# ASP.NET

**Unser Online-Tipp**
**für noch mehr Wissen ...**


... aktuelles Fachwissen rund
um die Uhr – zum Probelesen,
Downloaden oder auch auf Papier.

**www.InformIT.de**

PEARSON Education

ADDISON-WESLEY

Christian Wenz
Andreas Kordwig
Christian Trennhaus

# ASP.NET


ADDISON-WESLEY

An imprint of Pearson Education

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

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.ddb.de* abrufbar.



Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.

10 9 8 7 6 5 4 3 2 1

06 05 04 03

ISBN 3-8273-2048-8


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


| | |
|---|---|
| Einbandgestaltung: | Vera Zimmermann, Mainz |
| Lektorat: | Frank Eller, feller@pearson.de |
| Korrektorat: | Simone Meißner, Fürstenfeldbruck |
| Herstellung: | Philipp Burkart, pburkart@pearson.de |
| Satz und Layout: | mediaService, Siegen (www.media-service.tv) |
| Druck und Verarbeitung: | Bercker, Kevelaer |

Printed in Germany

# Inhaltsverzeichnis

# Vorwort

Vor etwa anderthalb Jahren arbeiteten wir am Vorgängertitel »Nitty-Gritty ASP«. Die Konzeption war damals relativ schnell erledigt. ASP hat einen recht beschränkten Funktionsumfang, weswegen wir innerhalb des vom Reihenkonzept vorgegebenen Rahmens alle wesentlichen Bestandteile kurz und knapp unterbringen konnten. Trotz des Taschenbuchformats und der Beschränkung auf nicht viel mehr als 400 Seiten ist dieser Titel eine echte Referenz, denn (fast) alles steht drin.

Als es etwas später um die Konzeption dieses Titels ging, war alles viel schwieriger. »Leider« – oder eher: glücklicherweise – ist der Funktionsumfang von ASP.NET um ein Vielfaches größer als der von ASP. Das sich damit ergebende Problem ist freilich, dass in gut 400 Seiten das Thema nicht erschöpfend behandelt werden kann. Es ist uns auch noch kein 1000-seitiges Buch in die Hände geraten, das dies adäquat erfüllt hätte. Die Konsequenz war klar: Für die Inhalte mussten wir eine Auswahl treffen. Wir haben also zunächst einmal im Rahmen unserer Tätigkeiten als Entwickler und Projektmanager ASP.NET ausgiebig in Real-World-Projekten getestet. So konnten wir die folgenden Punkte erarbeiten:

- Welche Funktionalität von ASP.NET wird wirklich häufig benötigt, welche dagegen nie oder offensichtlich nur in der Online-Dokumentation?
- Wo treten Probleme auf?
- An welchen Punkten wird am häufigsten in der Dokumentation nachgeschlagen oder die Suchmaschine angeworfen?

Die Auswertung dieser Informationen war dann die Basis unserer Kapitelauswahl. Wir haben also versucht, die wesentlichen, entscheidenden, häufig benötigten Elemente von ASP.NET unterzubringen, erheben allerdings keinen Anspruch auf Vollständigkeit. Anders gesagt: Wenn Sie überprüfen wollen, ob in diesem Buch etwas fehlt, werden Sie sicherlich fündig. Wenn Sie sich dann allerdings fragen, ob Sie die fehlende Technik schon häufig in der Praxis eingesetzt haben, werden Sie dies unserer Erwartung nach verneinen müssen.

Dieses Buch basiert bereits auf der brandneuen Version 1.1 des .NET Frameworks, vermutlich als eines der ersten Bücher weltweit. Glücklicherweise sind die Unterschiede zur Vorgängerversion 1.0 minimal. Deswegen lässt sich dieses Buch für beide Versionen benutzen; auf Unterschiede und Fallstricke weisen wir selbstverständlich hin.

Trotz unserer intensiven Recherche kann es natürlich sein, dass wir (und unsere Probanden) die eine oder andere Technik unter- oder überschätzt haben. Wenn Ihnen also tatsächlich etwas Wichtiges fehlt, schreiben Sie uns (Kontaktmöglichkeiten siehe weiter unten). Wenn Sie im Gegenzug etwas für überflüssig halten, melden Sie sich ebenfalls. Sie helfen dadurch, eine mögliche nächste Auflage dieses Titels zu formen.

Das Konzept der Nitty-Gritty-Reihe ist das folgende: Das Taschenbuch soll sowohl als Informationsquelle als auch als Nachschlagewerk dienen. Sie finden also sowohl Einführungen in verschiedene Themen als auch einen Referenzteil. Besonders wichtig ist der Praxisbezug und dass jede Erklärung zumindest mit einem kurzen Codebeispiel untermauert ist. Sie können dann diese Listingfragmente direkt in Ihre eigenen Anwendungen übernehmen und an Ihre Anforderungen anpassen. Aus diesem Grund haben wir von der beliebten, aber umstrittenen Methodik, viele komplexe Beispiele – dürftig erklärt – einzusetzen, Abstand genommen. Stattdessen bringen wir immer kleine, einfache Beispiele, dafür aber sehr viele. Leserfeedback zu anderen Titeln, insbesondere dem zuvor erwähnten »Nitty-Gritty ASP«, hat uns bestärkt, diesen Weg auch weiterhin einzuschlagen. Einer der Hauptvorteile ist, dass Sie die kleineren Beispiele einfach übernehmen und sich so baukastenartig eine Anwendung zimmern können; bei komplexen, zu speziell gehaltenen Beispielen sind Sie den Vorgaben der Autoren hilflos ausgeliefert.

Aus Platzgründen mussten wir auf den Abdruck einiger Kapitel verzichten. Diese finden Sie auf der Webseite des Buchs unter *www.nitty-gritty.de*. Ein Abdruck hätte den Rahmen des Buchs definitiv gesprengt.

## Gliederung

Allen Büchern der Nitty-Gritty-Reihe gemein ist der dreiteilige Aufbau:

- *Start up!* enthält eine Einführung in das Thema. In unserem Fall geht es um die für ASP.NET notwendigen Installationen, eine Einführung in die neue Sprache C# sowie in die Konstrukte von ASP.NET allgemein
- *Take that!* ist als Referenz gedacht. Hier sind die wichtigsten (sprich: am häufigsten eingesetzten) Klassen von ASP.NET vorgestellt und stets mit Beispielen untermauert. Sie erfahren hier nicht nur, wie Sie diese Klassen einsetzen, sondern können auch stets dort wie in einer Referenz nachschlagen.
- *Go ahead!*, der dritte Teil, ist stark praxisorientiert. Hier werden mehrere Aspekte und Techniken der täglichen ASP.NET-Programmierung dargestellt. Auch hier gilt wieder: Viele kleine Beispiele, direkt zum Einsatz in Ihren Anwendungen geeignet.

Noch eine Anmerkung zum ersten Teil. Wie bereits erwähnt setzen wir auf die Sprache C#, die Microsoft neu mit der Entwicklung des .NET Frameworks vorgestellt hat. Da steckt weder eine persönliche Präferenz noch ein tatsächlicher Vorteil noch Ideologie dahinter. Fakt ist nur, dass wir uns für eine Sprache entscheiden mussten. Der Hauptkonkurrent von C#, Visual Basic .NET (kurz: VB.NET), ist mindestens gleichwertig. Verschiedene Erhebungen kamen jedoch zu dem Ergebnis, dass zurzeit C# knapp die Nase vorn hat, was sich natürlich auch ändern kann.

## Kontakt und Support

Nach dem Kauf dieses Buches lassen wir Sie natürlich nicht alleine. Für Fragen, Anregungen und auch Fehlerberichte sind wir stets dankbar. Erfahrungsgemäß wird jede E-Mail innerhalb von zwei Werktagen beantwortet, häufig auch deutlich schneller. Wir bitten Sie lediglich, die folgenden »Spielregeln« einzuhalten:

- HTML-Mails werden stets zuletzt beantwortet.
- E-Mails mit Wichtigkeit »hoch« werden nach den HTML-Mails bearbeitet.

- Geben Sie bei Fehlerberichten/Fragen bitte immer neben dem Buchtitel auch die Seite an, auf die Sie sich beziehen.
- Helfen Sie uns, Ihnen zu helfen! Die Aussage »Beispiel X läuft nicht« ist meist wenig hilfreich (insbesondere deswegen, weil alle Beispiele mehrfach getestet worden sind). Stattdessen verraten Sie uns, welche Versionen von .NET und Betriebssystem Sie verwenden, wie Ihr Setup aussieht, welche Fehlermeldung Sie genau erhalten, ob Sie die Beispiele modifiziert haben und so weiter. Mit diesen Angaben lässt sich fast jedes Problem lösen.

Fragen, die über das Buchthema hinausgehen (beispielsweise bei allgemeinen Fragen zu ASP.NET oder Problemen mit selbst erstellten Skripten), sind in den ASP.NET-Newsgroups besser aufgehoben. In den Hierarchien *microsoft.public.dotnet.framework.aspnet.** (englischsprachig) bzw. *microsoft.public.de.german.entwickler.dotnet.** (deutschsprachig) finden Sie mehrere Newsgroups. Wenn Sie keinen Newsgroup-Reader besitzen, können Sie beispielsweise unter *http://groups.google.de/* Nachrichten lesen und auch verfassen.

Wenn Sie uns per E-Mail erreichen möchten, verwenden Sie die Adresse *asp.net@gmx.net*. Diese hat den Vorteil, dass Sie alle drei Autoren erreichen und somit der für das betreffende Kapitel zuständige auf jeden Fall Ihre Anfrage erhält. Außerdem finden Sie unter *http://www.hauser-wenz.de/support/* alle Beispiele und Listings aus diesem Buch als Download vor. Unter *http://www.nitty-gritty.de/* erhalten Sie allgemeine Informationen zur Reihe, inklusive einer Übersicht aller weiteren Reihentitel.

Abschließend noch ein Zitat aus einer Speisekarte eines bürgerlich-bayerischen Restaurants: Wenn es Ihnen nicht gefallen hat, sagen Sie's uns; wenn es Ihnen gefallen hat, sagen Sie's anderen. Also: Viel Vergnügen bei der Lektüre und viel Erfolg bei der Programmierung mit ASP.NET. Wir freuen uns auf Ihr Feedback.

Christian Wenz, Andreas Kordwig, Christian Trennhaus

München, März 2003

PS.: Wie üblich Dank an alle, die uns bei diesem Projekt unterstützt haben. Und vorsorglich »Nix für ungut« an all diejenigen, die wir in den Beispielen als Testdatensätze missbraucht haben.

# Teil I


# Start up!

# 1 Einführung

Bereits im Frühjahr 2001 stellte Microsoft seine neue Strategie ».NET« der Weltöffentlichkeit vor. Zunächst waren für Interessierte jedoch nur viele Marketinginformationen, aber keinerlei praktisch relevante Auskünfte über dieses ominöse .NET zu erhalten. Und auch jetzt erklärt Microsoft das Prinzip hinter .NET nicht so offensichtlich, dass die Vorteile dieser Strategie augenfällig sind.

## 1.1 .NET-Grundlagen

Hinter dem Schlagwort .NET verbergen sich zugleich Basisplattformen, Applikationsserver und eine vereinheitlichende Softwarearchitektur. Damit ist eine neue Plattform geschaffen worden, in der Anwendungen und Dienste integriert sind, wodurch verschiedenartigste Anforderungen immer auf eine einheitliche Basis zurückgreifen können. So umfasst .NET

- das .NET Framework
- .NET Enterprise Server
- Betriebssysteme für unterschiedliche Geräteplattformen
- Web Services
- und Entwicklungswerkzeuge zur Erstellung von .NET-Anwendungen.

Das wohl zentralste Element der .NET-Strategie ist das *.NET Framework*. Es stellt eine umfangreiche Klassenbibliothek dar, die unabhängig von der vom Programmierer eingesetzten Programmiersprache eine Vielzahl an Funktionalität zur Verfügung stellt.

Unter *.NET Enterprise Servern* fasst Microsoft alle diejenigen neuen Serverprodukte zusammen, die sich auf der gleichen Plattform basierend mit einheitlichen Mitteln individuell anpassen und erweitern lassen. Ein besonderes Merkmal dieser Generation von Servern ist auch ihre einfache Skalierbarkeit.

Um eine zentrale Klassenbibliothek auch sinnvoll nutzen zu können, wird dieses Framework in Zukunft in diverse *Betriebssysteme für unterschiedliche Geräteplattformen* integriert sein. Dadurch lassen

sich .NET Anwendungen dann durch einfaches Kopieren auf ein anderes System bereits installieren.

*Web Services* sind bereits grundlegend in die .NET-Architektur integriert, so dass sich Web Services sehr leicht erstellen und verarbeiten lassen. Web Services ist ein eigener Abschnitt im Praxisteil dieses Buches gewidmet, so dass an dieser Stelle nicht genauer darauf eingegangen wird.

Beinahe selbstverständlich ist es schon, dass Microsoft zu seiner Plattform passend *Entwicklungswerkzeuge* bereitstellt. So ist Visual Studio.NET eine umfassende Entwicklungshilfe zum Erstellen von .NET-Anwendungen, Web Matrix eine speziell auf Internet-Anwendungen zugeschnittenes Werkzeug, das zudem kostenlos ist.

Das Schlagwort .NET bezeichnet also eine gut durchdachte, integrierte Strategie, in der komplexe Anwendungen mit Hilfe großzügiger Standards leicht und vor allem nach einem einheitlichen Modus erstellt werden können. Einen besonders ansehnlichen Einblick in die .NET-Strategie bietet das folgende Diagramm der .NET-Architektur:



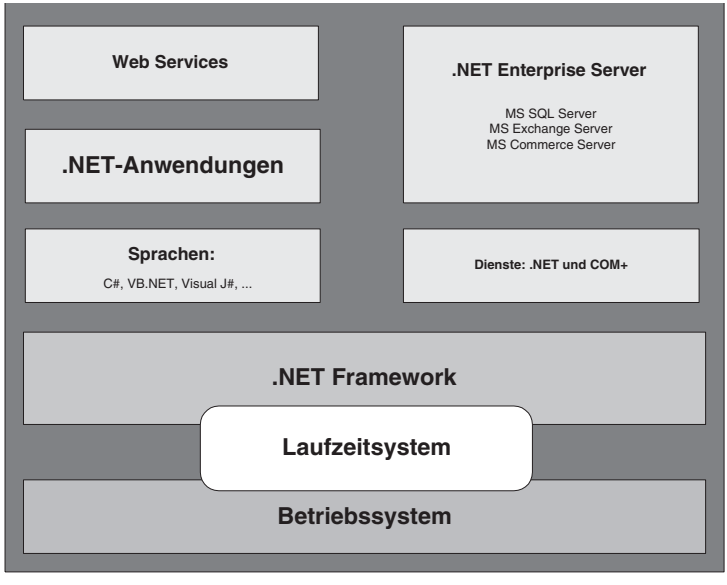



*Bild 1.1: Die .NET-Architektur*

Wie Sie sehen, ist diese Architektur nicht auf eine spezielle Programmiersprache ausgelegt. Die .NET-Architektur können Sie mit jeder Programmiersprache nutzen, die mit dem .NET Framework arbeiten kann. Dazu gehören die Microsoft Programmiersprachen C#, VB.NET und Visual J# genauso wie Programmiersprachen unabhängiger Softwareanbieter. Beispiele hierfür sind Perl.NET oder auch COBOL.NET.

## 1.2 Das .NET Framework

Wie aus der .NET-Architektur deutlich hervorgeht, ist das .NET Framework die Basis jeder .NET-Anwendung. Dieses Framework selbst besteht wieder aus mehreren Komponenten:

- einer zentralen Klassenbibliothek
- der Laufzeitumgebung

**Die Klassenbibliothek**

Die zentrale Klassenbibliothek des .NET Frameworks bietet alle grundlegenden Funktionalitäten für die Programmierung – und das für klassische Windowsanwendungen genauso wie für Webapplikationen. Dies bietet gleich mehrere Vorteile:

Zum einen müssen Sie diese grundlegenden Funktionalitäten nicht erst selbst programmieren, so dass Sie sich auf die eigentliche Anwendung und nicht auf einzelne Grundelemente konzentrieren können. Dies führt direkt zu einer Verringerung der Implementationszeit. Und ganz nebenbei wird Ihr Code von unnötigem Ballast befreit und deutlich übersichtlicher.

Des Weiteren stehen diese Basisklassen natürlich nicht nur C#, sondern jeder Programmiersprache innerhalb des .NET Frameworks zur Verfügung. Genau so, wie Sie unter C# einzelne Klassen und Methoden benutzen und ansprechen, würde das auch mit VB.NET oder Visual J# gemacht werden. Einzelne Anwendungen können dadurch bei Bedarf schnell und leicht von einer in eine andere .NET-Programmiersprache umgeschrieben werden.

Die zentrale Klassenbibliothek ist in mehrere logische Abschnitte unterteilt. Ein einzelner Abschnitt wird von Microsoft mit *Namespace* bezeichnet. Innerhalb eines solchen Namespaces sind dann alle zu

einem Thema gehörenden Klassen und Methoden zusammengefügt. So enthält der Namespace `System.Text` beispielsweise Klassen und Methoden zur Manipulation von Zeichenketten, der Namespace `System.XML` Funktionalität zur Zusammenarbeit mit XML-Dokumenten. Näheres zu den einzelnen Namespaces erfahren Sie in den Abschnitten zwei und drei dieses Buchs.

### Die Laufzeitumgebung

C# ist genau wie alle anderen .NET-Programmiersprachen eine kompilierte Sprache, so dass der nach dem Kompilieren entstandene Code innerhalb einer Laufzeitumgebung ausgeführt werden muss. War bislang je Programmiersprache noch eine spezifische Laufzeitumgebung erforderlich, so bietet das .NET Framework eine so genannte *Common Language Runtime* – eine Laufzeitumgebung für alle .NET-Programmiersprachen. Um eine .NET-Anwendung auf einen neuen Server zu portieren, müssen Sie lediglich sicherstellen, dass auf diesem Server die gemeinsame Laufzeitumgebung als Bestandteil des .NET Frameworks installiert ist, eine sprachenspezifische Erweiterung ist nicht erforderlich.



Aber die gemeinsame Laufzeit bringt noch einen weiteren, großen Vorteil mit sich: Objekte, die in unterschiedlichen Programmiersprachen erstellt wurden, laufen nicht nur vollkommen problemlos nebeneinander ab, sondern können sehr einfach miteinander kommunizieren. Denn durch die gemeinsame Laufzeitumgebung können Sie von jeder Programmiersprache aus auf alle anderen Objekte innerhalb der Laufzeitumgebung zugreifen, vollkommen unabhängig von der Programmiersprache, in der einzelne Objekte erstellt wurden. Dies ermöglicht eine neue, wesentlich erleichterte Zusammenarbeit in größeren Projekten.

Der Grund dafür, dass verschiedene Programmiersprachen so einfach zusammen in einer gemeinsamen Laufzeitumgebung miteinander operieren, liegt in dem Konstrukt der *Intermediate Language*. Wird ein .NET-Programm kompiliert, so wird es nicht direkt in einen nativen Maschinencode, sondern in eine Zwischensprache übersetzt. Diese Zwischensprache (oder eben englisch *intermediate language*) ist für alle .NET-Programmiersprachen dieselbe. Es bleibt sich daher gleich, in welcher Hochsprache ein Objekt erstellt wurde, denn innerhalb dieser Zwischenschicht sind alle Objekte in einer einzigen Sprache vorhanden.

Code in einer Zwischensprache lässt sich wie Sie sicher wissen nicht ausführen. Daher sorgt ein *Just-in-Time*-Compiler (JIT) dafür, dass jedes Objekt, das in der Intermediate Language vorliegt, bei Bedarf in Maschinencode kompiliert wird. Dieser Maschinencode bleibt dann im Arbeitsspeicher des Systems so lange resistent, bis eine neuere Version des Objekts vorliegt. Der große Vorteil gerade gegenüber anderen Programmiersystemen ist, dass das Vorhalten eines kompilierten Objekts im Speicher enorme Performancevorteile bringt. Denn gerade bei komplexeren Objekten kann die Kompilierung durchaus aufwändig sein, so dass man diesen Vorgang sinnvollerweise nur einmal durchführt.

Im Vergleich zu klassischem Java bietet die .NET-Plattform mit den Konstrukten der Intermediate Language den Vorteil, dass verschiedene Programmiersprachen in einer gemeinsamen Laufzeitumgebung performant ausgeführt werden, denn .NET-Code wird grundsätzlich kompiliert und nicht nur interpretiert.



## 1.3 Was ist ASP.NET?

Wann immer Sie eine dynamische Website erstellen möchten, müssen Sie zunächst eine Frage klären: Soll der dynamische Teil Ihrer Anwendung auf dem Client – also dem Browser – ausgeführt werden, oder soll Ihre Anwendung serverseitig mit dynamischen Elementen ergänzt werden?

Beide Ansätze haben ihre Anwendungsfälle. Wenn Sie z.B. ein kleines Spiel anbieten möchten, das im Browser gespielt wird, so bietet sich an, dieses Spiel mit Hilfe eines Flash-Plugins clientseitig ausführen zu lassen. Möchten Sie allerdings eine Intranet-Applikation erstellen, auf deren Homepage automatisch immer die neuesten Branchenneuigkeiten zu lesen sind, so macht es mehr Sinn, diese Neuigkeiten zentral zu pflegen und serverseitig in die Homepage zu integrieren.

ASP.NET bietet nun die Möglichkeit, Webapplikationen serverseitig dynamisch zu gestalten. Das Besondere an ASP.NET ist hier, dass dabei Microsofts .NET-Technologie verwendet wird und somit eine Reihe von neuen Möglichkeiten in die Erstellung dynamischer Websites mit einfließt.

## 1.4 ASP und ASP.NET: Wo liegt der Unterschied?

Wenn Sie bislang Erfahrungen mit ASP gesammelt haben, stellt sich sicher die Frage, wo nun der Unterschied zwischen ASP und ASP.NET liegt. Wenngleich der Name dieser beiden Technologien noch sehr ähnlich klingt, so sind ASP und ASP.NET doch sehr verschieden.

**Objektorientierung**

Der beim Programmieren wohl offensichtlichste Unterschied zwischen ASP und ASP.NET liegt in der stark objektorientierten Arbeitsweise, die mit ASP.NET gefordert wird. Dies liegt einfach daran, dass fast alle Funktionen, die für das Erstellen einer dynamischen Website erforderlich sind, als Methoden von Klassen des .NET Frameworks vorhanden sind. Gab es bei ASP lediglich fünf Objekte, so hat ASP.NET mit dem .NET Framework ein Vielfaches dessen. So werden Sie sich also in der Programmierung von Anwendungen etwas umstellen müssen. Unserer Erfahrung nach ist dieser Schritt aber sehr leicht vollzogen. Im Gegenzug gewinnen Sie mit ASP.NET-Code, weil er deutlich übersichtlicher und strukturierter ist als klassischer ASP-Code.



**Wahl der Programmiersprache**

ASP-Dokumente bestehen aus HTML-Code und Teilen aus VBScript oder JScript. Um eine ASP.NET-Anwendung zu erstellen, benötigen Sie neben den Kenntnissen in HTML noch Know-how in einer der .NET-Programmiersprachen. Sie können Ihre Anwendung also in C# oder VB.NET, genauso aber auch in J#, PHP.NET oder Corba.NET erstellen. Wie Sie bereits wissen, wird aus all diesen Programmiersprachen der gleiche Zwischencode erzeugt, so dass die Wahl der Programmiersprache für den einzelnen Programmierer wirklich nur auf dessen Vorkenntnissen beruhen muss.

**Einheitliche Basis auch für Backend-Operationen**

Auch wenn Sie innerhalb einer ASP.NET-Anwendung unterschiedliche Programmiersprachen einsetzen könnten, müssen Sie dies selbstverständlich nicht. Im Gegenteil: Sie können mit Hilfe des .NET Frameworks nun auch viele Backend-Operationen direkt mit Ihrer für die Webapplikation verwendeten Programmiersprache ausführen. Denn wofür unter ASP noch einzelne *.dll*-Dateien mit Hilfe einer betriebssys-

temnahen Sprache wie C++ erforderlich waren, können Sie heute mit Hilfe des Frameworks und den angeschlossenen Application Servern sehr einfach Backend-Routinen ausführen. In den meisten Fällen reicht es, dazu eine entsprechende Klasse des .NET Frameworks zu nutzen.

**Performance**

ASP.NET bietet gegenüber klassischem ASP einen deutlichen Performancegewinn. Ein Großteil davon rührt daher, dass der Code, den Sie in Ihrer ASP.NET-Anwendung erstellt haben, nicht nur interpretiert, sondern kompiliert und ausgeführt wird. Durch die Verwendung der Intermediate Language und des Just-in-Time-Compilers des .NET Frameworks sind die erforderlichen Kompilierungsvorgänge nicht zu langwierig, und der Geschwindigkeitsvorteil einer kompilierten Anwendung gegenüber einer interpretierten kann voll greifen.

**Einfache Portierung**

Wer kennt das nicht: Eine komplexe, dynamische Website wurde unter hohem Aufwand auf einem Entwicklungs- und Testsystem entwickelt. Der Kunde ist zufrieden, und die Anwendung soll auf die Produktionssysteme portiert werden. Unter ASP müssen Sie jetzt dafür sorgen, dass alle Zusatzkomponenten und eigene dll-Dateien auf dem Produktionssystem installiert und registriert werden. Gerade aber das Installieren von zusätzlichen Komponenten auf einem bereits produktiven Server wird oft nur schwer oder schlimmstenfalls sogar gar nicht gestattet. Um die Anwendung doch noch nutzen zu können, sind entweder eigenständige Systeme oder viele Sitzungen mit der Administrationsgruppe erforderlich.

Mit ASP.NET ist eine Portierung von einem auf ein anderes System ganz einfach. Da das .NET Framework als Basis dient, sind keinerlei Zusatzkomponenten mehr erforderlich – denn diese sind ja bereits im .NET Framework integriert. Dadurch wird das Migrieren einer Anwendung von Test- auf Produktionssysteme im Normalfall ein einfaches Kopieren von Anwendungsdateien. Es ist nicht erforderlich, diese auf dem Zielsystem zu registrieren. Sie können die neue Anwendung sofort ansprechen.

## 1.5 Die Architektur von ASP.NET

Wie Sie bereits gelesen haben, arbeitet ASP.NET eng mit dem .NET Framework. Aber wie wird eine Seite der Anwendung denn nun genau erzeugt?

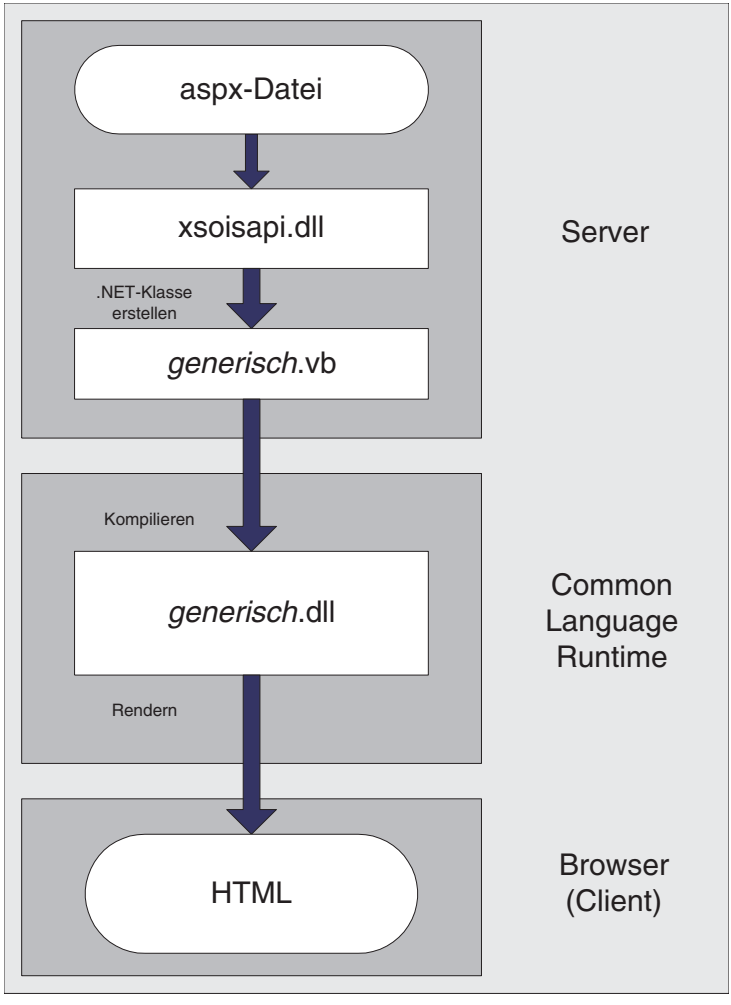



*Bild 1.2: Die ASP.NET-Architektur*

Fragt ein Besucher Ihrer Website ein ASP.NET-Dokument an, so wird diese Anfrage an eine ISAPI-Erweiterung des Webservers weitergeleitet. Diese Erweiterung des Webservers überprüft dann, ob der in der angefragten Seite enthaltene ASP.NET-Code bereits von einem anderen Benutzer aufgerufen wurde. Hat sich der Quellcode der ASP.NET-Seite im Vergleich zum letzten Aufruf nicht verändert, so ist die bereits in die Intermediate Language vorkompilierte Version des Dokuments noch aktuell, und sie kann erneut verwendet werden. Diese Vorgehensweise stellt sicher, dass auf der einen Seite immer nur der aktuellste Quellcode zur Ausführung kommt, auf der anderen Seite aber keine überflüssigen Kompilierungsvorgänge gestartet werden. Ist der Quellcode im angeforderten Dokument nämlich nicht mit dem des letzten Aufrufs identisch, wird das Dokument in die Intermediate Language vorkompiliert und die dort verfügbare alte Version dieses Dokuments überschrieben. Die nun im Speicher vorhandene Version des Dokuments wird über den JIT-Compiler ausgeführt und das Ergebnis dieses Vorgangs an den Besucher der Site zurückgesendet.

ASP.NET ist also ein deutlicher Fortschritt im Vergleich zu klassischem ASP. Die Performance ist höher, das Programmieren oft einfacher und auch übersichtlicher. Zudem lassen sich andere Anwendungen nun deutlich leichter in eine Website integrieren.

# 2 Installation

Um eine ASP.NET-Anwendung entwickeln und betreiben zu können, sind zwei Voraussetzungen zu erfüllen. Sie benötigen

- einen Webserver
- und das .NET Framework.

Dass ein Webserver nötig ist, um eine Webanwendung zu entwickeln und zu beitreiben, ist offensichtlich. Das .NET Framework ist, wie Sie aus der Einführung wissen, das zentrale Element der .NET-Plattform – und da die bislang verfügbaren Betriebssysteme dieses noch nicht integriert haben (in Microsofts demnächst verfügbarer Server-Produktfamilie der .NET-Server wird das Framework bereits inbegriffen sein), ist es im Nachhinein zu installieren. Microsoft stellt das .NET Framework für die aktuellen Betriebssysteme kostenlos zum Download bereit, für andere Betriebssysteme ist zumindest bislang keine Portierung geplant.



## 2.1 Einrichten einer Entwicklungsumgebung

Um also eine Entwicklungsumgebung für ASP.NET einzurichten, benötigen Sie zunächst einen Webserver für Ihre Windowsplattform, der dann im Anschluss vom .NET Framework ergänzt wird. Mit Windows 2000 und Windows XP wird bereits ein geeigneter Webserver ausgeliefert, der nun einfach nur noch installiert werden muss. Im Anschluss kann das .NET Framework installiert werden.

### 2.1.1 Voraussetzungen

Bevor Sie mit der Installation starten, sollten Sie überprüfen, ob Ihr Rechner die Mindestanforderungen von Webserver und .NET Framework erfüllt.

Speziell für die Softwareentwicklung ist es das .NET Framework SDK (*Software Development Kit*), das die höchsten Ansprüche an Ihr System stellt. Das SDK ist es jedoch, welches dank mitgelieferter Dokumentation und im Sourcecode vorliegenden Beispielen ideal für die

Entwicklung ist. Sie können diese Software unter Windows 2000, Windows XP und auf den Windows Server 2003 installieren.

Mindestvoraussetzungen
für Windows 2000 Professional und Windows XP:

- Prozessor: Intel Pentium Klasse, 90 MHz oder schneller
- RAM: 32 MB (96 MB oder mehr sind dringend empfohlen)
- Freier Festplattenplatz während der Installation: 145 MB
- Für die Software erforderlicher Festplattenplatz: 111 MB
- Video: 800x600, 256 Farben

Mindestvoraussetzungen
für Windows 2000-Server und die .NET-Server:

- Prozessor: Intel Pentium Klasse, 133 MHz oder schneller
- RAM: 128 MB (256 MB oder mehr sind dringend empfohlen)
- Freier Festplattenplatz während der Installation: 145 MB
- Für die Software erforderlicher Festplattenplatz: 111 MB
- Video: 800x600, 256 Farben

Die Systemvoraussetzungen für dieses Softwarepaket sind gemessen an den heute üblichen PCs nicht sehr hoch und stellen daher in den seltensten Fällen eine Hürde dar. Beachten Sie jedoch, dass etwas mehr Arbeitsspeicher gerade aufgrund der relativ häufigen Kompiliervorgänge sicher von Vorteil ist.



### 2.1.2 Microsofts Webserver installieren

Wenngleich Microsoft mit den Betriebssystemen Windows 2000 Professional und Windows XP einen Webserver auf der Betriebssystem-CD-ROM ausliefert, so ist nach einer Standardinstallation dieser Betriebssysteme der Webserver aber noch nicht auf einen Rechner installiert. Dies lässt sich aber schnell und einfach nachholen. Starten Sie über START/SYSTEMSTEUERUNG die Verwaltungsübersicht SOFTWARE.

Klicken Sie hier auf die Schaltfläche WINDOWS-KOMPONENTEN HINZUFÜGEN/ENTFERNEN.

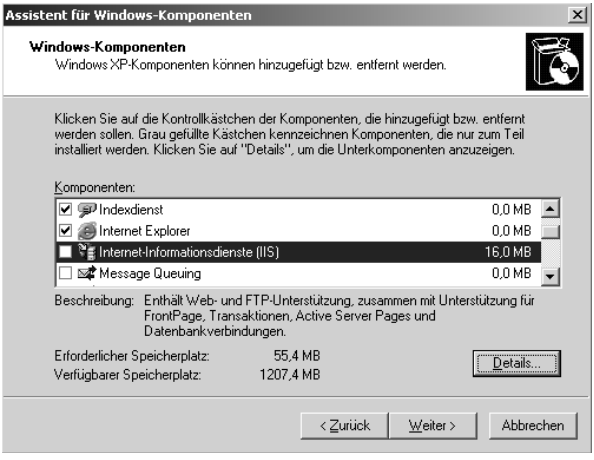



*Bild 2.1: Das Menü* WINDOWS-KOMPONENTEN HINZUFÜGEN/ENTFERNEN

Aus der Liste der hier wählbaren Programme selektieren Sie die Internet-Informationsdienste (IIS). Die Internet-Informationsdienste sind eine Gruppe von Programmen, die neben einem FTP- und einem Mailserver auch den Webserver mit einschließt. Klicken Sie jetzt auf WEITER.

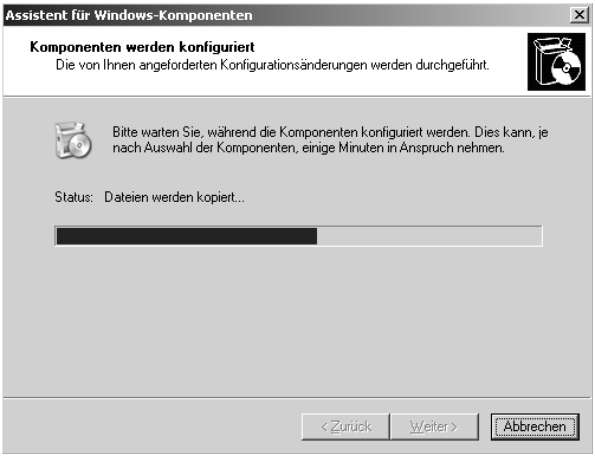



*Bild 2.2: Der* ASSISTENT FÜR WINDOWS-KOMPONENTEN *bei der Installation der Internet-Informationsdienste*

Der ASSISTENT FÜR WINDOWS-KOMPONENTEN installiert und konfiguriert nun den Microsoft Webserver IIS. Sobald dieser Schritt abgeschlossen ist, können Sie den Assistenten mit einem Klick auf FERTIG STELLEN beenden.


*Immer dann, wenn Sie Ihrem System neue Komponenten hinzufügen, sollten Sie über die Website http://www.windowsupdate.com nach aktuell verfügbaren Updates und Sicherheitspatches suchen. Gerade die Installation des Webservers ermöglicht Hackern einen potentiell leichteren Weg in Ihr System, so dass Sie hier besonders sorgsam aktuell verfügbare Patches einspielen sollten. Dazu gehört es auch, dass Sie auf Ihrem Rechner stets das aktuelle Service Pack installiert haben, denn dieses schließt eine Menge von Sicherheitslücken.*

Um Ihren Webserver zu verwalten, wurde während des Installationsvorgangs eine Steuerzentrale für die Webdienste eingerichtet. Sie erreichen diese Oberfläche über START/SYSTEMSTEUERUNG/VERWALTUNG/INTERNET-INFORMATIONSDIENSTE (bzw. in der Windows XP Kategorieansicht über START/SYSTEMSTEUERUNG/LEISTUNG UND WARTUNG/VERWALTUNG/INTERNET-INFORMATIONSDIENSTE).

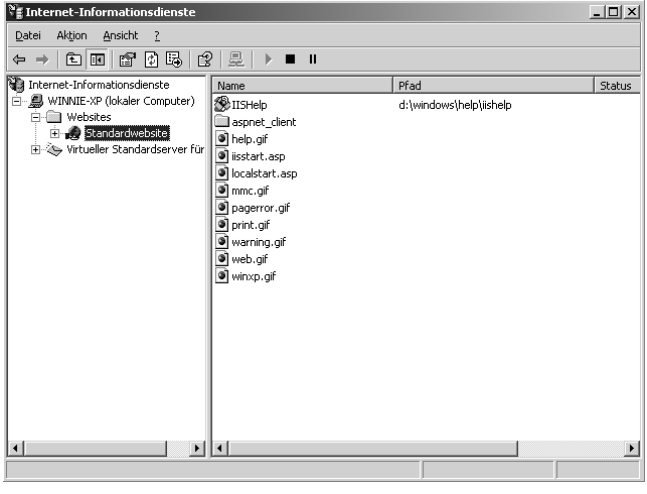



*Bild 2.3: Die Verwaltungsoberfläche der Internet-Informationsdienste*

In dieser Verwaltungsoberfläche können Sie den Webserver starten und stoppen, virtuelle Pfade und sogar virtuelle Webserver einrichten. Für die meisten Entwicklungsrechner reicht jedoch die Standardkonfiguration aus, so dass Sie hier nichts optimieren müssen.

### 2.1.3 Treiber für die Datenbankverbindungen aktualisieren

Um mit ASP.NET Datenbanken und XML-Dateien ansprechen zu können, müssen vor der Installation des .NET Frameworks die Treiber für Datenbankverbindungen aktualisiert werden. Microsoft stellt diese Treiber in einem Paket, den sog. Microsoft Data Access Components (MDAC), zur Verfügung. Unter Windows 2000 sollten Sie die Version 2.7 dieses Pakets installieren, Windows XP enthält diese Treiber bereits.

Die Microsoft Data Access Components können Sie unter der Adresse *http://www.microsoft.com/data/download_270RTM.htm* herunterladen. Wählen Sie dazu am unteren Ende der Seite die Sprache Ihres Betriebssystems aus und speichern Sie die zum Download angebotene Datei auf Ihrem PC.

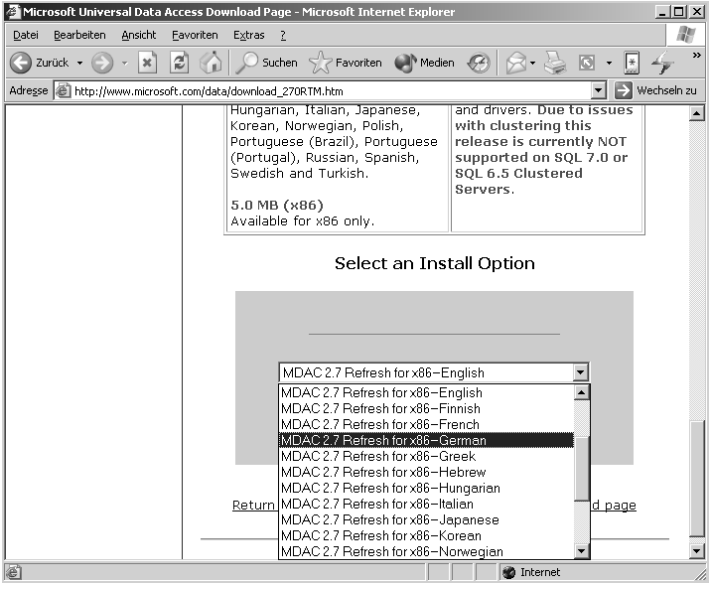



*Bild 2.4: Download der Data Access Components*

Sobald der Download abgeschlossen ist, starten Sie mit einem Doppelklick auf die neue Datei den Installationsvorgang. Nach dem Entpacken der Dateien nehmen Sie die Lizenzbedingungen an und klicken auf WEITER.

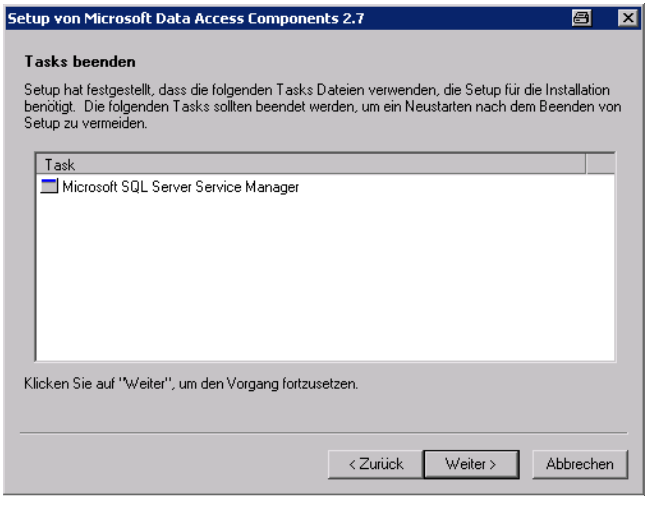



*Bild 2.5: Hinweis auf zu beendende Tasks als Vorbereitung zur Installation der Data Access Components*

Die Setup-Routine überprüft dann selbstständig die Systemumgebung und weist auf evtl. zu beendende Programme und Tasks hin.

HiN/WEiS *Sie müssen die von der Installationsroutine angemerkten Tasks nicht zwingend beenden, um die Installation vollziehen zu können. Wenn Sie Programme oder Tasks jedoch nicht beenden, dann müssen Sie vor der Installation des .NET Frameworks den Rechner zwingend neu starten.*

Klicken Sie auf WEITER und anschließend auf FERTIG STELLEN, so dass die Installation abgeschlossen werden kann. Nach dem Kopieren der erforderlichen Komponenten beenden Sie das Setup mit einem weiteren Klick auf SCHLIESSEN.

### 2.1.4 Das .NET Framework SDK installieren

Die aktuelle Version des .NET Framework SDK erhalten Sie ebenfalls frei zum Download. Unter *http://www.microsoft.com/germany/ms/entwicklerprodukte/downloads.htm* finden Sie eine Übersicht aller rund um .NET und das Framework verfügbaren Downloads. Wählen Sie hier das aktuelle .NET Framework SDK aus, und laden Sie dieses und evtl. verfügbare Service Packs auf Ihren Rechner. Achten Sie darauf, dass Sie nicht die weitervertreibbare Version des Frameworks wählen, da diese keinerlei Dokumentation enthält und allein schon deswegen zum Entwickeln wenig geeignet ist. Da das SDK insgesamt mehr als 130 MB groß ist, sollten Sie dieses jedoch über eine schnelle Verbindung zum Internet herunterladen – andernfalls ist dieser Download wohl eher eine Geduldsprobe.


*Alle Beispiele in diesem Buch wurden unter der Verwendung der Version 1.1 des .NET Frameworks entwickelt und getestet. Diese Version war vor der Drucklegung nur als Beta-Version verfügbar. Der Beispielcode ist selbstverständlich abwärtskompatibel zur Version 1.0, auf etwaige Unterschiede zwischen den Versionen weisen wir Sie an entsprechender Stelle hin.*

*Wenn Sie ebenfalls diese Vorabversion des .NET Frameworks installieren möchten, müssen Sie etwas anders als im Folgenden beschrieben vorgehen: Laden Sie die beiden unter http://msdn.microsoft.com/netframework/productinfo/next/download.asp angebotenen Softwarepakete herunter und installieren Sie zunächst das .NET Framework 1.1 Redistributable Beta (sofern noch nicht die finale Version erschienen ist). Erst im Anschluss an dieses Paket lässt sich das .NET Framework SDK 1.1 Beta einrichten. Die Vorgehensweise zum Aufsetzen des SDK 1.1 erfolgt analog zu der im Folgenden beschriebenen Installation des .NET Frameworks 1.0.*

Nach dem erfolgreichen Download starten Sie die Installation des .NET Framework SDK mit einem Doppelklick auf die Installationsdatei. Bestätigen Sie durch einen Klick auf JA im ersten Installations-Pop-Up, dass Sie die Installation jetzt starten wollen.

Nachdem die Setup-Routine gestartet wurde, klicken Sie zunächst auf WEITER, nehmen dann die Lizenzbedingungen an und klicken erneut auf WEITER. Im Anschluss können Sie die zu installierenden Komponenten wählen:


*Bild 2.6: Installation des .NET Framework SDKs*

Es empfiehlt sich, neben dem SDK auch die verfügbaren Beispiele zu installieren, da sich darunter auch eine Reihe von nützlichen ASP.NET-Beispielen befindet. Fahren Sie mit der Installation fort, indem Sie auf WEITER klicken. Wählen Sie im nächsten Fenster den Dateipfad, in dem die Dateien des .NET Frameworks installiert werden sollen, und setzen Sie die Installation erneut durch einen Klick auf WEITER fort. Komponenten und Beispiele des .NET Framework SDK werden nun installiert. Nachdem alle erforderlichen Komponenten auf Ihrem Rechner konfiguriert sind, erscheint ein Pop-Up, das Ihnen die Fertigstellung des Installationsvorganges meldet. Schließen Sie dieses Pop-Up mit einem Klick auf OK.


*Auch zu den Versionen des .NET Frameworks veröffentlicht Microsoft regelmäßig Updates und komplette Service Packs. Um evtl. Sicherheitslücken und Fehler zu vermeiden, sollten Sie regelmäßig auch diese Patches installieren. Eine Liste der aktuell verfügbaren und auf Ihrem Rechner noch nicht installierten Updates erhalten Sie wie gewohnt über die Website http://www.windowsupdate.com/.*

Zusammen mit dem .NET Framework SDK sind eine Reihe von Tools ausgeliefert worden, von denen besonders die *QuickStart Tutorials* sehr lehrreiche Beispiele und eine Menge Quellcode enthalten. Sie können diese Beispiele installieren, indem Sie unter START/PROGRAMME/MICROSOFT .NET FRAMEWORK SDK die Übersicht MICROSOFT .NET FRAMEWORK SDK QUICKSTARTS, TUTORIALS AND SAMPLES öffnen und den dort beschriebenen Anweisungen zur Installation der Quickstarts folgen. Eines der hilfreichsten Beispielprogramme, das mit den Quickstarts eingerichtet wird, ist der *.NET Framework Class Browser*. Diese Anwendung ist eine Kurzreferenz zu allen im Framework enthaltenen Klassen.

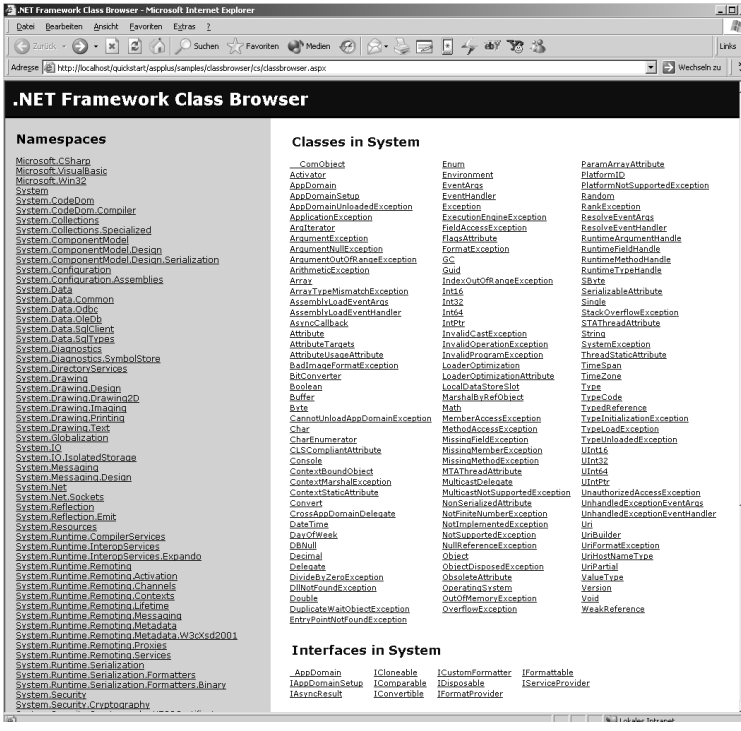



*Bild 2.7: Der .NET Framework Class Browser*

## 2.2 Produktivsysteme vorbereiten

Zwischen einem Entwicklungs- und einem Produktivsystem gibt es grundsätzlich einige wesentliche Unterschiede. So versucht man, auf einem Produktivsystem möglichst wenige Programme zu installieren, indem man sich auf die wirklich notwendigen Komponenten beschränkt. Der Vorteil dieser Vorgehensweise liegt darin, dass alle Ressourcen des Rechners von wenigen Programmen genutzt werden können. Zudem ist ein System mit wenigen Komponenten generell einfacher zu handhaben und meist sogar stabiler. Diesem Prinzip sollten auch Sie bei der Vorbereitung des Produktivsystems für ASP.NET-Anwendungen Folge leisten.

Installieren Sie auf einem Produktivsystem daher den Microsoft Webserver IIS und die aktualisierten Datenbanktreiber wie im vorangegangenen Abschnitt beschrieben. Da Sie auf dem Produktivsystem jedoch nicht entwickeln wollen, benötigen Sie als .NET-Grundlage nicht das gesamte Framework SDK. Stattdessen reicht die weitervertreibbare Version des .NET Frameworks vollkommen aus, die Sie ebenfalls auf der Seite *http://www.microsoft.com/germany/ms/entwicklerprodukte/downloads.htm* zum Download verlinkt finden. Diese Version des .NET Frameworks enthält alle zum Betrieb einer .NET-Anwendung erforderlichen Bestandteile, ist jedoch um jegliche Entwicklungstools und Dokumentation erleichtert.

Die Installation des weitervertreibbaren .NET Frameworks geschieht analog zur Installation des kompletten SDK, nur dass Sie nicht zwischen verschiedenen Installationsoptionen wählen können. Auf eine detaillierte Beschreibung wird hier daher verzichtet.

# 3 »Hallo Welt«

Der Begriff »Hallo Welt« hat historische Gründe. Am Anfang eines fast jeden didaktischen Konzepts zur Erklärung einer Programmiersprache steht ein einfaches Beispiel. Dieses Beispiel muss zwei Anforderungen erfüllen:

- Es muss möglichst einfach und übersichtlich sein.
- Damit das Ergebnis des Programms kontrolliert werden kann, ist eine Ausgabe Pflicht.

Die logische Konsequenz ist ein Programm, das nur ein Kommando ausführt: die Ausgabe eines Textes. Und irgendein Programmierer und/oder Autor hatte einst die Idee, als Text »Hello World« auszugeben, was von vielen anderen genauso übernommen worden ist. In einem deutschen Buch können Sie natürlich mit deutschen Textausgaben rechnen, in diesem Fall also: »Hallo Welt«.

Das Ziel dieses Kapitels ist also das Folgende: Sie erstellen eine simple Anwendung, die einen Text ausgibt. Dieses Beispiel wird in vielen der folgenden Kapitel als Ausgangsbasis verwendet. Dort wird dann zumeist anstelle eines statischen Textes ein berechneter Wert verwendet.

## 3.1 Eine einfache Konsolenanwendung

Beginnen wir mit dem einfachsten Fall – einer Windows-Konsolenanwendung. Da sich dieses Buch um ASP.NET dreht, wird dies das einzige derartige Beispiel bleiben.

Der folgende Code – dessen Interna sich Ihnen im Laufe der Spracheinführung vollkommen offenbaren werden – befindet sich im Web-Codearchiv unter dem Dateinamen *HalloWelt.cs*. Der entscheidende Code steht in Zeile 6: Hier wird der Text »Hallo Welt!« ausgegeben.

```
public class HalloWelt
{

  public static void Main()
  {
    System.Console.WriteLine("Hallo Welt!");
  }

}
```

*Listing 3.1: »Hallo Welt« als Konsolenanwendung (HalloWelt.cs)*

Um die Datei zu kompilieren (und daraus eine ausführbare *.exe*-Datei zu erzeugen), müssen Sie den C#-Compiler anwerfen. Dieser heißt *csc.exe* und befindet sich standardmäßig im Verzeichnis *%windir%\Microsoft.NET\Framework\vX.Y.ZZZZ*, also beispielsweise *C:\Windows\Microsoft.NET\Framework\v1.1.4322*. Normalerweise trägt das Installationsprogramm des .NET Frameworks dieses Verzeichnis automatisch in den Pfad ein, sodass ein einfacher Aufruf von *csc* genügt.

Dieser Compileraufruf hat die folgende Syntax:

```
csc /target:exe Datei.cs
```

Durch den Schalter `/target:exe` (oder kurz: `/t:exe`) sorgen Sie dafür, dass eine ausführbare Datei erzeugt wird (eine Alternative wäre beispielsweise eine Bibliothek bzw. DLL).


*Da die Standardeinstellung des Compilers jene ist, ausführbare Dateien zu erzeugen, genügt auch folgende Kurzform:*

```
csc Datei.cs
```


*Vorsicht, wenn Sie auch mit Visual Basic .NET arbeiten: Der dort verwendete Compiler, vbc.exe, hat eine andere Parameterreihenfolge: Erst der Dateiname, dann die Schalter:*

```
vbc Datei.vb /target:exe
```

Nach dem Kompilierungsvorgang befindet sich im aktuellen Verzeichnis die erzeugte *.exe*-Datei, im Hallo-Welt-Beispiel also *HalloWelt.exe*. Wenn Sie diese Datei ausführen, wird auch tatsächlich »Hallo Welt!« auf dem Bildschirm ausgegeben (siehe Bild 3.1).

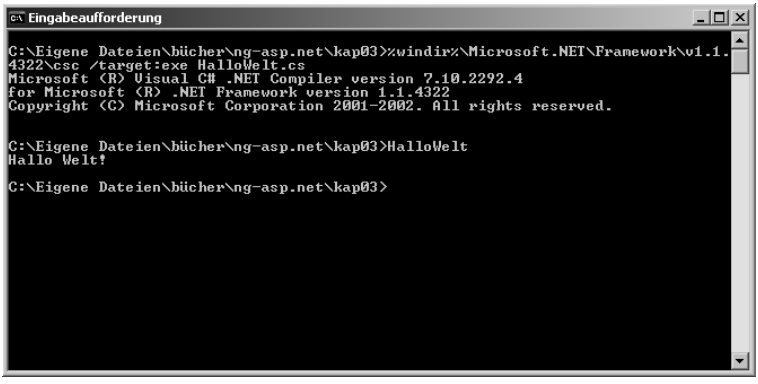



*Bild 3.1: Die Konsolenanwendung wird kompiliert und ausgeführt*


*Sie können den erzeugten Dateinamen auch direkt angeben, wenn Sie mit der Vorgabe – Skriptname mit Endung .exe statt .cs – nicht einverstanden sind:*

```
csc /out:AndererName.exe HalloWelt.cs
```

*Dieses Kommando erzeugt die ausführbare Datei AndererName.exe.*

Obwohl es sich um ein sehr einfaches Beispiel handelt, ist diese Datei nur auf Systemen ausführbar, auf denen das .NET Framework auch installiert ist. Der Versuch, die *.exe*-Datei auf einem System ohne .NET aufzurufen, führt zu einer kryptischen Fehlermeldung, wie in Bild 3.2 zu sehen. Je nach Version des Betriebssystems und des .NET Frameworks sieht die etwas anders aus.

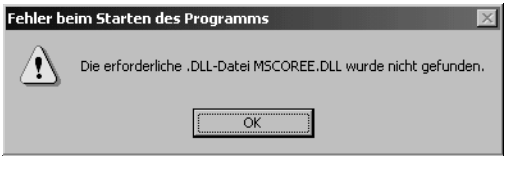



*Bild 3.2: Eine der möglichen Fehlermeldungen ohne .NET Framework*

Weitere potenzielle Stolperfalle: Die verschiedenen Versionen des .NET Frameworks sind nicht zueinander aufwärtskompatibel. Wenn Sie die Anwendung unter Framework-Version 1.1 kompilieren und die *.exe*-Datei dann auf einem System mit Version 1.0 zur Ausführung bringen möchten, erhalten Sie die Fehlermeldung aus Bild 3.3.

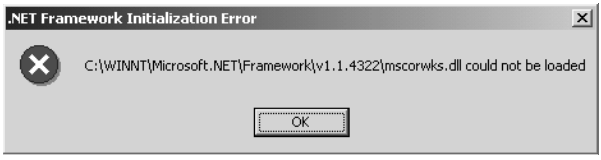



*Bild 3.3: Die installierte .NET-Version ist zu alt*

Kleines Trostpflaster: Die neueren Versionen des .NET Framework sind abwärtskompatibel, sprich: Wenn Sie ein Programm mit Version 1.0 kompilieren, läuft es auch mit Version 1.1.



## 3.2 Eine einfache ASP.NET-Webseite

Im Web ist die Situation natürlich einfacher, denn die Benutzer benötigen nur einen Webbrowser, das .NET Framework ist lediglich auf dem Webserver erforderlich. Die zu ASP.NET-Seiten zugehörige Dateiendung ist *.aspx*, was auch schon vom Namen her eine Erweiterung des Vorgängers ASP (Dateisuffix: *.asp*) markiert.

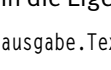
Eines der auffälligsten neuen Konzepte bei ASP.NET, insbesondere im Vergleich zu ASP, ist die mögliche komplette Trennung von Code und Content (Inhalt). Es gibt – bei entsprechender umsichtiger Programmierung – keine Vermischung von HTML- und Skriptcode mehr. Dies wird unter anderem dadurch erreicht, dass Sie in den Seitenkopf einer ASP.NET-Seite einen `<script>`-Bereich einfügen können. Durch das Attribut `runat="server"` geben Sie an, dass es sich hierbei um keine clientseitigen Skriptbefehle (etwa in JavaScript) handelt, sondern dass dieser Code auf dem Webserver ausgeführt wird:

```
<script runat="server">
...
</script>
```

Wie aber kann dieser Skriptbereich mit dem HTML-Anteil der Seite kommunizieren? Dazu gibt es zwei Möglichkeiten, HTML Controls und Web Controls. Ohne hier allzu weit vorzugreifen: Auch hier wird durch den Zusatz `runat="server"` ein Element auf einer HTML-Seite für den serverseitigen Zugriff freigeschaltet. Sie können dann aus dem `<script>`-Block heraus auf diese Elemente zugreifen und diese auch verändern.

Die folgende »Hallo Welt«-Seite macht genau das. Zunächst wird ein ASP.NET-Element, `<asp:Label>`, deklariert. Es erhält einen eindeutigen Bezeichner im `id`-Attribut sowie `runat="server"`:

```
<asp:Label id="ausgabe" runat="server" />
```




*Wie Sie sehen, wurde dieses Element XHTML-konform mit einem Schrägstrich abgeschlossen. Dies ist Pflicht, andernfalls erhalten Sie später eine ASP.NET-Fehlermeldung.*

Vom `<script>`-Block heraus wird nun auf dieses Element zugegriffen; in die Eigenschaft `Text` wird der entsprechende Wert geschrieben:

```
ausgabe.Text = "Hallo Welt!";
```

Nachfolgend der komplette Code. Sie finden hier noch einige zusätzliche Skriptelemente, die in den nachfolgenden Kapiteln erklärt werden:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  ausgabe.Text = "Hallo Welt!";
}
</script>
<html>
<head>
  <title>Hallo Welt!</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 3.2: »Hallo Welt« als ASP.NET-Seite (HalloWelt.aspx)*

*Beachten Sie die erste Zeile des Listings:*

```
<%@ Page Language="C#" %>
```

*Damit geben Sie an, dass es sich um ein C#-Script handelt. Die Standardsprache unter ASP.NET ist nämlich Visual Basic .NET, kurz VB.NET.*

*Um- bzw. Aufsteiger vom klassischen ASP sind gewohnt, dass Text auf eine andere Art und Weise ausgegeben wird:*

```
Response.Write("Hallo Welt!");
```

*Dies ist in ASP.NET natürlich auch möglich, durch die Trennung von Code und Content allerdings nicht mehr empfehlenswert und im professionellen Bereich auch nicht mehr in Gebrauch.*


*Bild 3.4: Die ASP.NET-Seite im Webbrowser*

Wenn Sie diese Seite in ein Verzeichnis unterhalb Ihres Webserver-Hauptordners (beispielsweise *c:\inetpub\wwwroot*) kopieren und im Webbrowser aufrufen, wird in der Tat »Hallo Welt!« ausgegeben. Im HTML-Quellcode der Seite tauchen dann der serverseitige Skriptblock und auch das `<asp:Label>`-Element nicht mehr auf:

```
<html>
<head>
  <title>Hallo Welt!</title>
</head>
```

```
<body>
<span id="ausgabe">Hallo Welt!</span>
</body>
</html>
```

Sie beobachten folgende Veränderungen:

- Der `<script>`-Block wurde entfernt.
- Das Element `<asp:Label>` wurde durch ein HTML-`<span>`-Element ersetzt.

Wenn Sie stattdessen eine Fehlermeldung im Webbrowser erhalten, ist etwas schief gegangen, womöglich haben Sie sich im Listing vertippt. Betrachten Sie beispielsweise Bild 3.5. Die (englischsprachige) Fehlermeldung sagt, dass »ausgabe« nicht gefunden werden konnte. Dieser Fehler passiert bei der händischen Erstellung von ASP.NET-Seiten relativ häufig – allzu oft wird nämlich einfach das `runat="server"` vergessen. Wenn dieser Parameter fehlt, kann das ASP.NET-Skript auch nicht serverseitig darauf zugreifen.

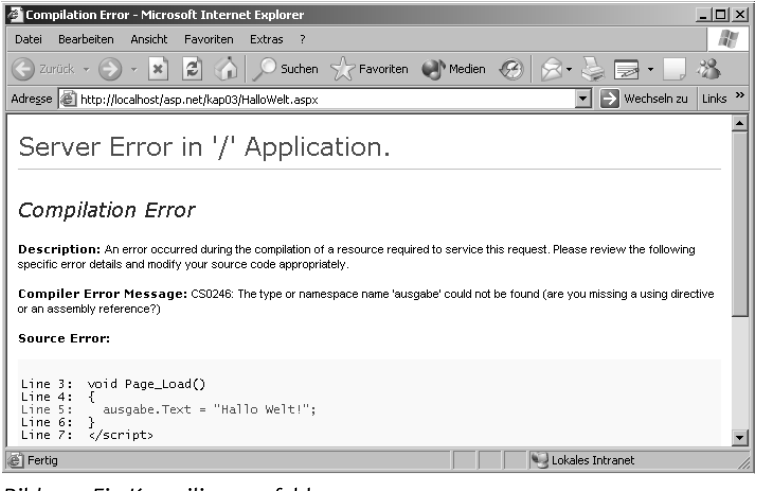



*Bild 3.5: Ein Kompilierungsfehler*

Einen weiteren »beliebten« Fehler können Sie Bild 3.6 entnehmen. Die Fehlerursache verbirgt sich hier in der Fehlernummer hinter Compiler Error Message. C#-Fehler beginnen immer mit CS. Das Präfix BC deutet darauf hin, dass der Visual Basic .NET-Compiler hier versucht hat, die Seite auszuführen. Höchstwahrscheinlich der Grund: Die Anweisung `<%@ Page Language="C#" %>` fehlt.

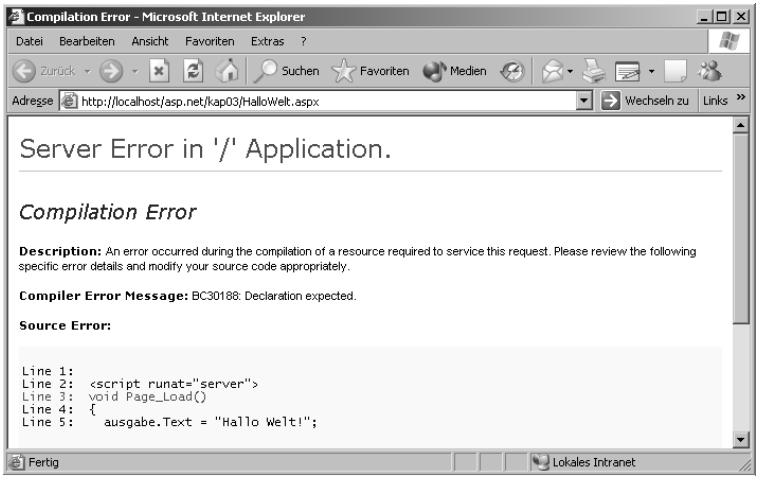



*Bild 3.6: Ein weiterer Kompilierungsfehler*

Obwohl die Hallo-Welt-Anwendung sehr einfach ist, dient sie doch als Ausgangsbasis für viele weitere kleine Beispiele in diesem Buch. In den folgenden Kapiteln erhalten Sie eine Spracheinführung in C#; dabei werden Sie jedoch hauptsächlich Webseiten erstellen und Ihre Ergebnisse im Browser betrachten. So müssen Sie sich nicht in der Kommandozeile die Finger wund tippen oder eine teure Entwicklungsumgebung anschaffen.

# 4 Einführung in C#

Mit Veröffentlichung der .NET-Strategie hat Microsoft auch gleich noch eine komplett neue Sprache vorgestellt: C# (gesprochen: C sharp). Der Sprachenname kann vielfältig interpretiert werden. Beispielsweise werden um einen Halbton erhöhte Noten mit einem Doppelkreuz (#) am Ende dargestellt; C# entspricht also dem Cis. Pikante weiterführende Interpretation: C# ist damit etwas höher angesiedelt als C. Die Verwandtschaft, Ähnlichkeit und gleichermaßen Weiterentwicklung der Programmiersprache C ist damit zum Ausdruck gebracht worden.

C# ist neben Visual Basic .NET Hauptsprache zur Programmierung im .NET Framework. Zwar stehen viele Alternativen zur Verfügung, beispielsweise JScript .NET, jedoch haben die nur einen sehr geringen Marktanteil. Anfangs war VB.NET klar der Sieger in Hinblick auf Verbreitung und Einsatz, insbesondere deswegen, weil alte ASP- und VB-Hasen sich natürlich Visual Basic .NET angenommen haben. Mittlerweile hat C# jedoch deutlich aufgeholt, auch ein Zeichen dafür, dass viele Quereinsteiger und Programmierer von anderen Programmiersprachen sich mit .NET beschäftigen.

In diesem Kapitel erhalten Sie eine Einführung in C# (wieso nicht in VB.NET, haben Sie im Vorwort erfahren). Natürlich ist es unmöglich, alle Aspekte der Sprache in einem einzigen Kapitel zu behandeln. Stattdessen beschränken wir uns darauf, die *wesentlichen* Bestandteile von C# zu behandeln, die Sie in Ihrer täglichen Arbeit benötigen. Das ist gewiss eine subjektive Auswahl, allerdings haben wir uns daran orientiert, was wir bei unserer täglichen Praxisarbeit mit C# benötigt haben, weswegen wir davon ausgehen, dass wir mit unserer Auswahl eine hohe Praxisrelevanz erreicht haben.



## 4.1 Anweisungen und Variablen

Eine der – insbesondere für Umsteiger von Visual Basic – wichtigsten und relevantesten Eigenheiten von C# besteht darin, dass jede Anweisung mit einem Semikolon beendet werden muss. Dadurch erkennt nämlich der C#-Compiler, wann das Ende eines Befehls erreicht ist. Sie

können im Gegenzug einen Befehl auf mehrere Zeilen umbrechen und somit den Code übersichtlicher gestalten (wir selbst machen in diesem Buch hin und wieder hiervon Gebrauch).

Beispielsweise sah im vorherigen Kapitel eine Anweisung wie folgt aus:

```
ausgabe.Text = "Hallo Welt!";
```

Auf zwei Zeilen aufgeteilt könnte diese Anweisung wie folgt aussehen:

```
ausgabe.Text =
  "Hallo Welt!";
```

Aufspalten ist also möglich, auch in noch mehr Zeilen. Eine Einrückung mit Leerzeichen (wie in der zweiten Zeile zu sehen) oder Tabs ist ebenfalls gestattet.


*Bei Visual Basic (und auch VB.NET) ist eine Aufteilung dieser Art nicht möglich; stattdessen müssen Sie hier durch einen Unterstrich andeuten, dass Sie das Kommando in der nächsten Zeile fortsetzen möchten.*



### 4.1.1 Variablen

Unter einer *Variablen*, krankhaft in *Veränderliche* eingedeutscht, versteht man in jeder Programmiersprache einen Art Datenspeicher. Während der Programmausführung werden in Variablen Werte zwischengespeichert, die Sie dann im Programm selbst verwenden. Erinnern Sie sich an die Schulzeit zurück; im Mathematikunterricht gab es Aufgaben, die Sie nicht »am Stück« durch wildes Tippen in den Taschenrechner lösen konnten; stattdessen mussten Sie schrittweise vorgehen, haben einzelne Zwischenergebnisse erzielt und aus diesen dann das eigentliche Ergebnis erhalten[1]. Genauso läuft es auch bei der Programmierung ab, die Zwischenergebnisse werden in Variablen gespeichert.

Um einer Variablen einen Wert zuzuweisen, benötigen Sie einen Zuweisungsoperator, das ist das Gleichheitszeichen. Die folgende Anweisung würde also die Variable `woodstock` mit dem Wert 1969 belegen:

```
woodstock = 1969;
```

1. Oder der Lehrer/die Lehrerin hat den Eindruck vermittelt, dies würde genau so gehen und sei sehr einfach, zumindest für ihn/sie.

Links vom Gleichheitszeichen steht also der Variablenname, rechts davon der Wert, den die Variable annehmen soll.

C# ist eine so genannte typisierte Sprache, sprich: Es muss vorher genau bekannt sein, von welchem Typ eine Variable ist, also beispielsweise ob Zahl oder Zeichenkette. In Abschnitt 4.1.2 werden wir näher hierauf eingehen. Zunächst ein einfaches Beispiel. Die Variable `woodstock` von zuvor enthält offensichtlich eine Zahl; ein Zahlen-Datentyp ist `int`, der Integerwerte (ganzzahlige Werte) aufnehmen kann.

Bei der C#-Umsetzung müssen Sie die Variable zunächst *deklarieren*. Damit teilen Sie dem Compiler mit, dass Sie beabsichtigen eine Variable `woodstock` einzusetzen, die vom Datentyp her ein Integer ist:

```
int woodstock;
```

Zunächst geben Sie also den Datentyp an, dann den Variablennamen. Nun können Sie auf die Variable zugreifen, wie oben gesehen.


*Sie können auch direkt bei der Variablendeklaration den (anfänglichen) Wert einer Variablen angeben, indem Sie den Zuweisungsoperator = direkt nach dem Variablennamen einsetzen:*

```
int woodstock = 1969;
```

Ein weiterer Datentyp sind Strings, auch Zeichenketten genannt. Besonderheit hier: Der Wert muss unbedingt in Anführungszeichen angegeben werden. Hier ein Beispiel:

```
String woodstock = "1969";
```

Die Variable `woodstock` enthält nun eine Zeichenkette. Diese darf aber natürlich ausschließlich aus Ziffern bestehen. Allerdings müssen Sie genau aufpassen, einer Variablen einen dem Typ entsprechenden Wert zuzuweisen. Folgende Anweisung würde zu einer Fehlermeldung führen:

```
String woodstock = 1969;
```

Nachfolgend ein Beispiel. Die (String-)Variable `woodstock` wird initialisiert und mit dem Wert `"1969"` belegt. Dann wird `ausgabe.Text` auf den Wert in der Variablen `woodstock` gesetzt:

```
ausgabe.Text = woodstock;
```

Dies führt im Endeffekt dazu, dass in dem <asp:Label>-Element der Wert in der Variablen ausgegeben wird.


*Näheres zu* <asp:Label> *erfahren Sie in der Referenz unter dem Punkt »Web Controls«.*

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String woodstock;
  woodstock = "1969";
  ausgabe.Text = woodstock;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.1: Eine Variable wird deklariert und ausgegeben (variable.aspx)*


*Bild 4.1: Der Wert der Variablen wird ausgegeben*

*Wenn Sie schon von vornherein genau wissen, dass Sie den Wert einer Variablen nie verändern werden, verwenden Sie das Schlüsselwort* `const`. *Damit geben Sie an, dass die Variable einen konstanten (beständigen) Wert hat. Sie können dieser Variablen dann keinen anderen Wert mehr geben. Dies bringt Ihnen ein wenig zusätzliche Performance:*

```
const String woodstock = "1969";
```

*Eine Abkürzung für Tippfaule: Sie können mehrere Variablen vom selben Datentyp in einer Anweisung deklarieren:*

```
int i, j = 42, k;
```

*Die drei Variablen* `i`, `j` *und* `k` *sind deklariert;* `j` *ist zusätzlich mit einem Wert vorbelegt.*

### 4.1.2 Datentypen

Wie bereits angedeutet, gibt es in C# eine ganze Reihe von Datentypen, die mit der Sprache mitgeliefert werden. Aufgrund des allumfassenden .NET Frameworks müssen Sie natürlich prinzipiell darauf achten, dass Sie Typen des Frameworks verwenden, denn nur dann ist eine Interoperabilität mit den anderen .NET-Sprachen möglich.

Die gute Nachricht: Die vordefinierten Datentypen in C# werden automatisch in .NET-Datentypen umgesetzt. Beginnen wir zunächst mit den ganzzahligen, numerischen Datentypen, in alphabetischer Reihenfolge:

| Datentyp | Entsprechender .NET-Typ | Suffix | Beschreibung | Wertebereich |
|---|---|---|---|---|
| `byte` | `Byte` | | 8-Bit-Integerwert | 0 bis 255 |
| `int` | `Int32` | | 32-Bit-Integerwert | -2.147.483.648 bis +2.147.483.647 |
| `long` | `Int64` | L | 64-Bit-Integerwert | -9.223.372.036.854.7775.808 bis +9.223.372.036.854.7775.807 |
| `short` | `Int16` | | 16-Bit-Integerwert | -32.768 bis +32.767 |

| Datentyp | Entsprechender .NET-Typ | Suffix | Beschreibung | Wertebereich |
|---|---|---|---|---|
| `sbyte` | `SByte` | | 8-Bit-Integerwert (mit Vorzeichen) | -128 bis +127 |
| `uint` | `UInt32` | U | 32-Bit-Integerwert (ohne Vorzeichen) | 0 bis 4.294.967.295 |
| `ulong` | `UInt64` | U | 64-Bit-Integerwert (ohne Vorzeichen) | 0 bis 18.446.744.073.709.551.615 |
| `ushort` | `UInt16` | | 16-Bit-Integerwert (ohne Vorzeichen) | 0 bis 65.353 |

*Tabelle 4.1: Die ganzzahligen Datentypen von C#*


*Lassen Sie sich durch die schiere Menge nicht verwirren; in der Praxis wird meistens* `int` *eingesetzt. Wenn Sie größere Werte benötigen, verwenden Sie* `long`*.*

Wenn Sie Dezimalzahlen (Fließkommazahlen) benötigen, müssen Sie einen der beiden folgenden Datentypen einsetzen:

| Datentyp | Entsprechender .NET-Typ | Suffix | Beschreibung | Wertebereich |
|---|---|---|---|---|
| `double` | `Double` | D | Dezimalzahl mit 15 bis 16 Stellen Genauigkeit | Kleinster Wert (betragsmäßig): 5 E-324<br>Größter Wert (betragsmäßig) 1,7 E308 |
| `float` | `Single` | F | Dezimalzahl mit 7 Stellen Genauigkeit | Kleinster Wert (betragsmäßig): 1,5 E-45<br>Größter Wert (betragsmäßig) 3,4 E38 |

*Tabelle 4.2: Die Dezimalzahl-Datentypen von C#*


*Auch hier ein Einsatztipp: Meistens wird auf* `double` *gesetzt, denn dort ist die Genauigkeit hoch und Sie können auch große Werte abspeichern.*


*In C# wird eine amerikanische Notation für Dezimalzahlen verwenden. Dort gibt es kein Dezimalkomma, sondern einen Dezimalpunkt! Um also eineinhalb in einer Variablen zu speichern, verwenden Sie folgenden Code:*

```
float eineinhalb = 1.5;
```

Bei numerischen Datentypen können Sie durch ein Suffix (Anhängsel) den Typ genau spezifizieren. In den beiden vorangegangenen Tabellen wurden diese Endungen aufgelistet. Beispielsweise wird nachfolgend die Zahl 8.0 explizit als Float angegeben:

```
float f = 8.0F;
```

Neben dem bereits bekannten Datentyp `String` für Zeichenketten (der .NET Datentyp heißt genauso) gibt es noch den booleschen Datentyp. Es handelt sich hierbei um einen Typ, den nur zwei Werte annehmen kann: `true` (wahr) und `false` (falsch). Man spricht hier auch von einem *Wahrheitswert*.

```
bool wahr = true;
bool unwahr = false;
```

Eine Besonderheit gibt es noch bei Strings. Innerhalb von einem String können Sie bestimmte Sonderzeichen verwenden. Die häufigsten sind:

- `\n` – neue Zeile
- `\t` – Tabulator

Der String `Woodstock\n` endet also mit einem Zeilensprung. Durch den vorangestellten Backslash wird das folgende n »entwertet«; also nicht als n wahrgenommen, sondern als Zeilensprung.



2. In Exponentialschreibweise angegeben.
   2E5 beispielsweise bedeutet $2 * 10^5 = 200000$.

Was aber nun, wenn in einem String ein Backslash ausgegeben werden soll? Berechtigte Frage mit einer einfachen Antwort: Setzen Sie ebenfalls einen Backslash voran:

```
String verzeichnis = "C:\\Inetpub\\wwwroot";
```


*Wenn Sie die Sonderbedeutung des Backslash ausnahmsweise aufheben wollen, stellen Sie der Zeichenkette einen Klammeraffen voran – wohlgemerkt vor den Anführungszeichen!*

```
String verzeichnis = @"C:\Inetpub\wwwroot";
```

### 4.1.3 Variablen umwandeln

Häufig stehen Sie vor der Herausforderung, dass Sie Variablen von einem Typ in einen anderen umwandeln möchten. Beispielsweise könnte es sein, dass Sie eine String-Variable haben, in die Sie einen Zahlenwert schreiben möchten. Der Typ der Stringvariablen kann im Nachhinein nicht mehr geändert werden, Sie müssen allerdings zuvor den Zahlenwert in eine Zeichenkette umwandeln.

In vielen anderen Sprachen ist dies unter bestimmten Voraussetzungen nicht notwendig, im Übrigen auch nicht in Visual Basic .NET. In C# ist das allerdings nötig, wie folgendes Beispiel zeigt:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  int woodstock;
  woodstock = 1969;
  ausgabe.Text = woodstock;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.2: Dieser Code funktioniert so nicht (datentyp.aspx)*

Die Ausgabe dieses Codes können Sie Bild 4.2 entnehmen. Der Compiler beschwert sich, dass er `ausgabe.Text` keinen Wert vom Typ `int` zuweisen kann, sondern nur Strings.

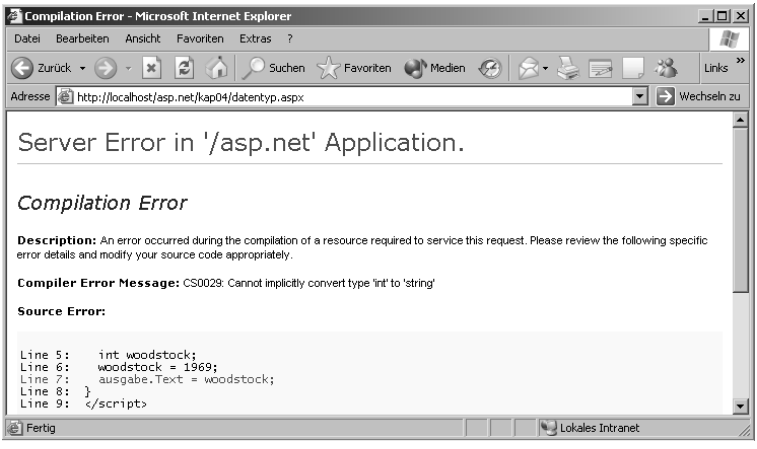



*Bild 4.2: Die Fehlermeldung*

Dieses Problem lässt sich zum Glück leicht beheben. Jeder Datentyp hat eine Methode `ToString()`, mit der der Variablenwert in eine Zeichenkette umgewandelt werden kann. Was eine Methode ist, werden Sie im nächsten Kapitel genau erfahren; zunächst einmal genügt die Information, dass Sie an den Variablennamen `.ToString()` (mit dem Punkt!) anhängen und eine Zeichenkette erhalten. Sie müssen also Zeile 7 im vorherigen Listing – die Zuweisung an `ausgabe.Text` – wie folgt ändern:

```
ausgabe.Text = woodstock.ToString();
```

Dann wird wie gewünscht der Text 1969 ausgegeben.


*Sie finden ein dementsprechend korrigiertes Listing unter dem Dateinamen datentyp-korrigiert.aspx im Webarchiv.*

Eine alternative Möglichkeit der Umwandlung besteht darin, den neuen Datentyp in Klammern vor die Variable zu schreiben. Wenn Sie beispielsweise einen ganzzahligen Wert in eine Fließkommazahl umwandeln möchten, können Sie das wie folgt durchführen (Fachterminus: *Casting*):

```
int i = 1969;
float f = (float)i;
```

Bei der Konzeption des .NET Frameworks haben die Entwickler an die Umwandlungsproblematik gedacht und über die `Convert`-Klasse eine Reihe von Methoden zur Variablenumwandlung zur Verfügung gestellt. Wenn Sie die QuickStart-Tutorials von ASP.NET installiert haben, finden Sie dort einen Class Browser, mit dem Sie einen Einblick in alle Klassen des .NET Frameworks erhalten (URL bei Standardinstallation: *http://localhost/quickstart/aspplus/samples/classbrowser/cs/class-browser.aspx*). Dort bekommen Sie auch einen Überblick über alle Funktionalitäten der *Convert*-Klasse.

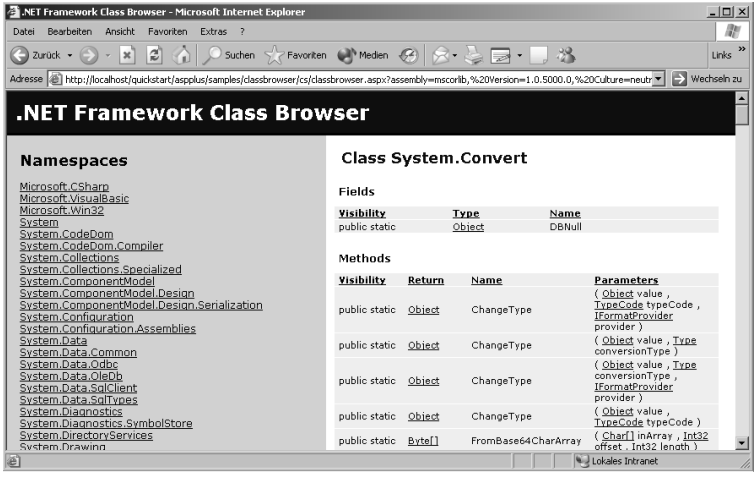



*Bild 4.3: Die Convert-Klasse im Class Browser von .NET*

Beispielsweise gibt es dort eine Methode `ToInt32()`, die eine Variable in einen `Int32`-Wert (in C#: `int`) umwandelt. Es ist somit auch möglich, eine Zeichenkette in einen Zahlenwert umzuwandeln (sofern die Zeichenkette auch tatsächlich eine Zahl enthält):

```
String s = "1969";
int i = Convert.ToInt32(s);
```

Die Variable `i` enthält nun den Wert 1969, als Zahl natürlich.

### 4.1.4 Kommentare

Zum Abschluss dieses Abschnitts noch ein Punkt, der vor allem bei komplexeren Programmen sehr wichtig ist: das Kommentieren. Sie können in Ihren Code Anmerkungen einfügen, die keine Auswirkungen auf den Code selbst haben, allerdings Ihnen später dabei helfen können, Ihren Code wieder zu verstehen.

Durch zwei Schrägstriche wird alles, was in derselben Zeile folgt, als Kommentar behandelt und damit ignoriert:

```
int woodstock = 1969;  //Das Jahr des Konzertes
```

Alternativ dazu können Sie auch Kommentare einsetzen, die sich über mehrere Zeilen erstrecken. Der Kommentar beginnt mit /* und endet mit */. Alles dazwischen ist nur dem Programmierer hilfreich, wird aber vom Compiler ignoriert:

```
/* ######################################
    In diesem Programm passiert etwas
   ###################################### */
```

Setzen Sie Kommentare stets mit Bedacht ein. Nicht jede einzelne Anweisung verdient einen Kommentar, aber ein Programm ganz ohne Kommentare muss schon sehr trivial sein, um auch so verstanden zu werden.


*Gerade bei Arbeiten im Team ist die Verwendung von aussagekräftigen, relevanten Kommentaren im Code sehr wichtig und vereinfacht die Zusammenarbeit.*

## 4.2 Operatoren und Operationen

Einen Operator von C# haben Sie bereits kennen gelernt – das Gleichheitszeichen, den Zuweisungsoperator. Allerdings gibt es noch eine Reihe weiterer Operatoren und zugehöriger Operationen, die im Folgenden vorgestellt werden.

### 4.2.1 Arithmetische Operatoren

Beim Arbeiten mit Zahlenwerten verwenden Sie arithmetische Operatoren; das sind die vier Grundrechenarten sowie der Modulooperator:

| Operator | Beschreibung | Beispiel | Ergebnis |
|---|---|---|---|
| + | Addition (plus) | 10 + 8 | 18 |
| - | Subtraktion (minus) | 10 - 8 | 2 |
| * | Multiplikation (mal) | 10 * 8 | 80 |
| / | Division (geteilt durch) | 10 / 8 | 1.25 |
| % | Modulo (Rest der Division) | 10 % 8 | 2 |

*Tabelle 4.3: Die arithmetischen Operatoren von C#*


*Der Additionsoperator (+) kann im Übrigen auch für Strings verwendet werden:*

```
String ws = "Wood" + "stock";
```

*Die Variable* `ws` *hat dann den Wert* `"Woodstock"`.



Im folgenden Listing werden die fünf exemplarischen Berechnungen aus Tabelle 4.3 sowie der String-Operator + eingesetzt:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String tabelle = "<table>";
  tabelle = tabelle + "<tr><th>Operation</th><th>Ergebnis</th></tr>";
  tabelle = tabelle + "<tr><td>10 + 8</td>";
  tabelle = tabelle + "<td>" + (10 + 8).ToString() + "</td></tr>";
  tabelle = tabelle + "<tr><td>10 - 8</td>";
  tabelle = tabelle + "<td>" + (10 - 8).ToString() + "</td></tr>";
  tabelle = tabelle + "<tr><td>10 * 8</td>";
  tabelle = tabelle + "<td>" + (10 * 8).ToString() + "</td></tr>";
  tabelle = tabelle + "<tr><td>10 / 8</td>";
  tabelle = tabelle + "<td>" + (10 / 8).ToString() + "</td></tr>";
  tabelle = tabelle + "<tr><td>10 % 8</td>";
  tabelle = tabelle + "<td>" + (10 % 8).ToString() + "</td></tr>";
  tabelle = tabelle + "</table>";
  ausgabe.Text = tabelle;
}
</script>
<html>
```

```
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.3: Die fünf Operationen (arithmetische_operatoren.aspx)*

Kernstück sind hier die Anweisungen `tabelle = tabelle + "text ..."`. Erinnern Sie sich: Links vom Gleichheitszeichen steht die Variable, rechts davon der zugewiesene Wert. Also wird an die Stringvariable `tabelle` der angegebene Text angefügt.


*Bild 4.4: Das Ergebnis der vier Operationen*

Beim Betrachten von Bild 4.4 fällt auf, dass `10/8` als Ergebnis 1 geliefert hat, also einen ganzzahligen Wert. Der Grund: Da als Parameter Integerwerte verwendet worden sind, wird automatisch angenommen, dass das Ergebnis auch vom Typ `int` ist. Es gibt zwei Möglichkeiten dies zu verhindern, also zum richtigen Ergebnis 1,25 zu gelangen:

- Speichern Sie zunächst das Divisionsergebnis in einer Fließkommavariablen:

```
double division = 10 / 8;
```

- Sorgen Sie dafür, dass einer der beiden Operanden ein Fließkommawert ist:

```
double division = 10 / 8.0;
```

*Übrigens, wenn Sie das Ergebnis mit* `ToString()` *in eine Zeichenkette umwandeln, wendet ASP.NET Ihre lokalen Spracheinstellungen an. Auf einem deutschen Betriebssystem wird also tatsächlich 1,25 ausgegeben, nicht 1.25 – auch, wenn eine englische Version des .NET Frameworks verwendet wird.*

### 4.2.2 Logische Operatoren

Bei booleschen Werten, also Wahrheitswerten, gibt es spezielle Operatoren, die so genannten *logischen Operatoren*. Davon stehen drei verschiedene zur Verfügung.



#### Logisches Und

Das logische Und ergibt nur dann `true`, wenn alle beteiligten Operanden `true` sind, ansonsten erhalten Sie `false`:

| a | b | a && b |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

*Tabelle 4.4: Das Ergebnis des logischen Und*

*C# verwendet einen kleinen Trick, um ein wenig Geschwindigkeit herauszuholen. Ausdrücke werden von links nach rechts ausgewertet. Wenn irgendwann das Ergebnis bereits feststeht, obwohl noch nicht alle Operanden ausgewertet worden sind, wird sofort das Ergebnis zurückgeliefert. Wenn beispielsweise* `false && a && b && c` *ermittelt werden soll, braucht C# gar nicht nachzuschauen, welche Werte* `a` *und* `b` *und* `c` *haben. Da der erste Operand bereits* `false` *ist, ist das Ergebnis sicherlich ebenfalls* `false`.

### Logisches Oder

Beim logischen Oder verhält es sich ähnlich, aber entscheidend anders. Hier wird nur genau dann `false` zurückgeliefert, wenn alle Operanden `false` sind, ansonsten erhalten Sie `true`.

| a | b | a \|\| b |
|---|---|---|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

*Tabelle 4.5: Das Ergebnis des logischen Oder*

Auch hier wird wieder der Abkürzungsmechanismus verwendet; `true || a || b || c` liefert auf jeden Fall `true`.

### Negation

Der einfachste Operator ist der Negationsoperator; dieser benötigt auch nur einen einzigen Operanden. Dieser negiert einfach den Wahrheitswert, aus `true` wird also `false`, aus `false` wird `true`.

| a | !a |
|---|---|
| true | false |
| false | true |

*Tabelle 4.6: Das Ergebnis der Negation*


*Die String-Repräsentationen von* `true` *und* `false` *(also* `true.ToString()` *bzw.* `false.ToString()`*) sind übrigens* `True` *und* `False`*, also mit Großbuchstaben.*

## 4.2.3 Spezielle Zuweisungsoperatoren

In Listing 4.3 haben Sie bereits Anweisungen der folgenden Machart gesehen:

```
tabelle = tabelle + "irgendein Text";
```

Diese Anweisung ist sehr häufig, dafür existiert ein Kürzel: +=. Mit folgender Anweisung erreichen Sie denselben Effekt wie oben:

```
tabelle += "irgendein Text";
```

Diese Kurzform steht für alle arithmetischen Operatoren zur Verfügung. Nachfolgend eine Übersicht (a sei 10, b sei 8):

| Operator | Beispiel | Langform | Neuer Wert von a |
|---|---|---|---|
| += | a += b | a = a + b | 18 |
| -= | a -= b | a = a - b | 2 |
| *= | a *= b | a = a * b | 80 |
| /= | a /= b | a = a / b | 1.25 |
| %= | a %= b | a = a % b | 2 |

*Tabelle 4.7: Die speziellen Zuweisungsoperatoren*



Besonders häufig ist dieser Spezialfall: Eine numerische Variable wird um genau 1 erhöht oder erniedrigt. Dafür gibt es die beiden Operatoren ++ und --:

```
int i = 1969;
i++; // i hat jetzt den Wert 1970
i--; // i hat jetzt (wieder) den Wert 1969
```

Im nachfolgenden Beispiel werden konsequent die Kurzformen verwendet:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  int a = 10, b = 8;
  String tabelle = "<table>";
  tabelle += "<tr><th>Operation (a=10, b=8)</th>
    <th>Neuer Wert von a</th></tr>";
  tabelle += "<tr><td>a += b</td>";
  a += b;
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a -= b;
  tabelle += "<tr><td>a -= b</td>";
```

```
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a *= b;
  tabelle += "<tr><td>a *= b</td>";
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a /= b;
  tabelle += "<tr><td>a /= b</td>";
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a -= b;
  tabelle += "<tr><td>a %= b</td>";
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a ++;
  tabelle += "<tr><td>a ++</td>";
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  a = 10;
  a --;
  tabelle += "<tr><td>a --</td>";
  tabelle += "<td>" + a.ToString() + "</td></tr>";
  tabelle += "</table>";
  ausgabe.Text = tabelle;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.4: Die Kurzformen im Einsatz (kurzformen.aspx)*


*Anstelle von Variablen können Sie natürlich auch normale Werte nehmen;* `i *= 2` *verdoppelt beispielsweise den Wert der Variablen* `i`.

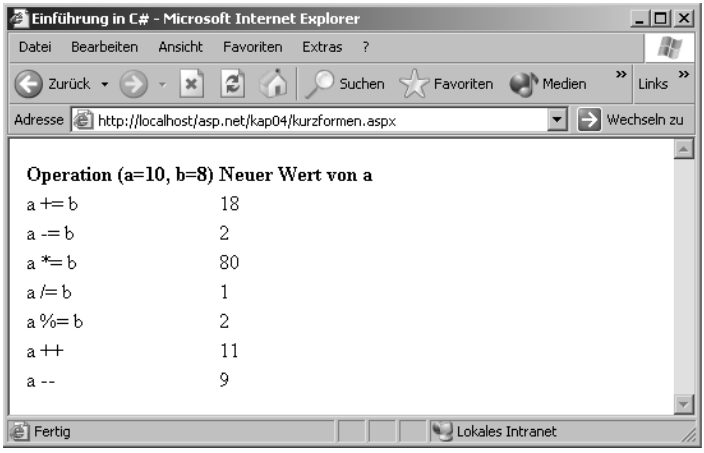



*Bild 4.5: Die sieben Kurzformen für Zuweisungsoperatoren*



### 4.2.4 Vergleichsoperatoren

Zum Abschluss dieses Abschnitts noch eine weitere Rubrik von Operatoren: Vergleichsoperatoren. Damit können Sie jedoch nur numerische Werte miteinander vergleichen (Ausnahme: ==). Für Strings sind diese Operatoren nicht verfügbar (im Gegensatz zu etlichen anderen Programmiersprachen).

Das Ergebnis einer Vergleichsoperation ist immer ein boolescher Wert: `true`, sofern der Vergleich zutreffend war, ansonsten `false`. Nachfolgende Tabelle enthält alle Möglichkeiten:

| Operator | Beschreibung | Beispiel | Ergebnis |
|---|---|---|---|
| > | Größer | `10 > 8` | `true` |
| >= | Größer oder gleich | `10 >= 8` | `true` |
| < | Kleiner | `10 < 8` | `false` |
| <= | Kleiner oder gleich | `10 <= 8` | `false` |
| == | Gleich | `10 == 8` | `false` |
| != | Ungleich | `10 != 8` | `true` |

*Tabelle 4.8: Die Vergleichsoperatoren von C#*

*Ein häufiger Fehler besteht darin, dass für Gleichheit der falsche Operator verwendet wird. In vielen Programmiersprachen, beispielsweise in Visual Basic (.NET), ist = sowohl Zuweisungsoperator als auch Vergleichsoperator. Bei C# ist das anders, hier funktioniert der Vergleich nur mit ==.*

## 4.3 Schleifen

Ein weiteres wesentliches Element vieler Programmiersprachen sind *Schleifen*. Dabei handelt es sich um Programmierkonstrukte, die eine wiederholte Ausführung von Anweisungen ermöglichen. Im weiteren Verlauf des Buches werden Sie immer wieder auf Schleifen stoßen; an dieser Stelle finden Sie aus Gründen der Übersichtlichkeit eher triviale Beispiele.

### 4.3.1 for

Die wohl häufigste Schleifenform in C# (und auch in vielen anderen Programmiersprachen) ist die `for`-Schleife, die folgende Syntax hat:

```
for (Start; Überprüfung; Ende) {
  // Befehle ...
}
```

Die Angaben in der runden Klammer haben die folgende Bedeutung:

- `Start`: Anweisung, die beim erstmaligen Durchlaufen der `for`-Schleife ausgeführt wird.
- `Ende`: Anweisung, die am Ende jedes Schleifendurchlaufs durchgeführt wird.
- `Überprüfung`: Bedingung, die `true` liefern muss, wenn die Schleife durchlaufen werden soll.

Hier ein einfaches Beispiel:

```
String s = "";
for (int i=1; i<3; i++) {
  s += "Woodstock\n";
}
```

Gehen wir diese Schleife schrittweise durch. Am Anfang wird das Startkommando ausgeführt, also `int i=1`. Die Variable `i` wird dekla-

riert und mit dem Wert 1 initialisiert. Bevor aber der Code innerhalb der Schleife (also in den geschweiften Klammern) ausgeführt wird, muss zunächst die Schleifenbedingung überprüft werden: `i<3`. Dies ist offensichtlich erfüllt, also wird an `s` der Text `"Woodstock"` nebst Zeilenwechsel angehängt. Danach wird das Endkommando ausgeführt, `i++`. Die Variable `i` hat nun den Wert 2.

Als Nächstes wird wieder die Schleifenbedingung geprüft. Gilt immer noch `i<2`? Offensichtlich ja, also wird erneut Text an die Stringvariable `s` angehängt. Am Ende wird `i` um 1 erhöht, hat also den Wert 3. Damit ist `i<3` nicht mehr erfüllt. Konsequenz: Die Schleife wird verlassen, der Code direkt nach der Schleife wird ausgeführt. Die Variable s hat den folgenden Wert:

```
Woodstock
Woodstock
```


*Wie Sie gesehen haben, wurde die Variable `i` innerhalb der Schleife deklariert. Dies ist nicht Pflicht, sie muss nur allgemein deklariert werden. Dies kann auch vor der Schleife geschehen.*

Nach dieser ausführlichen Einleitung ein komplettes, im Browser lauffähiges Beispiel. Dabei wird die Variable `i` – oft auch *Zählvariable* genannt, da sie die Schleifendurchläufe »mitzählt« – innerhalb der Schleife verwendet. Nachfolgend die ersten zehn Kubikzahlen:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String s = "";
  for (int i=1; i<=10; i++) {
    s += (i*i*i).ToString() + "<br />";
  }
  ausgabe.Text = s;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
```

```
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.5: Die ersten zehn Kubikzahlen (for-kubik.aspx)*

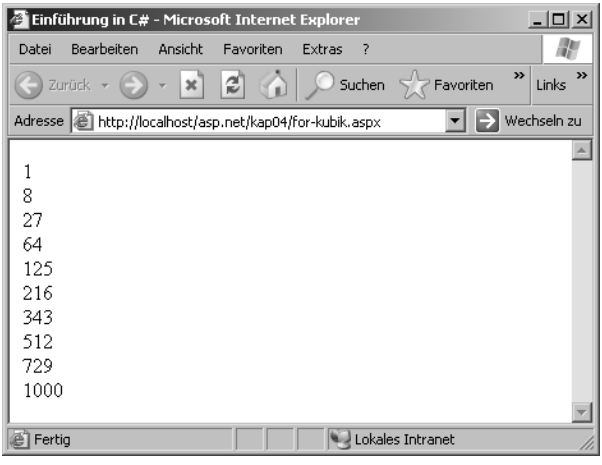



*Bild 4.6: Eine Schleife gibt die ersten zehn Kubikzahlen aus*

Eine weitere Anwendungsmöglichkeit für Schleifen ergibt sich bei mathematischen Berechnungen. Ein geradezu klassisches Beispiel ist die Ermittlung der Fakultät. Die Fakultät einer natürlichen Zahl *n*, auf *n!* geschrieben, ist das Produkt aller natürlichen Zahlen von 1 bis *n*. Beispielsweise ergibt 5! als Ergebnis 120, denn 1*2*3*4*5 = 120. Eine Schleife kann dies erledigen:

```
int zahl = 5; // Zahl, deren Fakultät errechnet wird
int fak = 1;
for (int i=2; i<=zahl; i++) {
  fak *= i;
}
```

Nach Ausführung dieses Codefragments enthält die Variable `fak` die Fakultät der angegebenen Zahl.

Bei der Wahl der drei Angaben in der Klammer hinter `for` haben Sie freie Wahl, Sie müssen beispielsweise die Zählvariable nicht nur um 1 erhöhen. Genauer gesagt benötigen Sie nicht einmal diese Angaben, lediglich die Strichpunkte sind Pflicht. Hier die Minimalversion der Schleife:

```
for (;;) {
  //irgendwelche Befehle
}
```

Diese Schleife hat keine Abbruchbedingung, läuft also endlos. In Abschnitt 4.3.4 erfahren Sie, wie Sie eine solche Schleife dennoch verlassen können.


*Wenn innerhalb einer Schleife nur ein einziges Kommando steht, könnten Sie die geschweiften Klammern weglassen – beachten Sie den Konjunktiv. Es ist viel weniger fehlerträchtig und hilft auch bei der Fehlersuche imminent, wenn Sie stets Klammern verwenden.*



### 4.3.2 do

Eine weitere Schleife ist die `do`-Schleife, die wie folgt aufgebaut ist:

```
do {
  //Anweisungen
} while (Bedingung);
```

Die Anweisungen innerhalb der geschweiften Klammern werden so lange ausgeführt, bis die am Ende angegebene Bedingung erfüllt ist. Um genau zu sein: Die Bedingung wird erst *nach* jedem Schleifendurchlauf überprüft. Die Schleife läuft also auf jeden Fall mindestens einmal durch. Dieser Nebeneffekt ist nicht immer erwünscht.

Nachfolgend das Kubikzahlbeispiel von zuvor, diesmal mit `do`:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String s = "";
  int i=0;
  do {
    i++;
    s += (i*i*i).ToString() + "<br />";
  } while (i<10);
```

```
  ausgabe.Text = s;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.6: Die zehn Kubikzahlen, diesmal mit do (do-kubik.aspx)*

In jedem Schleifendurchlauf wird eine Variable `i` um `1` erhöht und dann die zugehörige Kubikzahl ausgerechnet.

### 4.3.3 while

Die `while`-Schleife ist der `do`-Schleife ganz ähnlich, mit einem kleinen, aber häufig entscheidenden Unterschied: Die Schleifenbedingung wird *vor* jedem Durchlauf überprüft. Im Gegensatz zu `do` ist es bei `while` auch möglich, dass die Schleife nie durchlaufen wird.

Hier das Kubikzahlbeispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String s = "";
  int i=0;
  while (i<10) {
    i++;
    s += (i*i*i).ToString() + "<br />";
  }
  ausgabe.Text = s;
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
```

```
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.7: Die zehn Kubikzahlen, diesmal mit while (while-kubik.aspx)*

### 4.3.4 Kommandos für Schleifen

Normalerweise wird eine Schleife nur dann verlassen, wenn die Schleifenbedingung nicht mehr erfüllt ist. C# bietet hier zwei Kommandos an.

Zunächst einmal die Anweisung `break`, mit der eine Schleifenausführung vorzeitig beendet werden kann:

```
for (;;) {
  break;
}
```

Obwohl es sich um eine Endlosschleife handelt, wird sie schon beim ersten Durchlauf verlassen.

Zweites spezielles Schleifenkommando ist `continue`; dies sorgt dafür, dass der restliche Schleifencode übersprungen wird, C# die Endbedingung prüft und die Schleife gegebenenfalls erneut durchlaufen lässt. Hier ein kleines, zugegebenermaßen arg konstruiertes Beispiel:

```
String s = "";
int i = 1;
while (i <= 10) {
  i++;
  s += i.ToString() + "\n";
  i++;
  continue;
  s += i.ToString() + "\n";
}
```

Durch den Einsatz von `continue` wird die letzte Anweisung nie ausgeführt. Die String-Variable `s` enthält also nur gerade Zahlen, keine ungeraden.

## 4.4 Fallunterscheidungen

Bis dato sind alle Programme linear abgelaufen, will heißen: Der Code wurde von oben nach unten durchlaufen, jede einzelne Zeile wurde ausgeführt, das Ergebnis stand bereits vorher fest. Ein wesentlicher

Bestandteil von C# ist die Möglichkeit, Fallunterscheidungen durchzuführen. Das sind Wenn-Dann-Konstrukte. Nur so ist es möglich, auf bestimmte Begebenheiten – Benutzereingaben, das aktuelle Tagesdatum etc. – zu reagieren.

### 4.4.1 if

Die Syntax für eine Fallunterscheidung ist die folgende:

```
if (Bedingung) {
  // Anweisungen
}
```

Der Code ist selbst erklärend: Wenn die Bedingung in den (zwingenden) runden Klammern erfüllt ist, werden die Anweisungen ausgeführt, ansonsten nicht.

Für die folgenden Beispiele müssen wir einen Vorgriff auf die Referenz machen, um Fallunterscheidungen sinnvoll einsetzen zu können. `DateTime.Now.Minute` liefert die aktuelle Minute (zwischen 0 und 59) der aktuellen Uhrzeit auf dem Webserver zurück. Dieser Wert ändert sich natürlich alle 60 Sekunden, damit ist ein einfaches Testen möglich. Sofern Sie auf Ihrer lokalen Maschine testen möchten, können Sie dort auch die Uhrzeit direkt aus Windows heraus umstellen und so testen.

Nachfolgender Code überprüft zunächst, ob die aktuelle Uhrzeit zwischen 0 und 29 Minuten liegt – dann wird eine entsprechende Meldung ausgegeben. Analog dazu überprüft eine zweite `if`-Fallunterscheidung, ob die aktuelle Minutenzahl zwischen 30 und 59 liegt.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  ausgabe.Text = "Aktuelle Minutenzahl: " +
                 DateTime.Now.Minute.ToString();
  if (DateTime.Now.Minute >= 0 && DateTime.Now.Minute <= 29) {
    ausgabe.Text += "<br />Erste Hälfte der aktuellen Stunde";
  }
  if (DateTime.Now.Minute >= 30 && DateTime.Now.Minute <= 59) {
    ausgabe.Text += "<br />Zweite Hälfte der aktuellen Stunde";
  }
}
```

```
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.8: Fallunterscheidung mit* `if` *(if.aspx)*


*Streng genommen sind die geschweiften Klammern nur dann nötig, wenn mehr als eine Anweisung ausgeführt werden soll (ähnlich zu den Schleifen). Es ist aber guter Stil, prinzipiell Klammern einzusetzen.*

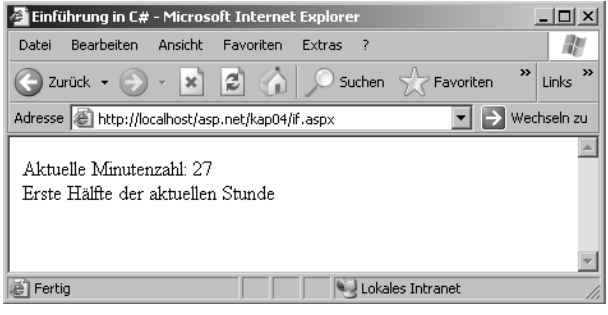



*Bild 4.7: Wir befinden uns noch in der ersten Stundenhälfte*

### 4.4.2 if-else

Der Code kann aber abgekürzt werden. Für den Fall, dass die Bedingung nicht erfüllt ist, können Sie alternative C#-Kommandos angeben. Dies geht mit `else`.

```
if (Bedingung) {
  // Anweisungen
} else {
  // alternative Anweisungen
}
```

Der `Page_Load()`-Teil im vorherigen Listing kann also wie folgt umgeschrieben werden:

```
void Page_Load()
{
  ausgabe.Text = "Aktuelle Minutenzahl: " +
    DateTime.Now.Minute.ToString();
  if (DateTime.Now.Minute >= 0 &&
      DateTime.Now.Minute <= 29) {
    ausgabe.Text +=
      "<br />Erste Hälfte der aktuellen Stunde";
  } else {
    ausgabe.Text +=
      "<br />Zweite Hälfte der aktuellen Stunde";
  }
}
```

Sie finden ein komplettes Skript im Webarchiv unter dem Dateinamen *if-else.aspx*.


*Auf* `else` *muss nicht einmal eine zweite* `if`*-Fallunterscheidung folgen, denn die aktuelle Minutenzahl muss ja systembedingt zwischen 0 und 59 liegen. Ist der Wert also nicht zwischen 0 und 29, ist er zwingenderweise zwischen 30 und 59. Ausführlich hätte das so aussehen können:*

```
if (DateTime.Now.Minute >= 0 &&
    DateTime.Now.Minute <= 29) {
  ausgabe.Text +=
    "<br />Erste Hälfte der aktuellen Stunde";
} else {
  if (DateTime.Now.Minute >=30 &&
      DateTime.Now.Minute <= 59) {
    ausgabe.Text +=
      "<br />Zweite Hälfte der aktuellen Stunde";
  }
}
```

*Im Zusammenhang mit* `if-else` *gibt es noch eine ganz besondere Kurzform. Betrachten Sie den folgenden Ausdruck:*

`(Bedingung) ? Wert1 : Wert2`

*Ist die Bedingung erfüllt, wird Wert1 zurückgeliefert, ansonsten Wert2. Ein kleines Beispiel:*

```
ausgabe.Text = (DateTime.Now.Minute % 2 == 0) ? "gerade" :
"ungerade";
```

*Hat die aktuelle Minutenzahl den Zweierrest 0, ist also gerade, wird* `"gerade"` *ausgegeben, ansonsten* `"ungerade"`.

*Diese Kurzform – die nur bestimmte* `if-else`*-Anweisungen ersetzen kann – scheidet die Geister. Einige Programmierer schwören darauf, andere wiederum vermeiden den* `?`*-Operator.*



### 4.4.3 switch

Wie Sie gerade gesehen haben, sorgt jedes zusätzliche `if` bei `else` für unübersichtlicheren Code, was Sie auch durch die gezeigte Einrückung mit Leerzeichen nicht sehr gut beheben können. Fallunterscheidungen lassen sich jedoch beliebig tief verschachteln. Wenn Sie allerdings einen einzigen Wert mehrfach überprüfen möchten, steht mit dem `switch`-Konstrukt eine elegante Alternative zur Verfügung. Hier zunächst die Syntax:

```
switch (Variable) {
  case Wert1:
    // Anweisungen für Variable == Wert1
    break;
  case Wert2:
    // Anweisungen für Variable == Wert2
    break;
  // usw.
  default:
    // Anweisungen, falls alles andere unzutreffend
    break;
}
```

Sie überprüfen also eine Variable (oder einen Ausdruck) gegen mehrere Werte. Sobald einer der Werte mit dem Variablenwert überein-

stimmt, wird der zugehörige Code ausgeführt. Für den Fall, dass keiner der Werte passt, haben Sie am Ende optional noch die Möglichkeit, Anweisungen durchführen zu lassen.

Um beim Beispiel mit der aktuellen Minutenzahl zu bleiben: Hier ein kleines Skript, bei dem an vier Zeitpunkten pro Stunde ein besonderer Text ausgegeben wird:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  ausgabe.Text = "Aktuelle Minutenzahl: " +
    DateTime.Now.Minute.ToString();
  switch (DateTime.Now.Minute) {
    case 0:
      ausgabe.Text += "<br />Volle Stunde";
      break;
    case 15:
      ausgabe.Text += "<br />Viertel nach";
      break;
    case 30:
      ausgabe.Text += "<br />Halbzeit";
      break;

    case 45:
      ausgabe.Text += "<br />Viertel vor";
      break;
    default:
      ausgabe.Text += "<br />Nichts Besonderes";
      break;
  }
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.9: Fallunterscheidungen mit* `switch` *(switch.aspx)*


*Bild 4.8: Fünfzehn Minuten vor der vollen Stunde*

*Die Verwendung von* `break` *am Ende jedes* `case`*-Abschnitts ist zwingend, sonst würden die nachfolgenden Anweisungen für die anderen* `case`*-Abschnitte auch ausgeführt werden. C# geht sogar noch einen Schritt weiter und verlangt auch am Ende des* `default`*-Abschnitts ein* `break`*, andernfalls erhalten Sie eine Compiler-Fehlermeldung.*



## 4.5 Fehler abfangen

In Bild 4.2 haben Sie bereits eine der gefürchteten Fehlermeldungen von ASP.NET gesehen[3]. Diese ist zwar zum Testen für Sie sehr sinnvoll, allerdings sollten solche Fehler vor Ihren Besuchern tunlichst vermieden werden. Bei Stellen in Ihrem Code, wo etwas schief gehen kann (beispielsweise später bei Datenbankabfragen – die Datenbank könnte ja theoretisch ausgefallen sein), sollten Sie besondere Vorkehrungen treffen.

In C# steht das `try-catch`-Konstrukt zur Verfügung, das die folgende Syntax hat:

3. Um genau zu sein, handelte es sich hierbei um eine Compiler-Fehlermeldung, also einen Fehler, der schon beim Kompilieren entdeckt worden ist (und den Sie unverzüglich korrigieren können). In diesem Abschnitt erfahren Sie, wie Sie Laufzeitfehler abfangen können, also einen Fehler, der erst während der Ausführung des Skripts selbst in Erscheinung tritt.

```
try {
  // "bedenklicher" Codeblock
} catch (Exception ex) {
  // Code bei Auftreten des Fehlers
}
```

Im `catch`-Abschnitt können Sie den Fehler einfach übergehen, ihn per E-Mail an den Webmaster schicken oder dem Nutzer eine freundliche Meldung ausgeben. In diesem Fall geben wir die Fehlermeldung selbst aus; Sie erhalten sie über `ex.Message` (`ex` ist der Variablenname für den aufgetretenen Fehler, den wir hinter `catch` in der runden Klammer deklarieren). Einen Fehler erzeugen wir dadurch, dass wir mit `Convert.ToInt32()` versuchen, eine Zeichenkette, die nur aus Buchstaben besteht, in einen Integerwert umzuwandeln:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  try {
    ausgabe.Text = Convert.ToInt32("woodstock").ToString();
  } catch (Exception ex) {

    ausgabe.Text = "Es ist leider ein Fehler aufgetreten: " +
                   ex.Message;
  }
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.10: In diesem Codestück steckt ein Fehler (try-catch.aspx)*

Die Ausgabe dieses Skripts können Sie Bild 4.9 entnehmen. Je nach Sprachversion des .NET Frameworks kann die Meldung auch eine andere Sprache haben.


*Bild 4.9: Der Fehler wurde abgefangen (und ausgegeben)*

Erwähnenswert ist weiterhin `finally`. Damit können Sie – nach dem `catch`-Block – Code angeben, der auf jeden Fall ausgeführt wird, egal ob ein Fehler aufgetreten ist oder nicht. Anstelle einer Weiterführung des Beispiels von gerade eben hier ein theoretisches Stück Code, der Ihnen zeigt, wo `finally` sinnvoll sein kann:

```
try {
  // Verbindung zur Datenbank öffnen
} catch (Exception ex) {
  // Fehlermeldung ausgeben, Mail an Webmaster
} finally {
  // Verbindung zur Datenbank wieder schließen
}
```

Dadurch, dass Sie auf jeden Fall die Verbindung zur Datenbank wieder schließen, räumen Sie auch bei einem ausgefallenen SQL-Server auf und entfernen die (erfolglose) Datenbankverbindung aus dem Speicher.

## 4.6 Arrays

Eine spezielle Form von Datenkonstrukt sind so genannte Arrays, auf Deutsch (aber ungebräuchlich): Felder. Anstelle von einem einzigen Wert enthält ein Array mehrere Werte.

### 4.6.1 Arrays erstellen

So ein Array wird wie folgt erstellt:

```
Datentyp[] Variablenname = new Datentyp[Elementzahl];
```

Etwas weniger abstrakt: Folgende Anweisung erzeugt ein String-Array mit fünf Elementen:

```
String[] woodstock = new String[5];
```

Sie können nun über `woodstock[x]` auf die einzelnen Elemente im Array zugreifen. Der Wert x innerhalb der eckigen Klammern ist der so genannte *Index* des Arrayelements. Kleine Spitzfindigkeit: Das erste Arrayelement hat den Index 0, das zweite Arrayelement den Index 1 und so weiter. Das Array `woodstock`, das oben als fünfelementig deklariert worden ist, enthält also Elemente `woodstock[0]` bis `woodstock[4]`. Sie können nun Werte zuweisen:

```
woodstock[0] = "Jimi Hendrix";
woodstock[1] = "The Who";
woodstock[2] = "Ten Years After";
woodstock[3] = "Santana";
woodstock[4] = "Joe Cocker";
```

### 4.6.2 Arrayelemente anzeigen

Das Anzeigen von Arrayelementen geht wie erwartet, beispielsweise `ausgabe.Text = woodstock[4]`, um Joe Cocker in das Feld mit `id="ausgabe"` zu schreiben. Häufig lautet jedoch die Anforderung, alle Elemente eines Arrays auszugeben. Klarer Fall für eine Schleife:

```
for (int i=0; i<5; i++) {
  ausgabe.Text += woodstock[i] + "\n";
}
```

Was aber, wenn Sie nicht genau wissen, wie viele Elemente das Array hat? Gucken Sie einfach nach. Hängen Sie an den Namen der Arrayvariablen `.Length` (inklusive Punkt) an, und Sie erhalten den gewünschten Wert, hier 5. Sie müssen also eine Schleife von 0 bis 4 laufen lassen, um Zugriff auf alle Elemente zu erhalten:

```
for (int i=0; i<woodstock.Length; i++) {
  ausgabe.Text += woodstock[i] + "\n";
}
```

Eine Alternative besteht darin, eine spezielle C#-Schleife zu verwenden, die wir Ihnen zuvor vorenthalten haben: `foreach`. Sie hat folgende Syntax:

```
foreach (Datentyp Element in Array) {
  // Element verarbeiten
}
```

Im konkreten Fall:

```
foreach (String element in woodstock) {
  // ...
}
```

Damit können Sie alle Elemente im Array durchlaufen und sie nacheinander betrachten. Nachfolgend ein komplettes Listing mit Browserausgabe:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String[] woodstock = new String[5];
  woodstock[0] = "Jimi Hendrix";
  woodstock[1] = "The Who";
  woodstock[2] = "Ten Years After";
  woodstock[3] = "Santana";
  woodstock[4] = "Joe Cocker";

  foreach (String element in woodstock) {
    ausgabe.Text += element + "<br />";
  }
}
</script>
<html>
<head>
  <title>Einf&uuml;hrung in C#</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 4.11: Ausgabe aller Arrayelemente mit* `foreach` *(foreach.aspx)*

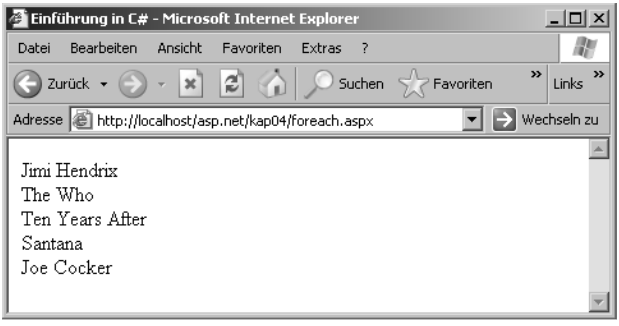



*Bild 4.10: Die fünf Arrayelemente*

## 4.7 Funktionen

Zum Abschluss noch ein weiteres Sprachkonstrukt, das Sie bereits seit Kapitel 3 verwenden; eine Erklärung sind wir bis dato schuldig geblieben. Die Rede ist von Funktionen[4], einer Möglichkeit, Code zu strukturieren und wiederzuverwerten. Die Grundidee ist, Code auszulagern und »auf Knopfdruck«, sprich bei Bedarf, aufzurufen.

### 4.7.1 Eine Funktion erstellen

Eine Funktion wird wie folgt definiert:

```
Datentyp Funktionsname()
{

}
```

Vor dem Funktionsnamen kommt der Datentyp, den die Funktion zurückliefert – wie in der Mathematik auch kann eine Funktion einen Rückgabewert haben. Der Unterschied ist das *kann*; denn eine C#-Funktion *muss* nichts zurückliefern. In diesem Fall geben Sie als Datentyp `void` an, wie im folgenden Beispiel:

4. Der C#-Terminus dafür heißt Methode, ein Begriff, der ursprünglich aus der Objektorientierten Programmierung (OOP) stammt. In vielen anderen Programmiersprachen ist jedoch auch von Funktionen die Rede, weswegen dieser Terminus eingesetzt wird.

```
void HalloWelt() {
  ausgabe.Text = "Hallo Welt!";
}
```

Sie rufen diese Funktion von jeder beliebigen anderen Stelle in Ihrem Code auf, indem Sie folgendes Kommando einfügen:

```
HalloWelt();
```

Das war es auch schon! Einzige Besonderheit: Bei C#-Seiten müssen Sie den kompletten Code in Funktionen packen[5]. Die Funktion `Page_Load()`, die wir bis dato bei jedem Beispiel verwendet haben, wird beim Laden der Seite automatisch ausgeführt. In ihr müssen Sie den kompletten Code platzieren. Um `Page_Load()` nicht allzu lang und unübersichtlich werden zu lassen, können Sie Elemente in andere Funktionen auslagern und diese dann aus `Page_Load()` heraus aufrufen.


*Ein wichtiger Punkt zu Variablen: Variablen, die Sie innerhalb einer Funktion definieren, sind nur innerhalb der Funktion gültig; man spricht hier auch von lokalen Variablen. Variablen, die Sie irgendwo im `<script>`-Bereich erzeugen, sind globale Variablen und auch innerhalb von Funktionen sichtbar.*

### 4.7.2 Funktion mit Rückgabewert

Wenn eine Funktion einen Rückgabewert hat, müssen Sie zunächst den zugehörigen Datentyp vor den Funktionsnamen schreiben. Innerhalb der Funktion geben Sie dann einen Wert mit der `return`-Anweisung zurück. Hier ein kleines Beispiel:

```
String HalloWelt()
{
  return "Hallo Welt!";
}
```

Eine Möglichkeit, diese Funktion aufzurufen und in Ihre Seiten einzubauen, wäre beispielsweise folgende Anweisung in `Page_Load()`:

```
ausgabe.Text = HalloWelt();
```

5. Ausnahme: Variablendeklarationen


*Vorsicht, wenn Sie innerhalb einer* `if`*-Fallunterscheidung einen Rückgabewert angeben. Achten Sie darauf, auch im* `else`*-Zweig einen Rückgabewert anzugeben, denn einen Rückgabewert muss die Funktion ja haben. Andernfalls weist Sie der C#-Compiler darauf hin.*


*Wenn Sie* `return` *verwenden, wird die Funktion sofort verlassen. Sie können* `return` *deswegen auch in Funktionen ohne Rückgabewert verwenden; folgende Anweisung verlässt die Funktion sofort:*

```
return;
```

### 4.7.3 Funktionsparameter

Erinnern Sie sich beispielsweise an `Convert.ToInt32()`? Das ist auch so etwas wie eine Funktion; sie dient zur Umwandlung von Zeichenketten in Integerwerte. Sie haben dort in Klammern die Zeichenkette angegeben, die in eine Zahl umgewandelt werden soll. Dies nennt man Parameter. Sie können sich bei der Definition einer Funktion dafür entscheiden, dass diese Parameter enthält. Diese müssen Sie im Funktionskopf angeben, inklusive dem Datentyp[6]. Den Namen, den Sie für die Parameter wählen, können Sie innerhalb der Funktion auch verwenden. Oder anders gesagt: Parameter verhalten sich wie lokale Variablen der Funktion.

Hier ein Beispiel, das die zuvor schon einmal gezeigte Fakultätsberechnung so umsetzt, wie es auch sinnvoll ist: Als Funktion mit einem Parameter:

```
UInt64 fakultaet(UInt64 zahl)
{
  UInt64 ergebnis = 1;
  for (UInt64 i=2; i<=zahl; i++) {
    ergebnis *= i;
  }
  return ergebnis;
}
```

6. Mehrere Parameter sind natürlich möglich; trennen Sie sie mit Kommata voneinander.


*Wie Sie sehen, verwenden wir als Datentyp* `UInt64`*, denn die Fakultät kann ohnehin nur von nicht-negativen ganzen Zahlen ermittelt werden, also von 0, 1, 2 und so weiter. Integer-Datentypen, die nicht mit* `U` *beginnen, sind mit Vorzeichen, also eine reine Verschwendung.*

## 4.8 Ein abschließendes Beispiel

Zum Abschluss der Spracheinführung noch ein Beispiel, das möglichst viele Elemente der Spracheinführung verwendet. Außerdem noch zwei Vorgriffe auf den Referenzteil; es geht um Web Controls. Sie kennen ja bereits folgendes Element:

```
<asp:Label id="ausgabe" runat="server" />
```

Es handelt sich um einen Textcontainer, in den Sie mittels `ausgabe.Text` Daten schreiben können, die dann als HTML an den Browser ausgeliefert werden. Was Sie bis dato noch nicht gesehen haben, ist folgendes Element:

```
<asp:TextBox id="eingabe" runat="server" />
```

Dieses ASP.NET-Element führt zu einem HTML-Textfeld, also `<input type="text">`. Der Clou: Sie können innerhalb Ihres C#-Codes über `eingabe.Text` auf das zugreifen, was der Benutzer zuvor ins Textfeld eingegeben hat.

Dies wollen wir ausnutzen. Dazu fügen wir noch eine Schaltfläche hinzu, um das Formular versenden zu können. Anstelle von `<input type="submit">` wird wieder ein spezielles ASP.NET-Element eingesetzt:

```
<asp:Button Text="Berechnen" OnClick="Berechnen" runat="server" />
```

Dies führt zu einer Schaltfläche, die mit `"Berechnen"` beschriftet ist (Parameter `Text`). Sobald die Schaltfläche geklickt wird, wird die Funktion `Berechnen()` ausgeführt (Parameter `OnClick`). Diese sieht wie folgt aus:

```
void Berechnen(Object o, EventArgs e)
{
  try {
    ausgabe.Text = eingabe.Text + "! = " +
                   fakultaet(Convert.ToUInt64(eingabe.Text));
  } catch (Exception ex) {
```

```
    ausgabe.Text = "Bitte geben Sie eine natürliche Zahl ein!";
  }
}
```

Es wird also versucht, die Eingabe (`eingabe.Text`) in einen `UInt64`-Wert umzuwandeln und dann an `fakultaet()` zu übergeben und das Ergebnis in das Ausgabefeld zu schreiben. Wenn die Umwandlung nicht klappt, kommt es aber zu keiner ASP.NET-Fehlermeldung, denn dies wird mit `try-catch` abgefangen. Es erscheint also eine selbst definierte Fehlermeldung.


*Beachten Sie den Kopf der Funktion* `Berechnen()`*: Sie enthält automatisch zwei Parameter, einen vom Typ* `Object` *(der Ober-Datentyp im .NET Framework) und einen vom Typ* `EventArgs` *(der Aufschluss darüber liefern kann, wieso die Funktion überhaupt aufgerufen worden ist – Sie wissen es ja, durch Klick auf die Schaltfläche).*

Das Einzige, was noch fehlt: Damit diese Interaktion zwischen dem Textfeld, das ja auf der Clientseite Eingaben benötigt, und den serverseitigen Funktionen im `<script>`-Block möglich ist, müssen Sie Textfeld und Schaltfläche noch innerhalb von speziellen `<form>`-HTML-Elementen platzieren. Setzen Sie den Parameter `runat="server"`:

```
<form runat="server">
  <asp:TextBox id="eingabe" runat="server" />
  <asp:Button Text="Berechnen" OnClick="Berechnen" runat="server" />
</form>
```

Und das war's auch schon! Bild 4.11 zeigt die Mini-Anwendung zur Laufzeit.


*In Bild 4.11 sehen Sie im Übrigen einen der interessanten Vorteile der neuen ASP.NET-Komponenten: Das Textfeld wird beim Neuladen der Seite mit dem eingegebenen Wert vorausgefüllt. Mit dem alten ASP wäre das ungleich aufwändiger gewesen.*

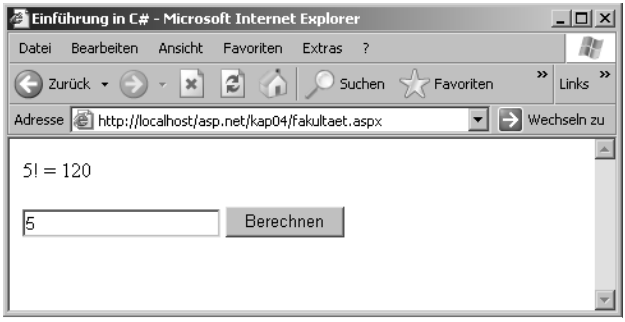



*Bild 4.11: Die Mini-ASP.NET-Anwendung*

# 5 OOP

Der Titel dieses Kapitels ist ein Kürzel, das für Objektorientierte Programmierung steht. Es handelt sich hierbei um ein allgegenwärtiges Konzept bei der Programmierung und wird in diesem Kapitel aus der Sicht von C# behandelt.

OOP kam insbesondere mit der Veröffentlichung von Java stark in Mode und genoss einen großen Hype, sprich: ohne OOP ging es nicht, Widerstand/-spruch war zwecklos. Mittlerweile ist wieder mehr Vernunft eingekehrt. OOP ist eine gute Sache, jedoch kein Allheilmittel und auch keine Eier legende Wollmilchsau; bedachter Einsatz (das heißt auch: Verzicht darauf) tut Not.


*In den beiden nächsten Kapiteln (6 und 7) erfahren Sie die häufigsten Einsatzgebiete für OOP bei ASP.NET. In diesem Kapitel spielen sich weiterhin alle Beispiele auf ein- und derselben .aspx-Seite ab.*

## 5.1 Worum geht es?

Das Grundprinzip der Objektorientierten Programmierung besteht darin, die wirkliche Welt in Infomationseinheiten darzustellen. Die allgemeine Einheit, ein Objekt, wird in einer Klasse dargestellt. Diese Klassen besitzen:

- *Eigenschaften* – das sind in der Klasse definierte Variablen
- *Methoden* – in der Klasse definierte Funktionen

Um das abstrakte Beispiel etwas anschaulicher zu machen: Eine mögliche Klasse könnte `Mensch` sein. Ein Mensch hat eine Reihe von Eigenschaften:

- Name
- Geburtsdatum
- Gewicht

Mögliche Methoden könnten die folgenden sein:

- Essen
- Trinken

Diese Methoden haben auch potenzielle Auswirkungen auf die Eigenschaften, beispielsweise könnte sich durch die Methode `Essen` das Gewicht der Person erhöhen.

## 5.2 Eine Klasse erstellen

Die Implementierung einer Klasse ist zunächst recht einfach. Sie benötigen das Schlüsselwort `class` und den Klassennamen. Die Klasse kann Daten in Form von Variablen sowie Funktionalität in Form von Funktionen enthalten. Mithilfe eines Schlüsselworts wird noch angegeben, ob die Variable bzw. Funktion von außen ausgelesen und verändert werden darf oder nicht. Bei `public` ist der Zugriff erlaubt, bei `private` können Sie nur innerhalb der Klasse darauf zugreifen.




*Wenn Sie nichts angeben, ist die Eigenschaft oder Methode automatisch als* `private` *deklariert, kann also nicht von außen abgefragt werden.*

So könnte die Beispielklasse also aussehen:

```
class Mensch
{
  public String Name;
  public DateTime Geburtsdatum;
  public double Gewicht;

  public void Essen(double Menge)
  {
    // ...
  }
}
```

Wie Sie eine Variable vom Typ einer Klasse erstellen (man spricht hier von *instanzieren*), haben Sie bereits in der Spracheinführung bei der Erstellung eines Arrays gesehen – ohne den Hinweis natürlich, dass hierbei ein Objekt erstellt wird. Verwenden Sie den `new`-Operator.

```
Mensch Bill = new Mensch();
Bill.Name = "Bill Gates";
Bill.Essen(0.25); // Wiener Schnitzel?!
```


*Um eine Objektvariable wieder loszuwerden und den damit belegten Speicher freizugeben, setzen Sie die Variable auf* `null`.

## 5.3 Data Hiding und Eigenschaften

Ein Grundparadigma von OOP ist, dem Nutzer keinen direkten Zugriff auf globale Klassenvariablen zu geben, sondern alles über Methoden zu regeln. Sie werden dies insbesondere bei Java-Programmen feststellen. Wenn es dort eine Variable `xyz` gibt, so hat der Programmierer häufig eine Methode `getXyz()` (zum Auslesen) und eine `setXyz()` (zum Setzen) erstellt, die beide `public` sind. Die Variable `xyz` jedoch ist als `private` markiert.

Dies ist natürlich auch unter C# möglich, allerdings gibt es hier das Konstrukt der Eigenschaft. Eigenschaften besitzen einen `get`- und einen `set`-Block, die zum Auslesen bzw. Setzen des Werts verwendet werden. Nach außen hin verhalten sich Eigenschaften wie Variablen. Das folgende Beispiel zeigt die Verwendung:

```
class Mensch
{
  private String _Name;

  public String Name
  {
    get {
      return _Name;
    }
    set {
      _Name = value;
    }
  }
}
```

Beachten Sie den `set`-Block. Wenn ein Benutzer von außen die Eigenschaft setzt, steht der angegebene Wert in `value` (und wird in diesem Beispiel direkt der Klasseneigenschaft zugewiesen).

Sie können die Eigenschaft wie gehabt verwenden:

```
Mensch Paul = new Mensch();
Bill.Name = "Paul Allen";
```

*Wenn Sie* `set` *weglassen, können Sie die Eigenschaft nur auslesen, nicht verändern – was durchaus gewollt sein kann.*

*An anderer Stelle in diesem Buch werden Sie Beispiele sehen, bei denen Sie bei* `set` *und* `get` *nicht einfach nur exakt die Daten in die Eigenschaft schreiben oder sie zurückgeben, sondern auch Modifikationen vornehmen.*

## 5.4 Methoden

Wenn Sie innerhalb einer Methode auf Objektvariablen, Funktionen oder Eigenschaften zugreifen möchten, gibt es die Möglichkeit, `this.` vor die Eigenschaft schreiben. `this` ist ein Verweis (eine *Referenz*) auf das aktuelle Klassenobjekt; darüber haben Sie auch Zugriff auf alle Eigenschaften (und natürlich auch Funktionen). Sie verwenden `this` stets innerhalb einer Klasse und können dann auch auf als `private` deklarierte Variablen und Methoden zugreifen.

Beispielsweise könnte die Funktion `Essen()` wie folgt implementiert werden[7]:

```
public void Essen(double Menge)
{
  this.Gewicht += Menge;
}
```

## 5.5 Konstruktor

Wenn Sie eine Klasse mit `new Klassenname()` erzeugen, wird im Hintergrund der so genannte Konstruktor der Klasse aufgerufen. Es handelt sich dabei um den Code, der bei der Initialisierung der Klasse ausgeführt wird. In den bisherigen Beispielen war so ein Konstruktor nicht

7. Die starke Vereinfachung in der Funktion sei hier gestattet.

vorhanden. Sie können aber selbst so eine Initialisierungsfunktion schreiben – oder gleich mehrere. Als Name der Funktion geben Sie einfach den Klassennamen an, Rückgabewert dürfen Sie keinen verwenden:

```
public Mensch(String name)
{
  this.Name = name;
}

public Mensch(String name, DateTime geburtsdatum)
{
  this.Name = name;
  this.Geburtsdatum = geburtsdatum;
}

public Mensch(String name, DateTime geburtsdatum, double gewicht)
{
  this.Name = name;
  this.Geburtsdatum = geburtsdatum;
  this.Gewicht = gewicht;
}
```

Das war jetzt nicht nur ein Konstruktor, sondern gleich drei. Da alle jedoch verschiedene Parameter erwarten, ist das kein Problem. Beim Aufruf entscheidet C# anhand der übergebenen Parameter, welchen der Konstruktoren er verwendet – wie beim Aufruf von Funktionen übrigens auch.

Sie können nun bei der Instanzierung einer Klasse auch Parameter angeben:

```
Mensch Steve = new Mensch("Steve Ballmer");
```

Die Variable `Steve` ist nun mit der Eigenschaft `Name` (=`"Steve Ballmer"`) vorbelegt.

Nachfolgend der komplette Code der Klasse, eingebettet in ein kleines Beispiel. Dabei wurde konsequent Data Hiding betrieben und alle Eigenschaftsvariablen als `private` deklariert.

```
<%@ Page Language="C#" %>
<script runat="server">
class Mensch
{
```

```
private String _Name;
private DateTime _Geburtsdatum;
private double _Gewicht;

// Konstruktoren
public Mensch(String name)
{
  this._Name = name;
}

public Mensch(String name, DateTime geburtsdatum)
{
  this._Name = name;
  this._Geburtsdatum = geburtsdatum;
}
public Mensch(String name, DateTime geburtsdatum, double gewicht)
{
  this._Name = name;
  this._Geburtsdatum = geburtsdatum;
  this._Gewicht = gewicht;
}
// Eigenschaften
public String Name
{
  get {
    return _Name;
  }
  set {
    _Name = value;
  }
}
public DateTime Geburtsdatum
{
  get {
    return _Geburtsdatum;
  }
  set {
    _Geburtsdatum = value;
  }
}
public double Gewicht
{
  get {
    return _Gewicht;
```

```
    }
    set {
      _Gewicht = value;
    }
  }
  // Methoden
  public void Essen(double Menge)
  {
    this._Gewicht += Menge;
  }
}
void Page_Load()
{
  Mensch Bill = new Mensch("Bill Gates");
  Bill.Gewicht = 70; // geraten
  Bill.Essen(0.15);  // Hamburger
  ausgabe.Text  = "<b>Name: </b>" + Bill.Name + "<br />";
  ausgabe.Text += "<b>Gewicht: </b>" + Bill.Gewicht.ToString();
}
</script>
<html>
<head>
  <title>OOP</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 5.1: Die Klasse* `Mensch` *(Mensch.aspx)*

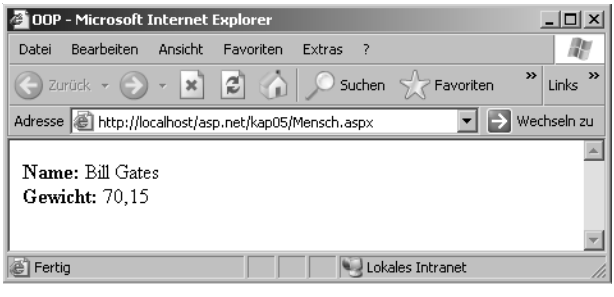



*Bild 5.1: Ein Hamburger schlägt sich im Gewicht nieder*

## 5.6 Vererbung

Die Klasse `Mensch` ist sehr allgemein gehalten. Viele Spezialisierungen sind möglich. Das Prinzip der OOP ist es, zunächst möglichst allgemeine Basisklassen zu erstellen und auf deren Basis dann spezialisierte Klassen. Im Beispielszenario sind geeignete spezialisiertere Klassen `Mann` und `Frau`. Ladies first, also beginnen wir mit der Frau. Die Klasse, von der geerbt wird, kommt hinter dem eigentlichen Klassennamen, getrennt von einem Doppelpunkt:

```
class Frau : Mensch
{
}
```

Die Klasse Frau *erbt* von der Klasse `Mensch` und hat zunächst genau dieselben Eigenschaften und Methoden wie die übergeordnete Klasse.

### 5.6.1 Funktionalität hinzufügen

Die Klasse Frau kann nun um zusätzliche Methoden erweitert werden. Ein Beispiel ist das Heiraten, denn dort ändert sich häufig der Nachname.


*Aus Gründen der Einfachheit sind in diesem Kapitel einige chauvinistische Vereinfachungen vorgenommen worden. Dies dient nur dazu, um die Konzepte anhand von annähernd realitätsnahen Szenarios demonstrieren zu können. Sie können beispielsweise anstelle von* `Frau` *dieselben Modifikationen bei einer Klasse* `Mann` *vornehmen.*

```
class Frau : Mensch
{
  public void Heiraten(Mensch Ehemann) {
    this.Name += "-" + Ehemann.Name;
  }
}
```

Wenn also Frau Lopez Herrn Affleck ehelicht, ist der neue Nachname Lopez-Affleck (erneut eine Vereinfachung). Ebenso wird vereinfacht angenommen, dass die Eigenschaft `Name` lediglich den Nachnamen enthält.

*Beachten Sie, dass hier auf* `this.Name` *zugegriffen worden ist, nicht auf* `this._Name`*. Denn:* `_Name` *ist eine vererbte Eigenschaft aus der Basisklasse, also nicht direkt in* `Frau` *definiert. Sie können nur darauf zugreifen, wenn sie entweder* `public` *oder* `protected` *ist – das ist beides nicht der Fall (*`_Name` *ist* `private`*). Die Eigenschaft* `Name` *ist dagegen public, also funktioniert dieser Zugriff.*

Auf diese Art und Weise können Sie leicht eine Spezialisierung vornehmen und neue Eigenschaften und Methoden zur Verfügung stellen.

### 5.6.2 Überschreiben

Bei einigen der spezialisierten Klassen ist es überaus sinnvoll, die Funktionalität bereits vorhandener Methoden zu verändern. Dies wird am Beispiel der Funktion `Essen()` vorgeführt. Aus dem weiblichen Freundeskreis des Autors dieser Zeilen kam die Beschwerde, dass Frauen angeblich doppelt so schnell zunehmen würden wie Männer. In C# lässt sich dieser Zusammenhang leichter nachbilden als im wahren Leben. Um eine Methode zu überschreiben, definieren Sie sie einfach erneut. Zwei Voraussetzungen:

- Sie müssen das Schlüsselwort `virtual` vor die ursprüngliche Methode in der Basisklasse schreiben.[8]
- Sie müssen das Schlüsselwort `override` vor die neue Methode schreiben.

```
override public void Essen(double Menge)
{
  this._Gewicht += 2 * Menge;
}
```

Es ist auch möglich, innerhalb der Funktion auch die ursprüngliche Funktion aus der übergeordneten Klasse aufzurufen. Mit `base` erhalten Sie eine Referenz auf eben jene. Die »feminine« `Essen()`-Methode könnte also auch folgendermaßen aussehen, mit demselben Ergebnis:

8. Wenn in der Basisklasse die Methode vollkommen ohne Code auskommt, können Sie dort auch `abstract` davor schreiben; dies ermöglicht ebenfalls Vererbung.

```
override public void Essen(double Menge)
{
  base.Essen(Menge);
  base.Essen(Menge);
}
```

Beim Klassenkonstruktor müssen Sie diese Technik sowieso meistens einsetzen, denn: Konstruktoren werden nicht vererbt! Sie können das allerdings wie folgt für die Klasse Frau erledigen:

```
public Frau(String name) : base(name)
{
}

public Frau(String name, DateTime geburtsdatum) : base(name, geburts-
datum)
{
}

public Frau(String name, DateTime geburtsdatum, double gewicht) :
base(name, geburtsdatum, gewicht)
{
}
```

Achten Sie also darauf, die richtigen Parameter in der richtigen Reihenfolge zu übergeben!

Nachfolgend die komplette Klasse Frau samt kleinem Beispiel. Das abgedruckte Listing enthält nur die wesentlichen Elemente, also nicht mehr die komplette Klasse `Mensch`; im Webarchiv erhalten Sie natürlich die vollständige Version:

```
<%@ Page Language="C#" %>
<script runat="server">
class Mensch
{
  // ...
  virtual public void Essen(double Menge)
  {
    this._Gewicht += Menge;
  }
}

class Frau : Mensch
{
  public Frau(String name) : base(name)
```

```
  {
  }
  public Frau(String name, DateTime geburtsdatum) : base(name, geburts-
datum)
  {
  }
  public Frau(String name, DateTime geburtsdatum, double gewicht) :
base(name, geburtsdatum, gewicht)
  {
  }

  public void Heiraten(Mensch Ehemann) {
    this.Name += "-" + Ehemann.Name;
  }

  override public void Essen(double Menge)
  {
    base.Essen(Menge);
    base.Essen(Menge);
  }
}


void Page_Load()
{
  Mensch Bill = new Mensch("Gates");
  Frau Melinda = new Frau("Mädchenname");
  Melinda.Gewicht = 60; // ebenfalls geraten
  Melinda.Essen(0.15); // Hamburger
  Melinda.Heiraten(Bill);
  ausgabe.Text  = "<b>Name: </b>" + Melinda.Name + "<br />";
  ausgabe.Text += "<b>Gewicht: </b>" + Melinda.Gewicht.ToString();
}
</script>
<html>
<head>
  <title>OOP</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 5.2: Die Klasse Frau erbt von* `Mensch` *(Frau.aspx)*


*Bild 5.2: Melinda Gates heißt natürlich nicht so*

So weit unser Ausblick auf die Objektorientierte Programmierung. In den nächsten beiden Kapiteln wird dieses Konzept in Hinblick auf ASP.NET Anwendung finden.

# 6 Code Behind

Anhand der vorangegangenen Beispiele haben Sie bereits gesehen, dass mit ASP.NET einer der größten Wünsche vieler Webentwickler in Erfüllung gegangen ist: die viel beschworene, komplette Trennung von Code und Inhalt. Eine ASP.NET-Seite bestand bis dato aus einem `<script>`-Block mit Code sowie einem HTML-Teil. Der HTML-Teil selbst war frei von Code. Allerdings konnten Sie aus dem `<script>`-Block heraus auf Elemente im HTML-Block zugreifen und diese so verändern.

## 6.1 Hintergrund

Obwohl alleine das schon ein gutes Konzept ist, ist eine noch strengere Trennung von Programmierung und Webdesign möglich. Das zugehörige Konzept heißt *Code Behind*, also »Hintergrund-Code«.

Worum geht es? Jeglicher C#- (oder VB.NET- oder JScript .NET- oder ...) Code wird in eine externe Klassendatei ausgelagert. Die ASP.NET-Seite erbt dann von dieser Klasse.

In der Praxis sieht das folgendermaßen aus: Sie verschieben Ihren kompletten C#-Code in eine neue Datei und betten diesen in eine Klasse ein. Das war es fast schon – aber eben nur fast. Da es sich um eine externe Klasse handelt, die unabhängig von einer ASP.NET-Seite steht, müssen Sie alle Web Controls und HTML Controls, auf die Sie in der ASP.NET-Seite zugreifen möchten, deklarieren, was etwas zusätzlichen Tippaufwand erfordert.

## 6.2 Ein Beispiel

Dies soll an einem kleinen Beispiel illustriert werden. Auf einer Website soll stets die aktuelle Uhrzeit ausgegeben werden. Dies kann – aufgrund der Trennung von Code und Content – nur auf dem Server geschehen, denn JavaScript (womit das clientseitig auch möglich wäre) kann im Browser deaktiviert werden. Dies wird in zwei Schritten umgesetzt:

- In allen entsprechenden ASP.NET-Seiten wird ein `<asp:Label>`-Element platziert, in dem letztendlich das Datum landen wird.
- In der Code-Behind-Datei schließlich wird die Uhrzeit in das `Label`-Element geschrieben.

Begonnen wird mit der Code-Behind-Datei. Da es sich um eine C#-Datei handelt, erhält sie die Dateiendung *.cs*. Als Dateiname wird *UhrzeitCB.cs* (*CB* steht für Code Behind) gewählt. Der Code darin wird nun stückweise aufgebaut.

Zunächst einmal wird eine Klassendefinition benötigt. Die Klasse heißt natürlich auch `UhrzeitCB` und erbt von `Page`, also der ASP.NET-Seitenklasse:

```
using System;

public class UhrzeitCB : System.Web.UI.Page
{
  // ...
}
```



Innerhalb dieser Klasse müssen Sie zunächst das `Label`-Element deklarieren, in das Sie schreiben möchten. Dieses Element verbirgt sich im Namespace `System.Web.UI.WebControls`.

```
  public System.Web.UI.WebControls.Label Uhrzeit;
```

Nun müssen Sie nur noch die Methode `Page_Load()` erstellen und in dieser die aktuelle Uhrzeit in das `Label`-Element schreiben. Dann wird diese beim Seitenaufruf angezeigt.

```
  public void Page_Load(Object o, EventArgs e)
  {
    if (Uhrzeit != null) {
      String uhr = DateTime.Now.ToShortTimeString();
      Uhrzeit.Text = "Aktuelle Zeit in München: " +
                     uhr;
    }
  }
```


*Natürlich können Sie in Code-Behind-Dateien auch eigene, neue Methoden deklarieren und diese dann von der ASP.NET-Seite aus aufrufen.*

Nachfolgend der komplette Code der Code-Behind-Datei:

```
using System;

public class UhrzeitCB : System.Web.UI.Page
{

  public System.Web.UI.WebControls.Label Uhrzeit;

  public void Page_Load(Object o, EventArgs e)
  {
    if (Uhrzeit != null) {
      String uhr = DateTime.Now.ToShortTimeString();
      Uhrzeit.Text = "Aktelle Zeit in München: " +
                     uhr;
    }
  }
}
```

*Listing 6.1: Die Code-Behind-Datei (UhrzeitCB.cs)*

Nun müssen Sie nur noch dafür sorgen, dass Ihre ASP.NET-Seiten die Code-Behind-Datei verwenden, also von der dort definierten Klasse erben. Dazu müssen Sie in der `Page`-Direktive im Seitenkopf Ihrer ASP.NET-Seiten zwei Parameter angeben:

- `Inherits`: Name der Klasse, von der geerbt werden soll
- `Src`: Name der Klassendatei, von der geerbt werden soll

Außerdem müssen Sie in der ASP.NET-Seite auch ein `Label`-Element unterbringen, in dem dann die Uhrzeit ausgegeben wird. Das id-Attribut dieses Elements muss `"Uhrzeit"` lauten, denn der Name des Labels in der Klasse ist ebenfalls `Uhrzeit`.

Nachfolgend eine einfache ASP.NET-Seite, die von *UhrzeitCB.cs* erbt und die Uhrzeit am rechten Rand ausgibt:

```
<%@ Page Language="C#"
  Inherits="UhrzeitCB" Src="UhrzeitCB.cs" %>
<html>
<head>
  <title>Code Behind</title>
</head>
<body>
```

```
<p align="right">
  <asp:Label id="Uhrzeit" runat="server" />
</p>
</body>
</html>
```

*Listing 6.2: Die ASP.NET-Seite mit der Uhrzeit (Uhrzeit.aspx)*


*Bild 6.1: Die Uhrzeit wird mit Code Behind eingeblendet*



Ist Ihnen im Code aufgefallen, dass wir explizit abgefragt haben, ob die Variable Uhrzeit den Wert `null` hat oder nicht?

```
if (Uhrzeit != null) {
  // ...
}
```

Dies hat einen einfachen Grund. Nehmen Sie einmal an, in der ASP.NET-Seite, die von der Code-Behind-Klasse erbt, befindet sich kein `Label`-Element `"Uhrzeit"` – aus welchen Gründen auch immer. Dann würde ein Zugriff auf `Uhrzeit.Text` zu einer Fehlermeldung führen. Deswegen die Abfrage. Aus diesem Grund wird auch bei folgender ASP.NET-Seite keine Fehlermeldung ausgegeben:

```
<%@ Page Language="C#"
  Inherits="UhrzeitCB" Src="UhrzeitCB.cs" %>
<html>
<head>
  <title>Code Behind</title>
</head>
<body>
```

```
<p align="right">
  Keine Uhr!
</p>
</body>
</html>
```

*Listing 6.3: Eine ASP.NET-Seite ohne* `Label`*-Element (KeineUhr.aspx)*

Die Seite enthält kein `Label`-Element, aufgrund der oben gezeigten Abfrage ist das jedoch kein Problem.


*Bild 6.2: Keine Fehlermeldung trotz fehlendem* `Label`*-Element*

*Wie zu sehen, haben Sie in der Code-Behind-Klasse eine Methode* `Page_Load()` *deklariert, was eine unter Umständen vorhandene lokale Funktion* `Page_Load()` *in der ASP.NET-Seite selbst überschreiben würde. Sie können dies aber beheben, wenn Sie in der Code-Behind-Klasse die lokale* `Page_Load()`*-Funktion aufrufen; fügen Sie dazu folgenden Code an den Anfang der Code-Behind-Methode* `Page_Load()` *ein:*

```
base.Page_Load(o, e);
```

*Das gilt natürlich auch für andere überschriebene Methoden/Funktionen.*

## 6.3 Code-Behind-Klassen kompilieren

Für eine höhere Performance ist es sinnvoll, Code-Behind-Klassen vorzukompilieren, also in eine DLL umzuwandeln. Es gestaltet sich dann der Aufruf einer Seite, die von der Code-Behind-Klasse erbt, schneller.

Dazu benötigen Sie den bereits bekannten C#-Compiler. Größter Unterschied zur Kompilierung einer Windows-Applikation (vergleiche Kapitel 3): Sie müssen angeben, dass eine DLL-Bibliothek erzeugt werden soll, was mit dem Schalter `/target:library` oder kürzer `/t:library` geht. Hier der Aufruf für die Kompilierung der Beispielklasse:

```
csc /target:library UhrzeitCB.cs
```

Die erzeugte Datei (*UhrzeitCB.dll*) kopieren Sie in das *bin*-Verzeichnis Ihrer Web-Anwendung. Von nun an können alle ASP.NET-Seiten der Anwendung von dieser Klasse erben. Lassen Sie dazu einfach den Parameter `Src` in der `Page`-Direktive weg und verwenden Sie nur `Inherits`. Im Webarchiv finden Sie eine entsprechend vorbereitete Datei *UhrzeitDLL.aspx*.

*Unter bestimmten Konfigurationen ist es nötig, beim Kompilieren einer C#-Datei noch anzugeben, welche Namespaces mit eingebunden werden sollen. Dies geht mit dem Schalter /reference oder kurz /r. Im obigen Beispiel wird beispielsweise der Namespace* `System` *und damit die Datei System.dll verwendet; die Kompilierung könnte dann wie folgt vonstatten gehen:*

```
csc /t:library /r:System.dll UhrzeitCB.cs
```

*Wenn Sie beim Kompilieren Fehlermeldungen erhalten, sollten Sie es so versuchen.*

## 6.4 Vorteile (und Nachteile)

Code Behind wird als eine große Errungenschaft von ASP.NET bezeichnet, allerdings ist nicht immer klar, wie viele dieser Aussagen nach reifer Überlegung und genauer Betrachtung gekommen sind und wie viele ohne Nachzudenken von ideologisch verbrämten Zeitgenossen getätigt wurden. An dieser Stelle deswegen einige Vorteile, wenn Code Behind eingesetzt wird:

- Tatsächliche Trennung von Code und Content
- Seitengestaltung fast komplett von einem Designer ohne ASP.NET-Kenntnisse durchführbar

- HTML-Anteil autark und unabhängig wartbar
- Zentrale Verwaltung von Code
- Schnelle Änderung von Kernfunktionalitäten für viele ASP.NET-Seiten auf einmal möglich, kein fehlerträchtiges Ausbessern in mehreren einzelnen Dateien
- Performancesteigerung, da zwischenkompilierte Version der Code-Behind-Datei im Cache/Speicher liegen könnte
- Performancesteigerung durch Vorkompilierung der Code-Behind-Datei in eine DLL

Es soll jedoch nicht verschwiegen werden, dass Code Behind auch ein paar kleine Nachteile mit sich bringen kann:

- Anstieg der Anforderungen an den Programmierer (in ASP.NET kann beispielsweise komplett ohne `class`-Konstrukte programmiert werden, aber nur ohne Code Behind)
- Microsoft ASP.NET Web Matrix unterstützt kein Code Behind.
- Schneller »Overkill« bei trivialen Skripten. Dies ist besonders gut bei Visual Studio .NET zu sehen, wo automatisch Code-Behind-Dateien erzeugt werden, auch wenn das gar nicht sinnvoll ist (hieran sind im Übrigen auch häufig gute und schlechte Zeitschriftenartikel zu unterscheiden).
- Ohne `Debug="True"` in der `Page`-Direktive keine aussagekräftigen Meldungen bei Laufzeitfehlern
- Erschwertes Debugging

Sie sehen: Code Behind ist sehr praktisch bei größeren Websites sowie zusammenhängenden ASP.NET-Seiten mit gemeinsamen Elementen. Bei kleinen Sites geht es auch problemlos ohne.

# 7 Eigene Steuerelemente

Das Code-Behind-Konzept aus dem vorangegangenen Kapitel kann konsequent weitergeführt werden. Es ist nicht nur möglich, Code-Behind-Klassen zu erstellen, sondern diese auch als eigenen Namespace zu verwenden und in die eigenen Seiten zu integrieren. Es ist auch möglich, eigene Steuerelemente zu programmieren – also selbst gemachte Web Controls.

## 7.1 Eigener Namespace

Im vorangegangenen Kapitel wurde eine Code-Behind-Klasse erstellt, in der auf allen Seiten automatisch die aktuelle Uhrzeit eingefügt worden ist. Diese Funktionalität – genauer gesagt die »Berechnung« der Uhrzeit – soll nun in eine Klasse ausgelagert und um eine Zusatzfunktion erweitert werden.

Die Klasse, die erstellt wird, heißt `UhrzeitNS` – `NS` steht für Namespace – und enthält eine String-Eigenschaft `zeit`, in der die aktuelle Uhrzeit steht. Der Aufbau der Datei ist wie der einer Klasse, nur dass um die Hauptklasse herum ein `namespace`-Konstrukt gebastelt wird. Als Namespace-Name wählen wir `NittyGrittyNS`. Der Torso der Datei kann dann ungefähr folgendermaßen aussehen:

```
using System;

namespace NittyGrittyNS
{

  public class UhrzeitNS
  {

    private String zeit;
    // ...

  }

}
```

Der Klassenkonstruktor setzt die Variable `zeit` auf die aktuelle Uhrzeit. Der Name des Konstruktors ist bekanntermaßen der Name der Klasse:

```
public UhrzeitNS()
{
  zeit = DateTime.Now.ToShortTimeString();
}
```

Als weiterer »Bonus« wird ein zusätzlicher Konstruktor erstellt. Bei diesem kann ein Text angegeben werden, der vor der Uhrzeit ausgegeben wird (im vorangegangenen Kapitel war das `"Aktuelle Zeit in München: "`).

```
public UhrzeitNS(String Praefix)
{
  zeit = Praefix + DateTime.Now.ToShortTimeString();
}
```

Allerdings ist die Eigenschaft `zeit` als `private` deklariert worden, kann also nicht direkt von außen gelesen werden. Damit die Klasse auch aus einer ASP.NET-Seite heraus verwendet werden kann, muss die Eigenschaft `zeit` les- und schreibbar sein. Dazu dient die Klasseneigenschaft `Zeit` (mit großem Z). Auch hier gilt: Wenn die Variable gesetzt wird, wird nur der Text vor der Zeit geändert.

```
public String Zeit
{
  set {
    zeit = value + DateTime.Now.ToShortTimeString();
  }
  get {
    return zeit;
  }
}
```

Nachfolgend der komplette Code für den (zugegebenermaßen sehr simplen) Namespace:

```
using System;

namespace NittyGrittyNS
{

  public class UhrzeitNS
  {
```

```
    private String zeit;

    public UhrzeitNS()
    {
      zeit = DateTime.Now.ToShortTimeString();
    }

    public UhrzeitNS(String Praefix)
    {
      zeit = Praefix + DateTime.Now.ToShortTimeString();
    }

    public String Zeit
    {
      set {
        zeit = value + DateTime.Now.ToShortTimeString();
      }
      get {
        return zeit;
      }
    }
  }
}
```

*Listing 7.1: Der komplette Namespace (NittyGrittyNS.cs)*

Im nächsten Schritt müssen Sie den Namespace kompilieren; hier kommt wieder *csc* zum Einsatz. Da Sie eine DLL-Datei erzeugen möchten, benötigen Sie den Schalter */t:library*:

```
csc /target:library NittyGrittyNS.cs
```

Sie erhalten eine Datei *NittyGrittyNS.dll*, die Sie in das *bin*-Verzeichnis Ihrer Webanwendung kopieren müssen, um sie einsetzen zu können.


*Der erzeugte Dateiname ist automatisch der alte Dateiname, ohne die Endung .cs, dafür mit der Erweiterung .dll. Wenn Sie einen anderen Namen wünschen, können Sie die Datei nach der Kompilierung umbenennen oder gleich den Schalter /out verwenden:*

```
csc /target:library /out: Name.dll NittyGrittyNS.cs
```

Nach erfolgreicher Kompilierung – falls es nicht klappt, Quellcode und *.dll*-Datei befinden sich auch im Webarchiv – können Sie die neue Klasse direkt in Ihre Seiten einbinden:

1. Laden Sie den Namespace in der ASP.NET-Seite per `Import`-Anweisung:

   ```
   <%@ Import Namespace="NittyGrittyNS" %>
   ```

2. Erzeugen Sie eine Instanz der im Namespace definierten Klasse und zwar im Format `Namespacename.Klassenname`.

   ```
   NittyGrittyNS.UhrzeitNS u =
     new NittyGrittyNS.UhrzeitNS();
   ```

Nachfolgend eine exemplarische ASP.NET-Datei, bei der die Klasse dreimal eingesetzt wird:

- Einmal wird sie bloß instanziert und dann verwendet.
- Beim zweiten Mal wird der alternative Konstruktor mit Präfix verwendet.
- Beim dritten Mal wird die Eigenschaft `Zeit` manuell gesetzt.

Hier der zugehörige Code:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="NittyGrittyNS" %>
<script runat="server">
void Page_Load()
{
  // 1. Direkte Verwendung
  NittyGrittyNS.UhrzeitNS u1 =
    new NittyGrittyNS.UhrzeitNS();
  Uhrzeit1.Text = u1.Zeit;

  // 2. Alternativer Konstruktor
  NittyGrittyNS.UhrzeitNS u2 =
    new NittyGrittyNS.UhrzeitNS(
      "Aktuelle Uhrzeit in München: ");
  Uhrzeit2.Text = u2.Zeit;

  // 3. Manuelles Setzen von "Zeit"
  NittyGrittyNS.UhrzeitNS u3 =
    new NittyGrittyNS.UhrzeitNS();
  u3.Zeit = "Aktuelle Uhrzeit in Ratingen: ";
  Uhrzeit3.Text = u3.Zeit;
}
```

```
</script>
<html>
<head>
  <title>Eigener Namespace</title>
</head>
<body>
<p>
  <asp:Label id="Uhrzeit1" runat="server" />
</p>
<p>
  <asp:Label id="Uhrzeit2" runat="server" />
</p>
<p>
  <asp:Label id="Uhrzeit3" runat="server" />
</p>
</body>
</html>
```

*Listing 7.2: Der eigene Namespace im Einsatz (UhrzeitNS.aspx)*

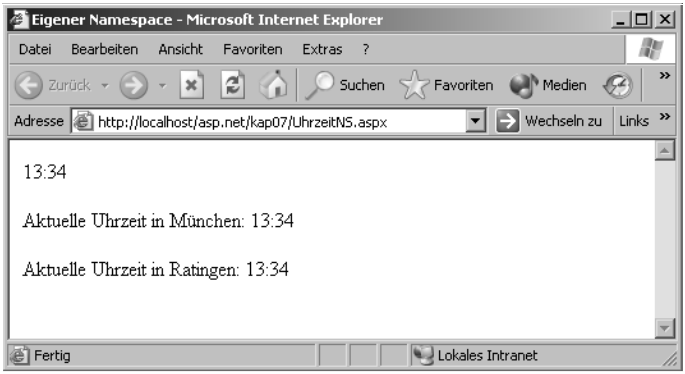



*Bild 7.1: Der eigene Namespace wird dreimal verwendet*


*Sie sehen an diesem Beispiel einen möglichen Einsatz für »Data Hiding«. Die Eigenschaft* `Zeit` *kann zwar gesetzt werden, die »eigentliche« Klassenvariable* `zeit` *jedoch nicht. So wird sichergestellt, dass der Benutzer lediglich einen Präfix vor die Uhrzeit schreiben kann, nicht jedoch die Uhrzeit überschreiben.*

## 7.2 Benutzersteuerelemente (User Controls)

Die Dateiendung *.ascx* ist für so genannte User Controls, Benutzersteuerelemente, reserviert. Dabei handelt es sich im Wesentlichen um eine weitere ASP.NET-Seite, die in andere ASP.NET-Seiten eingebunden werden kann. Allerdings ist ein programmtechnischer Zugriff auf Elemente des Steuerelements verfügbar, was einige komplexe Anwendungen ermöglicht.

### 7.2.1 User Control einbinden

Beginnen wir mit einem einfachen Beispiel. Die folgende Vorlage gibt – wieder einmal – die aktuelle Uhrzeit aus:

```
<%@ Control ClassName="UhrzeitUC1" Language="C#" %>
<script runat="server">
void Page_Load()
{
  Uhrzeit.Text = DateTime.Now.ToShortTimeString();
}
</script>

<asp:Label id="Uhrzeit" runat="server" />
```

*Listing 7.3: Das erste User Control (UhrzeitUC1.ascx)*

Im Wesentlichen besteht diese Datei aus einem `Label`-Element sowie Code, um in diesem Element die aktuelle Zeit auszugeben. Neu ist die erste Zeile, die `Control`-Direktive, mit folgenden beiden Parametern:

- `ClassName`: Name der Klasse, den können Sie hier festlegen (Sie benötigen ihn später in der ASP.NET-Datei).
- `Language`: Sprache, in der das Control verfasst worden ist, hier natürlich C#.

Um dieses Steuerelement in Ihre Seiten einzubauen, müssen Sie die `Register`-Direktive verwenden und dabei folgende Parameter setzen:

- `TagPrefix`: die Zeichenkette, die Sie vor den Namen des Controls schreiben werden (bei Web Controls ist das beispielsweise `asp`)
- `TagName`: der zuvor definierte Klassenname (also der Name des Controls) oder ein frei gewählter Name
- `Src`: der Dateiname der zugehörigen *.ascx*-Datei.

```
<%@ Register TagPrefix="NittyGritty"
             TagName="UhrzeitUC1"
             Src="UhrzeitUC1.ascx" %>
```

Um das Control einzubinden, müssen Sie lediglich die Syntax `TagPrefix:TagName` verwenden und dürfen das `runat="server"` nicht vergessen:

```
<NittyGritty:UhrzeitUC1 id="UC" runat="server" />
```

Hier der komplette Code, der die *.ascx*-Datei einbindet und die Uhrzeit einfügt:

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="NittyGritty"
             TagName="UhrzeitUC1"
             Src="UhrzeitUC1.ascx" %>
<html>
<head>
  <title>User Control</title>
</head>
<body>
<p>
  <NittyGritty:UhrzeitUC1 id="UC" runat="server" />
</p>
</body>
</html>
```

*Listing 7.4: Das User Control wird eingebunden (UhrzeitUC1.aspx)*

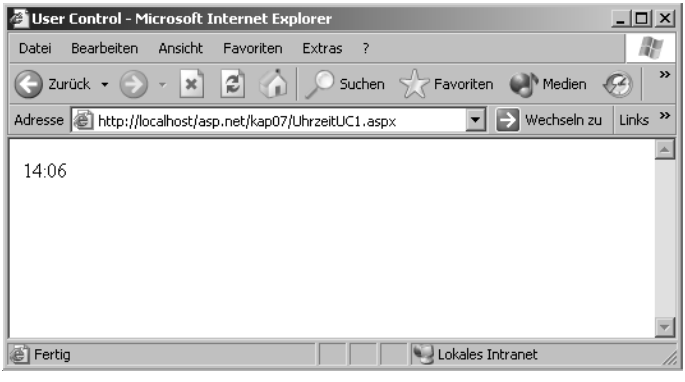



*Bild 7.2: Das User Control in Aktion*

### 7.2.2 User Control steuern

Auch bei User Controls können Sie mit Eigenschaften arbeiten und so die Ausgabe des Steuerelements anpassen. Um dies zu realisieren, wird zunächst eine etwas seltenere, noch aus ASP-Zeiten bekannte Form der Integration von Programmcode verwendet: `<% ... %>`.

```
<span><% =zeit %></span>
```

Nun müssen Sie nur noch wie in Abschnitt 7.1 gesehen eine Klassenvariable für die Uhrzeit sowie eine Eigenschaft mit Funktionalität zum Lesen und Setzen der Zeit implementieren. Nachfolgend der komplette Code:

```
<%@ Control ClassName="UhrzeitUC" Language="C#" %>
<script runat="server">
private String zeit = DateTime.Now.ToShortTimeString();

public String Zeit
{
  set {
    zeit = value + DateTime.Now.ToShortTimeString();
  }
  get {
    return zeit;
  }
}
</script>

<span><% =zeit %></span>
```

*Listing 7.5: Das erweiterte User Control (UhrzeitUC2.ascx)*

Beim Einbau in eine ASP.NET-Seite gibt es wie zuvor auch drei Möglichkeiten:

- Direkter Einbau; der Standardwert für die Uhrzeit (nur Minuten und Sekunden) wird ausgegeben.
- Einbau mit Setzen der Eigenschaft `Zeit` direkt im Control.
- Einbau mit programmtechnischem Setzen der Eigenschaft `Zeit`.

Im Listing sind alle drei Möglichkeiten verwendet worden:

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="NittyGritty"
             TagName="UhrzeitUC2"
             Src="UhrzeitUC2.ascx" %>
<script runat="server">
void Page_Load()
{
  UC3.Zeit = "Uhrzeit in Seefeld: ";
}
</script>
<html>
<head>
  <title>User Control</title>
</head>
<body>
<p>
  <NittyGritty:UhrzeitUC2 id="UC1" runat="server" />
</p>
<p>
  <NittyGritty:UhrzeitUC2 id="UC2"
    Zeit="Uhrzeit in München: " runat="server" />
</p>
<p>
  <NittyGritty:UhrzeitUC2 id="UC3" runat="server" />
</p>
</body>
</html>
```

*Listing 7.6: Drei Verwendungen für das User Control (UhrzeitUC2.aspx)*

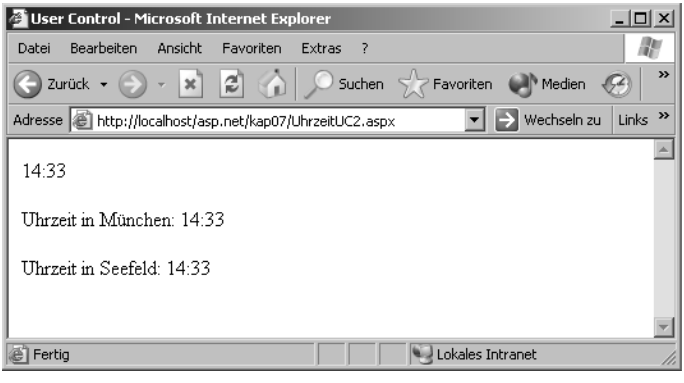



*Bild 7.3: Dreimal dasselbe Benutzersteuerelement*

Sie sehen: Mit Benutzersteuerelementen können Sie nicht nur Code auslagern, sondern auch Web Controls und damit auch Designelemente, was eine starke Modularisierung der ASP.NET-Seite ermöglicht.

## 7.3 Eigene Steuerelemente (Custom Controls)

Wenn Sie mehr Flexibilität wünschen, sollten Sie einen Blick auf Custom Controls werfen, sozusagen der große Bruder von User Controls. Hier läuft alles komplett objektorientiert ab – mit den damit verbundenen Vor- und Nachteilen. Sie haben maximale Kontrolle über das Steuerelement, müssen sich aber auch um alles kümmern – inklusive der Ausgabe. Aber der Reihe nach.

Der allgemeine Aufbau der Klasse sieht bekannt aus: Sie müssen wieder einen Namespace erzeugen und darin eine Klasse deklarieren; diese muss – und das ist neu – von `System.Web.UI.Control` abgeleitet werden, damit Sie sie als Web Control verwenden können:



```
using System;
using System.Web;
using System.Web.UI;

namespace NittyGrittyCC
{

  public class UhrzeitCC : Control
  {

    private String zeit;

    public String Zeit
    {
      set {
        zeit = value + DateTime.Now.ToShortTimeString();
      }
      get {
        return zeit;
      }
    }

  }

}
```

Was hier zunächst noch fehlt, ist der Konstruktor für die Klasse. Dieser wird bei der Initialisierung des Steuerelements ausgeführt; der zugehörige Methodenname ist `OnInit()`. Dieser kann wie folgt aussehen:

```
protected override void OnInit(EventArgs e)
{
  base.OnInit(e);
  if (zeit == null) {
    zeit = DateTime.Now.ToShortTimeString();
  }
}
```

Drei Dinge sind hier bemerkenswert:

- Die Methode `OnInit()` überschreibt natürlich die automatisch von der `Control`-Klasse vorgegebene Methode `OnInit()`; deswegen das Schlüsselwort `override`.
- Allerdings sollte natürlich die `Control`-Methode `OnInit()` trotzdem ausgeführt werden; dies geschieht durch `base.OnInit(e)`.
- Da im Control selbst auch das Präfix für die Uhrzeit gesetzt werden kann, muss explizit überprüft werden, ob die Eigenschaft `zeit` bereits einen Wert hat (dann darf nichts verändert werden):

  ```
  if (zeit == null) { }
  ```

Was nun noch fehlt, ist die Ausgabe des zugehörigen HTML-Codes. Dazu benötigen Sie ein `HtmlTextWriter`-Objekt, das sich unterhalb von `System.Web.UI` befindet. Dieser hat unter anderem die folgenden Methoden:

- `AddAttribute("Name", "Wert")`: Sorgt dafür, dass das als Nächstes ausgegebene Tag das Attribut `Name="Wert"` enthält.
- `RenderBeginTag("xyz")`: Beginnt das angegebene Tag, im Beispiel also `<xyz ...`
- `RenderEndTag()`: Beendet das zuletzt gestartete Tag.
- `Write()`: Gibt den angegebenen Text aus.

Alle Schreibausgaben des `HtmlTextWriter`-Objekts werden direkt an den Client, also im Normalfall den Webbrowser, geliefert. Für das kleine Uhrzeitbeispiel kann die Ausgabemethode, die im Übrigen `Render()` heißen muss, wie folgt aussehen:

```
protected override void Render(HtmlTextWriter h)
{
  h.RenderBeginTag("span");
  h.Write(HttpUtility.HtmlEncode(zeit));
  h.RenderEndTag();
}
```


*Da auch die* `Render()`*-Methode in der* `Control`*-Klasse vordefiniert ist, müssen Sie sie auch hier überschreiben und dazu das Schlüsselwort* `override` *verwenden.*

Nachfolgend das komplette Listing für das Custom Control:

```
using System;
using System.Web;
using System.Web.UI;

namespace NittyGrittyCC
{
  public class UhrzeitCC : Control
  {
    private String zeit;

    public String Zeit
    {
      set { zeit = value + DateTime.Now.ToShortTimeString(); }
      get { return zeit; }
    }
    protected override void OnInit(EventArgs e)
    {
      base.OnInit(e);
      if (zeit == null)
        zeit = DateTime.Now.ToShortTimeString();
    }
    protected override void Render(HtmlTextWriter h)
    {
      h.RenderBeginTag("span");
      h.Write(HttpUtility.HtmlEncode(zeit));
     h.RenderEndTag();
    }
  }
}
```

*Listing 7.7: Das komplette Custom Control (NittyGrittyCC.cs)*

Die erzeugte C#-Datei kompilieren Sie wie gehabt mit *csc*:

```
csc /t:library /out:UhrzeitCC.dll NittyGrittyCC.cs
```

*Auch hier gilt natürlich wieder: Bei obskuren Fehlermeldungen können Sie versuchen, die verwendeten DLLs des .NET Frameworks per /reference-Schalter einzubinden.*

```
csc /r:System.dll,System.Web.dll,System.Web.UI.dll
/t:library /out:UhrzeitCC.dll NittyGrittyCC.cs
```

Auch diese erzeugte DLL (*UhrzeitCC.dll*) gehört ins k-Verzeichnis der Webanwendung. Dann können Sie sie wieder mit der Register-Direktive in Ihre ASP.NET-Seiten einsetzen. Die folgenden drei Angaben müssen Sie dabei tätigen:

- `TagPrefix`: Zeichenkette vor dem Namen des Controls (bei den »offiziellen« .NET-Web-Controls ist das etwa `asp`)
- `Assembly`: der Dateiname der DLL, ohne Endung *.dll*!
- `Namespace`: der in der Klasse verwendete Namespace

```
<%@ Register TagPrefix="NittyGritty"
             Assembly="UhrzeitCC"
             Namespace="NittyGrittyCC" %>
```

Der Einbau geht dann im Format `TagPrefix:Klassenname` vonstatten. Auch hier gibt es wieder drei Einbaumöglichkeiten:

- Direkt, ohne Modifikation
- Direkt, mit Parametern
- Direkt, mit programmtechnischem Zugriff auf Eigenschaften des Steuerelements

Das folgende Listing zeigt alle drei Alternativen:

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="NittyGritty"
             Assembly="UhrzeitCC"
             Namespace="NittyGrittyCC" %>
<script runat="server">
void Page_Load()
{
  CC3.Zeit = "Uhrzeit in Starnberg: ";
}
</script>
```

```
<html>
<head>
  <title>User Control</title>
</head>
<body>
<p>
  <NittyGritty:UhrzeitCC id="CC1" runat="server" />
</p>
<p>
  <NittyGritty:UhrzeitCC id="CC2"
    Zeit="Uhrzeit in München: " runat="server" />
</p>
<p>
  <NittyGritty:UhrzeitCC id="CC3" runat="server" />
</p>
</body>
</html>
```

*Listing 7.8: Das Custom Control in der ASP.NET-Seite (UhrzeitCC.aspx)*



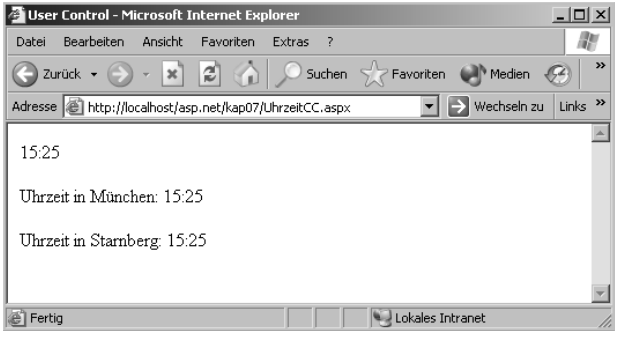



*Bild 7.4: Die drei Ausgaben des Steuerelements*

Damit haben Sie die ultimative Flexibilität. Wenn Sie einmal mit der Funktionalität in den Web Controls von ASP.NET nicht zufrieden sind, entwickeln Sie einfach eigene Steuerelemente.

# 8 C# vs. VB.NET

Wer denkt, Glaubenskriege werden hauptsächlich von der Open-Source-Fraktion initiiert, der irrt. Schon kurz nach Vorstellung der .NET-Strategie und der damit verbundenen Vorstellung der neuen Programmiersprache C# waren die Newsgroups voll von kindischen Anfeindungen. Die meisten ehemaligen C(++)- und Java-Programmierer fanden in C# viele ihnen bekannte Elemente wieder und sahen in Visual Basic .NET eine bloße Anpassung der »Anfängersprache« BASIC auf .NET-Verhältnisse. Eingefleischte Visual Basic-Anhänger verteufelten im Gegenzug den Microsoft-Neuling als kompliziert und unausgereift.

Diese Diskussion, die derart groteske Züge angenommen hatte, veranlasste den bekannten US-Autor Dan Appleman in seinem E-Book *»Visual Basic.NET or C#...Which to Choose?«* zu folgender treffenden Aussage:

> *Any of you who feel that the syntax:*
>
> ```
> if() { }
> ```
>
> *is somehow morally superior to:*
>
> ```
> If ... Then
> End If
> ```
>
> *are fools.*

Mit anderen Worten: Jeder, der seine Lieblingssprache auf einen künstlichen Sockel erhebt, hat das auch nötig. Ein sehr treffender Ausspruch.

Alleine ein Blick in den von Visual Basic .NET bzw. C# erzeugten MSIL-Code zeigt, dass die Sprachen quasi gleichwertig sind. Hin und wieder ist die Mär zu lesen, dass C# aufgrund der strengeren Typisierung und der Tatsache, dass bei Variablennamen nicht zwischen Groß- und Kleinschreibung unterschieden wird, etwas schneller ist.


*Sobald jedoch bei Visual Basic .NET die Anweisung* `Option Strict` *verwendet wird, verschwindet dieser Vorteil.*

In dieser Ausgabe des Buches wird – wie bereits im Vorwort erläutert – auf C# gesetzt. An dieser Stelle sollen kurz die wichtigsten Unterschiede zwischen dieser Sprache und Visual Basic .NET vorgestellt werden.

## 8.1 Allgemeine Unterschiede

Der auffälligste Unterschied – von der Sprachsyntax einmal abgesehen – ist die Unterscheidung zwischen Groß- und Kleinschreibung bei C#. Während Visual Basic .NET auch einen Zugriff auf `request.QUERYSTRING` ermöglicht, ist das unter C# nicht möglich; hier werden Sie also zu saubererem Programmieren gezwungen[9]. Bei vielen als *case-sensitive* konzipierten Sprachen ist das ein Problem – ein Tippfehler macht aus einer Variable `ausgabe` die Variable `Ausgabe`, welche dann eine komplett verschiedene ist. Nicht so bei C#, denn dort muss (wie in Visual Basic .NET) auch jede Variable zunächst deklariert werden.

Ein zweiter, erheblicher Unterschied ist die Umwandlung von einem Datentyp in einen anderen. Visual Basic .NET nimmt hier dem Programmierer in der Standardeinstellung eine ganze Menge an Arbeit ab. Bei C# ist das nicht so. Es wird eben nicht implizit alles in das jeweils gewünschte Format konvertiert. Das wird bereits beim Arbeiten mit numerischen Werten und Strings sichtbar. Betrachten Sie folgenden, in Visual Basic .NET geschriebenen Code:

```
<%@ Page Language="VB" %>
<script runat="server">
Sub Page_Load
  Dim i As String = "41"
  ausgabe.Text =
    String.Format("Die Antwort ist {0}", i+1))
End Sub
</script>
<html>
<head>
  <title>C# vs. VB.NET</title>
</head>
```

9. Außerdem muss jedem VB.NET-Entwickler klar sein, dass er so nicht CLS-konform programmiert.

```
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 8.1: Das Umwandlungsbeispiel in VB.NET (umwandlung.vb.aspx)*

Wie erwartet, wird im Browser der folgende Text ausgegeben:

```
Die Antwort ist 42
```

Bei der »Berechnung« von `i+1` wird automatisch der String in eine Zahl umgewandelt, damit `1` addiert werden kann.


*Streng genommen sollten Strings in Visual Basic .NET sowieso nur mit dem Operator `&` konkateniert (aneinander gefügt) werden, aber das ist ein anderes Thema.*

Was passiert aber nun, wenn dieser Code in C# umgewandelt wird? Zunächst das Listing:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  String i = "42";
  ausgabe.Text =
    String.Format("Die Antwort ist {0}", i+1);
}
</script>
<html>
<head>
  <title>C# vs. VB.NET</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 8.2: Das Umwandlungsbeispiel in C# (umwandlung.cs.aspx)*


*Bild 8.1: Die (überraschende) Ausgabe des C#-Listings*

Dieses Listing gibt – wie in Bild 8.1 zu sehen – als Ergebnis `421` aus, hier wurden also die Strings aneinander gehängt, nicht als Zahlen addiert. Warum? Nun, `String.Format()` erwartet als zweiten Parameter einen String, worum sich aber der VB.NET-Compiler wenig geschert hat. Der C#-Compiler hat dagegen eine strikte Umwandlung in eine Zeichenkette vorgenommen, würde aber als Parameter dennoch einen Zahlenwert akzeptieren.

Damit Sie vor solchen unliebsamen Überraschungen gefeit sind, sollten Sie – in beiden Sprachen! – stets eine entsprechende Typumwandlung vornehmen. In C# haben Sie hierzu zwei einfache Möglichkeiten:

- Sie verwenden die Methoden der `Convert`-Klasse: `Convert.ToInt32(i)` würde i in einen Integerwert umwandeln, sofern möglich.
- Sie »casten«, indem Sie den Datentyp davor schreiben: `(int)i`

Bei einer Umwandlung in einen String haben die meisten Datentypen sowieso eine Methode `ToString()` implementiert, die genau dieses erledigt.

Fazit: Wenn Sie in Visual Basic .NET hineinschnuppern möchten, werden Sie am Anfang vermutlich auf weniger Compiler-Fehlermeldungen stoßen als bei C#. Ihr Code ist dadurch allerdings nicht unbedingt sauber. Das Programmieren mit C# sorgt oft für mehr Frust, aber auch für besseren Code. Was allerdings nicht heißt, dass VB.NET-Code automatisch schlechter ist als C#-Code; es gibt viele exzellente Visual Basic .NET-Programmierer.

## 8.2 Fehlende/zusätzliche Funktionen

Häufig wird lamentiert, in C# würden bestimmte Funktionen von Visual Basic .NET fehlen. Ein Großteil dieser Beschwerden kann ignoriert werden, beispielsweise die String-Behandlungsfunktionen wie etwa `Left`, `Mid`, `Right` und Konsorten. Diese Funktionalität ist nämlich in der `String`-Klasse untergebracht und dort auch sprachunabhängig einsetzbar.


*Wer nicht darauf verzichten kann: Diese Funktionen gibt es sogar noch, und zwar in der Klasse* `Microsoft.VisualBasic.Strings`*!*

Allerdings gibt es zwei Beispiele für Funktionen, die doch hin und wieder schön wären:

`CType()` sorgt für eine Typumwandlung, beispielsweise können Sie so die Zeichenkette `"Normal"` in `System.Web.Mail.MailFormat.Normal` umwandeln (Hintergründe dazu in Kapitel 25). Mit den Casting-Möglichkeiten von C# ist das leider nicht möglich, `(System.Web.Mail.MailFormat)"Normal"` funktioniert leider nicht.

Eine weitere, sehr praktische Funktion ist `IsNumeric()`, die überprüft, ob ein Wert numerisch ist oder nicht. Dies ist insbesondere bei der Validierung von Formulareingaben praktisch und nützlich. Auch hierzu gibt es keine C#-Entsprechung, aber drei Alternativen:

1. Fangen Sie über `try-catch` beim Umwandeln mit der `Convert`-Klasse Umwandlungsfehler ab. Das ist ein gangbarer Weg, aber deutlich länger und komplizierter als unter VB.NET!

```
public static bool IsNumeric(Object o)
{
  try {
    double d = System.Double.Parse(o.ToString(),
      System.Globalization.NumberStyles.Any);
  }
  catch (Exception e) {
    return false;
  }
}
```


*Ein Blick in den generierten MSIL-Code zeigt jedoch, dass die oben vorgestellte C#-Variante effizienter zu sein scheint. Vermutlich liegt das daran, dass* `IsNumeric()` *zusätzliche Fehlerüberprüfungen vornimmt.*

2. Verwenden Sie einen regulären Ausdruck, aber auch dies ist nicht so trivial wie ein Aufruf von `IsNumeric()`.
3. So ähnlich wie der 1. Weg, nur ohne Abfangen: Verwenden Sie die Methode `TryParse()` der `Double`-Klasse:

```
bool istNumerisch = Double.Parse(
  i,
  System.Globalization.NumberStyles.Any,
  NumberFormatInfo.InvariantInfo,
  out ergebnis
);
```

Ein letztes Sprachelement, das nicht 1:1 von der einen in die andere Sprache übernommen werden kann und hier vorgestellt werden soll, ist `null`. Dies gibt es in der Tat nicht in Visual Basic .NET, aber es gibt einen Ersatz, `Nothing`. Ergebnis: Die folgende Abfrage:

```
if (liste.SelectedItem != null) { }
```

sieht wie folgt in Visual Basic .NET aus:

```
If Not liste.SelectedItem Is Nothing Then
  ' ...
End If
```

Damit sind aber auch die am häufigsten verwendeten Sprachelemente und -konstrukte abgehandelt.

## 8.3 Fazit

C# und Visual Basic .NET haben eine komplett unterschiedliche Syntax, sind aber vom Sprachumfang – vor allem dank des umfassenden .NET Frameworks – praktisch identisch. Die wenigen in der Praxis relevanten Unterschiede sind in diesem Kapitel aufgeführt; haben Sie sich einmal an den »Workaround« gewöhnt, umschiffen Sie diese Klippen zukünftig schnell und zuverlässig.

# Teil II


## Take that!

# 9 Die Klasse String

Falls Sie bereits Kontakt zu den früheren Varianten von ASP hatten, werden Sie sich vielleicht daran erinnern, dass es in VBScript eine ganze Reihe von Stringmanipulationsfunktionen gibt.

Mit der Einführung von ASP.NET sind diese sprachunabhängig implementiert und nebenbei noch einmal deutlich um viele Funktionen erweitert worden.

Der Auswertung von regulären Ausdrücken wurde durch eine eigene Klasse, die Klasse `Regex`, besonderes Augenmerk gewidmet. Da dieses Gebiet auf einem recht tiefen theoretischen Konzept beruht, haben wir ihm ein eigenes Kapitel gewidmet, wo wir Ihnen reguläre Ausdrücke im Allgemeinen und die speziellen Möglichkeiten in ASP.NET näher bringen werden.

In diesem Abschnitt beschäftigen wir uns mit den einfachen Stringmanipulations- und Stringuntersuchungs-Methoden, die uns die String-Klasse bietet.

Zunächst aber möchten wir Sie auf die Möglichkeit aufmerksam machen, einen String mit dem Wert `Empty` zu belegen. Mit diesem Wert wird ein leerer String repräsentiert. In einem engen Zusammenhang hierzu stehen eine Reihe von Eigenschaften.

## 9.1 Die Eigenschaften

Hier nun die beiden wichtigen Eigenschaften der Klasse `String`.

### 9.1.1 Char

Mit der Eigenschaft `Char` können Sie das Zeichen an einer von Ihnen festgesetzten Position eines Strings zurückgeliefert bekommen.

Diese Eigenschaft wird zur Indexierung der `String`-Klasse mit C# verwendet.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Beispiel;
  int Ergebnis;
  char Ergebnis2;

  Beispiel = "Dies ist ein Beispielstring.";
  a.Text = "Char- und Length-";
  a.Text += "Eigenschaften: <br /> ";

  Ergebnis2 = Beispiel[7];
  a.Text += "Zeichen an der Position 7: ";
  a.Text += Ergebnis2.ToString();
  a.Text += "<br />";

  Ergebnis = Beispiel.Length;
  a.Text += "Laenge von: ";
  a.Text += Beispiel + " ist: ";
  a.Text += Ergebnis.ToString() + "<br />";
}
</script>
<html>
<head><title>String-Eigenschaften</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.1: String-Eigenschaften (eigenschaften.aspx)*

Das Ergebnis dieses Codeblocks ist Folgendes:

```
Char- und Length-Eigenschaften:
Zeichen an der Position 7: t
Laenge von: dies ist ein Beispielstring. ist: 28
```

### 9.1.2 Length

Mit der Eigenschaft `Length` erhalten Sie die Anzahl von Zeichen, die die zu untersuchende Instanz lang ist. Hierbei werden auch Leerzeichen mitgezählt.

Das Beispiel hierzu finden Sie unter der Beschreibung zu `Char`.

## 9.2 Die Methoden

Weiterhin gibt es eine ganze Reihe von Methoden um Strings zu manipulieren oder auch um zwei Strings miteinander zu vergleichen.

Nachfolgend sind die wichtigsten Methoden zur Stringmanipulation kurz für Sie aufgelistet und erklärt.

### 9.2.1 Clone

Mit der Methode `Clone` können Sie eine zweite Referenzierung zu einer Instanz eines Strings herstellen. Sie verdoppeln sozusagen die Stringinformationen. Dies bedeutet auch, dass die Manipulationen an der einen Instanz bei der geklonten Instanz übernommen werden.

Falls Sie nur einen String verdoppeln wollen um ihn dann anderweitig zu verändern, sollten Sie dies beispielsweise über `Copy` machen.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Beispiel;
  object Clone;
  string Kopie;

  Beispiel = "Dies ist ein Beispielstring.";
  a.Text = "Clone vs. Copy-";
  a.Text += "Methode: <br /> ";

  Clone = Beispiel.Clone();
  Kopie = String.Copy(Beispiel + "und Mehr...");
  a.Text += "Original: <br />";
  a.Text += Beispiel + "<br />";
  a.Text += "Clone: <br />";
  a.Text += Clone + "<br />";
  a.Text += "Kopie: <br />";
  a.Text += Kopie + "<br />";
}
</script>
```

```
<html>
<head><title>Copy vs. Clone</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.2:* `Clone()` *vs.* `Copy()` *(clone.aspx)*

Das Ergebnis dieses Codeblocks ist das Folgende:

```
Clone vs. Copy-Methode:
Original:
Dies ist ein Beispielstring.
Clone:
Dies ist ein Beispielstring.
Kopie:
Dies ist ein Beispielstring.und Mehr...
```

### 9.2.2 Compare

Um zwei Strings miteinander zu vergleichen, steht Ihnen die Methode `Compare` zur Verfügung.

Das Ergebnis hat dabei die nachfolgenden Bedeutungen: Ist der Wert kleiner Null, so ist der erste angegebene String kleiner als der zweite. Ist das Ergebnis Null, so sind die beiden Strings gleich, ist das Ergebnis größer Null, so ist der zweite String größer.

Die Methode existiert in verschiedenen Varianten: Sie können beispielsweise als zusätzliche Parameter zwei optionale boolesche Werte angeben.

Der erste Wert legt fest, ob der Vergleich zwischen Großbuchstaben und Kleinbuchstaben unterscheiden soll. Mit dem zweiten Wert können Sie unterscheiden, ob für den Vergleich außerdem kulturspezifische Informationen (wie die Reihenfolgen von Tag, Monat und Jahr bei einer Datumsangabe) mit herangezogen werden sollen.

Als weitere Variante können Sie mittels `Compare` auch Teilstrings miteinander vergleichen. Hierbei wird nach den zu vergleichenden Strings jeweils ein Index angegeben, der anzeigt, ab welcher Position die beiden Strings zu vergleichen sind. Als weiterer Parameter ist anzugeben, wie viele Zeichen zu vergleichen sind.

Aus diesen Varianten ergeben sich u.a. die nachfolgenden Methodenaufrufe:

```
Ergebnis = String.Compare(String1, String2);
Ergebnis = String.Compare(String1, Index1 String2, Index2, Zeichenzahl);
```

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Beispiel;
  string Beispiel2;
  string Beispiel3;
  int Ergebnis;

  Beispiel = "Dies ist ein Beispielstring.";
  Beispiel2 = "DIES IST EIN BEISPIELSTRING.";
  Beispiel3 = "Dies ist ein anderer String.";
  a.Text = "Compare und CompareOrdinal-";
  a.Text += "Methode: <br /> ";

  Ergebnis = String.Compare(Beispiel, Beispiel2);
  a.Text += "String1: <br />";
  a.Text += Beispiel + "<br />";
  a.Text += "String2: <br />";
  a.Text += Beispiel2 + "<br />";
  a.Text += "Compare: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = String.CompareOrdinal(Beispiel, Beispiel2);
  a.Text += "CompareOrdinal: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = String.Compare(Beispiel, 5, Beispiel3, 5, 6);
  a.Text += "String1: <br />";
  a.Text += Beispiel + "<br />";
  a.Text += "String2: <br />";
  a.Text += Beispiel3 + "<br />";
  a.Text += "Compare 6 Zeichen ab Pos 5: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = String.Compare(Beispiel, 5, Beispiel3, 5, 15);
  a.Text += "Compare 15 Zeichen ab Pos 5: ";
  a.Text += Ergebnis + "<br />";
```

```
}
</script>
<html>
<head><title>Compare</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.3: Strings vergleichen (compare.aspx)*

Das Ergebnis dieses Codeblocks ist:

```
Compare und CompareOrdinal-Methode:
String1:
Dies ist ein Beispielstring.
String2:
DIES IST EIN BEISPIELSTRING.
Compare: -1
CompareOrdinal: 32
String1:
Dies ist ein Beispielstring.
String2:
Dies ist ein anderer String.
Compare 6 Zeichen ab Pos 5: 0
Compare 15 Zeichen ab Pos 5: 1
```



### 9.2.3 CompareOrdinal

Auch für diese Methode gibt es eine ganze Reihe von unterschiedlichen Ausprägungen. Gegenüber der `Compare`-Methode fallen allerdings die Möglichkeiten, Groß- und Kleinschreibung nicht zu beachten und Länderspezifika zu berücksichtigen, weg.

Es bleibt also zum einen die Möglichkeit zwei Strings zu vergleichen. Dabei ist das Ergebnis kleiner Null, wenn der erste String größer ist, das Ergebnis ist größer Null, wenn der zweite String größer ist, und gleich Null, wenn beide Strings gleich sind.

Zum anderen können Sie auch innerhalb eines Strings Teilstrings miteinander vergleichen. Die Syntax für diese Spezialität ist analog zur Syntax der `Compare`-Methode. Sie können sie auch dem nachfolgenden Beispiel entnehmen.

Beispiel: Siehe `Compare`-Methode

### 9.2.4 CompareTo

Mittels dieser Methode wird eine Instanz mit einem angegebenen Objekt verglichen. Sie haben verschiedene Möglichkeiten den Parameter zu übergeben. Der Parameter kann entweder ein beliebiges Objekt sein oder ein Objekt vom Typ `String`. Als Rückgabewert werden die nachfolgenden Ergebnisse zurückgeliefert:

Das Ergebnis ist Null, falls die Instanz und das Objekt gleich sind. Falls die Instanz kleiner als das Objekt ist, wird eine negative Zahl zurückgeliefert, ansonsten erhalten Sie als Resultat eine positive Zahl.

Hier noch einmal kurz ein Aufruf dieser Methode:

```
int Ergebnis;
Ergebnis = StrUntersuchung.CompareTo(Str_Vergleich);
```

### 9.2.5 Concat

Wenn Sie Strings oder die Inhalte von `String`-Objekten oder ein Array von Strings zu einem String zusammenfügen wollen, so liefern Ihnen die Methoden mit der Bezeichnung `Concat` die richtigen Lösungsmittel.

Grundsätzlich können Sie bis zu vier Strings oder Objekte als Parameter angeben, auf die diese Verknüpfung angewendet werden soll. Hiermit können Sie also, wenn Sie beispielsweise vier Stringvariablen haben und diese zu einem String zusammenfügen wollen, eine ganze Reihe von Befehlszeilen einsparen.

```
Ergebnis = String.Concat(String1, String2);
Ergebnis = String.Concat(String1, String2, String3);
```

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Teil1;
  string Teil2;
  string Ergebnis;

  Teil1 = "Dies ist der erste Teil";
  Teil2 = " Und nun der zweite Teil.";
  a.Text = "Concat, ";
  a.Text += "Methode: <br /> ";
```

```
  Ergebnis = String.Concat(Teil1, Teil2);
  a.Text += "Teil1: ";
  a.Text += Teil1 + "<br />";
  a.Text += "Teil2: ";
  a.Text += Teil2 + "<br />";
  a.Text += "Concat: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Teil2.Insert(10, Teil1);
  a.Text += "Insert an Position 10: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Teil1.PadLeft(35, '_');
  a.Text += "PadLeft mit _: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Ergebnis.PadRight(40, '_');
  a.Text += "PadRight mit _: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Ergebnis.Remove(10, 5);
  a.Text += "Remove 5 Zeichen: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Ergebnis.TrimEnd('_');
  a.Text += "Loesche die _ am Ende: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Ergebnis.TrimStart('_');
  a.Text += "Loesche die _ am Anfang: ";
  a.Text += Ergebnis + "<br />";
}
</script>
<html>
<head><title>Diverse Stringmethoden</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.4: Diverse Stringmethoden (concat.aspx)*

Die Ausgabe dieses Codeblocks ist Folgendes:

```
Concat, Methode:
Teil1: Dies ist der erste Teil
Teil2: Und nun der zweite Teil.
Concat: Dies ist der erste Teil Und nun der zweite Teil.
Insert an Position 10: Und nun dDies ist der erste Teiler zweite Teil.
PadLeft mit _: ___________Dies ist der erste Teil
PadRight mit _: ___________Dies ist der erste Teil_____
Remove 5 Zeichen: __________s ist der erste Teil_____
Loesche die _ am Ende: __________s ist der erste Teil
Loesche die _ am Anfang: s ist der erste Teil
```

### 9.2.6 Copy

Mit der `Copy`-Methode erhalten Sie die Möglichkeit eine neue Instanz eines Strings zu erzeugen. Dieser neue String hat denselben Inhalt wie der als Parameter übergebene String.

Beispiel: Siehe `Clone`-Methode.

### 9.2.7 CopyTo

Die `CopyTo`-Methode kopiert eine von Ihnen festgelegte Anzahl von Zeichen (auch die Startposition innerhalb des Strings legen Sie als Parameter fest) in ein Array von Zeichen. Sie können auch den Startindex innerhalb des Arrays festlegen.

Als Parameter geben Sie also zunächst die Startposition innerhalb der verwendeten Instanz an, danach folgen das Ziel-Objekt, der Startindex im Zielarray und als Letztes die Anzahl der zu kopierenden Zeichen an.

Aus diesen Parametrisierungen ergibt sich der folgende Aufruf:

```
Quellstring.CopyTo(Start, Zielstring, Zielstart, Anzahl);
```

### 9.2.8 EndsWith

Mit der Methode `EndsWith` können Sie untersuchen, ob die Instanz mit dem String, den Sie als Parameter übergeben haben, endet.


*Beachten Sie, dass ein Abschneiden von Leerzeichen vor der Verwendung dieser Methode eventuell hilfreich sein könnte.*

Die Rückgabewerte sind entweder `True` für ein positives Ergebnis oder `False` für ein negatives Ergebnis.

```
Ergebnis = Text.EndsWith(Vergleichsstring);
```

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Text1;
  string Text2;
  bool Ergebnis2;

  Text1 = "Dies ist ein Text.";
  Text2 = "Dies ist ein anderer Text.";
  a.Text = "EndsWith, Equal,... ";
  a.Text += "Methoden: <br /> ";

  Ergebnis2 = String.Equals(Text1, Text2);
  a.Text += "Text1: " + Text1;
  a.Text += "<br />Text2: " + Text2;
  a.Text += "<br />Equals: ";
  a.Text += Ergebnis2.ToString() + "<br />";

  Ergebnis2 = Text1.EndsWith("Text.");
  a.Text += "Text1 endet mit Text.: ";
  a.Text += Ergebnis2.ToString() + "<br />";

  Ergebnis2 = Text1.StartsWith("Text.");
  a.Text += "Text1 beginnt mit Text.: ";
  a.Text += Ergebnis2.ToString() + "<br />";
}
</script>
<html>
<head><title>EndsWith</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.5:* `EndsWith()` *und* `StartsWith()` *(endswith.aspx)*

Das Ergebnis dieses Codeblocks sieht folgendermaßen aus:

```
EndsWith, Equal, ... Methoden:
Text1: Dies ist ein Text.
Text2: Dies ist ein anderer Text.
Equals: False
Text1 endet mit Text.: True
Text1 beginnt mit Text.: False
```

### 9.2.9 Equals

Mittels der `String.Equals`-Methoden haben Sie die Möglichkeit eine Instanz mit einem Objekt vom Typ `String` oder mit einem String zu vergleichen. In beiden Fällen brauchen Sie nur einen Parameter anzugeben.

Falls Sie zwei Strings als Parameter angeben, können Sie auch diese mittels der `Equals`-Methode vergleichen. Der Rückgabewert ist entweder `True`, wenn die beiden `String`-Objekte übereinstimmen. Ansonsten erhalten Sie den booleschen Wert `False` zurückgeliefert.

Das Beispiel finden Sie im Abschnitt `EndsWith`.

### 9.2.10 Format

Jedes `String`-Objekt besteht nicht nur aus den einzelnen Textelementen, sondern enthält zusätzlich noch weitere Formatierungsinformationen. Diese Formatierungsinformationen können Sie mit Hilfe der `Format`-Methode auslesen oder manipulieren.

Sie haben mit dieser Methode beispielsweise die Möglichkeit Zahlenformate festzulegen. Hierbei stehen Ihnen eine ganze Reihe von Parametern zur Verfügung.

In der nachfolgenden Tabelle sind die wichtigsten Formatparameter zusammengefasst:

| Formatparameter | Beschreibung |
|---|---|
| C, c | Währungsformat |
| D, d | Dezimalzahl |
| E, e | Exponentialdarstellung |
| F, f | Feste Länge |
| G, g | Allgemein: Je nach Format wird eine andere Genauigkeit ausgewählt. |

| Formatparameter | Beschreibung |
|---|---|
| N, n | Ziffer |
| P, p | Prozentangabe |
| R, r | Mit diesem Parameter kann eine Konvertierung zu einem String wieder in das ursprüngliche Zahlenformat sichergestellt werden. |
| X, x | Hexadezimalzahl |

*Tabelle 9.1: Zahlenformate für Format oder ToString*

Neben diesen Formatierungen können Sie über den Formatparameter auch eher primitiv eine Variable in einen String einfügen.

Hier ein Aufruf der `Format`-Methode:

```
Ergebnis = String.Format(Quelle, Formateinlage);
```

Sie finden auch einige Formatparameter auf `ToString` angewendet und die Ausgabe mit Format im nachfolgenden Beispiel.

```
<%@ Page Language="C#" Debug="true"%>
<script runat="server">
void Page_Load()
{
  double Format;
  string Ergebnis;

  Format = 12345.678;
  a.Text = "Format ";
  a.Text += "Methode: <br /> ";

  Ergebnis = String.Format("Zahl {0} im String.", Format);
  a.Text += Ergebnis + "<br />";

  a.Text += "Formatierungen von ToString: <br />";
  a.Text += "Currency: ";
  Ergebnis = Format.ToString("c");
  a.Text += Ergebnis + "<br />";

  a.Text += "Fixed: ";
  Ergebnis = Format.ToString("f");
  a.Text += Ergebnis + "<br />";
```

```
  a.Text += "Wissenschaftlich: ";
  Ergebnis = Format.ToString("e");
  a.Text += Ergebnis + "<br />";
}
</script>
<html>
<head><title>Format</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.6: Strings formatieren (format.aspx)*

Dieses Listing erzeugt die nachfolgende Ausgabe:

```
Format Methode:
Zahl 12345,678 im String.
Formatierungen von ToString:
Currency: 12.345,68 DM
Fixed: 12345,68
Wissenschaftlich: 1,234568e+004
```

### 9.2.11 GetEnumerator

Mit der `GetEnumerator`-Methode erhalten Sie ein Objekt, das Ihnen die Möglichkeit bietet, einzelne Elemente schrittweise auszuwählen und anzuspringen. In der Klasse `String` wird jedes Zeichen in einem `String`-Objekt anwählbar.

### 9.2.12 GetHashCode

Mit der `GetHashCode`-Methode erhalten Sie einen Integerwert, der dem `HashCode` der verwendeten Instanz entspricht. Hiermit können Sie Zugriffe innerhalb des `String`-Objekts deutlich beschleunigen.

### 9.2.13 GetType

Mittels der `GetType`-Methode können Sie den Typ der angegebenen Instanz ermitteln.


*Diese Methode gibt es übrigens für alle Datentypen – dadurch erhalten Sie die Möglichkeit den Datentyp zu ermitteln, falls Sie ihn nicht kennen.*

Der Datentyp, in dem das Ergebnis zurückgeliefert wird, ist vom Typ `System.Type`.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Text;
  System.Type Ergebnis;
  System.TypeCode Ergebnis2;

  Text = "Dies ist ein Text.";
  a.Text = "Gettype und GetTypeCode ";
  a.Text += "Methoden: <br /> ";

  Ergebnis = Text.GetType();
  a.Text += "Text: ";
  a.Text += Text + "<br />";
  a.Text += "GetType liefert: ";
  a.Text += Ergebnis.ToString() + "<br />";

  Ergebnis2 = Text.GetTypeCode();
  a.Text += "GetTypeCode liefert: ";
  a.Text += Ergebnis2.ToString() + "<br />";
}
</script>
<html>
<head><title>GetType</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.7:* `GetType()` *(gettype.aspx)*

Das Ergebnis dieses Codeblocks ist Folgendes:

```
GetType und GetTypeCode Methoden:
Text: Dies ist ein Text.
GetType liefert: System.String
GetTypeCode liefert: String
```

### 9.2.14 GetTypeCode

Mit der Methode `GetTypeCode` erhalten Sie den zu ermittelnden Datentyp über einen Code. Diese Methode gibt es ebenfalls für alle wichtigen Datentypen.

Der Datentyp, in dem das Ergebnis zurückgeliefert wird, ist vom Typ `System.TypeCode`.

Das Beispiel hierzu finden Sie bei der Beschreibung der Methode `GetType`.

### 9.2.15 IndexOf

Mittels dieser Methode erhalten Sie den Index des ersten Auftretens der als Parameter angegebenen Zeichenkette in der zu untersuchenden Instanz.

Sie können als weitere Parameter zwei Integerwerte angeben, die die Startposition und die Anzahl der zu untersuchenden Zeichen spezifizieren.


*Ähnliche Methoden finden Sie beispielsweise auch in der Klasse* `Array` *oder in anderen Klassen für die Strukturierung von Daten.*

Nachfolgend sehen Sie einige beispielhafte Aufrufe dieser Methode:

```
Ergebnis = String1.IndexOf(Suchstring);
Ergebnis = String1.IndexOf(Suchstring, Startposition);
Ergebnis = String1.IndexOf(Suchstring, Startposition, Endposition);
```

### 9.2.16 IndexOfAny

Diese Methode ermöglicht es Ihnen in einem String eine Anzahl von Zeichen zu suchen, die in einem `Char`-Array, das als Parameter übergeben wird, aufgesplittet werden. Auch hier können Sie die Startposition und die Anzahl der zu untersuchenden Zeichen zusätzlich als Parameter mitgeben.

Das erste Auftreten wird als Rückgabewert zurückgeliefert.

### 9.2.17 Insert

Mit der Methode `Insert` können Sie in eine String-Instanz einen als Parameter übergebenen String einfügen.

Als zweiten Parameter müssen Sie die Position, an der der String einzufügen ist, angeben.

Falls dieser Wert größer ist als die Länge der Instanz, gibt es einen Fehler, den Sie entsprechend abfangen können.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

### 9.2.18 Intern

Mittels der Methode `Intern` können Sie die Systemreferenz zu einem String erhalten. Dies bedeutet, dass Sie daraus ermitteln können, ob es beispielsweise bereits eine andere Variable mit diesem Inhalt gibt.


*Üblicherweise werden intern Stringinhalte nur einmal abgelegt. Wollen Sie dies verhindern, so müssen Sie einem String explizit eine neue Instanz zuweisen, um diese dann mit dem gleichen Inhalt belegen zu können.*



### 9.2.19 IsInterned

Mittels dieser Methode können Sie die interne Referenz eines untersuchten Strings ermitteln. Die interne Referenz stellt eine Referenzierung dar, die in der Laufzeitbibliothek des ASP.NET-Programms von jedem String angelegt wird.

### 9.2.20 Join

Die `Join`-Methode ermöglicht Ihnen, ein Array von Strings mittels eines Verknüpfungszeichens (das auch Nichts sein kann) zu einem String zusammenzufassen.

Sie können sich hierbei auch mittels zweier zusätzlicher `Integer`-Parameter auf einen Teil des Arrays beschränken. Der erste Parameter gibt an, ab welchem Indexelement die Verbindung durchgeführt werden soll, und der zweite Parameter, wie viele Elemente er umfassen soll.

Nachfolgend sehen Sie einige beispielhafte Aufrufe der `Join`-Methode:

```
Ergebnis = String.Join(Verknüpfungsstring, Stringarray);

Ergebnis = String.Join(Verknüpfungsstring, Stringarray, Startelement,
  Elementezahl);
```

### 9.2.21 LastIndexOf

Mittels dieser Methoden können Sie das letzte Auftreten eines Zeichens oder einer Zeichenkette innerhalb der Instanz eines `String`-Objektes ermitteln. Sie haben die Möglichkeit über einen zweiten Parameter den Startpunkt der Suche innerhalb des Strings festzulegen.

Außerdem können Sie über einen weiteren `Integer`-Parameter noch festlegen, wie viele Zeichen Sie ab dem Startpunkt durchsuchen wollen.

### 9.2.22 LastIndexOfAny

Mit der `LastIndexOfAny`-Methode können Sie analog zur `IndexOfAny`-Methode in einem String eine Anzahl von Zeichen suchen, die in einem `Char`-Array als Parameter übergeben werden.

Auch hier können Sie die Startposition und die Anzahl der zu untersuchenden Zeichen zusätzlich als Parameter übergeben.

Das letzte Auftreten wird als Rückgabewert zurückgeliefert.

### 9.2.23 PadLeft

Die `PadLeft`-Methode hat zwei Parameter, die ihre Wirkung bestimmen. Der erste Parameter ist ein Integerwert, der festlegt, wie viele Zeichen der Ergebnisstring besitzen soll.

Als zweiten Parameter können Sie angeben, mit welchen Zeichen oder welcher Zeichenkette der String auf der linken Seite (also am Anfang) aufgefüllt werden soll.

In gewisser Weise stellt diese Methode also eine Umkehrung der `TrimStart`-Methode dar.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

### 9.2.24 PadRight

Die `PadRight`-Methode hat ebenfalls zwei Parameter, die ihre Wirkung bestimmen. Der erste Parameter ist wieder ein Integerwert, der festlegt, wie viele Zeichen der Ergebnisstring besitzen soll.

Als zweiten Parameter können Sie auch hier angeben, mit welchen Zeichen oder welcher Zeichenkette der String auf der rechten Seite (also am Ende) aufgefüllt werden soll.

In gewisser Weise ist diese Methode also eine Umkehrung der `TrimEnd`-Methode.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

### 9.2.25 Remove

Mittels der Methode `Remove` können Sie ab einer definierten Position die angegebene Anzahl Zeichen aus einer `String`-Instanz entfernen.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

### 9.2.26 Replace

Mittels der `Replace`-Methode können Sie innerhalb eines Strings eine Zeichenkette oder ein Zeichen durch eine andere Zeichenkette (oder Zeichen) ersetzen.

### 9.2.27 Split

Die `Split`-Methode stellt eine Art Umkehrung der `Join`-Methode dar. Mit ihrer Hilfe kann ein Text in einem String über ein Trennzeichen in ein Array von Strings aufgeteilt werden.

Das Trennzeichen wird hierbei nicht in das Array übernommen.

Anbei finden Sie einen beispielhaften Aufruf der `Split`-Methode:

```
string[] Aufteilung = String.Split(Quellstring, Parameter);
```

### 9.2.28 StartsWith

Analog zur `EndsWith`-Methode haben Sie hier die Möglichkeit zu überprüfen, ob eine `String`-Instanz mit einer bestimmten Zeichenkette endet.

Das Beispiel finden Sie im Abschnitt `EndsWith`.

### 9.2.29 Substring

Mittels dieser Methode können Sie aus einer `String`-Instanz einen Teilstring extrahieren.

Als Parameter müssen Sie die Startposition dieses Teilstrings angeben. Als zweiten Parameter können Sie optional die Anzahl der Zeichen ab der Startposition zusätzlich festlegen, die den Teilstring spezifizieren. Daraus ergibt sich der nachfolgende Methodenaufruf:

```
Pruefstring.Substring(Startposition, AnzahlZeichen);
```

### 9.2.30 ToCharArray

Mit der `ToCharArray`-Methode führen Sie eine Umwandlung einer String-Instanz in ein Array von einzelnen Zeichen durch. Dieses Array wird zurückgeliefert.

Wenn Sie zwei Zahlen als Parameter angeben, können Sie diese Umwandlung auf einen Teilstring anwenden, der durch die Startposition und die Anzahl der Zeichen spezifiziert wird. Ein Methodenaufruf mit allen möglichen Parametern sieht also wie folgt aus:

```
Ergebnis = Pruefstring.ToCharArray(Startposition, AnzahlZeichen);
```

oder:

```
Ergebnis = Pruefstring.ToCharArray();
```

### 9.2.31 ToLower

Die Methode `ToLower` erzeugt eine Kopie der Originalinstanz, in der alle Großbuchstaben in Kleinbuchstaben umgewandelt sind.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Text;
  string Ergebnis;

  Text = "Dies ist ein Text.";
  a.Text = "ToLower und ToUpper ";
  a.Text += "Methoden: <br /> ";
```

```
  Ergebnis = Text.ToLower();
  a.Text += "Text: " + Text + "<br />";
  a.Text += "ToLower liefert: ";
  a.Text += Ergebnis + "<br />";

  Ergebnis = Text.ToUpper();
  a.Text += "ToUpper liefert: ";
  a.Text += Ergebnis + "<br />";
}
</script>
<html>
<head><title>ToUpper, ToLower</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 9.8: Groß- und Kleinbuchstaben (tolower.aspx)*

Das Ergebnis dieses Codeblocks ist Folgendes:

```
ToLower und ToUpper Methoden:
Text: Dies ist ein Text.
ToLower liefert: dies ist ein text.
ToUpper liefert: DIES IST EIN TEXT.
```

### 9.2.32 ToString

Diese Methode ist eher aus Vollständigkeitsgründen implementiert. Eine Umwandlung von einem String in einen String hat keinen echten Effekt.


*Da es aber diese Methode in einer ganzen Reihe von Klassen gibt, können Sie diese auf unbekannte Objekte anwenden und eine Stringumwandlung erzwingen. Ist die Instanz schon ein String, wird kein Schaden angerichtet.*

Mittels der bei der `Format`-Methode vorgestellten Formatierungsmöglichkeiten können Sie einen Nicht-String für eine Ausgabe noch nach Ihren Wünschen formatieren.

### 9.2.33 ToUpper

Die Methode `ToUpper` erzeugt eine Kopie der Originalinstanz, in der alle Kleinbuchstaben in Großbuchstaben umgewandelt sind.

Das Beispiel finden Sie bei der `ToLower`-Methode.

### 9.2.34 Trim

Sie haben auch für diese Methode zwei unterschiedliche Möglichkeiten der Anwendung: Zum einen können Sie sie dafür verwenden (wenn Sie keinen weiteren Parameter angeben), am Anfang und am Ende des Strings eventuell stehende Leerzeichen zu eliminieren.

Geben Sie ein Zeichen als Parameter an, wird dieses Zeichen am Anfang und am Ende des Strings entfernt.

### 9.2.35 TrimEnd

Die `TrimEnd`-Methode funktioniert analog zur `Trim`-Methode, nur werden die Modifikationen nur am Ende eines Strings vorgenommen.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

### 9.2.36 TrimStart

Die `TrimStart`-Methode funktioniert analog zur `Trim`-Methode, aber die Modifikationen werden nur am Anfang eines Strings vorgenommen.

Das Beispiel hierzu finden Sie bei der Methode `Concat`.

## 9.3 Die Operatoren

Neben dem Operator +, der eine Zusammenführung von Strings bewirkt, gibt es noch weitere Operatoren, die wir Ihnen auch nicht vorenthalten möchten.

Diese beiden Operatoren stellen Vergleichsmöglichkeiten zwischen zwei Strings zur Verfügung.

### 9.3.1 Equality

Mit dem Operator `Equality` können Sie herausfinden, ob zwei `String`-Objekte denselben Wert besitzen oder nicht. Als Ergebnis wird entweder `True` geliefert oder `False`.

Hier ein kleiner beispielhafter Aufruf:

```
==(String1, String2)
```

### 9.3.2 Inequality

Um herauszufinden, ob zwei Strings ungleich sind, können Sie neben dem Operator `Equality` auch den Operator `Inequality` einsetzen. Das Ergebnis ist invertiert zum Operator `Equality`.

Die Rückgabewerte lauten also `True` bei Ungleichheit der `String`-Objekte und `False` bei Gleichheit der `String`-Objekte.

Hier ein kleiner beispielhafter Aufruf:

```
!=(String1, String2)
```

# 10 Die Klasse DateTime

In der Klasse `DateTime` werden Ihnen Möglichkeiten zur Verfügung gestellt, Teile eines Datums oder einer Uhrzeit abzufragen, aber auch Möglichkeiten, die Instanzen vom Datentyp `DateTime` so zu manipulieren, dass sie die unterschiedlichsten Darstellungsarten erzeugen.

Sie haben Methoden um die aktuelle Sekunde zu ermitteln (gemessen an der Systemzeit) oder auch das aktuelle Datum im Deutschland-typischen Format anzuzeigen und durch die Verwendung einer anders parametrisierten Methode auf das amerikanische Datumsformat zu wechseln.

Sie können für die Kalenderinformationen auf die unterschiedlichsten Kalenderarten zurückgreifen. Standardmäßig sind in der `System.Globalisation`-Klasse die nachfolgenden Kalendertypen hinterlegt, die verschiedene Besonderheiten und Ausprägungen besitzen:

- der Kalender `GregorianCalendar`,
- der Kalender `HebrewCalendar`,
- der Kalender `HijriCalendar`,
- der Kalender `JapaneseCalendar`,
- der Kalender `JulianCalendar`,
- der Kalender `KoreanCalendar`,
- der Kalender `TaiwanCalendar`,
- der Kalender `ThaiBuddhistCalendar`.

Die Beispiele für die unterschiedlichen `DateTime`-Datenypen und Methoden basieren auf dem gregorianischen Kalender.

## 10.1 Die DateTime-Datenfelder

Wir werden Ihnen im Folgenden die unterschiedlichen Datenfelder, die Sie abfragen können, und auch die verschiedenen Methoden zur Datumsmanipulation vorstellen.

Alle Datenfelder, die wir hier vorstellen, werden auch vom .NET Compact Framework bereitgestellt.

### 10.1.1 MaxValue

Mittels des Datenfeldes `MaxValue` können Sie den größten Wert ermitteln, der für den von Ihnen definierten Datentyp zulässig ist.

### 10.1.2 MinValue

Mittels des Datenfeldes `MinValue` können Sie den kleinsten Wert ermitteln, der für den von Ihnen definierten Datentyp zulässig ist.

## 10.2 Die Eigenschaften

Mittels der nachfolgenden Eigenschaften können Sie einzelne Teilelemente abfragen oder setzen, die einen Datumstypen repräsentieren.

Auch diese Eigenschaften werden innerhalb des .NET Compact Framework bereitgestellt.



### 10.2.1 Date

Mittels der `Date`-Eigenschaft ermitteln Sie das Datum Ihrer `DateTime` Instanz. Diese können Sie dann beispielsweise in einen String umwandeln und ausgeben. Genau dieses demonstrieren wir auch im nachfolgenden Beispiel.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  DateTime Datum;

  Datum = DateTime.Now;
  a.Text = "DateTime Eigenschaften: ";
  a.Text += "<br />Now: ";
  a.Text += Datum.ToString();
  a.Text += "<br />Date: ";
  a.Text += Datum.Date.ToString();
  a.Text += "<br />Day: ";
```

```
  a.Text += Datum.Day.ToString();
  a.Text += "<br />DayOfWeek: ";
  a.Text += Datum.DayOfWeek.ToString();
  a.Text += "<br />DayOfYear: ";
  a.Text += Datum.DayOfYear.ToString();
  a.Text += "<br />Hour: ";
  a.Text += Datum.Hour.ToString();
  a.Text += "<br />Millisecond: ";
  a.Text += Datum.Millisecond.ToString();
  a.Text += "<br />Minute: ";
  a.Text += Datum.Minute.ToString();
  a.Text += "<br />Month: ";
  a.Text += Datum.Month.ToString();
  a.Text += "<br />Second: ";
  a.Text += Datum.Second.ToString();
  a.Text += "<br />TimeOfDay: ";
  a.Text += Datum.TimeOfDay.ToString();
  a.Text += "<br />Ticks: ";
  a.Text += Datum.Ticks.ToString();
  a.Text += "<br />Today: ";
  a.Text += DateTime.Today.ToString();
  a.Text += "<br />UTCNow: ";
  a.Text += DateTime.UtcNow.ToString();
  a.Text += "<br />Year: ";
  a.Text += Datum.Year.ToString();
  a.Text += "<br />";
}
</script>
<html>
<head><title>DateTime Eigenschaften</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 10.1: Eigenschaften von* `DateTime` *(dateeigenschaft.aspx)*

Die Ausgabe dieses Codeblocks sehen Sie in Bild 10.1.

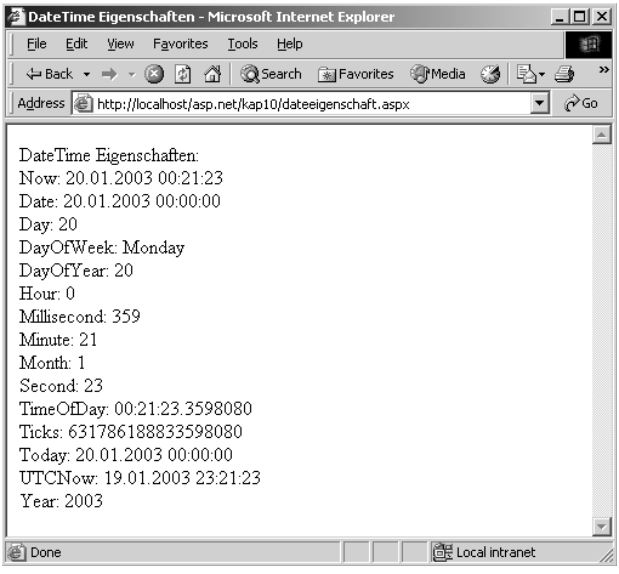



*Bild 10.1: Ausgabe von DateTime*



### 10.2.2 Day

Die `Day`-Eigenschaft ermittelt den aktuellen Tag innerhalb des aktuellen Monats (beim 20. Mai würde also 20 geliefert).

Auch diese wird im Beispiel zur Eigenschaft `Date` in einen String umgewandelt und dann ausgegeben.

### 10.2.3 DayOfWeek

Diese Eigenschaft liefert den Wochentag, der in der untersuchten Instanz hinterlegt ist.

Betrachten Sie auch für den Einsatz dieser Instanz das Beispiel im Abschnitt zur Eigenschaft `Date`.

### 10.2.4 DayOfYear

Mit der Eigenschaft `DayOfYear` erhalten Sie den Tag des Jahres, der sich aus dem Datum der untersuchten Instanz ermitteln lässt. Auch hierzu verweisen wir Sie auf das große Beispiel bei der Eigenschaft `Date`.

### 10.2.5 Hour

Die Stunde einer zu untersuchenden `DateTime`-Instanz erhalten Sie mittels dieser Eigenschaft.

Das Beispiel hierzu ist auch im Abschnitt `Date` zu finden.

### 10.2.6 Millisecond

Sie können die Analyse einer Datumsinstanz bis auf die Millisekunde herunter durchführen. Mittels der Eigenschaft `Millisecond` erhalten Sie genau diese.

Als Beispiel sehen Sie eine Ausgabe der Millisekunde im Abschnitt `Date`.

### 10.2.7 Minute

Auch die Minute einer `DateTime`-Instanz können Sie abfragen und gesondert weiterverwenden. Dies geschieht über die Eigenschaft `Minute`.

Das Beispiel hierzu finden Sie ebenfalls im Abschnitt `Date`.

### 10.2.8 Month

Die Komponente, die in einer `DateTime`-Instanz den Monat repräsentiert, kann über die `Month`-Eigenschaft abgefragt werden. Das zugehörige Beispiel finden Sie bei der `Date`-Eigenschaft.

### 10.2.9 Now

Mittels der `Now`-Eigenschaft können Sie eine Instanz vom Typ `DateTime` mit der aktuellen Systemzeit befüllen. Auch diese Eigenschaft erhalten Sie im Beispiel zu `Date` demonstriert.

### 10.2.10 Second

Mit der `Second`-Komponente ermitteln Sie die Sekunde innerhalb einer `DateTime`-Datenstruktur und können diese weiterverarbeiten. Das zugehörige Beispiel finden Sie bei der `Date`-Eigenschaft.

### 10.2.11 Ticks

Die `Ticks`-Eigenschaft berechnet die Anzahl der Systemtakte, die durch den Inhalt der `DateTime`-Struktur repräsentiert ist. Auch hierzu erhalten Sie ein Beispiel im Abschnitt `Date`.

### 10.2.12 TimeOfDay

Mittels der `TimeOfDay`-Eigenschaft können Sie die aktuelle Systemzeit ermitteln oder setzen. Auch hierzu möchten wir Sie auf das Beispiel im Abschnitt `Date` verweisen.

### 10.2.13 Today

Die Eigenschaft `Today` liest das aktuelle Systemdatum aus oder ermöglicht das Setzen eben dieses Datums. Das Beispiel hierzu befindet sich im `Date`-Abschnitt. Eine eventuelle Zeitangabe wird hier im Gegensatz zu `Now` nicht berücksichtigt.

### 10.2.14 UtcNow

Mit Hilfe der `UtcNow`-Eigenschaft erhalten Sie die aktuelle Uhrzeit als normalisierte Weltzeit, also UTC-Zeit (Universal Time Coordinated, früher Greenwich Mean Time).

### 10.2.15 Year

Mit der Eigenschaft `Year` ermitteln Sie das in einer `DateTime`-Datenstruktur hinterlegte Jahr. Ein Beispiel hierzu finden Sie im `Date`-Abschnitt.



## 10.3 Die Methoden

Nachdem wir die Eigenschaften der `DateTime`-Klasse kennen gelernt haben, kommen wir nun zu den Methoden dieser Klasse.

### 10.3.1 Add

Mittels der Methode `Add` können Sie einen als Parameter definierten Zeitraum einer Instanz vom Typ `DateTime` hinzufügen. Hiermit haben Sie die Möglichkeit neue Daten durch Hinzufügen (oder Entfernen) von Zeiträumen zu berechnen.

Der Datentyp für diese Operation muss vom Typ `System.TimeSpan` sein. Er besteht aus 4 Parametern, die für Tag, Stunde, Minute und Sekunde stehen.

Details hierzu können Sie dem angefügten Beispiel entnehmen.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  DateTime Datum;
  DateTime Ergebnis;
  TimeSpan Dauer;

  Dauer = new TimeSpan(10,0,0,0);
  Datum = DateTime.Now;
  a.Text = "Add und andere Methoden: ";
  a.Text += "<br /> ";
  a.Text += Datum.ToString();
  a.Text += "<br /> ";

  Ergebnis = Datum.Add(Dauer);
  a.Text += "<br />Add: 10 Tage: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddDays(6);
  a.Text += "<br />AddDays: 6 Tage: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddHours(5);
  a.Text += "<br />AddHours: 5 Stunden: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddMilliseconds(5000);
  a.Text += "<br />AddMilliseconds: 5000: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddMinutes(55);
  a.Text += "<br />AddMinutes: 55 Minuten: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddMonths(2);
  a.Text += "<br />AddMonths: 2 Monate: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddSeconds(25);
  a.Text += "<br />AddSeconds: 25 Sekunden: ";
  a.Text += Ergebnis.ToString();
```

```
  Ergebnis = Datum.AddTicks(25000000);
  a.Text += "<br />AddTicks: 25000000: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.AddYears(25);
  a.Text += "<br />AddYears: 25 Jahre: ";
  a.Text += Ergebnis.ToString();
  a.Text += "<br /> ";
}
</script>
<html>
<head><title>DateTime Eigenschaften</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 10.2: Methoden von* `DateTime` *(add.aspx)*

Dieses Listing erzeugt die nachfolgende Ausgabe:

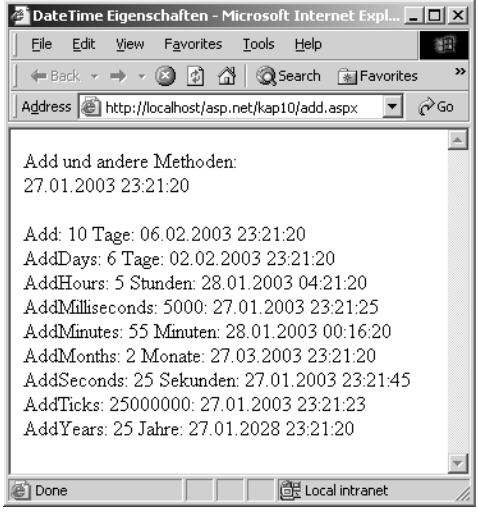



*Bild 10.2: Add und andere Methoden*

### 10.3.2 AddDays

Die AddDays-Methode ist eine Spezialform der Add-Methode, mit der Sie die Tage in einem Datum durch Hinzufügen oder Abziehen modifizieren können. Ein zugehöriges Beispiel finden Sie bei Add.

### 10.3.3 AddHours

Analog zu AddDays können Sie in diesem Fall eine definierte Anzahl von Stunden zu einem Datum hinzufügen oder abziehen. Ein Beispiel für die Benutzung sehen Sie bei der Add-Methode.

### 10.3.4 AddMilliseconds

Die Modifikation von DateTime-Instanzen durch Hinzufügen oder Entfernen von Zeiteinheiten kann auch auf der Ebene von Millisekunden durchgeführt werden. Das Programmbeispiel zu Add handelt auch diesen Punkt ab.

### 10.3.5 AddMinutes

Auch Minuten lassen sich zu DateTime-Instanzen hinzuaddieren oder subtrahieren. Dies erfolgt mittels der Methode AddMinutes. Das Beispiel hierzu ist auch bereits in der Add-Methode zu finden.

### 10.3.6 AddMonths

Als weitere Skalierungsmöglichkeit steht Ihnen die Möglichkeit zur Verfügung Daten auf Monatsebene mittels der AddMonths-Methode zu modifizieren. Beachten Sie auch hier das Beispiel in der Add-Methoden-Erläuterung.

### 10.3.7 AddSeconds

Die Modifikation Ihrer DateTime-Instanzen durch Hinzufügen oder Entfernen von Zeiteinheiten können Sie auch auf der Ebene von Sekunden durchführen. Das Programmbeispiel zur Add-Methode handelt auch diesen Punkt ab.

### 10.3.8 AddTicks

Die kleinste Einheit, um Daten durch Hinzufügen oder Entfernen von Zeiteinheiten zu verändern, bietet die Methode AddTicks an. Ein Beispiel finden Sie bei der Add-Methode.

### 10.3.9 AddYears

Auch ganze Jahre können Sie einer `DateTime`-Instanz zufügen oder entfernen. Dies erfolgt mit Hilfe der `AddYears`-Methode. Auch hierzu finden Sie ein Beispiel unter der `Add`-Methode.

### 10.3.10 Compare

Mit der `Compare`-Methode haben Sie die Möglichkeit zwei `DateTime`-Instanzen zu vergleichen. Der Rückgabewert ist kleiner Null, wenn das erste als Parameter angegebene Datum früher ist als das zweite Datum. Wenn beide Daten gleich sind, dann ist der Rückgabewert Null, ansonsten ist der Rückgabewert größer Null.

### 10.3.11 CompareTo

Mit der `CompareTo`-Methode kann eine Instanz vom Typ `DateTime` mit einem `DateTime`-Datum als Parameter verglichen werden. Falls die Instanz kleiner als der Parameter ist, wird ein Wert kleiner Null zurückgeliefert, bei gleichen Daten erhalten Sie Null, ansonsten wird ein Wert größer Null zurückgegeben.



### 10.3.12 DaysInMonth

Die Methode benötigt zwei Parameter, das Jahr und den Monat, die auszuwerten sind. Diese werden als Integerwerte angegeben.

Der Rückgabewert sind die Anzahl der Tage, die dieser Monat hatte – Sie haben damit die Möglichkeit z.B. Schaltjahre zu ermitteln.

Der Codeabschnitt

```
Erg = DaysInMonth(1996,2);
```

liefert den Wert 29 zurück. Hiermit ist dann klar, dass 1996 ein Schaltjahr ist.

### 10.3.13 Equals

Mit dieser Methode können Sie einen Vergleich zwischen einem als Parameter angegebenen Objekt und einer `DateTime`-Instanz anstellen. Das Ergebnis ist ein boolescher Ausdruck, also entweder `True` oder `False`.

Alternativ können Sie auch zwei Objekte als Parameter angeben, die dann miteinander verglichen werden. Auch hierbei ist der Rückgabewert entweder `True` oder `False`.

### 10.3.14 FromFileTime

Mittels der `FromFileTime`-Methode können Sie den Zeitstempel eines Windows Files (also einen 64Bit Integerwert) in ein `DateTime`-Format umwandeln.

### 10.3.15 FromFileTimeUtc

Auch mit der Methode `FromFileTimeUtc` haben Sie die Möglichkeit einen Windows File Zeitstempel in ein `DateTime`-Objekt umzuwandeln. Der Datentyp wird dabei auf UTC (also Unified Time Coordinated) normiert. Das Beispiel hierzu sehen Sie bei der `FromFileTime`-Methode.

### 10.3.16 FromOADate

Durch Anwendung der `FromOADate`-Methode führen Sie eine Formatumwandlung von einem OLE Automation-Datum in ein `DateTime`-Datumsformat durch.

### 10.3.17 GetDateTimeFormats

Mit Hilfe der Methode `GetDateTimeFormats` können Sie Daten aus dem Datentyp `DateTime` in einen String formatieren. Die Ausgabeform wird über einen möglichen Culture-Parameter festgelegt. Dieser Parameter stellt eine Möglichkeit dar, länderspezifische Ausgaben zu deklarieren und zu verwenden.

### 10.3.18 GetHashCode

Mittels dieser Methode erhalten Sie den HashCode dieser `DateTime`-Instanz.

### 10.3.19 GetType

Durch den Gebrauch dieser Methode können Sie den Datentyp der zu untersuchenden Instanz ermitteln.

Diese Methode gibt es übrigens für alle wichtigen Datentypen – dadurch erhalten Sie die Möglichkeit den Datentyp zu ermitteln, falls Sie ihn nicht kennen.

### 10.3.20 GetTypeCode

Mit der Methode `GetTypeCode` erhalten Sie den zu ermittelnden Datentyp über einen Code. Diese Methode gibt es ebenfalls für alle wichtigen Datentypen.

### 10.3.21 IsLeapYear

Diese Methode liefert Ihnen bei der Angabe einer Jahreszahl einen booleschen Wert zurück, der aussagt, ob das angegebene Jahr ein Schaltjahr ist oder nicht.

### 10.3.22 Parse

Mittels dieser Methode können Sie einen String, der ein Datum und eine Uhrzeit angibt, in eine Datenstruktur vom Typ `DateTime` umwandeln. Hierbei kann der angegebene String auch ein Datumsformat darstellen, das nicht den auf dem Server voreingestellten länderspezifischen Einstellungen entspricht.



### 10.3.23 ParseExact

Auch diese Methode ermöglicht Ihnen eine Umwandlung eines Strings in den Datentyp `DateTime`.

Hierbei muss allerdings das Datenformat exakt den Vorgaben entsprechen. Insbesondere müssen auch die länderspezifischen Einstellungen berücksichtigt werden.

### 10.3.24 Subtract

Mittels dieser Methode können Sie Zeiten oder Zeitdauern von einer von Ihnen angegebenen `DateTime`-Instanz abziehen.

### 10.3.25 ToFileTime

Mittels dieser Methode können Sie das `DateTime`-Datenformat in das Zeitformat des Filesystems umwandeln. Dies entspricht einem Integer-

wert, der seit dem 1.1.1601 0:00 Uhr immer im 200-Nanosekunden-Rhythmus um eins erhöht wird.

Beispiel:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  DateTime Datum;
  DateTime Ergebnis;
  long Ergebnis2;

  Datum = DateTime.Now;
  a.Text = "Datumsumwandlungs-Methoden: ";
  a.Text += "<br /> ";
  a.Text += Datum.ToString();

  Ergebnis2 = Datum.ToFileTime();
  a.Text += "<br />ToFileTime: ";
  a.Text += Ergebnis2.ToString();

  Ergebnis2 = Datum.ToFileTimeUtc();
  a.Text += "<br />ToFileTimeUTC: ";
  a.Text += Ergebnis2.ToString();

  Ergebnis = Datum.ToLocalTime();
  a.Text += "<br />ToLocalTime: ";
  a.Text += Ergebnis.ToString();

  Ergebnis = Datum.ToUniversalTime();
  a.Text += "<br />ToUniversalTime: ";
  a.Text += Ergebnis.ToString();

  a.Text += "<br />ToLongDateString: ";
  a.Text += Datum.ToLongDateString();

  a.Text += "<br />ToLongTimeString: ";
  a.Text += Datum.ToLongTimeString();

  a.Text += "<br />ToShortDateString: ";
  a.Text += Datum.ToShortDateString();

  a.Text += "<br />ToShortTimeString: ";
  a.Text += Datum.ToShortTimeString();
```

```
}
</script>
<html>
<head><title>DateTime Methoden</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 10.3: Weitere Methoden von* `DateTime` *(tofiletime.aspx)*

Die Bildschirmausgabe dieses Programms sieht folgendermaßen aus:

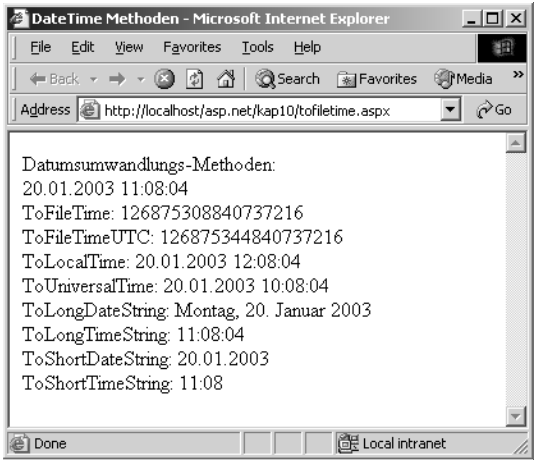



*Bild 10.3: Ausgabe von Datumsumwandlungs-Methoden*

### 10.3.26 ToFileTimeUtc

Auch mit der `ToFileTimeUtc`-Methode führen Sie eine Umwandlung in das Dateisystem-Zeitformat durch. Dieses Mal wird allerdings die Zeit auf die GMT-Zeit normiert.

### 10.3.27 ToLocalTime

Mittels dieser Methode wird die Instanz, die Sie angeben, aus der UTC-Zeit in die lokale Zeit der Zeitzone, die Sie für Ihren Server angegeben haben, umgewandelt. Die Methode überprüft dabei nicht, ob die Instanz tatsächlich ein Datum der UTC-Zeit enthalten hat.

### 10.3.28 ToLongDateString

Wenn Sie eine `DateTime`-Instanz in einen String umwandeln wollen, stehen Ihnen eine ganze Reihe von Methoden zur Verfügung, je nachdem welches Darstellungsformat Sie erreichen wollen. Eine grundsätzliche Vorauswahl wird über die Verwendung der Methode sichergestellt.

Mit der Methode `ToLongDateString` konvertieren Sie den Datumsanteil in einen String, der den Konventionen eines langen Datums folgt. Sie können denselben Effekt auch über die Methode `ToString` erreichen.

### 10.3.29 ToLongTimeString

Diese Methode konvertiert einen `DateTime`-Datentyp in einen String, der die Zeitangabe als langes Datenfeld enthält. Sie können diese Konvertierung auch über die Methode `ToString` erhalten, wenn Sie den entsprechenden Parameter setzen. Diese Parameter sind im dortigen Abschnitt beschrieben.

### 10.3.30 ToOADate

Mittels dieser Methode können Sie ein OLE-Datum aus einer `DateTime`-Instanz erzeugen.

### 10.3.31 ToShortDateString

Mit der Methode `ToShortDateString` konvertieren Sie den Datumsanteil in einen String, der den Konventionen eines kurzen Datums folgt. Sie können denselben Effekt auch über die Methode `ToString` erreichen. Für die Bedeutung eventueller Parameter schauen Sie bitte dort nach.

### 10.3.32 ToShortTimeString

Diese Methode konvertiert einen `DateTime`-Datentyp in einen String, der die Zeitangabe als kurzes Datenfeld enthält. Sie können diese Umwandlung auch mit der Methode `ToString` erhalten, wenn Sie den entsprechenden Parameter setzen. Die Parameter sind im dortigen Abschnitt beschrieben.

### 10.3.33 ToString

Diese Methode wird Ihnen auch in einer ganzen Reihe von anderen Klassen begegnen, da sie universell eingesetzt wird um aus einem beliebigen Datenformat einen String zu erhalten.

Im Falle einer Konvertierung aus dem `DateTime`-Format können Sie als Parameter auch noch angeben, welcher Konvertierungsregel gefolgt wird. Die vorhergehenden Methoden hatten diese Konvertierungsregeln bereits fest vorgegeben.

Die nachfolgende Tabelle gibt Ihnen einen Überblick über die Konvertierungsregeln, und das nachfolgende Programm zeigt Ihnen ein Beispiel – mit der zugehörigen Ausgabe.

Beachten Sie, dass in diesem Fall Unterschiede zwischen Groß- und Kleinschreibung bestehen.

| Format | Beschreibung |
|---|---|
| d | Ausgabe im Standardformat ShortDate, das je nach lokalen Einstellungen durchaus variieren kann. |
| D | Ausgabe im Standardformat LongDate, das je nach lokalen Einstellungen durchaus variieren kann. |
| f | Ausgabe von Datum und Uhrzeit, dabei wird LongDate und ShortTime verwendet. |
| F | Ausgabe von Datum und Uhrzeit, dabei wird LongDate und LongTime verwendet. |
| g | Ausgabe von Datum und Uhrzeit, dabei wird ShortDate und ShortTime verwendet. |
| G | Ausgabe von Datum und Uhrzeit, dabei wird ShortDate und LongTime verwendet. |
| m, M | Ausgabe des Monats und des Tages nach den lokalen Länderkonventionen |
| r, R | Mit dieser Formatierungsregel wird das länderspezifische Datumsformat ausgegeben, das dem sog. IETF (Internet Engineering Task Force) RFC1123 Standard genügt. |
| s | Das hier gewählte Ausgabeformat ist sortierbar, das heißt, die längste Zeiteinheit ist an erster Stelle, dann folgen die weiteren Zeiteinheiten in absteigender Reihenfolge, wobei die Anzeige erneut über die kulturspezifischen Einstellungen festgelegt wird (Standard ist die lokale Zeit). |
| t | Ausgabe der Uhrzeit, dabei wird ShortTime verwendet. |
| T | Ausgabe der Uhrzeit, dabei wird LongTime verwendet. |

| Format | Beschreibung |
|---|---|
| u | Sortierbares Ausgabeformat analog zum Parameter s, allerdings wird als Zeitstandard die UTC-Weltzeit verwendet. |
| U | Ausgabe von Datum und Uhrzeit, dabei wird auf die UTC-Zeit normiert. |
| y, Y | Ausgabe von Jahr und Monat auf der Basis der kulturspezifischen Informationen |

*Tabelle 10.1: Formatierungszeichen für Standard-Datumsformate*

Neben diesen Standard-Ausgabearten können Sie auch ein Ausgabeformat selbst definieren. Hierfür stehen Ihnen auch viele unterschiedliche Formatierungsparameter zur Verfügung, die in der nachfolgenden Tabelle aufgelistet sind.

| Format | Beschreibung |
|---|---|
| d | Tag eines Monats als Zahl, einstellige Tage werden einstellig angezeigt. |
| dd | Tag eines Monats als Zahl, einstellige Tage werden mit führender Null angezeigt. |
| ddd | Abgekürzter Tag eines Monats |
| dddd | Ausgeschriebener Tag eines Monats abhängig von der eingestellten Kultur |
| M | Monat als Zahl, einstellige Monate werden einstellig angezeigt. |
| MM | Monat als Zahl, einstellige Monate werden mit führender Null angezeigt. |
| MMM | Abgekürzte Monatsbezeichnung |
| MMMM | Volle Monatsbezeichnung |
| y | Anzeige des Jahres (ohne Jahrhunderte). Bei Zahlen kleiner zehn wird keine führende Null vorangestellt. |
| yy | Anzeige des Jahres (ohne Jahrhunderte). Bei Zahlen kleiner zehn wird eine führende Null vorangestellt. |
| yyyy | Vollständige Ausgabe der Jahreszahl |

| Format | Beschreibung |
|---|---|
| gg | Zeitenwendenangabe (z. B.: v. Chr.), wenn diese Teil der Kulturparameter ist |
| h | Stunden im 12-Stunden-Format ohne führende Nullen |
| hh | Stunden im 12-Stunden-Format mit führenden Nullen |
| H | Stunden im 24-Stunden-Format ohne führende Nullen |
| HH | Stunden im 24-Stunden-Format mit führenden Nullen |
| m | Minute, einstellige Minuten ohne führende Nullen |
| mm | Minute, einstellige Minuten mit führenden Nullen |
| s | Sekunde, einstellige Sekunden ohne führende Nullen |
| ss | Sekunde, einstellige Sekunden mit führenden Nullen |
| f | Zehntel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| ff | Hundertstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| fff | Tausendstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| ffff | Zehntausendstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| fffff | Hunderttausendstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| ffffff | Millionstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| fffffff | Zehnmillionstel Sekunden Genauigkeit. Der Rest der Zeitangabe wird abgeschnitten. |
| t | Falls es eine Unterscheidung zwischen Vormittag und Nachmittag gibt (in den Kulturinformationen festgelegt), wird die erste Stelle hiermit angezeigt. |
| tt | Falls es eine Unterscheidung zwischen Vormittag und Nachmittag gibt (in den Kulturinformationen festgelegt), wird sie hiermit angezeigt. |
| z | Anzeige der Zeitzonenstunde ohne führende Null (z.B.: +1 für Mitteleuropäische Zeit) |

| Format | Beschreibung |
|---|---|
| zz | Anzeige der Zeitzonenstunde mit führender Null (z.B.: +01 für Mitteleuropäische Zeit) |
| zzz | Anzeige der Zeitzone mit führender Null (z.B.: +01:00 für Mitteleuropäische Zeit). Beachten Sie: Es gibt Zeitzonen, die eine halbe Stunde Abweichung haben können. |
| : | Standardtrennzeichen zwischen Zeiten |
| / | Standardtrennzeichen zwischen Datumsangaben |
| % c, c bedeutet ein Muster-zeichen aus den beiden Tabellen. | Erzeugung eines Formats – falls ausschließlich die Standardmuster (siehe vorherige Tabelle) verwendet werden, ist das Zeichen % nicht zwingend. |
| \ c, c bedeutet ein beliebiges Zeichen. | Erzwungene Anzeige genau dieses Zeichens |

*Tabelle 10.2: Formatierungszeichen für eigene Datumsformate*

Im nachfolgenden Beispiel demonstrieren wir Ihnen eine Reihe unterschiedlicher Datumsformate.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  DateTime Datum;
  Datum = DateTime.Now;
  a.Text = "Datums-Formate: <br />";
  a.Text += Datum.ToString();

  a.Text += "<br />Formatierung d: ";
  a.Text += Datum.ToString("d");
  a.Text += "<br />Formatierung D: ";
  a.Text += Datum.ToString("D");
  a.Text += "<br />Formatierung f: ";
  a.Text += Datum.ToString("f");
  a.Text += "<br />Formatierung F: ";
  a.Text += Datum.ToString("F");
```

```
  a.Text += "<br />Formatierung g: ";
  a.Text += Datum.ToString("g");
  a.Text += "<br />Formatierung G: ";
  a.Text += Datum.ToString("G");
  a.Text += "<br />Formatierung m: ";
  a.Text += Datum.ToString("m");
  a.Text += "<br />Formatierung r: ";
  a.Text += Datum.ToString("r");
  a.Text += "<br />Formatierung U: ";
  a.Text += Datum.ToString("U");
  a.Text += "<br />Format dddd, dd.MMMM yyyy: ";
  a.Text += Datum.ToString("dddd, dd.MMMM yyyy");
}
</script>
<html>
<head><title>DateTime Formatierungen</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 10.4: Datumsformate (datumsformate.aspx)*



Hier die zugehörige Bildschirmausgabe:

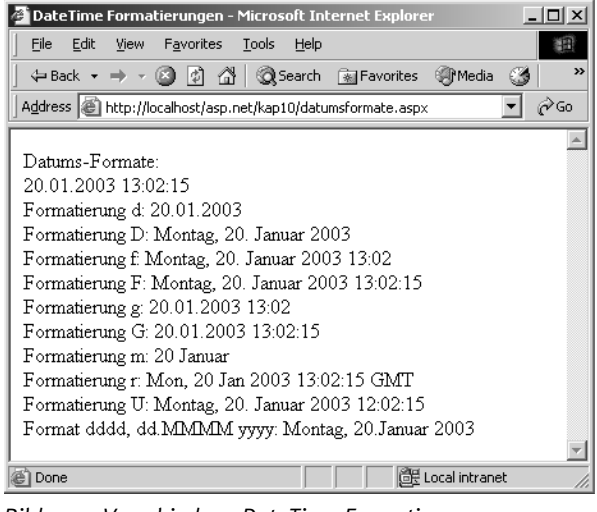



*Bild 10.4: Verschiedene DateTime-Formatierungen*

### 10.3.34 ToUniversalTime

Diese Methode wandelt die lokale Zeit in die Standard-Weltzeit um.

Das heißt, wenn z.B. der Server als Zeitzone die Mitteleuropäische Sommerzeit eingestellt hat, werden vom Datum zwei Stunden abgezogen und das neue Datum ausgegeben.

## 10.4 Die Operatoren

Für Rechenoperationen im Format von `DateTime` stehen Ihnen die Operatoren für Addition und Subtraktion sowie Vergleichsoperatoren für kleiner, kleiner oder gleich, größer, größer oder gleich, gleich sowie ungleich zur Verfügung.

Damit haben Sie dieselben Möglichkeiten wie bei einfachen numerischen Berechnungen und Vergleichen.

# 11 Reguläre Ausdrücke

Reguläre Ausdrücke haben aus der Sprachwissenschaft über die Theoretische Informatik auch in das tägliche Leben eines Programmierers Eingang gefunden.

Sie sind ein sehr mächtiges Werkzeug um Texte zu analysieren (zu parsen) und die Ergebnisse dann sinnvoll weiter zu verwenden. Mit der Verwendung von regulären Ausdrücken können Sie beispielsweise überprüfen, ob ein Satz bestimmten Regeln entspricht oder, was bei Webanwendungen sicherlich häufiger vorkommt, ob beispielsweise eine Emailadresse im Wesentlichen korrekt angegeben wurde.

Auf die Analyse einer Emailadresse werden wir später noch mehrfach zurückkommen, da sich diese als Demonstrationsbeispiel, was Sie mit regulären Ausdrücken alles anstellen können, ideal eignet.

Microsoft hat für die Behandlung von regulären Ausdrücken eine eigene Klasse bereitgestellt, die Klasse `Regex`. Wir werden Ihnen auch die zugehörigen Methoden und Eigenschaften dieser Klasse kurz vorstellen.



## 11.1 Was sind reguläre Ausdrücke?

Nun sollten wir zunächst einmal darauf eingehen, was eigentlich reguläre Ausdrücke sind. Um eine Definition aus der theoretischen Informatik zu verwenden sind reguläre Ausdrücke Zeichenketten in der Form, die sich durch einen endlichen Automaten (ein Konstrukt, das eine Textanalyse nach bestimmten Regeln durchführt) beschreiben lassen und die (deswegen) bestimmten Regeln gehorchen, sog. Grammatiken.

Eine ganz einfache Grammatik wäre beispielweise `[a-zA-Z]*nf`, diese beschreibt alle »Wörter«, die mit nf enden. Auf die Bedeutung der Syntax, die wir Ihnen vorgeführt haben, werden wir später noch einmal eingehen. Wörter haben wir deswegen in Anführungszeichen gesetzt, da beispielsweise neben den echten Wörtern wie Hanf oder Senf auch Wörter wie Anf über diese Regel erfasst würden.

Ein anderes Beispiel für eine Grammatik, die einen regulären Ausdruck beschreibt, wäre `^NittyGritty$`. In diesem Fall würde ausschließlich

ein Wort, nämlich »NittyGritty« als korrekter regulärer Ausdruck in einer Zeichenkette erkannt.

Tatsächlich sind reguläre Ausdrücke bei der syntaktischen Analyse eines Textes sehr hilfreich.

Die Methoden, mit denen reguläre Ausdrücke ausgewertet werden, sind so genannte endliche Automaten. Es gibt sie in verschiedenen Ausprägungen. Die Implementierung von Microsoft simuliert einen nicht-deterministischen endlichen Automaten, der vor allem den Vorteil besitzt, dass eine Art Rückreferenzierung möglich ist. Leider ist die Laufzeit einer Auswertung in den allgemein üblichen Auswertungsalgorithmen nicht linear (wie sie bei einem deterministischen endlichen Automaten wäre) – dies hat den Nachteil, dass die Ausführungszeit bei langen zu untersuchenden Ausdrücken durchaus langwierig werden kann.

Doch genug von der Theorie – wenn Sie Interesse an weiterführenden Informationen haben, können Sie einige einschlägige Literatur hierzu finden.



## 11.2 Die Zeichensprache für reguläre Ausdrücke

Kommen wir nun zu den einzelnen Zeichen, die in regulären Ausdrücken zur Verfügung stehen, und ihrer Bedeutung.

Es gibt eine ganze Reihe von Sonderzeichen, die bestimmte Regeln bedeuten. Die erste große und wichtigste Gruppe sind die so genannten Multiplikatoren, die festlegen, ob und wie oft eine bestimmte Gruppe von Zeichen wiederholt werden soll. Beispielsweise bedeutet A?, dass der Buchstabe A null oder einmal an der bezeichneten Stelle vorkommt. A+ bedeutet dagegen, dass der Buchstabe A mindestens einmal vorkommt – seine Anzahl ist aber grundsätzlich unbegrenzt.

Die nachfolgende Tabelle gibt einen Überblick über die Multiplikatoren, die es in der ASP.NET-Implementierung regulärer Ausdrücke gibt.

| Multi-plikator | Bedeutung | Beispiel |
|---|---|---|
| * | Beliebige Anzahl (auch nullmal) | A*C: C, AC, AAC, ... |
| + | Mindestens einmal | A+C : AC, AAC, AAAC; AAAAC,... |
| ? | Null oder einmal | A?BC: BC, ABC |
| {n} | Genau n-mal | A{3}B: AAAB |
| {n,} | Mindestens n-mal | A{2,}B: AAB, AAAB, AAAAB,... |
| {n,m} | Mindestens n-mal, höchstens m-mal | A{2,3}B: AAB, AAAB |
| *? | Erste Übereinstimmung mit so wenig Wiederholungen wie möglich | |
| +? | Erste Übereinstimmung mit so wenig Wiederholungen wie möglich (aber mindestens einmal vorkommend) | |
| ?? | Null oder eine Wiederholung | |

*Tabelle 11.1: Multiplikatoren bei regulären Ausdrücken*

Als weiteres wesentliches Sprachelement gibt es die Möglichkeit Zeichengruppen anzugeben. Beispielsweise bedeutet [ABC], dass entweder A, B oder C ein gültiges Zeichen an dieser Position darstellt.

Auch hierfür haben wir Ihnen in einer Tabelle die möglichen Gruppierungsmöglichkeiten, die in ASP.NET zur Verfügung stehen, in einer Tabelle zusammengefasst.

| Gruppen-zeichen | Beschreibung | Beispiel |
|---|---|---|
| . | Alle Zeichen außer \n (Zeilenvorschub) sind gültig. | .A: AA, BA, bA, _A,... |
| [] | Eines der in der Klammer angegebenen Zeichen kann verwendet werden. | [ASP]B: AB, SB, PB |
| [^] | Die in der Klammer angegebenen Zeichen können nicht verwendet werden. | [^ASP]B: BB, CB, DB, ... |

| Gruppen-zeichen | Beschreibung | Beispiel |
|---|---|---|
| - | Erlaubt als Von-Bis-Zeichen eine Abkürzung der Schreibweise innerhalb der eckigen Klammern. | [1-9]: 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| \p{name} | Festlegung einer Klasse von Zeichen, die zutreffend ist (Unicode-Bezeichner oder auch Gruppen von Zeichen sind erlaubt). | \p{IsGreek}: µ,... |
| \P{name} | Festlegung einer Klasse von Zeichen, die nicht zutreffend ist. | \P{IsGreek}: alle nicht griechischen Zeichen |
| \w | Jedes alphabetische Zeichen wird gefunden. | |
| \W | Jedes nicht alphabetische Zeichen wird gefunden. | \W: 1, 2, §,... |
| \s | Jedes Sonderzeichen, das keinen Text ausgibt (also sog. »Whitespaces«), wird gefunden. | |
| \S | Invertierung von \s | |
| \d | Jede Zahl wird gefunden. | |
| \D | Invertierung von \d | |

*Tabelle 11.2: Gruppierungsmöglichkeiten regulärer Ausdrücke*



Nachdem Sie nun die Hauptregeln kennen gelernt haben, sollten wir noch kurz auf die Besonderheiten eingehen, wie die Frage, wie kann ich Zeichen darstellen, die eigentlich Teil der Beschreibung des regulären Ausdrucks sind, wie z.B. das Zeichen [ ?

Diese Zeichen werden durch einen vorangestellten Backslash gekennzeichnet. Dies bedeutet, dass sie nicht als Regel interpretiert werden sollen, sondern als das Zeichen, das sie nebenbei auch noch darstellen.

Weiterhin lassen sich auch bestimmte spezielle Sonderzeichen (der Systemsound beispielsweise) auf diese Art in regulären Ausdrücken beschreiben.

In der nachfolgenden Tabelle haben wir dargestellt, welche Zeichenkombination welches – besondere – Zeichen bedeutet.

| Darstellung | Sonderzeichen |
|---|---|
| \ | Wenn Sie beispielsweise \+ schreiben, dann wird das +-Zeichen dargestellt. Für \\ wird also das \-Zeichen dargestellt. |
| \040 | Das ASCII-Zeichen, das im Oktal-Zahlenformat die Stelle 040 belegt (die Space-Taste) – Sie können so alle beliebigen Zeichen benennen, die Zahlen müssen für den Oktalwert immer dreistellig sein. |
| \a | Systemton |
| \b | Backspace-Taste |
| \cA | Zeichen, das gemeinsam mit der Ctrl (oder Strg)-Taste gedrückt wird. In diesem Falle also Ctrl-A. |
| \e | Escape-Taste |
| \f | Seitenvorschub (auch Form Feed genannt) |
| \n | Zeilenvorschub (auch Line Feed genannt) |
| \r | Wagenrücklauf (auch Carriage Return genannt) |
| \t | Tabulator-Zeichen |
| \v | Vertikaler Tabulator |
| \x20 | Das ASCII-Zeichen, das im Hexadezimal-Zahlenformat die Stelle 20 belegt (die Space-Taste) – Sie können so alle beliebigen Zeichen benennen, die Zahlen müssen für den Hexadezimalwert immer zweistellig sein und ein führendes x haben. |

*Tabelle 11.3: Sonderzeichendarstellung für reguläre Ausdrücke*

Nun zeigen wir Ihnen nur noch einige wenige Besonderheiten, bevor wir Ihnen dann an einem praktischen Beispiel die Verwendung dieser Regeln für reguläre Ausdrücke demonstrieren.

Bisher sind wir beispielsweise noch nicht auf das Zeichen | eingegangen, das eine Alternative darstellen soll: A|B bedeutet also, dass entweder A oder B als korrektes Zeichen angesehen werden kann.

Beachten Sie, dass eine Alternative `AB|C` bedeutet, dass es die Alternativen `AB` oder `C` gibt und nicht `AB` oder `BC`!

Diese Besonderheiten haben wir in der nachfolgenden Tabelle zusammengefasst.

| Zeichen | Beschreibung | Beispiel |
|---|---|---|
| \| | Alternative | [A\|B]: A, B |
| ^ | (am Anfang) Festlegung, dass die Übereinstimmung am Anfang der Zeichenkette auftreten muss | ^A*: A, AA, ... |
| ^ | (Innerhalb von Klammern) Logisches Nicht | [^a]: Alle einbuchstabigen Wörter außer a |
| $ | Das Zeichen oder die Zeichenkette muss am Stringende auftreten. | *A$: A, BA, BBA,... |
| () | Klammerungen analog zu mathematischen Termen | B(A+)B: BAB, BAAB, BAAAB,... |
| \1 | Rückreferenzierungen: Wenn ein Term aufgetreten ist, muss er doppelt auftreten. | (AAB)\1A: AABAABA |

*Tabelle 11.4: Besondere Zeichen für reguläre Ausdrücke*



Nun haben wir alles, was an Grund-»Vokabeln« notwendig ist, beisammen.

Eine letzte Tabelle können wir Ihnen dennoch nicht ersparen, in der Sie die Optionen sehen können, die die grundsätzlichen Analyseregeln festlegen. Dann kommen wir aber endgültig auf unser Beispiel zurück.

| Option | Beschreibung |
|---|---|
| `None` | Es sind keine besonderen Optionen festgelegt. |
| `IgnoreCase` | Die Treffer sollen festgestellt werden, unabhängig von Groß- oder Kleinschreibung. |
| `Multiline` | Mit dieser Option wird festgelegt, dass das Erkennen von regulären Ausdrücken auch über mehrere Zeilen durchgeführt werden kann. |
| `ExplicitCapture` | Mittels dieser Option können Sie festlegen, dass die Ziele explizit angegeben werden können. |

| Option | Beschreibung |
|---|---|
| `Compiled` | Ermöglicht das Kompilieren der regulären Ausdrücke, dadurch wird die Startzeit verlängert, aber dafür die Ausführungszeit deutlich verkürzt. |
| `Singleline` | Hiermit wird festgelegt, dass die Auswertung nur in einer Zeile erfolgen soll. |
| `IgnorePattern-Whitespace` | Bei der Treffersuche wird ignoriert, ob ein Leerzeichen, ein Tabulator oder Ähnliches gesetzt wurde oder nicht. |
| `RightToLeft` | Statt von links nach rechts wird ein Text von rechts nach links untersucht. Hierbei müssen Sie aber darauf achten, dass der Suchcursor auch am Ende der Textzeile steht. |

*Tabelle 11.5: Optionen für die Behandlung von regulären Ausdrücken*

Aber jetzt zu dem versprochenen Beispiel, einer Überprüfung einer E-Mail-Adresse:

Werfen wir Sie ins kalte Wasser – hier ist der reguläre Ausdruck, der eine E-Mail-Adresse auf verschiedene notwendige Eigenschaften überprüft. Wir werden nachfolgend die einzelnen Elemente noch einmal erklären.

```
^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.[a-zA-Z]{2,}$
```

Überprüfen wir diesen regulären Ausdruck: Am Anfang und am Ende stehen jeweils ein `^` und ein `$`. Dies bedeutet, dass der zu untersuchende reguläre Ausdruck genau die Länge des zu untersuchenden Strings haben soll.

Der Abschnitt `[a-zA-Z._\-]`, der doppelt vorkommt, erlaubt neben dem Alphabet von A bis Z (auch als Kleinbuchstaben) auch die Zeichen für Punkt, Unterstreichungsstrich sowie das Minuszeichen (durch den Backslash wurde sichergestellt, dass dieses Zeichen gemeint war). Der dritte, ähnliche Abschnitt `[a-zA-Z]` erlaubt also nur Buchstaben.

Die Zeichen nach der eckigen Klammer geben jeweils Auskunft darüber, wie oft die Zeichen in den Klammern auftauchen dürfen, damit sie tatsächlich reguläre Ausdrücke darstellen. Die erste Zeichengruppe darf mindestens einmal auftreten (aber sie muss auch mindestens ein-

mal auftreten). Die zweite und die dritte Gruppe müssen mindestens zweimal auftreten (also aus mindestens zwei Zeichen bestehen.

Zu guter Letzt wird noch festgelegt, dass genau einmal ein `@` nach der ersten Buchstabengruppe und ein Punkt vor der dritten Buchstabengruppe auftreten muss.

Schon haben Sie einen recht komplexen regulären Ausdruck entworfen.

## 11.3 Regex-Eigenschaften

Kommen wir nun zu den Eigenschaften der Klasse `Regex`.

### 11.3.1 Options

Mittels dieser Eigenschaft können Sie die Optionen, die zum Regelwerk der regulären Ausdrücke gehören, setzten oder löschen sowie überprüfen, wie sie gerade gesetzt sind.

### 11.3.2 RightToLeft

Dieser Wert liefert zurück, ob die Suche innerhalb des Dokuments von links nach rechts oder von rechts nach links durchgeführt werden soll. Falls der Automat zur Überprüfung der regulären Ausdrücke mit der Option `RightToLeft` erstellt wurde.



## 11.4 Die Regex-Methoden

Hier noch die Methoden der `Regex`-Klasse.

### 11.4.1 CompileToAssembly

Mittels dieser Methode können Sie den regulären Ausdruck kompilieren und auf Festplatte für ein späteres Einbinden abspeichern.

### 11.4.2 Escape

Mittels dieser Methode können Sie einen Text so umformatieren, dass eine bestimmte Gruppe von Zeichen (\, *, +, ?, |, {, [, (,), ^, $, ., #, Leerzeichen sowie Tabulatorzeichen) durch einen vorangestellten Backslash gekennzeichnet sind.

Sie müssen den String, der entsprechend umzuwandeln ist, als Parameter angeben.

Ein Aufruf sieht also wie folgt aus:

```
Ergebnis = Regex.Escape(Umzuwandelnder_String);
```

### 11.4.3 GetGroupNames

Für jede Treffergruppe wird über diese Methode der vergebene Name ermittelt (falls keine explizite Namensbildung festgelegt wurde, steht in dem Array, das zurückgeliefert wird, eine Zahl als Benennung).

### 11.4.4 GetGroupNumbers

Alternativ haben Sie eine Liste als Array von Integerwerten, die die Gruppennummern bedeuten. Ein Zugriff über ein Integerarray ist schneller als ein Zugriff über ein Array von Strings.

### 11.4.5 GroupNameFromNumber

Mittels dieser Methode können Sie eine Referenz von der Gruppennummer zum Gruppennamen herstellen.

### 11.4.6 GroupNumberFromName

Diese Methode stellt die Umkehrung von `GroupNameFromNumber` dar. Es wird also eine Referenz von Gruppenname zur Gruppennummer vorgenommen.

### 11.4.7 IsMatch

Mittels dieser Methoden können Sie überprüfen, ob ein erzeugter regulärer Ausdruck in einem als Parameter definierten String zu finden ist. Sie haben weiterhin die Möglichkeit eine Startposition für die Suche innerhalb des Strings anzugeben.

Als weitere Möglichkeit können Sie auch den regulären Ausdruck als String und zusätzlich die Kompilierungsoptionen zusätzlich mit angeben.

Die Rückgabewerte sind in diesem Fall boolesche Ausdrücke. Ein Aufruf würde also die nachfolgende Form haben:

```
Ergebnis = Regex.IsMatch (String, RegExp);
```

### 11.4.8 Match

Analog zu `IsMatch` können Sie mit der `Match`-Methode einen String nach einem regulären Ausdruck durchsuchen. Das Ergebnis wird in einem `Match`-Objekt zurückgegeben.

Darüber können Sie beispielsweise die Gruppen von Treffern ermitteln und haben weitere Strukturinformationen, die Ihnen mit einer einfachen `IsMatch`-Auswertung nicht zur Verfügung stehen würden.

Das `Match`-Objekt stellt Ihnen durch Mehrfachaufruf der `Match`-Methode eine verkettete Liste zur Verfügung, über die Sie beispielsweise alle zulässigen regulären Ausdrücke in einem String ermitteln können. Der Aufruf kann z.B. wie folgt durchgeführt werden:

```
Ergebnis = Regex.Match (String, RegExp);
```

### 11.4.9 Matches

Mit der `Matches`-Methode erhalten Sie eine Kollektion von `Match`-Objekten, die alle Treffer des regulären Ausdrucks in dem untersuchten String enthält. Hiermit vermeiden Sie den lästigen Mehrfachaufruf von `Match`.

### 11.4.10 Replace

Alle gefundenen regulären Ausdrücke in einem String werden durch einen anderen String ersetzt. Hierbei ist der erste Parameter der String, in dem der reguläre Ausdruck zu finden ist, der zweite Parameter der reguläre Ausdruck und der dritte Parameter der String, der als Ersetzung zu verwenden ist.

Ein Aufruf sieht also beispielsweise so aus.

```
Ergebnis = Regex.Replace(Text, Regex, Ersetzung);
```

### 11.4.11 Split

Ein String wird in ein Array von Strings aufgeteilt. Die Trennung in die Teilstrings erfolgt an den durch den regulären Ausdruck definierten Stellen.

```
Stringarray = Regex.Split(Text, Regex);
```

### 11.4.12 ToString

Der reguläre Ausdruck wird in einen String umgewandelt.

### 11.4.13 Unescape

Zeichen in einer Zeichenkette, die mit einem vorangestellten Backslash versehen sind, verlieren diesen Backslash und werden – falls sie durch ihre Hexadezimal- oder Oktalcodes repräsentiert werden – in ein lesbares Standardzeichen umgewandelt.

Hier ein Beispielaufruf:

```
Ergebnis = Regex.Unescape(Text);
```

## 11.5 Beispielprogramm

Dieses – zugegebenermaßen kleine – Beispielprogramm wertet einen fest vorgegebenen String mit einem regulären Ausdruck aus. Vorher wird der String noch »Escaped« und wieder »Ent-Escaped«.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  string Text;
  string Ergebnis1;
  string Ergebnis2;
  string Regexp;
  Match Ergebnis3;
  Match Ergebnis5;
  MatchCollection Ergebnis4;
  int i;

  Text = "Hallo dieses waren 0.25 $ und 1 Cent.";
  Regexp = "[a-zA-Z]{4,5}(|.)";
  a.Text = "Regexp: <br />";

  a.Text += "Text ist: <br />";
  a.Text += Text + " <br />";
  a.Text += "Escaped: <br />";
  Ergebnis1 = Regex.Escape(Text);
  a.Text += Ergebnis1 + " <br />";
  a.Text += "Unescaped: <br />";
```

```
  Ergebnis2 = Regex.Unescape(Ergebnis1);
  a.Text += Ergebnis2 + " <br />";
  a.Text += "Match: <br />";
  Ergebnis3 = Regex.Match(Text,Regexp);
  a.Text += Ergebnis3.ToString() + " <br />";
  a.Text += "Matches: <br />";
  Ergebnis4 = Regex.Matches(Text,Regexp);
  a.Text += "Anzahl Treffer: <br />";
  i = Ergebnis4.Count;
  a.Text += i.ToString() + " <br />";
  Ergebnis3 = Regex.Match(Text,Regexp);
  for (i=0;i<Ergebnis4.Count; i++)
    {
      a.Text += Ergebnis3.ToString();
      a.Text +=  "<br />";
      Ergebnis3 =Ergebnis3.NextMatch();
    }
}
</script>
<html>
<head><title>Regex-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 11.1:* Regex-*Funktionen (regex.aspx)*

Der reguläre Ausdruck soll alle Wörter mit mindestens vier oder fünf Buchstaben finden und diese ausgeben (daher wurden vom sechsbuchstabigen Wert »dieses« nur fünf Buchstaben, nämlich »diese« ausgegeben). Es ist ein Treffer des angegebenen regulären Ausdrucks.

Wir hoffen, dass diese »Spielerei« mit regulären Ausdrücken Ihnen einen Eindruck vermittelt hat, was man mit diesem mächtigen Werkzeug alles anfangen kann.

# 12 HTML Controls

Was sind HTML Controls? Einfach ausgedrückt sind HTML Controls spezielle HTML-Elemente, die serverseitig ausgeführt werden. Ein einfaches HTML Control ist beispielweise die Textausgabe mit dem nachfolgenden kleinen Beispielprogramm.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  ausgeben.InnerText = "Eine Textausgabe.";
}
</script>
<html>
<head><title>Textausgabe</title></head>
<body>
<p Id="ausgeben" runat="server"></p>
</body>
</html>
```

*Listing 12.1: Eine einfache Textausgabe (textausgabe.aspx)*

Dieses Programm gibt einen Text aus, indem das HTML-Tag `<p>` mit einem Identifikator `id` versehen und spezifiziert wird, dass der Tag serverseitig ausgeführt werden soll (`runat="server"`). Das serverseitige Programm versieht die Eigenschaft `InnerText` mit einem String. Dieser String wird dann ausgegeben. Den dadurch erzeugten HTML-Quelltext sehen Sie in den nächsten Zeilen.

```
<html>
<head><title>Textausgabe</title></head>
<body>
<p id="ausgeben">Eine Textausgabe.</p>
</body>
</html>
```

Natürlich können Sie wesentlich mehr über HTML Controls steuern und anlegen. Sie sind prädestiniert für Formularelemente, die vorausgefüllt sein können oder nach der Eingabe serverseitig überprüft werden können.

Auch ein Dateihandling (ein Dateiupload) können Sie über HTML Controls einfach realisieren. Aber stellen wir Ihnen nun schrittweise die wichtigsten Elemente vor.

Dieses kleine Programm enthält im Deklarationsteil der Prozedur ein weiteres wesentliches Element. Es ist ein Ereignis darin definiert worden. Auf diese Ereignisse wollen wir als Erstes einmal einen Blick werfen.

## 12.1 Ereignisse und ihre Verarbeitung

Für die Verarbeitung von Programmen sind neben dem Standard-Programmablauf noch andere Dinge wichtig. Z.B. sollen eventuell bestimmte Dinge durchgeführt werden, wenn die Maus gedrückt wird oder wenn eine Dateiverbindung zusammenbricht. Solche Dinge werden Ereignisse genannt.

Wir haben in den letzten Listings bereits immer wieder einen Vertreter von Ereignissen verwendet.



Wir haben das Ereignis `ServerClick` ausgenutzt, um sicherzustellen, dass, wenn eine Maustaste gedrückt wurde, eine Aktion auf dem Server ausgeführt wurde. Wir haben dann eine Prozedur aufgerufen, die durch das Ereignis abgearbeitet werden konnte. Die Methode `OnServerClick` wird bei Buttons ausgelöst.

Ein anderes Ereignis löst beispielsweise `OnClick` aus, das durch das Klicken eines ImageButtons oder eines Buttons erzeugt wird.

Über ein solches Ereignis können Sie beispielsweise ein anderes wichtiges Ereignis auslösen, das `PostBack`-Ereignis, das eine erneute Ausführung der ASP.NET-Seite auslöst. Auch dieses Ereignis ist in der täglichen Anwendung sehr hilfreich.

Einen Beispielcode für den Aufruf einer Prozedur, die durch ein Ereignis aufgerufen wird, sehen Sie im nachfolgenden Codeblock.

```
void Ereignis(Object o, EventArgs e)
{
  ...
}
<button Id="S" runat="server"
OnServerClick="Ereignis">Ereignis</button>
```

In der nachfolgenden Übersicht sehen Sie die wichtigsten Ereignisse und in welchen Klassen des `HtmlControl`-Namespaces sie zu finden sind.

| Ereignis | Beschreibung | Zugeordnete Klassen |
|---|---|---|
| Data-Binding | Dieses Ereignis tritt auf, wenn ein Control mit einer Datenquelle verknüpft wird | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |
| Disposed | Dieses Ereignis tritt auf, wenn ein Control aus dem Speicher wieder gelöscht wird | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |
| Init | Tritt bei der Initialisierung des Controls auf | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |
| Load | Tritt auf , wenn das Control in das Page-Objekt integriert wird | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |
| PreRender | Tritt in dem Augenblick auf, wenn das Rendern des Page-Objekts beginnt | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |

| Ereignis | Beschreibung | Zugeordnete Klassen |
|---|---|---|
| Server-Change | Tritt auf, wenn sich der Wert des Controls ändert und dieser an den Server gesandt wird | HtmlInputCheckBox, HtmlInputHidden, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTextArea |
| ServerClick | Tritt auf, wenn das Control angeklickt wird | HtmlAnchor, HtmlButton, HtmlInputButton, HtmlInputImage |
| Unload | Tritt auf, wenn das Control aus dem Speicher entfernt wird | HtmlAnchor, HtmlButton, HtmlForm, HtmlImage, HtmlInputButton, HtmlInputCheckBox, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputRadioButton, HtmlInputText, HtmlSelect, HtmlTable, HtmlTableCell, HtmlTableRow, HtmlTextArea |

*Tabelle 12.1: Ereignisse im HtmlControls-Namespace*



## 12.2 Die Klassen des HtmlControls-Namespaces

Kommen wir nun aber zu den einzelnen HTML-Controls des `HtmlControls`-Namespaces.

### 12.2.1 HtmlAnchor

Mittels dieses Controls haben Sie Zugriff auf den `<a>`-Tag, der einen Link darstellt.

Sie können dieses Control verwenden, um einen Link, den Sie dem Benutzer anzeigen wollen, serverseitig dynamisch zu generieren.

Das nachfolgende kleine Beispiel macht genau dies und gibt einen – auf dem Server definierten – Link aus.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
```

```
  link.HRef = "http://www.addison-wesley.de";
  link.InnerText = "Addison Wesley";
  a.Text = "Hyperlink <br />";
}
</script>
<html>
<head><title>Hyperlink</title></head>
<body>
<asp:Label Id="a" runat="server" />
<a Id="link" runat="server"></a>
</body>
</html>
```

*Listing 12.2: Link (anchor.aspx)*

### 12.2.2 HtmlButton

Mittels dieses Controls wird ein `<button>`-Tag ausgewertet. Die einzelnen Radiobuttons werden dabei anhand der `Id`-Parameter unterschieden. Diese müssen Sie also immer mit angeben.


*Diese Parameter müssen übrigens in ASP.NET so geschrieben sein, dass sie keine Leerzeichen oder Sonderzeichen enthalten.*

Hier nun ein kleines Beispiel, bei dem Sie einen Buttondruck serverseitig auswerten und anzeigen, welcher Button gedrückt wurde.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Knopf";
}
void Click1(Object o, EventArgs e)
{
  a.Text = "Knopf 1 gedrueckt";
}
void Click2(Object o, EventArgs e)
{
  a.Text = "Knopf 2 gedrueckt";
}
</script>
```

```
<html>
<head><title>Knopf-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<button Id="Knopf1" runat="server"
OnServerClick="Click1">Knopf 1</button>
<button Id="Knopf2" runat="server"
OnServerClick="Click2">Knopf 2</button>
</form>
</body>
</html>
```

*Listing 12.3: Schaltflächen (button.aspx)*

Hier ist noch einmal kurz zusammengefasst, was dieses Programm macht: Es wartet serverseitig auf einen Druck auf einen der beiden Knöpfe. Je nachdem welcher der Knöpfe gedrückt wurde, wird ein Text ausgegeben.


*Um dieses Control zu verwenden müssen Sie (beziehungsweise die Besucher Ihrer Website) JavaScript aktiviert haben. Dies bedeutet auch, dass dieses Control nur in Browsern funktioniert, die JavaScript unterstützen.*

*Wenn Sie sichergehen wollen, dass Ihre Buttons in jedem Fall funktionsfähig sind, dann verwenden Sie zum Beispiel das* `HtmlInputButton`*-Control, das wir in diesem Kapitel noch vorstellen werden.*

### 12.2.3 HtmlForm

Mittels des `HtmlForm`-Controls haben Sie Kontrolle über die `<form>`-Tags. Damit können Sie Formulare serverseitig versenden bzw. vor dem Versand überprüfen oder auch vorausfüllen. Dieses HTML-Control bettet alle einzelnen HTML-Tags, die in Formularen verwendet werden, ein.

Ein Aufruf erfolgt, wie im nachfolgenden Codefragment demonstriert.

```
<form runat="server">
...
</form>
```

Dieses Control wird im Wesentlichen für die Verwendung von Postback-Mechanismen für nicht-postback-fähige Controls benötigt.

Postback-Mechanismen bedeuten dabei die Möglichkeit, nach dem Auftreten bestimmter Ereignisse den Programmcode mit den Ergebnissen dieser Ereignisse auszustatten und erneut ablaufen zu lassen.

Die Controls, die diese Mechanismen verwenden, müssen innerhalb eines `HtmlForm`-Controls stehen.

Ein Beispiel für eine Anwendung sehen Sie bei `HtmlButton`.

### 12.2.4 HtmlImage

Über `HtmlImage`-Controls erhalten Sie serverseitigen Zugriff auf `<img>`-Tags. Auf diese Weise können Sie Bilder dynamisch in eine Webseite einbinden.

### 12.2.5 HtmlInputButton Control

Über das `HtmlInputButton`-Control können Sie serverseitig Zugriff auf die nachfolgenden Html-Tags erhalten: `<input type="button">`, `<input type="submit">` sowie `<input type="reset">`. Die Funktionsweise ist analog zu `HtmlButton`-Controls.

Zusätzlich haben Sie auf der Programmebene die Möglichkeit eine `Value`-Eigenschaft zu setzen und damit unterschiedliche Werte zu übergeben. Die `Value`-Eigenschaft muss vom Typ String sein.

Anbei ein kleines Beispiel.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Input-Knopf <br />";
}
void Click1(Object o, EventArgs e)
{
  Knopf1.Value = "gedrückt";
  Knopf2.Value = "nicht gedrückt";
}
void Click2(Object o, EventArgs e)
{
```

```
  Knopf2.Value = "gedrückt";
  Knopf1.Value = "nicht gedrückt";
}
</script>
<html>
<head><title>Input-Button-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<input type="button" Id="Knopf1" runat="server"
OnServerClick="Click1">Knopf 1</input>
<input type="button" Id="Knopf2" runat="server"
OnServerClick="Click2">Knopf 2</input>
</form>
</body>
</html>
```

*Listing 12.4:* `<input>`*-Schaltflächen (inputbutton.aspx)*

### 12.2.6 HtmlInputCheckBox Control

Mit dem `HtmlInputCheckBox`-Control haben Sie Zugriff auf die HTML-Tags vom Typ `<input type="checkbox">`. Nachfolgend finden Sie auch hier ein kleines Beispiel, das wir Ihnen in seiner Funktionalität kurz erläutern werden.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Checkboxen <br />";
}
void Senden(Object o, EventArgs e)
{
  if (Box1.Checked == true) {
    a.Text += "Box 1 gedrueckt <br />";
  }
  if (Box2.Checked == true) {
    a.Text += "Box 2 gedrueckt <br />";
  }
}
</script>
<html>
<head><title>Checkbox-Funktionen</title></head>
<body>
```

```
<asp:Label Id="a" runat="server" />
<form runat="server">
<input Type="checkbox" Id="Box1" runat="server">Knopf 1</input>
<input Type="checkbox" Id="Box2" runat="server">Knopf 2</input>
<button Id="S" runat="server"
OnServerClick="Senden">Senden</button>
</form>
</body>
</html>
```

*Listing 12.5: Checkboxen (checkbox.aspx)*

Mittels der Eigenschaft `Checked` wird überprüft, ob die Checkbox gesetzt ist oder nicht. Falls ja, wird ein Text ausgegeben.

### 12.2.7 HtmlInputFile

Mittels des `HtmlInputFile`-Controls können Sie auf Html-Tags vom Typ `<input type="file">` serverseitig zugreifen. Sie können dadurch beispielsweise dynamisch Dateien laden und anzeigen oder auch nach dem Laden manipulieren.

Beachten Sie, dass Sie, wenn Sie eine Anzeige eines Files durchführen wollen, das Tag `enctype` auf den Wert `"multipart/form-data"` setzen müssen.

Die Eigenschaften, die mit dieser Klasse verknüpft sind, legen die MIME-Typen fest, die akzeptiert werden (`Accept`). Über die nur lesbare Eigenschaft `PostedFile` erhalten Sie Zugriff auf die übertragene Datei.

Dieser Zugriff wird im Wesentlichen über die Klasse `HtmlPostedFile` bereitgestellt. Über die Eigenschaft `HtmlPostedFile.InputStream` wird Ihnen ein (nur lesbarer) Stream bereitgestellt, der den Zugriff auf die Dateiinhalte zulässt.

### 12.2.8 HtmlInputHidden

Tags, die versteckt sind – also den Typ `<input type="hidden">` besitzen – können serverseitig mittels dieses Controls manipuliert werden.

### 12.2.9 HtmlInputImage

Tags, die Bilder enthalten – also den Typ `<input type="image">` besitzen – können serverseitig mittels dieses Controls manipuliert werden.

Damit haben Sie auch für über Bilder generierte Knöpfe die Möglichkeit Bilder oder Aktionen zu gestalten. Über die Eigenschaft `Src` legen Sie den Pfad des Bildes fest. Diesen Pfad können Sie auch über den Parameter `src` setzen.

### 12.2.10 HtmlInputRadioButton

Kommen wir nun wieder zu einem Control, dem wir ein Beispiel zuordnen. Über das `HtmlInputRadioButton`-Control haben Sie die Möglichkeit Radiobuttons serverseitig zu manipulieren. Das zugehörige HTML-Tag ist `<input type="radio">`.

Auch hier haben Sie über die Eigenschaft `Checked` die Möglichkeit abzufragen, ob ein Button gesetzt ist oder nicht.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Radiobuttons <br />";
}
void Senden(Object o, EventArgs e)
{
  if (Box1.Checked == true) {
    a.Text += "Alternative 1 <br />";
  }
  if (Box2.Checked == true) {
    a.Text += "Alternative 2 <br />";
  }
  if (!(Box2.Checked)&&!(Box1.Checked)) {
    a.Text += "Druecken Sie einen Button! <br />";
  }
}
</script>
<html>
<head><title>Radiobutton-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<input type="radio" Id="Box1" runat="server">Alternative 1</input>
```

```
<input type="radio" Id="Box2" runat="server">Alternative 2</input>
<button Id="S" runat="server" OnServerClick="Senden">Senden</button>
</form>
</body>
</html>
```

*Listing 12.6: Radiobuttons (radiobutton.aspx)*

In diesem Beispiel haben wir zwei Radiobuttons mit einem Text versehen; wir überprüfen nun, ob der Anwender einen der beiden Radiobuttons angewählt hat, und geben im entsprechenden Fall den angewählten Radiobutton aus.

### 12.2.11 HtmlInputText

Über das `HtmlInputText`-Control haben Sie die Herrschaft über die Html-Tags `<input type="text">` und `<input type="password">`. Sie können beispielsweise einen Text bereits auf der Serverseite in ein Textfeld hineinsetzen.


*Bei Passwortfeldern funktioniert das serverseitige Vorbelegen übrigens aus Sicherheitsgründen nicht.*

Der Text, der in dieses Textfeld eingegeben wurde, kann über die `Text`-Eigenschaft wieder ausgelesen und weiterverarbeitet werden.

Im nachfolgenden Beispiel haben wir ein Textfeld zur Eingabe bereitgestellt. Nach Absenden geben wir den Text des Textfeldes aus.

```
<%@ Page Language="#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  text.Size = 30;
  text.MaxLength = 10;
  a.Text = "Textfeld <br>";
}
void Senden(Object o, EventArgs e)
{
  a.Text += text.Value + " <br>";
}
</script>
```

```
<html>
<head><title>Textfeld-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<input tpe="text" Id="text" runat="server">
Bitte geben Sie einen Text ein.</input>
<button Id="S" runat="server"
OnServerClick="Senden">Senden</button>
</form>
</body>
</html>
```

*Listing 12.7: Textfelder (text.aspx)*

Sie haben übrigens über die Methoden und die Eigenschaften der `HtmlInputText`-Klasse die Möglichkeit, beispielsweise auch die maximale Anzahl von Zeichen oder die Größe des Textfeldes serverseitig im Programm zu bestimmen – dies haben wir auch in unserem Beispiel getan.



### 12.2.12 HtmlSelect

Über das `HtmlSelect`-Control haben Sie Zugriff auf das `<select>`-Zeichen. Damit können Sie serverseitig die Auswertung von Auswahllisten kontrollieren.

Für die Verwendung von Auswahllisten benötigen Sie, da es ja sein kann, dass mehrere Felder angewählt wurden, eine einfache Möglichkeit dies zu überprüfen. Dies erfolgt über die Eigenschaft `Auswahlfeld.Items[Position].Selected`.

`Auswahlfeld` ist dabei der Name, den Sie Ihrem Auswahlfeld gegeben haben. `Position` ist die Position, an der sich der entsprechende Text befindet.

Falls Sie nicht wissen, welcher Text sich an der gesuchten Position befindet, so haben Sie die Möglichkeit, ihn mit `Auswahlfeld.Items[Position].Value` zu ermitteln.

Voraussetzung hierfür ist natürlich, dass der bei `Value` gesetzte Wert mit dem angezeigten Text übereinstimmt.

Kommen wir nun zu unserem kurzen Beispiel, das Ihnen vermutlich mehr helfen kann.

```
<%@ Page Language="cs" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Auswahlliste <br>";
}
void Senden(Object o, EventArgs e)
{
  a.Text += "Sie haben ausgewaehlt: <br />";
  for (int i=0;i<Box.Items.Count; i++)
  {
  if(Box.Items[i].Selected) {
      a.Text += Box.Items[i].Value + "<br />";
    }
  }
}
</script>
<html>
<head><title>Auswahlliste-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<select Id="Box" multiple runat="server">
<option Value="Wert1">Wert1</option>
<option Value="Wert2">Wert2</option>
<option Value="Wert3">Wert3</option>
<option Value="Wert4">Wert4</option>
</select>
<button Id="S" runat="server"
OnServerClick="Senden">Senden</button>
</form>
</body>
</html>
```

*Listing 12.8: Auswahllisten (auswahlliste.aspx)*


*Beachten Sie, dass Ihr Index unbedingt innerhalb der Zählgrenzen bleiben muss – also Anzahl der Elemente minus eins, da Sie ansonsten einen Laufzeitfehler erzeugen!*

### 12.2.13 HtmlTable

Die nächsten drei Controls ermöglichen Ihnen die Kontrolle von HTML-Tabellen (bzw. einzelnen Teilen dieser Tabellen). Über das `HtmlTable`-Control haben Sie die Möglichkeit auf das `<table>`-Tag Kontrolle auszuüben.

### 12.2.14 HtmlTableCell

Mit der `HtmlTableCell`-Control können Sie auf mehrere Tabellenzellen serverseitig die Kontrolle ausüben. Dies betrifft die beiden HTML-Tags `<td>` und `<th>`.

### 12.2.15 HtmlTableRow

Wenn Sie auf das Tag `<tr>` serverseitig zugreifen wollen, dann müssen Sie dies mit den Eigenschaften und Methoden des `HtmlTableRow`-Controls durchführen.

### 12.2.16 HtmlTextArea

Für die Kontrolle eines mehrzeiligen Texfeldes, das über das `<textarea>`-Tag deklariert ist, stehen Ihnen die Methoden und Eigenschaften des `HtmlTextArea`-Controls zur Verfügung.

Dieses Control stellt beispielsweise Eigenschaften für die Festlegung der Größe (`Cols` für die Breite und `Rows` für die Höhe), aber auch die Möglichkeit das Aussehen über CascadingStyleSheets (CSS) zu bestimmen.

Hier ein kleines Beispiel für eine `HtmlTextArea`, deren Größe wir festlegen und deren Eingabe wir dann nach Druck auf den Sende-Button ausgeben. Der Inhalt ist mit einem Text fest vorbelegt – allerdings nicht serverseitig.

Über die `Value`-Eigenschaft können Sie zum einen die eingegebenen Daten auslesen, aber auch das Textfeld mit selbst definierten Ausdrücken vorbelegen.

```
<%@ Page Language="cs" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
```

```
  textarea.Cols = 15;
  textarea.Rows = 10;
  a.Text = "Textfeld <br>";
}
void Senden(Object o, EventArgs e)
{
  a.Text += textarea.Value + " <br>";
}
</script>
<html>
<head><title>Textfeld-Funktionen</title></head>
<body>
<asp:Label Id="a" runat="server" />
<form runat="server">
<textarea Id="textarea" runat="server">Bitte geben Sie einen Text ein.
</textarea>
<button Id="S" runat="server" OnServerClick="Senden">
Senden</button>
</form>
</body>
</html>
```

*Listing 12.9:* `<textarea>`*-Elemente (textarea.aspx)*

# 13 Web Controls

Web Controls haben im ursprünglichen Konzept von ASP bereits eine Rolle gespielt. Sie sind sehr bedeutend und daher ein eigenes Kapitel für einen Überblick wert.

Wir haben Ihnen in den vorhergehenden Beispielen übrigens bereits laufend ein Web Control untergeschoben, das eine Textausgabe macht.

Hier ist noch einmal ganz kurz dieses Web Control vorgestellt:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Eine Textausgabe.";
}
</script>
<html>
<head><title>Textausgabe</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 13.1: Eine einfache Textausgabe (textausgabe_kurz.aspx)*

Hier sehen Sie, wie einfach Sie Web Controls erkennen können. Die Deklaration erfolgt im HTML-Teil mit dem nachfolgenden Tag

```
<asp:XXX Id="a" runat="server" />
```

XXX steht hierbei für das Web Control, das repräsentiert werden soll. In unserem Beispiel also das Web Control `Label`.

Web Controls sind einerseits recht ähnlich zu HTML Controls, die wir Ihnen im vorherigen Kapitel näher gebracht haben, andererseits sind sie weitaus mächtiger und ein wenig weiter variabel als HTML Controls. Mit Web Controls kann die HTML-Repräsentation eines Controls aus mehreren Elementen bestehen, während bei HTML Controls jedes Control immer für ein Element steht. Im weiteren Verlauf werden Sie sehen, was hiermit gemeint ist.

Zunächst werden wir Ihnen einen kurzen Überblick über alle Web Controls geben um dann in den jeweiligen Unterkapiteln detailliert auf Eigenschaften und Methoden einzugehen. Jede Behandlung eines der Controls und seiner Methoden enthält ein kurzes Beispiel, das Sie auch in die Verwendung der Komponenten im praktischen Gebrauch einführen soll.

Nachfolgend werden wir Ihnen erst einmal die den Web Controls zugeordneten Standardeigenschaften vorstellen.

## 13.1 Web Control-Ereignisse

Auch für Web Controls gibt es eine ganze Reihe von Ereignissen, die innerhalb des Namespaces deklariert sind.

Diese sind in der nachfolgenden Tabelle kurz aufgelistet.

| Ereignis | Beschreibung |
|---|---|
| DataBinding | Verknüpfung eines Web Controls mit einer Datenquelle |
| Disposed | Löschen eines Controls aus dem Speicher |
| Init | Initialisierung eines Controls |
| Load | Integration des Controls in das Page-Objekt |
| PreRender | Beginn des Renderns des Page-Objekts |
| Unload | Entfernen des Controls aus dem Speicher |

*Tabelle 13.1: Ereignisse von Web Controls*

Alle weiteren Ereignisse sind bei den einzelnen Klassen kurz aufgeführt.

## 13.2 Web Control Standardeigenschaften

Mit Hilfe der nachfolgenden Eigenschaften werden grundlegende Elemente für alle Web Controls festgelegt. Für die Controls `Literal`, `PlaceHolder`, `Repeator` und `Xml` gelten diese Standardeigenschaften allerdings nicht.

### 13.2.1 AccessKey

Mittels dieser Eigenschaft können Sie einen so genannten Shortcut einer Web Control zuordnen. Ein Shortcut ist dabei eine beliebige Taste, die gemeinsam mit der Taste `Alt` gedrückt wird.


*Wenn der Browser älter als der IE 4 ist, dann steht Ihnen diese Funktionalität nicht zur Verfügung.*

### 13.2.2 Attributes

Die Liste, die über die Kollektion Attributes zur Verfügung gestellt wird, stellt alle Attribute bereit, die innerhalb des Öffnungstags durch den Programmierer deklariert wurden und keine Standardattribute des Web Controls darstellen.

### 13.2.3 BackColor

Mittels dieser Eigenschaft können Sie für Web Controls, die sich wie Tabellen verhalten, eine Hintergrundfarbe festlegen. Für die Benennung der Farbe haben Sie entweder die Möglichkeit, diese als sechsstellige Hexadezimalzahl anzugeben oder alternativ auch die Standard-HTML-Bezeichner für Farben zu verwenden (z.B.: green).

### 13.2.4 BorderColor

Diese Eigenschaft gibt Ihnen die Möglichkeit eine Rahmenfarbe (analog zur Hintergrundfarbe *BackColor*) festzulegen. Beachten Sie, dass diese Eigenschaft, da eine Rahmenfarbe nicht dem normalen HTML-Standard entspricht, nur mit Microsoft-Browsern Wirkung zeigt.

### 13.2.5 BorderWidth

Sie können eine Rahmenbreite für Ihre Web Controls definieren, die Sie über die Eigenschaft angeben. Die Einheit für die Rahmenbreite ist Pixel.

### 13.2.6 BorderStyle

Für einen Tabellenrahmen können Sie außerdem einen Zeichenstil setzen. Als Parameter stehen Ihnen dabei die folgenden Möglichkeiten zur Verfügung.

| Rahmenstil | Bedeutung |
|---|---|
| NotSet | Kein besonderer Rahmenstil gesetzt |
| None | Kein Rahmenstil gesetzt |
| Dotted | Gepunkteter Rahmen |
| Dashed | Gestrichelter Rahmen |
| Solid | Durchgängiger Rahmen |
| Double | Doppelter Rahmen |
| Groove | Vertiefter Rahmen |
| Ridge | Erhöhter Rahmen |
| Inset | Abgesenkter Rahmen |
| Outset | Erhöhter Rahmen |

*Tabelle 13.2: Verschiedene Rahmenstile und ihre Bedeutung*

Experimentieren Sie ein wenig um den besten Rahmenstil herauszubekommen.



### 13.2.7 CssClass

Mittels dieser Eigenschaft können Sie eine Klasse, die ein Cascading Style Sheet deklariert, einem Web Control zuordnen.

### 13.2.8 Style

Die `Style`-Eigenschaft bildet eine Kollektion von Textbausteinen, die als CSS-Styles gerendert werden.

Falls Sie eine der Methoden verwenden, die einen Style-Wert setzt, wird der Wert auch in dieser Eigenschaft geändert.

### 13.2.9 Enabled

Mittels dieser Eigenschaft können Sie ein Web Control aktiv oder inaktiv schalten. Inaktiv schalten bedeutet hierbei nicht, dass das Web Control nicht mehr sichtbar ist. Es wird nur anders dargestellt, und man kann mit ihm keine Aktionen auf der Clientseite durchführen.

### 13.2.10 Font

Mittels dieser Eigenschaft können Sie Zeichensatzinformationen für das zu deklarierende Web Control festlegen.

### 13.2.11 ForeColor

Über die Eigenschaft `ForeColor` legen Sie die Vordergrundfarbe des Web Controls fest.

### 13.2.12 Height

Mit dieser Eigenschaft können Sie die Höhe eines Web Controls festlegen.

### 13.2.13 TabIndex

Mittels dieser Eigenschaft können Sie die Reihenfolge festlegen, mit der der Fokus der einzelnen Web Controls über die Tabulatortaste verschoben werden kann.

### 13.2.14 ToolTip

Mit der `ToolTip`-Eigenschaft können Sie einen Text festlegen, der angezeigt wird, wenn der Mauszeiger sich über dem Web Control befindet und eine Zeit lang nicht bewegt wird.


*Diese Möglichkeit funktioniert übrigens ausschließlich mit dem Internet Explorer. Bei allen anderen Browsern wird diese Eigenschaft ignoriert.*

### 13.2.15 Width

Mittels dieser Methode legen Sie die Breite des Web Controls fest.

Nun kommen wir zu den eigentlichen Web Controls.

## 13.3 AdRotator Web Control

Mittels dieses Web Controls können Sie Werbebanner einfach in Ihre Webseiten einbauen. Über die in diesem Control bereitgestellten Mechanismen lassen sich beispielsweise die Werbebanner fortlaufend rollieren oder austauschen.

Die eigentlichen Werbeinformationen werden dabei in einem eigenständigen XML-File gespeichert. Dieses XML-File enthält eine Liste von bestimmten Attributen wie das Werbebild (`ImageUrl`), einen Link (`NavigateUrl`), einen Indikator, der die relative Häufigkeit des Werbe-

banners anzeigt (`Impressions`), Schlüsselwort (`Keyword`) sowie einen alternativen Text, der angezeigt wird, wenn das Bild nicht angezeigt werden kann (`AlternateText`).

Einen beispielhaften Aufbau eines solchen XML-Files können Sie dem nachfolgenden Codeabschnitt entnehmen:

```
<Advertisements>
<Ad>
<ImageUrl>Bild1.jpg</ImageUrl>
<NavigateUrl>http://www.nitty-gritty.de</NavigateUrl>
<Impressions>1</Impressions>
<Keyword>Thema1</Keyword>
<AlternateText>Nitty Gritty</AlternateText>
</Ad>
<Ad>
<ImageUrl>Bild2.jpg</ImageUrl>
<NavigateUrl>http://www.addison-wesley.de</NavigateUrl>
<Impressions>1</Impressions>
<Keyword>Thema2</Keyword>
<AlternateText>Addison Wesley</AlternateText>
</Ad>
</Advertisements>
```

Der Aufruf eines solchen Werbe-XML-Files würde über das nachfolgende Web Control erfolgen:

```
<asp:AdRotator Id="Werbung" runat="server"
AdvertisementFile="Werbefile.xml"/>
```

Im Control ist das Ereignis `AdCreated` enthalten. Dieses Ereignis tritt jedes Mal auf, wenn ein Durchlauf durch die Webebanner gestartet wird (bevor das Rendering startet).

## 13.4 Button Web Control

Mittels dieses Web Controls können Sie einen Button erzeugen und verwalten.

Über die Eigenschaft `Text`, die Sie auch beim Aufruf als Parameter verwenden können, haben Sie die Möglichkeit einen Text auf den Button, den Sie erzeugen, zu schreiben.

Außerdem können Sie über den Eventhandler, der durch das `Click`-Ereignis (durch `OnClick`) ausgelöst wird, erkennen, wann der erzeugte Button gedrückt wurde, und dieses Ereignis dann serverseitig abarbeiten. Als weiteres Ereignis steht Ihnen `OnCommand` zur Verfügung. Sie können einem Button mit der Eigenschaft `CommandName` ein »Kommando« zuordnen. Dieses Kommando wird ausgeführt, falls der entsprechende Button gedrückt wurde.

Genaueres geht sicherlich aus dem nachfolgenden Beispielprogramm hervor.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Buttons: <br />";
}
void gedrueckt1(Object o, EventArgs e)
{
  a.Text += "Button 1 gedrueckt.<br />";
  knopf1.Enabled = false;
  knopf2.Enabled = true;
}
void gedrueckt2(Object o, EventArgs e)
{
  a.Text += "Button 2 gedrueckt.<br />";
  knopf2.Enabled = false;
  knopf1.Enabled = true;
}
</script>
<html>
<head><title>Button</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:Button Id="knopf1" Text="Senden 1"
OnClick="gedrueckt1" runat="server" />
<asp:Button Id="knopf2" Text="Senden 2"
OnClick="gedrueckt2" runat="server" />
</form>
</body>
</html>
```

*Listing 13.2: Schaltflächen (button.aspx)*

Beachten Sie, dass dieses Web Control immer innerhalb eines `<form runat="server"></form>` stehen muss – wie alle Formular-Web Controls.

Weiterhin haben wir hier – so ganz nebenbei – die `Enabled`-Eigenschaft und ihren Effekt demonstriert.

Ansonsten sehen Sie, dass es eine große Ähnlichkeit zu HTML Controls gibt, allerdings ist die Handhabung ein wenig einfacher.

Im Control sind die Ereignisse `Click` und `Command` deklariert. Diese Ereignisse treten auf, wenn ein Button angeklickt wurde.

## 13.5 Calendar Web Control

Mittels des `Calendar` Web Controls haben Sie die Möglichkeit einfach einen Monatskalender anzuzeigen und den User einen Tag auswählen zu lassen. Sie haben weiterhin die Möglichkeit Monate vor- oder zurückzuspringen, ganz wie Sie es brauchen.

Für die Darstellung der einzelnen Kalenderelemente steht Ihnen eine ganze Reihe von Eigenschaften zur Verfügung. So können Sie die Vordergrund- (`ForeColor`) und Hintergrundfarben (`BackColor`) der einzelnen Elemente variieren oder den ausgewählten Tag markieren.

Als Manipulationsmöglichkeit für die einzelnen Kalenderelemente können Sie beispielsweise die Erscheinungsform der Kopfzeile (`ShowDayHeader`, `ShowTitle`, `ShowNextPrevMonth`), der ersten Wochentage (`FirstDayOfWeek`), des einzelnen Tages (`DayNameFormat`), des nächsten Monats (`NextMonthText`), und vieles andere definieren.

Nachfolgend sehen Sie ein kleines Beispiel – falls Sie nicht durch die Farbwahl erblinden.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Kalender: <br />";
}
</script>
<html>
<head><title>Button</title></head>
<body>
```

```
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:Calendar Id="kalender" runat="server">
<DayStyle BackColor="lightgreen"></DayStyle>
<TodayDayStyle ForeColor="red"></TodayDayStyle>
<WeekendDayStyle BackColor="green"></WeekendDayStyle>
<OtherMonthDayStyle ForeColor="blue">
</OtherMonthDayStyle>
</asp:Calendar>
</form>
</body>
</html>
```

*Listing 13.3: Kalender (kalender.aspx)*

Sie ersparen sich mit diesem Web Control sicherlich einiges an Programmierarbeit, allerdings sind die – dennoch zahlreichen – Variationsmöglichkeiten etwas eingeschränkt.

Im `Calendar`-Control ist eine ganze Reihe von Ereignissen deklariert. Mit dem Ereignis `DayRender` haben Sie die Möglichkeit auf jede Erzeugung eines Tages einzuwirken. Das Ereignis `SelectionChanged` tritt auf, wenn ein Tag, eine Woche oder ein Monat ausgewählt wird. Das Ereignis `VisibleMonthChanged` tritt auf, wenn über die Monatsnavigation der aktive Monat gewechselt wird.

## 13.6 CheckBox Web Control

Mit dem `CheckBox` Web Control haben Sie die Möglichkeiten Check Boxen zu deklarieren und auszuwerten.

Dabei geben Sie mit dem Parameter `Text` an, welcher Text neben der Check Box ausgegeben werden soll (dieser Parameter steht Ihnen auch als Eigenschaft zur Verfügung) – dies kann sowohl clientseitig als auch serverseitig geschehen, wie im nachfolgenden Beispiel demonstriert. Sie haben natürlich auch weitere verschiedene Gestaltungsmöglichkeiten, auf die wir aber nicht im Detail eingehen wollen.

Beispielsweise können Sie über den Parameter `Checked` mittels `true` oder `false` festlegen, ob die Check Box bei der erstmaligen Anzeige angeklickt oder gelöscht sein soll.

Hier nun das Beispiellisting:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Checkboxen: <br />";
}
void gedrueckt(Object o, EventArgs e)
{
  if (box.Checked) {
    a.Text += "Check Box gesetzt.<br />";
  }
  else {
    a.Text += "Check Box nicht gesetzt.<br />";
  }
}
</script>
<html>
<head><title>Check Box</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:CheckBox Id="box" Text="Ich will!" runat="server" />
<asp:Button Id="knopf" Text="Senden"
OnClick="gedrueckt" runat="server" />
</form>
</body>
</html>
```

*Listing 13.4: Checkboxen (checkbox.aspx)*

Dieses Listing erzeugt eine Check Box, die, wenn das Formular gesendet wird, serverseitig überprüft wird, ob sie gesetzt ist oder nicht. Dies erfolgt über die Eigenschaft `Checked`. Die übrigen Elemente dürften Ihnen bereits bekannt sein.

Im Control ist das Ereignis `CheckedChanged` deklariert. Dieses Ereignis tritt auf, wenn eine Check Box an- oder ausgeklickt wurde.

## 13.7 CheckBoxList Web Control

Eine Variante des `CheckBox`-Web Controls stellt das `CheckBoxList`-Control dar. Es ermöglicht Check Boxen, die thematisch zusammengehören, in einer Liste zu gruppieren und auch über die Ergebnisse gruppierte Auswertungen durchzuführen.

Die Gruppierung erfolgt hierbei über das Element `ListItem`.

Im Gegensatz zu `CheckBox` wird hier ein »Ausgewählt« allerdings über die `Selected`-Eigenschaft angezeigt. Ansonsten stehen Ihnen auch hier die üblichen Verdächtigen als Eigenschaften zur Verfügung: `Text` für die Beschriftung der Check Boxen, `Value` für die Werte, die an den Server beim Setzen übergeben werden, und vieles andere mehr.

Ein typisches Programmelement einer Liste sehen Sie nachfolgend angedeutet:

```
<asp:CheckBoxList Id="Identifikator" runat="server">
<asp:ListItem Text="Text1" />
</asp:CheckBoxList>
```

So haben Sie die Möglichkeit, die Check Boxen zu gruppieren (standardmäßig sind sie auf eine Gruppierung in Tabellenform eingestellt) oder einfach als »Fließtext« darzustellen. Diese Möglichkeit bietet sich Ihnen über die Eigenschaft `RepeatLayout`, die als zulässige Werte `Table` oder `Flow` erlaubt.

Hier ein kleines Listing, das die Verwendung der `CheckBoxList` anschaulich macht.

```
<%@ Page Language="C#" %>
<script runat="server">
void gedrueckt(Object o, EventArgs e)
{
  box.CellPadding = 4;
  box.CellSpacing = 4;
  a.Text = "Check Boxen: <br />";
  for (int i = 0; i < box.Items.Count;i++) {
    if (box.Items[i].Selected == true) {
      a.Text += box.Items[i].Text;
      a.Text += ": gesetzt.<br />";
    }
    else {
      a.Text += box.Items[i].Text;
```

```
      a.Text += ": nicht gesetzt.<br />";
    }
  }
}
</script>
<html>
<head><title>Check Box-Liste</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server"/>
<asp:CheckBoxList Id="box"
RepeatColumns="3"
runat="server">
<asp:ListItem Text="Box 1" />
<asp:ListItem Text="Box 2" />
<asp:ListItem Text="Box 3" />
<asp:ListItem Text="Box 4" />
<asp:ListItem Text="Box 5" />
<asp:ListItem Text="Box 6" />
<asp:ListItem Text="Box 7" />
</asp:CheckBoxList>
<asp:Button Id="knopf" Text="Senden" OnClick="gedrueckt" runat="server" />
</form>
</body>
</html>
```

*Listing 13.5: Checkbox-Liste (checkboxlist.aspx)*

Im Control ist das Ereignis `SelectedIndexChanged` deklariert. Dieses Ereignis tritt auf, wenn eine Check Box der Liste angeklickt wurde.

## 13.8 DataGrid Web Control

Mit diesem Web Control werden Daten aus einer Datenquelle – wie einer Datenbank oder einer Datei, in eine Tabelle übernommen und angezeigt.

Da dieses Control eng mit den Datenbankzugriffen verknüpft ist, gehen wir später in Kapitel 17.4 im Detail auf dieses Web Control ein.

## 13.9 DataList Web Control

Auch das DataList-Control widmet sich insbesondere der Verarbeitung und Anzeige von Daten aus einer Datenbank – in diesem Fall lassen sich beispielsweise Ausgaben aus Tabellenspalten mit diesem Control je nach Bedarf anpassen und verändern.

Auch hierzu später mehr im Bereich Datenbankzugriffe ab Kapitel 17.

## 13.10 DropDownList Web Control

Auch die DropDownList-Controls haben ihren Weg aus den HTML Controls zu den Web Controls gefunden. Sie lassen sich analog zur CheckBoxList anlegen und verwalten.

Die einzelnen Elemente der Liste werden über die ListItem Tags angelegt. Ein beispielhaftes Fragment demonstriert Ihnen den Aufbau eines DropDownList Web Controls:

```
<asp:DropDownList Id="box" runat="server">
<asp:ListItem Text="Wert 1" value=1 />
...
</asp:DropDownList>
```

Die DropDownList stellt eine Liste zur Verfügung, aus der sich jeweils nur ein Element auswählen lässt. Wollen Sie eine Liste mit multiplen Auswahlmöglichkeiten erzeugen, müssen Sie das Control ListBox verwenden.

Sie haben die Selected-Eigenschaft zur Verfügung, um zu erkennen, welches Element angewählt wurde. Jedes Element kann mit einem Text und einem zugehörigen Value-Tag versehen werden, so dass Ihnen eine Verarbeitung von Text und Value möglich ist.

Außerdem steht Ihnen wieder eine Reihe von Parametrisierungsmöglichkeiten für die Gestaltung der HTML-Elemente des DropDownList-Controls zur Verfügung.

Im Control ist das Ereignis SelectedIndexChanged deklariert. Dieses Ereignis tritt auf, wenn die Auswahl der DropDownList verändert wird.

Im Beispiel haben wir von einigen dieser Gestaltungsmöglichkeiten Gebrauch gemacht.

```
<%@ Page Language="C#" %>
<script runat="server">
void gedrueckt(Object o, EventArgs e)
{
box.Width = 100;
box.ToolTip = "Waehlen Sie Parameter aus!";
a.Text = "DropDown-Listen: <br />";
for (int i = 0; i < box.Items.Count;i++) {
  if (box.Items[i].Selected == true) {
    a.Text = a.Text + box.Items[i].Text;
    a.Text = a.Text + ": gesetzt.<br />";
    }
  else {
    a.Text = a.Text + box.Items[i].Text;
    a.Text = a.Text + ": nicht gesetzt.<br />";
    }
  }
}
</script>
<html>
<head><title>Dropdown-Liste</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:DropDownList Id="box" BackColor="LightGray" runat="server">
<asp:ListItem Text="Wert 1" value=1 />
<asp:ListItem Text="Wert 2" value=2 />
<asp:ListItem Text="Wert 3" value=3 />
<asp:ListItem Text="Wert 4" value=4 />
<asp:ListItem Text="Wert 5" value=5 />
<asp:ListItem Text="Wert 6" value=6 />
<asp:ListItem Text="Wert 7" value=7 />
</asp:DropDownList>
<asp:Button Id="knopf" Text="Senden"
OnClick="gedrueckt" runat="server" />
</form>
</body>
</html>
```

*Listing 13.6: Dropdown-Listen (dropdownlist.aspx)*

## 13.11 HyperLink Web Control

Mit dem `HyperLink`-Control haben Sie die Möglichkeit die Erstellung und Verwaltung von Hyperlinks anstatt durch HTML Controls auch durch Web Controls durchführen zu lassen.

Die Verwendung gestaltet sich einfach z.B. durch die Einbindung des Tags

```
<asp:HyperLink Id="hl" runat="server">
</asp:HyperLink>
```

Im Gegensatz zu den meisten anderen Web Controls wird durch das `HyperLink`-Control kein `PostBack`-Ereignis ausgelöst, sondern einfach auf die neue Seite weitergeleitet.

Eine Verknüpfung mit Datenbanken oder ein dynamischer Aufbau der URL ermöglicht Ihnen eine große Vielzahl von Variationsmöglichkeiten, die dieses einfache Control sehr nützlich im Tagesgebrauch machen.

Auch Bilder als Links lassen sich über dieses Control einfach und elegant einbinden.

## 13.12 Image Web Control

Mit dem `Image`-Control können Sie ein Bild anzeigen. Sie können einige Manipulationen in der Anzeige vornehmen. Sie haben die Möglichkeit festzulegen, ob das Bild linksbündig, rechtsbündig oder zentriert angezeigt wird, aber auch wenn es nicht angezeigt werden kann, weil der Browser das Bildformat nicht unterstützt, welcher alternative Text angezeigt werden soll.

Ansonsten sind die Möglichkeiten dieses Controls aber beschränkt. Es werden keine Ereignisse ausgelöst, wenn Sie z.B. mit der Maus auf das Bild klicken, Ihnen stehen damit nur sehr eingeschränkte Manipulationsmöglichkeiten zur Verfügung.

Wollen Sie eine interaktive Verwendung eines Bildes, dann benutzen Sie am besten das `ImageButton`-Control, das Ihnen genau diese Möglichkeit bietet, da es die Standard-Ereignisse auslösen kann.

Hier noch die Einbindung dieses Controls:

```
<asp:Image Id="i" ImageUrl="Url_des_Bilds" runat="server"/>
```

## 13.13 ImageButton Web Control

Mit dem `ImageButton`-Control haben Sie die Möglichkeit ein Bild als einen Button darzustellen. Die Einbindung eines solchen Controls erfolgt mit dem nachfolgenden Codeblock:

```
<asp:ImageButton Id="ib" runat="server"/>
```

Um den Ort des Bildes zu spezifizieren müssen Sie entweder im Deklarationsteil oder aber auch im Programmteil die URL des Bildes angeben. Mit der oben genannten `Id` würden Sie z.B. im C#-Codeblock den folgenden Code die URL des Bildes festlegen.

```
ib.ImageUrl="Url_des_Bilds"
```

Im Control sind die Ereignisse `Click` und `Command` deklariert. Diese Ereignisse treten auf, wenn der `ImageButton` gedrückt wird. Beim `Command`-Ereignis können Sie weitere Parameter über die Eigenschaften `CommandName` (die eine Unterscheidung der `ImageButtons` auf einer Seite möglich macht) und `CommandArgument` festlegen.



## 13.14 Label Web Control

Bereits in der Einführung zu diesem Kapitel ist Ihnen das `Label`-Control über den Weg gelaufen. Mit ihm lassen sich einfach Texte, die serverseitig generiert und verarbeitet wurden, ausgeben. Die Texte, die durch ein `Label`-Control generiert werden, sind statisch und lassen sich nicht durch den Benutzer editieren.

Die Einbindung in den HTML-Code erfolgt über

```
<asp:Label Id="a" runat="server" />
```

Wir haben hiervon bereits sehr ausführlich Gebrauch gemacht. Sie haben dann die Möglichkeit über die Text-Eigenschaft dieses Label mit Inhalten zu befüllen.

Im nachfolgenden Beispiel haben wir ein wenig mit den Eigenschaften, die dieses Label bereitstellt, die Ausgabe des Textes verändert. Dies haben wir über ein neues Stylesheet gemacht, das durch die Tags `<style></style>` deklariert wird.

Spielen Sie ein wenig mit den Möglichkeiten, die sich Ihnen über die Änderung von CSS bieten! Es lohnt sich.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Eine Textausgabe.";
  a.CssClass = "CssStyle";
}
</script>
<style>
.CssStyle
{
  font: 12pt arial;
  font-weight:900;
  color:darkblue;
}
</style>
<html>
<head><title>Label</title></head>
<body>
<asp:Label Id="a" runat="server" />
</body>
</html>
```

*Listing 13.7: Textausgabe mit* `Label` *(label.aspx)*

## 13.15 LinkButton Web Control

Mit diesem Control steht Ihnen eine Vermischung des `Button`-Controls mit dem `HyperLink`-Control zur Verfügung.

Das Aussehen eines `LinkButton`-Controls stellt sich analog zu einem `HyperLink`-Control dar, allerdings ist sein Verhalten das eines `Button`-Controls.

Damit stehen Ihnen auch die Ereignisse, die ein Button unterstützt, für Links zur Verfügung. Sie können die Ereignisbehandlung `OnClick` wie auch `Command` verwenden und darüber hinaus das `PostBack`-Ereignis.

Des Weiteren gibt es die Gestaltungseingenschaften der Klasse `WebControl`, die wir Ihnen bereits vorgestellt haben.

*Sie müssen, um dieses Control nutzen zu können, JavaScript auf dem Client-Browser aktiviert haben.*

Nachfolgend stellen wir `Button` und `LinkButton` in einem Beispiel gegenüber.

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  a.Text = "Linkbuttons: <br />";
}
void gedrueckt1(Object o, EventArgs e)
{
  a.Text += "LinkButton gedrueckt.<br />";
  knopf1.Enabled=false;
  knopf2.Enabled=true;
}
void gedrueckt2(Object o, EventArgs e)
{
  a.Text += "Button gedrueckt.<br />";
  knopf2.Enabled=false;
  knopf1.Enabled=true;
}
</script>
<html>
<head><title>Linkbutton</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:LinkButton Id="knopf1" Text="Senden 1"
OnClick="gedrueckt1" runat="server" />
<asp:Button Id="knopf2" Text="Senden 2"
OnClick="gedrueckt2" runat="server" />
</form>
</body>
</html>
```

*Listing 13.8: Link-Schaltfläche (linkbutton.aspx)*

## 13.16 ListBox Web Control

Mit dem `ListBox`-Control haben Sie die Möglichkeit eine Drop-Down-Liste in einer Listbox zu erstellen. Die Ähnlichkeit zum `DropDownList`-Control ist daher sehr groß. Allerdings haben Sie mit dem `ListBox`-Control eine Reihe von zusätzlichen Manipulationsmöglichkeiten. Sie können beispielsweise Mehrfachselektionen zulassen.

Diese steuern Sie über den Parameter `SelectionMode`. Als Werte sind hierbei `Single` (ein Element kann ausgewählt werden) und `Multiple` (mehrere Elemente sind zulässig) möglich.

Die Anzahl der anzuzeigenden Zeilen steuern Sie über den Parameter `Rows`.

Wollen Sie einen Parameter bereits vorselektieren, so geschieht dies über den Parameter `selected`, der entweder auf `true` oder `false` gesetzt werden kann. Dieser Parameter kann bei jedem `ListItem` angegeben werden.

Im Control ist das Ereignis `SelectedIndexChanged` deklariert. Dieses Ereignis tritt auf, wenn die Auswahl der `ListBox` verändert wird.

Ein möglicher Aufruf hat also die nachfolgende Form:

```
<asp:ListBox Id="box"
SelectionMode="Multiple"
runat="server">
<asp:ListItem Text="Wert 1" Value="1" />
...
</asp:ListBox>
```

Nachfolgend ein Beispiel, das die Verwendung des `ListBox` Web Controls demonstriert.

```
<%@ Page Language="C#" %>
<script runat="server">
void gedrueckt(Object o, EventArgs e)
{
box.Width = 200;
box.ToolTip = "Waehlen Sie Parameter aus!";
a.Text = "Listbox-Listen: <br />";
for (int i = 0; i < box.Items.Count;i++) {
  if (box.Items[i].Selected == true) {
    a.Text = a.Text + box.Items[i].Text;
```

```
    a.Text = a.Text + ": gesetzt.<br />";
    }
  else {
    a.Text = a.Text + box.Items[i].Text;
    a.Text = a.Text + ": nicht gesetzt.<br />";
    }
  }
}
</script>
<html>
<head><title>Dropdown-Liste</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server"/>
<asp:ListBox Id="box" SelectionMode="Multiple" Rows="3"
runat="server">
<asp:ListItem Text="Wert 1" value=1 />
<asp:ListItem Text="Wert 2" value=2 />
<asp:ListItem Text="Wert 3" value=3 />
<asp:ListItem Text="Wert 4" value=4 />
<asp:ListItem Text="Wert 5" value=5 />
<asp:ListItem Text="Wert 6" value=6 />
<asp:ListItem Text="Wert 7" value=7 />
</asp:ListBox>
<asp:Button Id="knopf" Text="Senden" OnClick="gedrueckt" runat="server" />
</form>
</body>
</html>
```

*Listing 13.9: Listbox (listbox.aspx)*

## 13.17 Literal Web Control

`Literal`-Controls sind `Label`-Controls sehr ähnlich. Ein Zugriff erfolgt über die nachfolgende Syntax:

```
<asp:Literal Id="l" Text="XX" runat="server" />
```

Der Unterschied zu `Label`-Controls ist, dass Sie einem `Literal` keine Styles zuordnen können.

Wir verzichten in diesem Fall auf ein Beispiel.

## 13.18 Panel Web Control

Das `Panel`-Control ermöglicht es Web Controls zusammenzufassen und zu gruppieren. Werden Attribute eines Panels geändert, so wirkt sich dies auf alle innerhalb des Panels deklarierten Web Controls aus.

Eine praktische Möglichkeit zur Anwendung ist, diese Art Gruppen von Web Controls inaktiv zu schalten. Die Tags zur Deklaration eines Panels haben die nachfolgende Form:

```
<asp:Panel Id="Panel" runat="server">
Hier können Sie weitere Controls deklarieren,
die zum Panel gehören.
</asp:Panel>
```

## 13.19 RadioButton Web Control

Das `RadioButton` Web Control hat als Gestaltungsmöglichkeiten exakt dieselben wie z.B. das `CheckBox` Web Control. Sie können über den Parameter `checked` steuern, ob der Radio Button ausgewählt sein soll.

Einen einzelnen Radio Button anzuzeigen macht nicht immer Sinn. In den meisten Fällen wollen Sie voraussichtlich mehrere Buttons zu Gruppen zusammenfassen. Dies erfolgt über die Eigenschaft `GroupName`, die Sie auch als Parameter angeben können.

Einfacher können Sie Gruppen von Radiobuttons erzeugen, wenn Sie das Web Control `RadioButtonList` verwenden. Es schränkt aber die Gestaltungsmöglichkeiten ein wenig ein.

Sie können mit dem Parameter `TextAlign` auch die Ausrichtung des Radio Buttons festlegen und haben ansonsten noch alle üblichen aus den Web Control-Eigenschaften abgeleiteten Gestaltungsmöglichkeiten, wie beispielsweise die Festlegung von Rahmen und Hintergrundfarbe.

Im Control ist das Ereignis `CheckedChanged` deklariert. Dieses Ereignis tritt auf, wenn die Auswahl des `RadioButton` verändert wird.

Die Syntax für den Aufruf des `RadioButton` Web Controls ist die nachfolgende:

```
<asp:RadioButton Id="ButtonID" Text="ButtonText"
GroupName="ButtonGruppe" runat="server" />
```

Hier ein kleines Beispiel, das eine Gruppe von Radio Buttons anlegt und anzeigt, welcher ausgewählt wurde.

```
<%@ Page Language="C#" %>
<script runat="server">
void gedrueckt(Object o, EventArgs e)
{
a.Text = "Radio Buttons: <br />";
if (b1.Checked == true) {
  a.Text = a.Text + "Button 1 gesetzt.<br />";
  }
if (b2.Checked == true) {
  a.Text = a.Text + "Button 2 gesetzt.<br />";
  }
if (b3.Checked == true) {
  a.Text = a.Text + "Button 3 gesetzt.<br />";
  }
}
</script>
<html>
<head><title>Radio Button</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:RadioButton Id="b1" Text="Button 1"
GroupName="Gruppe" runat="server" />
<asp:RadioButton Id="b2" Text="Button 2"
GroupName="Gruppe" runat="server" />
<asp:RadioButton Id="b3" Text="Button 3"
GroupName="Gruppe" runat="server" />
<asp:Button Id="knopf" Text="Senden"
OnClick="gedrueckt" runat="server" />
</form>
</body>
</html>
```

*Listing 13.10: Radiobuttons (radiobutton.aspx)*

## 13.20 RadioButtonList Web Control

Mit diesem Web Control können Sie eine Gruppe von Radio Buttons darstellen. Die Syntax und die Verwendung der Parameter und Eigenschaften ist sehr ähnlich zur Syntax der `CheckBoxList`-Syntax.

Die Variationsmöglichkeiten sind nicht ganz so groß wie in dem Fall, wenn Sie gruppierte `RadioButton`-Controls verwenden würden, dafür ist die Syntax kürzer und übersichtlicher in der Programmierung.

Im Control ist das Ereignis `SelectedIndexChanged` deklariert. Dieses Ereignis tritt auf, wenn die Auswahl der `RadioButtonList` verändert wird.

Auch hier ordnen Sie die Liste über das `ListItem`-Tag an. Ein Aufruf des Controls hat die nachfolgende Form:

```
<asp:RadioButtonList Id="ButtonID" runat="server">
<asp:ListItem Text="ButtonText" />
...
</asp:RadioButtonList>
```

Hier ein Beispiel, das genau wie das Beispiel zum `RadioButton`-Control funktioniert.

```
<%@ Page Language="C#" %>
<script runat="server">
void gedrueckt(Object o, EventArgs e)
{
a.Text = "Radio Buttons: <br />";
for (int i = 0; i < rl.Items.Count;i++) {
  if (rl.Items[i].Selected == true) {
    a.Text = a.Text + rl.Items[i].Text;
    a.Text = a.Text + ": gewaehlt.<br />";
    }
  }
}
</script>
<html>
<head><title>Radio Button</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
<asp:RadioButtonList Id="rl" runat="server">
<asp:ListItem Text="Button 1" />
<asp:ListItem Text="Button 2" />
<asp:ListItem Text="Button 3" />
</asp:RadioButtonList>
<asp:Button Id="knopf" Text="Senden"
OnClick="gedrueckt" runat="server" />
```

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

*Listing 13.11: Eine Radiobutton-Liste (radiobuttonlist.aspx)*

## 13.21 Repeater Web Control

Dieses Control stellt weitere Möglichkeiten zur Verfügung, Listen von Daten anzuzeigen und strukturiert auszugeben.

Da in diesem Control die Freiheitsgrade hoch sind, ist seine Anwendung etwas aufwändig, aber dafür für Ihre individuellen Gestaltungsmöglichkeiten sicherlich sehr interessant.

## 13.22 Table Web Control

Mit dem `Table`-Control können Sie eine Tabelle anzeigen und programmseitig ihre Inhalte manipulieren.



Das `Table`-Control besitzt einige Eigenschaften, die z.B. die Gitterlinien, `GridLines` mit den möglichen Ausprägungen `None`, `Horizontal`, `Vertical` und `Both` (also horizontal und vertikal) festlegen. Eine andere Eigenschaft ermöglicht das Festlegen eines Hintergrundbildes (`BackImageUrl`).

Auch die horizontale Position relativ zu den übrigen Daten kann über die Eigenschaft `HorizontalAlign` (mit den Parametern `NotSet`, `Left`, `Right`, `Center`) festgelegt werden.

Über die Eigenschaft `Rows` erhalten Sie ein Array vom Datentyp `TableRowCollection`, diese Kollektion enthält die Zeilen der Tabelle, die darüber lesbar sind (aber nicht veränderbar).

Ein Aufruf einer serverseitig erzeugten Tabelle erfolgt über die nachfolgenden Tags:

```
<asp:Table Id="Tabelle" runat="server"
GridLines="Both"
HorizontalAlign="Center">
...
</asp:Table>
```

## 13.23 TableCell Web Control

Für die Manipulation einzelner Tabellenzellen ist das `TableCell` Web Control geschaffen worden.

Hier stehen Ihnen Eigenschaften zur Verfügung, die Ihnen Informationen über die Anzahl der Spalten, über die eine Tabellenzelle reicht (`ColumnSpan`), die Zeilen, die in der Zelle zusammengefasst sind (`RowSpan`), den Textinhalt der Zelle (`Text`) oder die vertikale Ausrichtung (`VerticalAlign` mit den Parametern: `Top`, `Middle` und `Bottom`) bzw. die horizontale Ausrichtung (`HorizontalAlign` mit den Parametern: `NotSet`, `Left`, `Right`, `Center` und `Justify` – das ist Blocksatz) einer Tabellenzelle geben.

Für die Kopfzeile einer Tabelle gibt es die etwas spezialisiertere ergänzende Klasse `TableHeaderCell` (also die Elemente, die mit dem `<th>` Tag verknüpft sind. Auch für diese Zellen gibt es dieselben Eigenschaften wie für die gewöhnlichen Tabellenzellen.

Hier noch ein kleines Beispiel, wie eine Tabellenzelle mit Web Controls erzeugt wird:

```
<asp:TableCell>
...
</asp:TableCell>
```

## 13.24 TableRow Web Control

Für die Bearbeitung von ganzen Tabellenzeilen bietet sich das `TableRow`-Control an. Die mit diesem Control verknüpften Eigenschaften sind zum einen ein Array vom Datentyp `TableCellCollection`, das den Namen `Cells` hat und die Tabellenzellen enthält.

Zum anderen haben Sie Eigenschaften, die die horizontale oder die vertikale Ausrichtung festlegen (`HorizontalAlign` und `VerticalAlign`). Die Parameter dieser Eigenschaften sind dieselben wie bei der entsprechenden `TableCell`-Eigenschaft.

Eng verknüpft mit diesem Control ist die Klasse `TableCellCollection`, diese liefert Eigenschaften wie z.B. die Anzahl der Zellen in einer Tabelle (`Count`). Mit Methoden wie `Add(Zelle)` oder `Remove(Zelle)` lassen sich Zellen zu einer Kollektion hinzufügen oder entfernen.

Über die AddAt-Methode lässt sich in einer bestimmten Indexposition eine Zelle hinzufügen, über RemoveAt an einer Indexposition eine Zelle wieder entfernen.

Nachfolgend der Aufruf einer TableRow-Control:

```
<asp:TableRow>
...
</asp:TableRow>
```

Beachten Sie, dass eine Tabellenzelle immer innerhalb einer Tabellenzeile und diese innerhalb einer Tabelle angeordnet sein müssen.

## 13.25 TextBox Web Control

Mit dem TextBox-Control haben Sie die Möglichkeit auch Textboxen über Web Controls zu steuern.

Sie haben alle Formatierungsmöglichkeiten, wie Stylesheets oder Vorder- und Hintergrundfarbe, Sie können die Textbox mit einem Text vorbelegen, den der Benutzer dann ändern oder übernehmen kann.

Sie können die Textboxen in der Höhe und in der Breite unterschiedlich gestalten – kurz: Für die Eingabe von Texten haben Sie mit diesem Web Control eine Vielzahl von Möglichkeiten.

Als grundsätzliche Typen stehen Ihnen einzeilige (SingleLine), mehrzeilige (MultiLine) und Passwort-Textboxen (Password) zur Verfügung. Die Auswahl erfolgt über die Eigenschaft TextMode, die auch als Parameter verfügbar ist.

Über die Eigenschaft MaxLength steuern Sie, wie viele Zeichen maximal eingegeben werden dürfen.

Die Eigenschaften Columns und Rows stehen für die Breite und die Höhe der Anzeigen.

Im Control ist das Ereignis TextChanged deklariert. Dieses Ereignis tritt auf, wenn der Text in der TextBox verändert wird.

Ein Aufruf hat die nachfolgende Form:

```
<asp:TextBox Id="Identifikator"
TextMode="TextModus" runat="server" />
```

Hier ein kleines Beispiel mit den meisten dieser Parameter:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load(Object o, EventArgs e)
{
  tm.Rows =10;
  tm.Columns = 20;
  a.Text = "Textboxen."+ " <br />";
  a.Text += ts.Text + " <br />";
  a.Text += tm.Text + " <br />";
  a.Text += tp.Text + " <br />";
}
</script>
<html>
<head><title>Textboxen</title></head>
<body>
<form runat="server">
<asp:Label Id="a" runat="server" />
Singeline-Textbox:
<asp:TextBox Id="ts"
Text="Bitte aendern"
TextMode="SingleLine" runat="server" />
<br />
Multiline-Textbox: <br />
<asp:TextBox Id="tm" Text="Bitte aendern"
TextMode="MultiLine" runat="server" />
<br />
Passwort-Textbox:
<asp:TextBox Id="tp" TextMode="Password" runat="server" />
<br />
<asp:Button Id="s" Text="senden" OnClick="Page_Load"
runat="server" />
</form>
</body>
</html>
```

*Listing 13.12: Textfelder (textbox.aspx)*

## 13.26 XML Web Control

Mit dem XML Web Control haben Sie die Möglichkeit XML-Elemente in Ihre Webseiten zu integrieren. Da dieser Bereich sehr umfangreich und interessant ist, haben wir ihm ein eigenes Kapitel (Kapitel 18) gewidmet.

# 14 Validation Controls

Im folgenden Abschnitt werden wir Ihnen die Überprüfung von Eingabefeldern über Validation Controls näher bringen. Nach einer kurzen Erklärung, was hinter diesem Konzept steckt, stellen wir Ihnen die Controls vor und zeigen Ihnen eine kurze Anwendung für den Einsatz dieser Controls.

## 14.1 Was sind Validation-Controls?

Validation Controls bieten Ihnen die Möglichkeit Kontrolle über die Formularelemente, die Sie in Ihren Webseiten verwenden, zu erlangen.

Sie können mit diesen Controls beispielsweise überprüfen, ob ein Eingabefeld einen Text enthält oder ob es leer ist. Sie können aber auch komplexere Überprüfungen durchführen, die den Anwender zwingen eine vorgeschriebene Standardeingabe durchzuführen. Sie können über diese Controls beispielsweise sicherstellen, dass eine einzugebende Emailadresse ein »@« oder mindestens einen Punkt nach diesem Klammeraffen enthält.

Ein Parameter der Validation Controls gibt immer einfach an, welches Element Ihrer Formularseite zu überprüfen ist. Eventuelle Fehlermeldungen für unvollständige Formularelemente werden über einen weiteren Parameter gesteuert.

Eine Validierung kann natürlich nur erfolgen, wenn die entsprechenden Programmelemente serverseitig implementiert sind.

Sie können einen ersten Überblick über die Validierungen, die Sie ausführen können, in der nachfolgenden Tabelle gewinnen.

| Validierungsart | Control Name | Kurzbeschreibung |
|---|---|---|
| Benutzer-definiert | `CustomValidator` | Sie können eigene selbst definierte Validierungen erstellen. |
| Pflichtfeld | `RequiredField-Validator` | dient der Kontrolle, dass Felder mit Inhalten befüllt werden |
| Regulärer Ausdruck | `RegularExpression-Validator` | Überprüfung, ob ein regulärer Ausdruck mit der Benutzereingabe übereinstimmt |

| Validierungsart | Control Name | Kurzbeschreibung |
|---|---|---|
| Wertebereich | `RangeValidator` | Aufgrund festgelegter Wertegrenzen werden die Usereingaben geprüft. |
| Wertevergleich | `CompareValidator` | Hiermit können Sie Vergleiche mit Einträgen aus anderen Controls, oder einfach mit anderen Variablen oder Konstanten durchführen. |

*Tabelle 14.1: Die unterschiedlichen möglichen Validierungsarten*

Die Kontrolle der Serverelemente ist noch einmal auf einige wenige eingeschränkt. Welche Serverelemente über die `ValidationControls` kontrolliert werden können, entnehmen Sie der nachfolgende Tabelle. Wir werden auf die Details im Folgenden noch einmal eingehen.

| Control-Name | Programmcode | Kurzbeschreibung |
|---|---|---|
| DropDownList | `<asp:dropdownlist>` | Auswahlliste von Einträgen, ein Web Control |
| HtmlInputFile | `<input Type="file">` | Dateiimport, ein HTML Control |
| HtmlInputText | `<input Type="text">`<br>`<input Type="password">` | Texteingabe, entweder einfache Texte oder alternativ Passwörter, ein HTML Control |
| HtmlSelect | `<select>` | Auswahlliste von Einträgen, ein HTML Control |
| HtmlTextarea | `<textarea>` | Ein evtl. mehrzeiliges Textfeld, ein HTML Control |
| ListBox | `<asp:listbox>` | Eine Listbox mit mehrfacher Auswahlmöglichkeit, ein Web Control |
| RadioButtonList | `<asp:radiobuttonlist>` | Eine Liste aus Radio Buttons, ein Web Control |
| TextBox | `<asp:textbox>` | Eine einzeilige, mehrzeilige oder Passwörter enthaltende Textbox, ein Web Control |

*Tabelle 14.2: Die validierbaren ServerControls*

## 14.2 Validierungsarten

Nachdem wir Ihnen einen Überblick über die Verwendungsmöglichkeiten von Validation Controls gegeben haben, gehen wir nun kurz auf die einzelnen Validatoren ein.


*Ein grundsätzlicher Hinweis ist noch angebracht: Da die Validierung mit sofortiger Einblendung der Fehlermeldung clientseitig durchgeführt und diese über JavaScript gesteuert wird, funktionieren alle clientseitigen Validierungen nur mit JavaScript-fähigen Browsern, bei denen diese Fähigkeit auch aktiviert ist. Ansonsten findet die Validierung serverseitig statt.*

### 14.2.1 Pflichtfelder

Pflichtfelder werden Sie vermutlich in Ihrer Entwicklerlaufbahn am häufigsten überprüfen. Wenn Sie Eingabemasken erstellen, ist es häufig so, dass bestimmte Angaben gemacht sein müssen, damit eine Übertragung auf den Server und eine weitere Bearbeitung überhaupt einen Sinn haben. Eine Pflichtfeldüberprüfung testet also ab, ob in bestimmte Felder nichts eingegeben ist.

Das Validation Control, das diese Validierung ausführt, heißt – wie bereits im Überblick angedeutet – `RequiredFieldValidator`.

Als Parameter müssen Sie die `Id` (den Identifikator) des Controls angeben, das überprüft werden soll (der aufgerufene Parameter hierfür heißt `ControlToValidate`), außerdem einen Parameter, der eine Fehlermeldung ausgibt (`ErrorMessage`).

Ein Control-Aufruf würde also beispielsweise so aussehen:

```
<asp:RequiredFieldValidator
ControlToValidate="t1"
ErrorMessage ="Bitte eingeben!"
Runat="server" />
```

Übrigens wird bei der Validierung ein Trim-Vorgang durchgeführt, alle Returns, Leerzeichen oder Tabulatoren würden also am Anfang und am Ende der Eingabe entfernt – bleibt dann nichts übrig, würde dies als keine Eingabe interpretiert und es würde die Fehlermeldung angezeigt werden.

### 14.2.2 Regulärer Ausdruck

Sie haben die Wahl einen regulären Ausdruck zu definieren und ihn als Überprüfungskriterium für die Eingabe des Users herzunehmen. Hiermit lassen sich beispielsweise einfach E-Mail-Adressen überprüfen.

Wenn Sie wissen wollen, wie man reguläre Ausdrücke aufbaut sowie welche weiteren Möglichkeiten sich hinter diesen Ausdrücken verbergen, so finden Sie diese Informationen in Kapitel 11.

Als Parameter für den `RegularExpressionValidator` benötigen Sie neben den bereits bekannten Feldern `ControlToValidate` und `ErrorMessage` auch noch einen Parameter `ValidationExpression`, der den regulären Ausdruck, der zur Überprüfung herangezogen werden soll, enthält.

Ein möglicher Aufruf einer Validierung hätte also diese Form:

```
<asp:RegularExpressionValidator
ControlToValidate = "t1"
ErrorMessage ="Ihre Eingabe ist inkorrekt!"
Validationexpression ="[A-D]{2}"
Runat="server" />
```

Dieses Control würde im Control mit der ID `t1` nur zweibuchstabige Werte zulassen, die aus den Großbuchstaben A bis D bestehen.

### 14.2.3 Wertebereich

Für die nachfolgenden Datentypen können Sie Wertebereiche definieren, auf denen dann eine Überprüfung mit dem `RangeValidator`-Control wirken würde:

- Currency
- DateTime
- Double
- Integer
- String

Einer dieser Datentypen muss als Parameter genannt werden. Ansonsten sind wieder der Identifikator des zu validierenden Controls, eine Fehlermeldung sowie Untergrenze und Obergrenze des zulässigen Wertebereichs aufzulisten.

Ein Control-Aufruf hätte dann etwa die nachfolgende Form:

```
<asp:RangeValidator
ControlToValidate="t1"
ErrorMessage ="Erlaubt sind nur Werte von 2 bis 20!"
Type="Integer"
MinimumValue = "2"
MaximumValue = "20"
Runat="server" />
```

Diese Parameter stehen Ihnen auch als Eigenschaften auf der Serverseite zur Verfügung, so dass Sie die Werte auch dynamisch setzen können.

Beachten Sie, dass keine Eingabe innerhalb der zulässigen Grenze liegen würde. Für die Überprüfung, ob auch eine Eingabe vorgenommen wurde, benötigen Sie den `RequiredFieldValidator`.

### 14.2.4 Wertevergleich

Mit dem `CompareValidator`-Control können Sie zwei Controls miteinander vergleichen. Dabei steht Ihnen nicht nur die Möglichkeit zur Verfügung, zu sehen, ob zwei Controls gleiche Werte besitzen (`Equal`), Sie können auch prüfen, ob die Datentypen gleich sind (`DataTypeCheck`), ob der erste Wert größer ist (`GreaterThan`), ob der zweite Wert größer ist (`LessThan`), ob der erste Wert größer oder gleich ist (`GreaterThanEqual`), ob der zweite Wert größer oder gleich ist (`LessThanEqual`) und zu guter Letzt ob die beiden Werte ungleich sind (`NotEqual`).

Ein Aufruf des `CompareValidator`-Controls würde dann etwa so aussehen:

```
<asp:CompareValidator
ControlToValidate = "t1"
ControlToCompare = "t2"
ErrorMessage ="Ihr Datum muss nach dem Datum aus t2 liegen!"
Type ="DateTime"
Operator = "GreaterThanEqual"
Runat="server" />
```

### 14.2.5 Benutzerdefiniert

Benutzerdefiniert heißt in diesem Fall, dass Sie eine serverseitige Funktion schreiben können, die eine Überprüfung durchführt. Dieser Funktionsname muss im Parameter `OnServerValidate` übergeben werden.

Der Rückgabewert dieser Funktion ist die Eigenschaft `IsValid`. Diese muss entweder `true` oder `false` sein. Ist sie `false`, wird die in `ErrorMessage` angegebene Fehlermeldung ausgegeben. Natürlich müssen Sie auch wieder den Identifikator des zu überprüfenden Controls angeben.

Hiermit ergibt sich der folgende Control-Aufruf für den benutzerdefinierten Validator `CustomValidator`.

```
void Funktion1(Object o, ServerValidateEventArgs e) {
  if (...) {
    e.IsValid = true;
  }
  else {
    e.IsValid = false;
  }
}
<asp:CustomValidator
ControlToValidate = "t1"
ErrorMessage ="Ihre Eingabe ist inkorrekt!"
OnServerValidate ="Funktion1"
Runat="server" />
```

Wir haben auch noch ein Fragment der aufgerufenen Funktion mit eingefügt.

## 14.3 Fehlerausgabe

Neben der einfachen Variante Fehlermeldungen über einen Fehlertext, den Parameter `ErrorMessage`, auszugeben, haben Sie auch noch andere Möglichkeiten, eine Fehlermeldung oder eine Liste von Fehlermeldungen auszugeben, die das Ergebnis von fehlgeschlagenen Validierungen sind.

Sie können auch über die Eigenschaft `Text` eine Fehlermeldung ausgeben. Die Ausgabe erfolgt in Form eines Labels.

Die Ausgabe ist damit auch unabhängig von der Behandlung der Ausgabe von `ErrorMessage`.

Die Möglichkeit, eine Validierung zu gruppieren und in einer Liste auszugeben, demonstrieren wir Ihnen im nachfolgenden Abschnitt.

### 14.3.1 ValidationSummary: Zusammenfassung von Fehlern

Für eine strukturiertere Fehlermeldung steht Ihnen das Control `ValidationSummary` zur Verfügung. Mit ihr werden alle Fehlermeldungen zusammengefasst.

Die Form der Anzeige wird dabei über einen Parameter gesteuert, der `DisplayMode` heißt.

Die möglichen Arten des `DisplayMode` und eine Beschreibung dieser Arten können Sie der nachfolgenden Tabelle entnehmen.

| Modusart | Beschreibung |
|---|---|
| `BulletList` | Eine Liste, die mit einem Bullet begonnen wird |
| `List` | Eine Liste ohne ein besonderes Listenstartkennzeichen. |
| `SingleParagraph` | Die Fehlermeldungen werden unstrukturiert hintereinander ausgegeben. |

*Tabelle 14.3: Die Modi des DisplayMode*

Weiterhin können Sie eine Zusammenfassung anzeigen (Parameter `Showsummary`) und die Fehlermeldung in eine Messagebox umleiten.

Wollen Sie die Ausgabe der Fehlermeldung im Fenster verhindern, so können Sie dies dadurch erreichen, dass der Parameter Display auf den Wert `None` gesetzt wird. Dann wird eine Fehlermeldung nur durch eine `ValidationSummary` ausgegeben. Als weitere Werte sind `Static` (ein fester Platz für die Fehlermeldung wird innerhalb des HTML-Textes reserviert) und `Dynamic` (die Fehlermeldung wird dynamisch eingebunden, d.h. das Layout der Website wird durch die Fehlermeldung verändert) möglich.

Ein möglicher Aufruf des `ValidationSummary`-Controls hätte die nachfolgende Form:

```
<asp:ValidationSummary
Id="t1"
DisplayMode="BulletList"
ShowMessageBox="true"
HeaderText="Diese Eingaben waren fehlerhaft:"
Runat="server"/>
```

## 14.4 Weitere Eigenschaften von Validation Controls

Wir sind bereits auf die Eigenschaften `Text` und `ErrorMessage`, die Validation Controls besitzen, eingegangen. Es gibt noch eine Reihe weiterer Eigenschaften für die Gestaltung von Validierungen, die wir Ihnen nachfolgend vorstellen wollen.

Etliche Eigenschaften kennen Sie bereits von HTML Controls und Web Controls her. So können Sie mittels `BackColor` die Hintergrundfarbe, mit `BorderColor` die Rahmenfarbe, mit `CssClass` ein Stylesheet in einem Validation Control festlegen. Damit stehen Ihnen viele Gestaltungsmöglichkeiten des Controls offen. Da sich die Eigenschaften im Wesentlichen an den Web Control-Eigenschaften orientieren, verweisen wir Sie auf die dortigen Erläuterungen.

## 14.5 Ein Beispielprogramm

Nachfolgend finden Sie ein kleines Beispielprogramm, das einige Felder überprüft und die Fehlermeldungen hierzu ausgibt.



```
<%@ Page Language="C#" %>
<script Runat="server">
void Page_Load(Object o, EventArgs e)
{
  a.Text = "Validierungen von Email ";
  a.Text += " und einer Zahl. <br />";
}
void Mult(Object o, ServerValidateEventArgs e)
{
  String str;
  int Rest;

  int b = Convert.ToInt32(tviel.Text);
  int Ergebnis = Math.DivRem(b, 127, out Rest);

  if (Rest == 0) {
    e.IsValid = true;
  }
  else {
    e.IsValid = false;
  }
}
```

```
</script>
<html>
<head><title>Validierungen</title></head>
<body>
<form Runat="server">
<asp:Label Id="a" Runat="server" />
Geben Sie Ihre Emailadresse ein:
<asp:TextBox Id="temail"
TextMode = "SingleLine" Runat="server" />
<br />
Geben Sie ein Vielfaches von 127 an (min. 127): <br />
<asp:TextBox Id="tviel"
TextMode = "SingleLine" Runat="server" />
<br />
<asp:Button Id="s" Text = "senden" OnClick="Page_Load"
Runat="server" />

<asp:RequiredFieldValidator ControlToValidate="tviel"
ErrorMessage="Keine Zahl eingegeben"
Runat="server" />

<asp:RequiredFieldValidator ControlToValidate="temail"
ErrorMessage="Keine Email eingegeben"
Runat="server" />

<asp:RegularExpressionValidator ControlToValidate="tviel"
ErrorMessage="Keine Ziffern eingegeben"
Text="Sie müssen Ziffern eingeben"
ValidationExpression="[0-9]+"
Runat="server" />

<asp:RegularExpressionValidator ControlToValidate="temail"
ErrorMessage="Email falsch eingegeben"
ValidationExpression=
"^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.[a-zA-Z]{2,}$"
Runat="server" />

<asp:RangeValidator ControlToValidate="tviel"
ErrorMessage="Min 127 und max 99999!"
Type= "Integer"
Display = "None"
MinimumValue = "127"
MaximumValue = "99999"
Runat="server" />
```

```
<asp:CustomValidator ControlToValidate="tviel"
ErrorMessage="kein Vielfaches!"
OnServerValidate = "Mult"
Runat="server" />

<asp:ValidationSummary
HeaderText="Das ist falsch:"
DisplayMode = "BulletList"
ShowMessageBox = "true"
Runat="server" />
</form>
</body>
</html>
```

*Listing 14.1: Das Validierungs-Beispielprogramm (validation.aspx)*

Zusammengefasst macht dieses Programm das Folgende:

Wir haben zwei Eingabefelder deklariert. In eines soll eine Zahl (ein Vielfaches von 127) und in das andere eine E-Mail-Adresse eingegeben werden.

Die Überprüfungen sind nun die folgenden:

Bei der Email wird erst überprüft, ob überhaupt eine Eingabe stattgefunden hat. Danach wird anhand einer Überprüfung über einen regulären Ausdruck ein Grundmuster von Emailadressen gecheckt.

Bei der Zahl gibt es eine Reihe von weiteren Überprüfungen: Ist ein Wert eingegeben? Mit einem regulären Ausdruck wird überprüft, ob es sich auch wirklich um eine Zahl handelt (dabei sind führende Nullen zugelassen). Dann wird festgestellt ob die Zahl zwischen den Grenzen 127 und 99999 liegt. Zu guter Letzt wird eine selbst geschriebene Überprüfungsroutine aufgerufen, die nachschaut, ob die eingegebene Zahl ein Vielfaches von 127 ist.

Die Fehlermeldungen werden als Zusammenfassung in einem JavaScript-Popup dargestellt.

# Teil III


# Go ahead!

# 15 Sessions und Cookies

Webapplikationen haben im Vergleich zu Windowsanwendungen einen großen Nachteil: Da Webapplikationen über das Protokoll HTTP übermittelt werden, haben sie keinen Status.

Dieser Umstand ist in vielen Anwendungen nicht relevant, in einigen jedoch ist es wichtig, für jeden Benutzer der Site ein individuelles Handling zu ermöglichen. Dazu dienen so genannte Sessions – zu Deutsch Anwendersitzungen. Innerhalb einer Anwendersitzung können Sie den Besucher Ihrer Site auf seinen Wegen nachverfolgen und so bestimmte Aktionen definiert ausführen.

Das für eine Anwendersitzung gebräuchlichste Beispiel ist wohl der Warenkorb. Sinnvoll ist ein Warenkorb nur dann, wenn jeder Besucher Ihrer Site einen eigenen Warenkorb hat und die dort abgelegten Artikel so lange gespeichert werden, bis der Besucher einen Bestellvorgang abschließt. Dabei darf es keine Rolle spielen, auf welchen Seiten Ihrer Anwendung sich der Besucher in der Zwischenzeit getummelt hatte, seine Anwendersitzung muss immer aktiv bleiben.

Um Anwendersitzungen zu ermöglichen, gibt es in ASP.NET zwei Möglichkeiten. So können Sie eine Session mit oder ohne Cookies verarbeiten.



## 15.1 Cookies

Ein Cookie ist eine kleine Textinformation, die beim Aufruf einer Seite vom Webserver an den Browser geschickt wird. Diese Textdatei wird in einem speziellen Verzeichnis abgelegt und ist dort vom Webserver aus über den Browser les- und schreibbar (beim Netscape: Alle Cookies in einer einzigen Textdatei). Wenngleich es zur Einführung der Cookies große Sicherheitsbedenken gab, lassen heute die meisten Benutzer den Einsatz von Cookies zu.

Der große Vorteil von Cookies besteht darin, dass die kleine Textdatei (auf Wunsch) auch nach dem Verlassen einer Website beim Benutzer gespeichert bleibt. So ist es möglich, einen Benutzer auch nach einigen Tagen beim Wiederkehren auf eine Site zu erkennen und mit den bereits über diesen Benutzer gespeicherten Informationen weiterzuarbeiten.

### Einschränkungen

Cookies unterliegen ganz generell einigen Beschränkungen, die Sie bei deren Verwendung kennen sollten:

- Cookies dürfen nur von der gleichen Domain geschrieben und wieder ausgelesen werden. So kann ein Webserver der Domain *microsoft.com* beispielsweise kein Cookie der Domain *netscape.com* auslesen oder gar manipulieren.
- Ein einzelnes Cookie hat eine maximale Größe von 4Kbyte.
- Die Browser erlauben es nicht, mehr als 300 unterschiedliche Cookies zu speichern. Wird nach dieser Obergrenze ein neues Cookie von einem Webserver gesendet, so löscht der Browser automatisch das älteste vorhandene Cookie.
- Pro Domain (z.B. addison-wesley.de) speichert ein Browser maximal 20 unterschiedliche Cookies. Auch hier wird automatisch das älteste Cookie gelöscht, wenn diese Obergrenze erreicht wird.
- Cookies können an einen Pfad des Webservers gebunden werden. So können Sie beim Setzen eines Cookies beispielsweise den Pfad */asp.net* angeben. Dieses Cookie kann dann nur mehr aus diesem oder einem Unterverzeichnis gelesen und geschrieben werden. Aus dem Verzeichnis */homepage* ist der Zugriff auf dieses Cookie unterbunden.



### Cookies schreiben

Im .NET Framework gibt es mit der Klasse `HttpCookie` eine spezielle Klasse für das Verarbeiten von Cookies. Um ein Cookie also schreiben zu können, muss dieses zunächst über einen Konstruktor erstellt werden.

```
HttpCookie cookie = new HttpCookie("meincookie");
```

Das Cookie `cookie` hat den Namen `meincookie`. Cookies können eine Reihe von Wertepaaren speichern. Dazu benutzen Sie die `Set`-Methode der `Value`-Eigenschaft eines Cookies:

```
cookie.Values.Set("Farbe", "lila");
```

Ein Cookie wird über einen Webserver gesetzt. Dazu sendet der Webserver das Cookie als Teil des Headers mit an den Browser, der das Cookie dann abhängig von der Konfiguration annimmt. Für das Senden von HTTP-Elementen ist das `Response`-Objekt zuständig. Um ein Cookie nun zu setzen, verwenden Sie die Methode `SetCookie` dieses Objekts:

```
Response.SetCookie(cookie);
```

Der gesamte Code, um ein einfaches Cookie in einer ASP.NET-Seite zu setzen, lautet somit:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  HttpCookie cookie = new HttpCookie("meincookie");
  cookie.Values.Set("Farbe", "lila");
  Response.SetCookie(cookie);
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
</body>
</html>
```

*Listing 15.1: Ein Cookie setzen (cookie_setzen.aspx)*

Ob das Cookie nun wirklich geschrieben wurde. lässt sich leicht überprüfen. Sobald die automatische Cookie-Behandlung im Browser abgeschaltet ist, sehen Sie Inhalt und Herkunft des gesendeten Cookies (Bild 15.1).

Dieses Cookie ist nun so lange auf dem Rechner des Besuchers verfügbar, wie der Browser geöffnet ist. Da das Cookie Teil der Header-Information ist, die an den Browser geschickt wird, müssen Sie das Cookie bereits vor jeglichem HTML-Code schreiben.

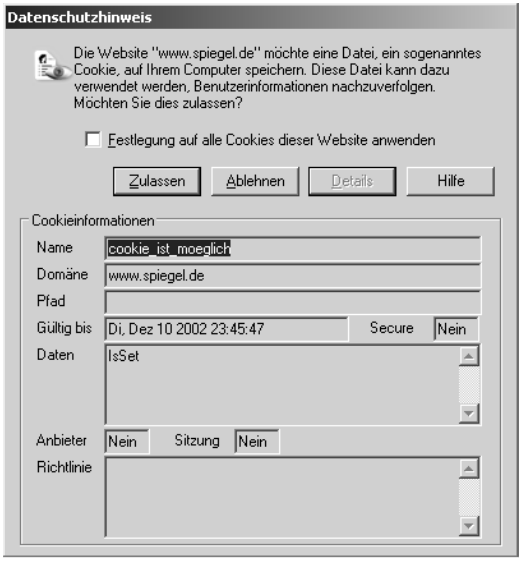



*Bild 15.1: Eingabeaufforderung zur Annahme eines Cookies*



### Ablaufdatum

Natürlich macht es nicht immer Sinn, dass die beim Besucher der Website gespeicherten Daten gelöscht werden, sobald dieser seinen Browser schließt. Wenn Sie die Lebensdauer des Cookies beeinflussen wollen, können Sie die `Expires`-Eigenschaft des `HttpCookie`-Objekts nutzen:

```
cookie.Expires = DateTime.MaxValue;
```

Das Cookie würde nun nicht mehr gelöscht werden, Sie haben ein sog. permanentes oder persistentes Cookie gesetzt. Ob dies besonders sinnvoll ist, bleibt jedoch zu bezweifeln. Denn zum einen ist die Information, die auf dem Benutzerrechner hinterlegt wird, in den seltensten Fällen dauerhaft relevant. Zum anderen speichert ein Browser ja nur eine begrenzte Anzahl von Cookies – ein persistentes Cookie wird also allein schon von daher irgendwann gelöscht werden.

Weitaus gebräuchlicher und auch sinnvoller ist eine Anweisung, mit der das Cookie für eine bestimmte Anzahl von Tagen nach dem Setzen

gültig ist. Wenn Ihr Cookie z.B. 15 Tage auf dem Rechner des Besuchers gespeichert bleiben soll, dann verwenden Sie folgende Anweisung:

```
cookie.Expires = DateTime.Now.AddDays(15);
```

Setzen Sie das Cookie einfach bei jedem Besuch der Website erneut mit einem derartig aktualisiertem Ablaufdatum, und das Cookie bleibt in den Browsern der Besucher gespeichert.

Das folgende Beispiel sorgt nun dafür, dass ein Wertepaar als Cookie zum Browser geschickt wird und dort 15 Tage gespeichert bleibt. Ruft ein Besucher diese Seite erneut auf, so wird das Gültigkeitsdatum erneut auf 15 Tage von jetzt an gesetzt. Wichtig dabei ist, dass Sie nicht nur das Ablaufdatum, sondern immer auch alle Werte setzen:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  HttpCookie cookie = new HttpCookie("meincookie");
  cookie.Values.Set("Farbe", "lila");
  cookie.Expires = DateTime.Now.AddDays(15);
  Response.SetCookie(cookie);
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
</body>
</html>
```

*Listing 15.2: Ein Cookie mit Ablaufdatum setzen (mit_ablaufdatum.aspx)*

### Cookies lesen

Cookies dienen in den meisten Fällen dazu, auf den geschriebenen Informationen beim Besuch der nächsten Seiten definierte Aktionen auszuführen. Damit Sie wissen, welche Aktion genau zu folgen hat, müssen Sie das Cookie auslesen.

Bevor Sie die Werte Ihres Cookies auslesen können, sollten Sie zunächst prüfen, ob Ihr Cookie überhaupt gesetzt ist.

```
if (Request.Cookies["meincookie"] != null) {

}
```

Wie Sie sehen, erfolgt der Zugriff auf Cookies mit Hilfe des `Request`-Objekts. Denn dieses Objekt fängt alle Daten auf, die an den Webserver vom Browser gesendet wurden. Um auf die im Cookie gespeicherten Werte zuzugreifen, benutzen Sie einfach die `Value`-Eigenschaft der `CookieCollection`-Klasse.

```
cookieinhalt = Request.Cookies["meincookie"].Value
```

Mit dem nun folgenden Listing werden die in Listing 15.2 gesetzten Cookie-Werte ausgelesen. Im Browser werden die im Cookie geschriebenen Werte angezeigt:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  if (Request.Cookies["meincookie"] != null) {
    String cookieinhalt =
     Request.Cookies["meincookie"].Value;
    ausgabe.Text = cookieinhalt;
  }
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 15.3: Auslesen eines Cookies (cookie_lesen.aspx)*

### Cookies ändern

Natürlich können Sie den Inhalt von Cookies auch jederzeit ändern. Im Grunde funktioniert dies genau wie das Setzen von Cookies. Nur dass Sie zusätzlich zur `Set`-Methode zum Setzen einzelner Werte eines Cookies mit Hilfe der `Add`-Methode mehrere Werte zu einem Cookie hinzufügen können.

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  if (Request.Cookies["meincookie"] != null) {
    HttpCookie cookie = Request.Cookies["meincookie"];
    cookie.Values.Add("Form", "rund");
    cookie.Expires = DateTime.Now.AddDays(15);
    Response.SetCookie(cookie);
  }
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
</body>
</html>
```

*Listing 15.4: Ein Cookie ändern (cookie_aendern.aspx)*

In diesem Listing wird zunächst geprüft, ob das Cookie `meincookie` bereits besteht. Falls dem so ist, wird ein zusätzliches Wertepaar hinzugefügt.


*Immer wenn Sie die* `Set`*-Eigenschaft der* `Value`*-Methode eines Cookies verwenden, werden alle bis dahin im Cookie gespeicherten Werte gelöscht und durch ein neues Wertepaar ersetzt.*

### Cookies löschen

Zum sauberen Programmieren gehört es auch, nicht mehr benötigte Informationen – wie Cookies – zu löschen. Da Cookies wesentlich über ihr Ablaufdatum gesteuert werden, ist das Löschen eines Cookies sehr einfach.

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
```

```
 if (Request.Cookies["meincookie"] != null) {
    HttpCookie cookie = Request.Cookies["meincookie"];
    cookie.Expires = DateTime.Now;
    Response.SetCookie(cookie);
  }
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
</body>
</html>
```

*Listing 15.5: Ein Cookie löschen (cookie_loeschen.aspx)*

In diesem Listing wird das Ablaufdatum des Cookies auf die aktuelle Sekunde gesetzt, folglich wird es vom Browser in der folgenden Sekunde automatisch gelöscht.

### Pfadangaben

Um den Zugriff auf ein Cookie zu steuern, können Sie mit dem Cookie eine Pfadangabe speichern. So ist ohne diese Angabe der Standardpfad /, also der unterste Pfad Ihrer Anwendung, gesetzt. Dies bedeutet, dass das Cookie von jedem Skript innerhalb Ihrer Anwendung gelesen und manipuliert werden kann. Manchmal ist es jedoch sinnvoll, den Zugriff auf das Cookie nur für Seiten innerhalb eines Bereichs Ihrer Website zuzulassen, z.B. den Bereich `/asp.net/sessions`.

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  HttpCookie cookie = new HttpCookie("meincookie");
  cookie.Values.Set("Farbe", "lila");
  cookie.Expires = DateTime.Now.AddDays(15);
  cookie.Path = "/asp.net/sessions";
  Response.SetCookie(cookie);
}
</script>
<html>
```

```
<head>
<title>Cookies</title>
</head>
<body>
</body>
</html>
```

*Listing 15.6: Ein Cookie mit einer Pfadeinschränkung setzen (pfadangabe.aspx)*

Wie Sie sehen, gibt es beim Setzen eines Cookies mit einer Pfadeinschränkung lediglich eine neue Zeile, in der Sie das `Path`-Attribut des Cookies setzen.


*Die Verwendung von Pfadangaben in Cookies hat zur Folge, dass das Cookie vom Browser nur mehr beim Aufruf von Dokumenten innerhalb des gesetzten Pfades gesendet wird. Damit wird es unmöglich, das Cookie aus dem falschen Bereich einer Anwendung zu lesen.*

**Domain**

Sie können ein Cookie immer nur dann auslesen, wenn Sie das Cookie von einem Server der gleichen Domäne geschrieben haben. Diese Eigenschaft ist ein elementarer Bestandteil der Definition von Cookies, denn so wird ein Missbrauch von Cookies weitestgehend eingeschränkt. Nicht auszudenken, wenn Sie mit einem Skript auf dem Server *www.meinedomain.de* Cookies mit interessanten Informationen anderer Domains wie *www.diebank.de* auslesen könnten. Allerdings führt diese Einschränkung auch zu unangenehmen Nebeneffekten. Setzen Sie Ihr Cookie z.B. auf dem Server *www.meinedomain.de*, so können Sie dieses je nach Browserversion von Ihrem eigenen zweiten Server *apps.meinedomain.de* nicht unbedingt auslesen. Um dieses zu ermöglichen, verwenden Sie die Domain-Eigenschaft des Cookies:

```
HttpCookie cookie = new HttpCookie("meincookie");
cookie.Values.Set("Farbe", "lila");
cookie.Domain = "www.meinedomain.de";
cookie.Domain = "apps.meinedomain.de ";
cookie.Expires = DateTime.Now.AddDays(15);
Response.SetCookie(cookie);
```


*Im Gegensatz zu klassischem ASP können Sie mit der Domain-Eigenschaft keine fremden Domains mehr schreiben. Ist Ihr Rechner Mitglied der Domain meinedomain.de, so wird die Angabe eines Rechners einer zweiten Domain meinedomain.at nicht an den Browser übermittelt. Die Domain-Eigenschaft dient in ASP.NET nur mehr dazu, den Zugriff auf das Cookie von unterschiedlichen Servern der gleichen Domain sicherzustellen, in der das Cookie gesetzt wurde.*

### Cookie-Unterstützung prüfen

Wie Sie sehen, ist der Umgang mit Cookies an sich sehr einfach. Bevor Sie jedoch eine Anwendung entwickeln, die die Verwendung von Cookies voraussetzt, sollten Sie sich immer einer Tatsache bewusst sein: Viele Benutzer schalten die Unterstützung von Cookies im Browser ab. Um dennoch einen planbaren Anwendungsablauf zu erreichen, müssen Sie also die Cookie-Unterstützung jedes Besuchers Ihrer Site testen.

Die Vorgehensweise hierzu ist denkbar einfach. Sie setzen auf einer Seite ein Cookie und versuchen, dieses Cookie auf einer zweiten Seite auszulesen, auf die Sie den Besucher weitergeleitet haben.

Zunächst wird also ein Cookie gesetzt und auf die nächste Seite weitergeleitet:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  HttpCookie cookie = new HttpCookie("meincookie");
  cookie.Values.Set("Cookietest", "ok");
  Response.SetCookie(cookie);
  Response.Redirect("cookietest_teil2.aspx");
}
</script>
```

*Listing 15.7: Teil 1 des Cookietests: Das Setzen eines Cookies (cookietest_teil1.aspx)*

Das vom vorangegangenen Skript aufgerufene Dokument überprüft nun, ob das zuvor gesendete Cookie vom Browser an den Server zurückgeschickt wird, also ob das Setzen des Cookies erfolgreich war:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  if (Request.Cookies["meincookie"] != null) {
    if (Request.Cookies["meincookie"].
     Values["Cookietest"] == "ok") {
      ausgabe.Text = "Der Besucher akzeptiert
       Cookies";
    } else {
      ausgabe.Text = "Cookies werden abgelehnt";
    }
  }
}
</script>
<html>
<head>
<title>Cookies</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 15.8: Teil 2 des Cookietests: die eigentliche Überprüfung (cookietest_teil2.aspx)*

Wenn das Schreiben des Cookies im ersten Skript erfolgreich war, wird eine entsprechende Nachricht ausgegeben. Andernfalls wird der Fehlschlag bekannt gegeben. In einer reellen Anwendung werden Sie wohl kaum derartige Textmeldungen an die Besucher ausgeben. Mit der gewonnenen Information, ob Cookies vom aktuellen Besucher unterstützt werden oder nicht, können Sie jedoch innerhalb Ihrer Anwendung angemessen weiterverfahren.


*Natürlich wäre es sehr schön, mit einer einzigen Seite einen Cookietest durchführen zu können. Dies ist leider nicht möglich, denn das Auslesen von Cookie-Informationen geht nur vor dem Schreiben des ersten Header-Elements, also des ersten Cookies.*

## 15.2 Sessions

Unter einer Benutzersession versteht man das Verweilen eines Besuchers auf einer Website und dessen Bewegungen innerhalb dieser Anwendung. Um nun einer Anwendung zu ermöglichen, genau dieses zu verfolgen, gibt es das passende Konstrukt der Session. Mit Hilfe dieser ist es möglich, jeden Nutzer während seines Verweilens auf der Site eineindeutig zu identifizieren. Dadurch lässt sich ein großer Nachteil des HTTP-Protokolls – dass das Protokoll keinen Status hat – zu einem gewissen Teil umgehen.

Interessant wird die Information darüber, wo sich ein Anwender gerade innerhalb der Site befindet und wie er sich bewegt, immer dann, wenn die Applikation für diesen einzelnen Anwender relevante Daten verarbeitet. Die klassischen Beispiele hierfür sind E-Commerce-Systeme und Portalanwendungen mit Personalisierungskomponenten. Ein Beispiel zu einer Personalisierungskomponente wird am Ende des Kapitels erläutert, vorweg aber noch ein wenig Theorie.



### 15.2.1 Das Session-Objekt

Im .NET Framework wird ein komplettes Sessionmanagement-System ausgeliefert. Dieses kümmert sich um die Erstellung eineindeutiger Identifikatoren für die Bestimmung von Benutzern genauso wie um die Nachverfolgung dieser auf deren Wegen über die Website. Dazu ist das Sessionmanagement in einigen Parametern sehr flexibel und ermöglicht das Management von Sessions unabhängig davon, ob Sie Ihre Website mit nur einem einzelnen Webserver oder mit einer ganzen Serverfarm betreiben.

Wird eine Anwendersitzung gestartet, so wird dem Besucher zunächst ein eindeutiger Identifikator, eine *SessionID*, zugewiesen. Diese ID besteht aus einem String, der einerseits sehr lang und andererseits

über einen Zufallsgenerator generiert wurde. Dadurch soll vermieden werden, dass Sessions von externen Programmen außerhalb des Systems erzeugt werden können – die Anwendung wäre dann sehr leicht zu hacken.

Zu einer *SessionID* können dann benutzerspezifische Daten gespeichert werden. Dazu gehört zum einen der Weg, den der Anwender durch die Site nimmt, zum anderen können Sie auch selbst benutzerspezifische Informationen zu einer Session hinzufügen und verarbeiten. Dazu später mehr.

### 15.2.2 Session-Behandlung einrichten

Um mit Sessions arbeiten zu können, müssen Sie Ihre Anwendung zunächst dafür konfigurieren.

#### Sessions mit Cookies

Wie bereits auch in klassischem ASP arbeitet das Sessionmanagement standardmäßig mit Cookies, um die *SessionID* beim Benutzer zu speichern und dadurch eine eindeutige Verfolgung dessen zu ermöglichen. Um diese Art von Sessionmanagement zu starten, genügt bereits die Standardkonfiguration jeder Webanwendung in ASP.NET. Um sicherzugehen, dass das Sessionmanagement aktiviert ist (bei manchen Providern ist dies in der Grundeinstellung deaktiviert), können Sie dies auch gezielt für Ihre Anwendung erzwingen.

Mit der Konfigurationsdatei *web.config* können Sie Ihre Webapplikation in Grundzügen einstellen. Die Datei *web.config* ist eine einfache XML-Datei im Stammverzeichnis Ihrer Anwendung, welche vom Webserver beim Start der Anwendung eingelesen und verarbeitet wird. Um das Sessionmanagement in der Grundeinstellung zu starten, genügen folgende Einträge in der Konfigurationsdatei:

```
<configuration>
  <system.web>
    <sessionState mode="InProc" />
  </system.web>
</configuration>
```

*Listing 15.9: web.config zum Erzwingen von Sessionmanagement (sessionmanagement_web.config)*

Rufen Sie nun eine ASP.NET-Seite von Ihrem Webserver auf, die eine Session verwendet, so sendet der Server automatisch ein Cookie mit Ihrer SessionID.


*Bild 15.2: Ein Session-Cookie von ASP.NET*

Dieses Cookie dient ab nun dazu, Sie zu identifizieren. Alle weiteren Informationen außer dieser SessionID werden auf dem Webserver gespeichert.

### Sessions ohne Cookies

Wie Sie bereits aus dem vorangegangenen Abschnitt zu Cookies wissen, kann das Annehmen von Cookies von jedem Benutzer selbst abgelehnt werden. Wenn Ihre Anwendung die Verwendung von Sessions zwingend erfordert (weil Sie z.B. einen Warenkorb aufbauen), dann ist es also nicht ratsam, sich auf Sessions mit Cookies zu verlassen. In klassischem ASP mussten Sie eigene Konstrukte verwenden, um den-

noch sicher eine *SessionID* von Seite zu Seite weiterzuleiten. ASP.NET unterstützt nun von sich aus Sessions ohne Cookies.

Um das Management von Sessions ohne Cookies zu aktivieren, müssen Sie lediglich einen Schlüsseleintrag in Ihre Konfigurationsdatei *web.config* einfügen:

```
<configuration>
  <system.web>
    <sessionState cookieless="true" />
  </system.web>
</configuration>
```

*Listing 15.10: web.config zum Aktivieren von Sessions ohne Cookies (cookieless_web.config)*

Durch den Eintrag `cookieless="true"` in der Konfigurationsdatei wird erreicht, dass die SessionID automatisch als Teil der URL an den Browser übergeben wird. Darüber hinaus wird die SessionID auch automatisch in alle Links innerhalb Ihrer Anwendung eingefügt, so dass beim Aufruf einer anderen Seite mit dem `GET`-Request des Browsers die SessionID des Besuchers übergeben wird.

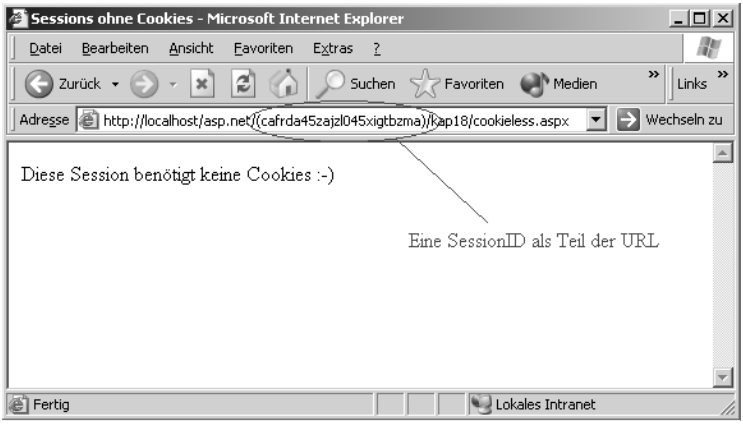



*Bild 15.3: SessionID als Teil der URL*

Der Vorteil dieser Vorgehensweise liegt auf der Hand: Durch die automatische Verquickung von bestehender URL und SessionID wird die

wichtige ID unabhängig von Browserart und -konfiguration von Seite zu Seite mit übertragen, Sie können mit Sessions frei agieren.


*Wenn Sie mit Sessions ohne Cookies arbeiten, müssen Sie beachten, dass Sie innerhalb Ihrer Anwendung nur relative Links einsetzen. Denn nur bei der Verwendung von relativen Links kann ASP.NET beim dynamischen Generieren des HTML-Streams, der an den Besucher Ihrer Site gesendet wird, gewährleisten, dass SessionID korrekt in den Linkpfad integriert wird. Zudem geht ASP.NET davon aus, dass beim Setzen von absoluten Pfaden ein Verlassen der Anwendung gewünscht ist, so dass Ihre Sessioninformationen gelöscht werden.*

### Weitere Konfigurationsmöglichkeiten

Sessions dienen dazu, für einzelne Besucher der Website individuelle Informationen bereitzuhalten. Mit ASP.NET haben Sie die Möglichkeit, den Speicherort für diese Informationen festzulegen. Diese Konfiguration geschieht über Einträge in der Datei *web.config* über den `mode`-Parameter des Attributs `sessionState`.

```
<configuration>
  <system.web>
    <sessionState mode="InProc" />
  </system.web>
</configuration>
```

*Listing 15.11: Standardeinstellung des Speicherorts von Sessioninformationen (session_lokal_web.config)*

Die Grundkonfiguration für den Speicherort von Sessioninformationen lautet `InProc` und bedeutet, dass alle Sessioninformationen im Arbeitsspeicher des aktuellen Servers gehalten werden. Diese Konfiguration ist für einfache Umgebungen, bei denen nur ein Webserver und eine geringe bis mittlere Anzahl von Besuchern vorhanden sind. Durch das Speichern der Informationen im Arbeitsspeicher des Servers sind diese Daten sehr performant abgreifbar. Allerdings beschränkt der Arbeitsspeicher auch die Anzahl an zugleich haltbaren Informationen. Da Sessioninformationen jedoch einen geringen Speicherbedarf haben, ist dieser Umstand nicht zu kritisch.

Wenn Sie Anwendungen auf mehreren Servern parallel betreiben, müssen Sie Sessions zentral managen. Dazu können Sie über die Konfigurationsdatei *web.config* festlegen, dass ein Server das Sessionmanagement übernimmt. Auf diesen Server greifen dann alle Webserver zu.

```
<configuration>
  <system.web>
    <sessionState mode="SessionState"
     stateConnectionString="tcpip=10.0.0.1:42424" />
  </system.web>
</configuration>
```

*Listing 15.12: Sessioninformationen auf einem zentralen Server speichern (session_remote_web.config)*

Mit dieser Konfiguration werden Sessions zentral vom Server mit der IP-Adresse 10.0.0.1 verwaltet. So werden dort alle SessionIDs generiert und zusätzliche Informationen auf diesem Server im Arbeitsspeicher gehalten.

Noch flexibler und skalierbarer sind Sie, wenn Sie Sessioninformationen in einer Datenbank ablegen. ASP.NET unterstützt deswegen die Verwendung eines SQL-Servers zu diesem Zweck.

```
<configuration>
  <system.web>
    <sessionState mode="SqlServer"
     sslConnectionString="data source=10.0.0.1;
     database=state; user id=session; password=pwd"
    />
  </system.web>
</configuration>
```

*Listing 15.13: Sessioninformationen in einer Datenbank verwalten (session_db_web.config)*

Alle Sessions werden nun in einer Datenbank gespeichert. Dadurch kann auch eine große Webserverfarm auf gemeinsame Sessioninformationen zugreifen. Der Nachteil dieser Vorgehensweise ist jedoch, dass durch die zusätzlichen Datenbankabfragen etwas an Performance verloren geht.

### 15.2.3 Mit Sessions arbeiten

Nachdem Sie Ihre Webanwendung für die Verwendung von Sessions vorbereitet haben, kann es nun endlich losgehen.

#### Sessioninformationen schreiben

Sessioninformationen bestehen aus Paaren von Schlüsseln und Werten. Diese lassen sich einfach durch den Aufruf

```
Session["Hallo"] = "Welt";
```

schreiben. Sobald Sie in einer Anwendung das erste Mal Sessioninformationen schreiben, wird die Session gestartet. Mit einem einfachen Skript können Sie also bereits erste Informationen in eine Session schreiben:

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  Session["Startzeit"] = DateTime.Now.Ticks;
}
</script>
<html>
<head>
<title>Erste Sessions</title>
</head>
<body>
<a href="session_lesen.aspx ">
 Verweildauer auf der Site</a>
</body>
</html>
```

*Listing 15.14: Eine erste Sessioninformation schreiben (session_schreiben.aspx)*

#### Sessioninformationen lesen

Um die zuvor in die Session geschriebene Information wieder auszulesen, genügt ebenfalls eine einfache Zuweisung:

```
String meinevar = ((String) Session["Hallo"]);
```

Die Variable `meinevar` enthält nun den Wert `Hallo`. Im folgenden Beispiel wird die Sessioninformation aus Listing 15.14 dazu genutzt, die Zeit zu berechnen, die der Besucher seit dem Aufruf des vorigen Listings auf der Site verbracht hat.

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  Int64 anfang = ((Int64) Session["Startzeit"]);
  DateTime anfang_date = new System.DateTime(anfang);
  TimeSpan zeit =
   DateTime.Now.Subtract(anfang_date);
  ausgabe.Text = zeit.ToString();
}
</script>
<html>
<head>
<title>Verweildauer</title>
</head>
<body>
<asp:Label id="ausgabe" runat="Server" />
</body>
</html>
```

*Listing 15.15: Auslesen einer Session-Variablen (session_lesen.aspx)*

### Sessioninformationen löschen

Neben dem Schreiben von Sessioninformationen besteht natürlich auch die Möglichkeit, einzelne Sessioninformationen zu löschen.

```
Session.Remove["Hallo"];
```

Mit diesem einfachen Befehl löschen Sie die Sessionvariable `Hallo`.

Verlässt ein Benutzer Ihre Website, dann ist es sinnvoll, alle seine Sessioninformationen zu löschen. Statt diese einzeln über den `Session.Remove`-Befehl zu entfernen, können Sie die Methode `Session.Clear()` einsetzen.

```
<%@ Page language="C#" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  Session.Clear();
}
</script>
<html>
<head>
<title>Alle Sessioninformationen löschen</title>
</head>
<body>
Alle individuellen Daten wurden gelöscht.
</body>
</html>
```

*Listing 15.16: Löschen aller Sessioninformationen (sessions_loeschen.aspx)*


*Achten Sie darauf, dass Sie Sessioninformationen immer aktiv beim Verlassen des Besuchers löschen. Auf diese Art und Weise stellen Sie sicher, dass Ihr Speicher für Sessioninformationen nur aktuell verwendbare Informationen enthält.*

# 16 Dateizugriff

Die einfachste und oft auch effizienteste Art, Daten zu speichern, ist das Ablegen dieser Daten in einfachen Textdateien. Ob es sich nur um Log-Daten oder um komplette Foren handelt – oft stehen hinter einer Anwendung eine oder mehrere Dateien. Einer der wichtigsten Gründe dafür ist, dass das Lesen und Schreiben von Dateien in freigegebenen Verzeichnissen von nahezu jedem Provider schon mit den kleinsten Hosting-Paketen gestattet wird. Hinzu kommt, dass Sie sich für die eine oder andere Anwendung vielleicht eine Datenbankunterstützung wünschen würden – eine Textdatei erfüllt aber oft den gleichen Zweck bei wesentlich geringerem Aufwand.

Mit ASP.NET können Sie selbstverständlich auf das Dateisystem des Servers zugreifen. Sofern der ASP.NET-User die entsprechenden Berechtigungen hat, können Sie Dateien und Verzeichnisse erstellen und löschen, umbenennen und auf Dateien lesend und schreibend zugreifen. Wie Sie dies im Detail umsetzen können, zeigt dieses Kapitel.


*Bei allen Operationen auf dem Dateisystem verwenden Sie Klassen und Methoden des Namespaces* `System.IO`. *Diesen müssen Sie zuerst in Ihr Script einbinden, denn sonst findet der .NET-Compiler die verwendeten Elemente des Namespaces nicht und es kommt zu einem Kompilierungsfehler. Zum Einbinden dient folgender Code:*

```
<%@ Import Namespace="System.IO" %>
```

## 16.1 Aus Dateien lesen

Wollen Sie lediglich den Inhalt einer Datei lesen, dann können Sie die Klasse `StreamReader` dazu verwenden.

### 16.1.1 Eine gesamte Textdatei lesen

Als Beispiel soll eine Textdatei *datei.txt* mit zwei Zeilen Inhalt ausgelesen werden.

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.IO" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String pfad = "c:\\datei.txt";
  StreamReader reader = new StreamReader(pfad);
  ausgabe.Text = reader.ReadToEnd();
  reader.Close();
}
</script>
<html>
<head>
<title>Auslesen einer Textdatei</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 16.1: Eine Textdatei lesen (datei_lesen.aspx)*

Zunächst wird ein neues `StreamReader`-Objekt instanziert und im Anschluss der gesamte Inhalt des Objektes mit Hilfe der `ReadToEnd`-Methode ausgelesen. Mit der Methode `Close` wird der Zugriff auf die Datei beendet.

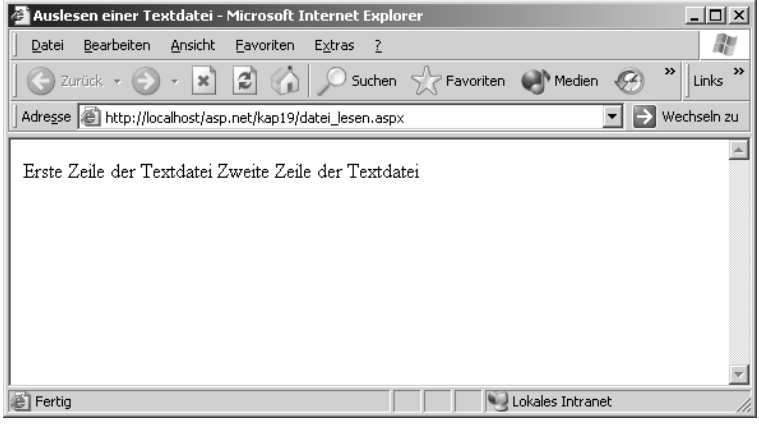



*Bild 16.1: Ausgabe der eingelesenen Datei*

Auch wenn mit dem kleinen Script schon der erste Schritt gemacht wurde, so gibt es doch noch einige kleinere Unwägbarkeiten, die abzustellen sind.

So fällt beim Lesen des Quellcodes als Erstes auf, dass der Pfad zur ausgelesenen Textdatei fest codiert ist. Auf diese Art und Weise mag das Beispiel auf Ihrem Entwicklungsrechner vielleicht funktionieren. Sobald das Script aber auf einen fremden Server kopiert wird, müssen Sie vielleicht eine andere Festplattenpartition oder andere Pfade nutzen. Von daher empfiehlt es sich, dass Sie den Pfad der einzulesenden Datei relativ zu Ihrer ASP.NET-Seite angeben und die Datei entsprechend speichern. Mit

```
Server.MapPath("daten/datei.txt")
```

oder ganz ausführlich

```
HttpServerUtility.MapPath("daten/datei.txt")
```

geben Sie an, dass die Textdatei *datei.txt* im Unterverzeichnis *daten* relativ zum ausgeführten Script liegt. Dabei spielt es gar keine Rolle, in welchem absoluten Verzeichnis sich ihr eigentliches Script befindet, die Anwendung wird also auf andere Server portierbar.

Ein weiterer Fehler am ursprünglichen Script ist auch, dass der gesamte Inhalt der Textdatei in einer Zeile ausgegeben wird. Der Zeilenumbruch wurde jedoch nicht ignoriert, sondern schlichtweg nicht in das statt eines Zeilenumbruchs in HTML erforderliche Steuerzeichen `<br />` umgewandelt.

### 16.1.2 Zeilenweise lesen

Der einfachste Weg, die in der Textdatei enthaltenen Zeilenumbrüche wieder korrekt mit auszugeben, ist es, die Textdatei Zeile für Zeile einzulesen und nach jedem Zeilenende einen HTML-Zeilenumbruch einzufügen. Die Methode `ReadLine` liest genau eine Zeile des `StreamReader`-Objekts ein.

```
inhalt += reader.ReadLine();
inhalt += "<br />";
```

Mit diesen zwei Zeilen Code ist eine Zeile eingelesen und der Zeilenumbruch formatiert. Um jede Zeile der Datei auszulesen, muss diese Codesequenz nun in eine Schleife eingebunden werden.

```
while (reader.Peek() != -1) {
...
}
```

Die Methode `Peek` liefert das nächste folgende Zeichen eines `StreamReader`-Objekts, ohne dieses jedoch tatsächlich zu lesen. Dadurch eignet sich die Methode hervorragend, um das Dateiende einer Textdatei aufzuspüren. Denn ist das nächste folgende Zeichen ungültig (also -1), so ist die Textdatei zu Ende.

Der gesamte Code zum Zeilenweisen Einlesen und korrekten Ausgeben einer Textdatei lautet dann:

```
<%@ Page language="C#" %>
<% @ Import Namespace="System.IO" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String pfad = Server.MapPath("daten/datei.txt");
  String inhalt = "";
  StreamReader reader = new StreamReader(pfad);
  while (reader.Peek() != -1) {
    inhalt += reader.ReadLine();
    inhalt += "<br />";
  }
  ausgabe.Text = inhalt;
  reader.Close();
}
</script>
<html>
<head>
<title>Auslesen einer Textdatei</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 16.2: Zeilenweises Lesen einer Textdatei (zeilenweise_lesen.aspx)*

## 16.2 In Dateien schreiben

Um Textdateien zu schreiben, genügt es, die Klasse `StreamWriter` zu nutzen. Das folgende Script schreibt mit Hilfe der `WriteLine`-Methode zwei Zeilen in eine Datei:

```
<%@ Page language="C#" %>
<% @ Import Namespace="System.IO" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String pfad = Server.MapPath("daten/log.txt");
  StreamWriter writer = new StreamWriter(pfad);
  writer.WriteLine("Eine Zeile");
  writer.WriteLine("Eine weitere Zeile");
  writer.Close();
}
</script>
<html>
<head>
<title>Schreiben einer Textdatei</title>
</head>
<body>
</body>
</html>
```

*Listing 16.3: Einfaches Schreiben einer Datei (datei_schreiben.aspx)*

Wie Sie sehen, ist das Vorgehen der Methoden aus den Klassen `StreamReader` und `StreamWriter` sehr ähnlich. Den Konstruktoren beider Klassen übergeben Sie den Pfad zur Datei, den Lese- bzw. Schreibvorgang schließt immer die Methode `Close` ab.


*Der Aufruf der Methode* `Close` *bewirkt bei* `StreamWriter`*-Objekten, dass das tatsächliche Schreiben von Daten erfolgt und die Objektinstanz beendet wird. Vor dem Aufruf der Methode befinden sich die zu schreibenden Daten im Speicher der Anwendung.*

*Der Aufruf der Methode* `Flush` *löst den tatsächlichen Schreibvorgang genau wie die* `Close`*-Methode aus, nur dass der verwendete* `StreamWriter` *durch die Methode* `Flush` *nicht beendet wird, sondern weiterhin verwendet werden kann.*

```
writer.Flush();
```


*Gerade beim Schreiben von Dateien treten häufig unerwartete Probleme auf. Zumeist ist die Ursache dafür jedoch relativ einfach: Der ASP.NET-Benutzer hat nicht die erforderlichen Schreibberechtigungen auf dem gewünschten Verzeichnis. Prüfen Sie daher Berechtigungen lieber einmal zu oft als zu selten.*

**Daten zu einer Datei hinzufügen**

Eine häufig gestellte Anforderung ist das Schreiben von Daten, die ein Besucher der Site in ein Formular eingegeben hatte, in eine Art Log-Datei. So könnte das Formular etwa persönliche Daten zur Teilnahme an einem Gewinnspiel abfragen. Ein derartiges Formular mit integrierter Speicherung der Daten soll nun erstellt werden.

Der zunächst einfachste Teil der Aufgabe ist das Erstellen eines entsprechenden Formulars. Um das Beispiel übersichtlich zu halten, werden lediglich eine E-Mail-Adresse und das Geburtsdatum des Teilnehmers abgefragt:

```
<form runat="server">
  E-Mail:
  <asp:TextBox id="email" runat="server" /><br />
  Geburtstag:
  <asp:TextBox id="gebtag" runat="server" /><br />
  <asp:Button text="Teilnehmen" onclick="speichern" runat="server" />
</form>
```

Das Formular ist als serverseitig auszuführendes Formular angelegt. Durch einen Klick auf den Button `Teilnehmen` wird die Prozedur `speichern` aufgerufen:

```
void speichern(Object Sender, EventArgs e) {
  String pfad = Server.MapPath("daten/spiel.txt");
  String zeile = "";
  StreamWriter writer = new StreamWriter(pfad, true);
  zeile = email.Text + ";" + gebtag.Text;
  writer.WriteLine(zeile);
  writer.Close();
}
```

Die meisten Elemente der Prozedur sind Ihnen bereits bekannt. Neu, aber dennoch entscheidend ist der zweite Parameter `true` beim Konstruktor des `StreamWriter`-Objekts. Durch diesen Parameter legen Sie fest, dass die Datei zum Anhängen von Daten geöffnet wird. Dadurch ist sichergestellt, dass nicht jeder neue Teilnehmer am Gewinnspiel die Daten seines Vorgängers überschreibt, sondern dass wie gewünscht eine Liste mit den Daten aller Teilnehmer entsteht. Der komplette Quellcode für dieses Beispiel lautet dann:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.IO" %>
<script runat="server">
void speichern(Object Sender, EventArgs e)
{
  String pfad = Server.MapPath("daten/spiel.txt");
  String zeile = "";
  StreamWriter writer = new StreamWriter(pfad, true);
  zeile = email.Text + ";" + gebtag.Text;
  writer.WriteLine(zeile);
  writer.Close();
  formular.Visible = false;
  ausgabe.Text = "Herzlichen Glückwunsch - Sie ";
  ausgabe.Text += "nehmen am Gewinnspiel teil! ";
  ausgabe.Text += "<br />Ihre Daten wurden ";
  ausgabe.Text += "erfolgreich gespeichert.";
}
</script>
<html>
<head>
<title>teilnahme am Gewinnspiel</title>
</head>
<body>
<div id="formular" runat="server">
  Bitte geben Sie Ihre Daten zur Teilnahme am
  Gewinnspiel an:
  <form runat="server">
    E-Mail:
    <asp:TextBox id="email" runat="server" /><br />
    Geburtstag:
    <asp:TextBox id="gebtag" runat="server" /><br />
    <asp:Button text="Teilnehmen"
     onclick="speichern" runat="server" />
  </form>
```

```
</div>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 16.4: Speichern von Daten aus einem Formular in einer Datei (datei_aendern.aspx)*

## 16.3 Erweitertes Dateihandling

Neben den bereits bekannten Klassen `StreamReader` und `StreamWriter` ist die Klasse `File` beim Umgang mit Dateien sehr nützlich. Mit den Methoden dieser Klasse können Sie Dateien zur weiteren Verarbeitung öffnen, kopieren, umbenennen, löschen und diverse Dateiattribute abfragen. Die nachfolgende Tabelle gibt eine Übersicht über die Methoden der Klasse:

| Methode | Funktion |
|---|---|
| `Exists(dateiname)` | Prüft, ob eine Datei `dateiname` besteht, und liefert einen booleschen Wert zurück |
| `Open(dateiname)` | Öffnet die Datei `dateiname` |
| `Copy(dateiname, kopiename)` | Kopiert eine Datei `dateiname` in die Datei `kopiename` |
| `Create(dateiname)` | Erstellt eine Datei `dateiname` |
| `Delete(dateiname)` | Löscht die Datei `dateiname` |
| `Move(dateiname, kopiename)` | Verschiebt die Datei `dateiname` in die Datei `kopiename`. Mit dieser Methode können Sie Dateien auch umbenennen. |
| `GetCreationTime(dateiname)` | Liefert das Erstellungsdatum einer Datei `dateiname` zurück |
| `GetLastAccessTime(dateiname)` | Gibt den Wert des letzten Zugriffs auf die Datei `dateiname` aus |
| `GetLastWriteTime(dateiname)` | Liefert den Zeitpunkt der letzten Änderung der Datei `dateiname`. |

*Tabelle 16.1: Die wichtigsten Methoden der* `File`*-Klasse*

Wollen Sie beispielsweise eine Datei kopieren, so können Sie mit folgendem ASP.NET-Skript automatisieren:

```
<%@ Page language="C#" %>
<% @ Import Namespace="System.IO" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String orig = Server.MapPath("daten/log.txt");
  String kopie = Server.MapPath("daten/log1.txt");
  File.Copy(orig, kopie);
}
</script>
<html>
<head>
<title>Kopieren einer Datei</title>
</head>
<body>
</body>
</html>
```

*Listing 16.5: Kopieren einer Datei (datei_kopieren.aspx)*

## 16.4 Mit Verzeichnissen arbeiten

Bei komplexeren Anwendungen (z.B. einer Administrationsoberfläche für eine dynamische Website) kann es erforderliche sein, neben dem einfachen Arbeiten mit Dateien ganze Verzeichnisbäume zu erstellen, zu bewegen oder auch nur auszuwerten. In diesen Fällen können Sie mit den Methoden der `Directory`-Klasse eine ganze Menge erreichen.

| Methode | Funktion |
|---|---|
| `Exists(pfad)` | Prüft, ob das Verzeichnis `pfad` existiert |
| `CreateDirectory(pfad)` | Erstellt das Verzeichnis `pfad` |
| `Delete(pfad)` | Löscht das Verzeichnis `pfad` |
| `Move(pfad, ziel)` | Verschiebt das Verzeichnis `pfad` nach ziel |
| `GetDirectories(pfad)` | Liefert ein String-Array mit den Namen aller Unterverzeichnisse des Verzeichnisses `pfad` zurück |

| Methode | Funktion |
|---|---|
| `GetDirectoryRoot(pfad)` | Gibt den Namen des Stammverzeichnisses von `pfad` zurück |
| `GetFiles(pfad)` | Liefert ein String-Array mit den Namen aller im Verzeichnis `pfad` enthaltenen Dateien |
| `GetLogicalDrives()` | Gibt ein String-Array mit den Namen aller auf dem Rechner verfügbaren logischen Laufwerke (z.B. »c:\«) aus |
| `GetParent(pfad)` | Benennt das Oberverzeichnis des Verzeichnisses `pfad` |

*Tabelle 16.2: Die wichtigsten Methoden der Klasse* `Directory`

Wollen Sie also den Inhalt eines Verzeichnisses auslesen, dann genügt folgendes Skript:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.IO" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String pfad = Server.MapPath(".");
  String inhalt = "<p><b>Unterverzeichnisse von ";
  inhalt += Server.MapPath(".").ToString();
  inhalt += ":</b><br />";
  String[] unterverzeichnisse =
   Directory.GetDirectories(pfad);
  foreach(String verzeichnis in unterverzeichnisse){
    inhalt += " " + verzeichnis;
    inhalt += "<br />";
  }
  inhalt += "</p><p><b>Dateien in ";
  inhalt += Server.MapPath(".").ToString();
  inhalt += ":</b><br />";
  String[] dateien = Directory.GetFiles(pfad);
  foreach (String datei in dateien) {
    inhalt += " " + datei;
    inhalt += "<br />";
  }
  inhalt += "</p>";
  ausgabe.Text = inhalt;
}
```

```
</script>
<html>
<head>
<title>Dateien und Unterverzeichnisse</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 16.6: Dateien und Unterverzeichnisse des aktuellen Verzeichnisses (verzeichnis_lesen.aspx)*

Das eigentliche Auslesen der gewünschten Inhalte erfolgt hier mit den beiden Methoden `Directory.GetDirectories(pfad)` und `Directory.GetFiles(pfad)`. Der Rest des Skriptes dient lediglich der sauberen Ausgabe der beiden Arrays, die von diesen Methoden zurückgegeben werden. Auch die Klasse `Directory` ist ein Teil des `System.IO`-Namespaces. Vergessen Sie also nicht, diesen bei Datei- und Verzeichnisoperationen immer einzubinden (mit `<%@ Import Namespace="System.IO" %>`).

# 17 Datenbanken

Datenbanken spielen eine wichtige Rolle bei der Erstellung dynamischer Websites. So lassen sich dort Inhalte von Teilen der Website strukturiert speichern und bei Bedarf abrufen, und auch Benutzerdaten werden wegen der einfacheren Auswertbarkeit oft in Datenbanken gehalten.

Gerade die großen Datenbanksysteme wie der SQL Server 2000 oder Oracle 9i bieten durch die Kombination von Performance, Hochverfügbarkeit und Skalierbarkeit ein nahezu perfektes Backend für jede große Webapplikation. Wie Sie Datenbanken ansprechen und ihre Funktionen für eine ASP.NET-Anwendung nutzen können, zeigt dieses Kapitel.

## 17.1 Zugriff auf Datenbanken

Der erste Schritt beim Arbeiten mit einer Backend-Anwendung ist immer, den Zugriff auf diese Anwendung sicherzustellen. Um die Verknüpfung von ASP.NET und den verschiedenen Datenbanken etwas leichter zu verstehen, erfolgt zunächst ein kleiner Ausflug in ADO.NET.

### 17.1.1 ADO.NET

ADO.NET ist die Weiterentwicklung von Microsofts ActiveX Data Objects (ADO), einer Schnittstelle zum Anbinden relationaler Datenquellen. Mit ADO.NET steht Ihnen als .NET-Entwickler eine Architektur zur Verfügung, die das Anbinden von Datenbanken, XML-Dateien und anderen Backend-Anwendungen vereinheitlicht und deutlich erleichtert.

#### Die Architektur von ADO.NET

Als Weiterentwicklung von ADO beruht auch ADO.NET darauf, Ihnen als Entwickler eine Reihe von möglichst einfach zu bedienenden Basisklassen zur Verfügung zu stellen, mit deren Hilfe Sie einheitlich auf die unterschiedlichsten Datenquellen zugreifen können. So sollte es zur Programmentwicklung möglichst keinen Unterschied machen, ob Sie mit einer einfachen MS-Access Datenbank oder einem hochperformanten SQL-Server Cluster als Backend-Applikation arbeiten, denn im Grunde stellt beides für Sie nur eine relationale Datenbank dar.

Darüber hinaus möchte man mit ADO.NET noch einen weiteren entscheidenden Schritt gehen – die eigentliche Anwendung soll nicht zwingend von einer Verbindung zur Datenquelle abhängig sein müssen. So ist es gerade bei Internetanwendungen fast unmöglich, zu regelmäßigen Zeiten auf Daten aus einer Backend-Applikation zu verzichten. Nur müssen andererseits Datenbanken zur Wartung meist kurz offline geschalten werden, ein Konflikt ist vorprogrammiert. Die Lösung bei ADO.NET besteht nun darin, dass ADO.NET eine Zwischenschicht bereitstellt, in der Daten aus einer Datenbank vorgepuffert werden können. Die eigentliche ASP.NET-Anwendung greift dann auf den Puffer zu, die ständige Abhängigkeit von der Erreichbarkeit der eigentlichen Datenbank ist nicht mehr gegeben.

ADO.NET folgt folgender Architektur:

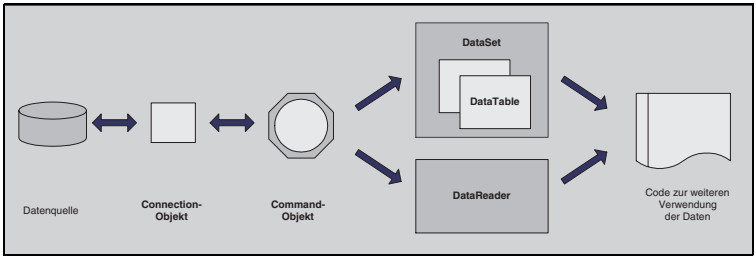



*Bild 17.1: Die Architektur von ADO.NET*

Die zentralen Elemente von ADO.NET sind der *.NET Data Provider* und das `DataSet`. Der .NET Data Provider dient als die zuvor beschriebene Pufferschicht. Über ihn können Datensätze aus der ursprünglichen Datenquelle gespeichert und verarbeitet werden. Um ein möglichst gutes Abbild der ursprünglichen Datenquelle zu erreichen, können einzelne Daten in einer relationalen Struktur im so genannten `DataSet` abgelegt werden. Zusätzlich wird über den .NET Data Provider auch die eigentliche Verbindung zur jeweiligen Datenquelle aufgebaut.

### Basisklassen in ADO.NET

Wie bereits aus der Architektur ersichtlich wird, so hat ADO.NET einen fundamentalen Kern von Basisklassen. Diese Basisklassen finden unabhängig von der Art der Datenquelle Verwendung, wodurch ein

Wechsel von einer auf eine andere Backend-Architektur unter ASP.NET kein großes Dilemma mehr ist.

Die fünf wichtigsten Basisklassen von ADO.NET sind:

1. `Connection`

   Das `Connection`-Objekt stellt die eigentliche Verbindung zwischen ADO.NET und der jeweiligen Datenbank zur Verfügung. Über dieses Objekt lassen sich sämtliche Datenbanktransaktionen abbilden. Unterschiedliche Datenquellen bedürfen zwangsweise unterschiedlicher Arten des Verbindungsaufbaus. So muss für eine optimale Performance eine SQL-Server Datenbank anders angesprochen werden als ein SQL-Server. Zu diesem Zweck sind die Methoden und Eigenschaften des `Connection`-Objekts in speziell auf die Datenquelle abgestimmten Namespaces ausgelagert. Zur Entwicklung aber unterscheiden sich diese Objekte lediglich durch den Namespace, da unabhängig von der Datenquelle die gleichen Methoden und Eigenschaften gleich bedient werden können.

2. `Command`

   Mit Hilfe des `Command`-Objektes können Sie Befehle an die Datenquelle senden. Daher benutzt man die Methoden dieser Klasse dazu, um Daten in der Backend-Datenquelle zu ändern. Abhängig von der Datenquelle unterstützen die Methoden dieser Klasse zum Teil auch komplexere Techniken wie das Verarbeiten von gespeicherten Prozeduren.

3. `DataReader`

   Die Klasse `DataReader` ist eine performance-optimierte Klasse zum Auslesen von Datensätzen aus einer Datenbank. Die Methoden dieser Klasse sind sehr leicht zu bedienen, und da im Arbeitsspeicher immer nur ein Datensatz zur gleichen Zeit gehalten wird, ist diese Klasse prädestiniert für das Verarbeiten größerer Datenmengen.

4. `DataSet`

   Das Konstrukt des `DataSet`-Objekts ermöglicht es, ganze Datenbanksysteme virtuell im Speicher eines Anwendungs-Servers darzustellen. Denn ein `DataSet`-Objekt besteht aus einer Vielzahl von `DataTable`-Objekten und deren Verknüpfungen. So ist es möglich,

dass eine ASP.NET-Anwendung autark auf die Tabellen und deren Verknüpfungen im `DataSet` zurückgreift und dadurch die Webapplikation komplett von der ständigen Anbindung unabhängig wird. Für den Entwickler verhält sich das `DataSet`-Objekt genauso wie eine relationale Datenbank, nur dass Sie ein `DataSet` zu jeder Zeit auch noch als XML-Datei ablegen, oder weitere Daten aus XML zum vorhandenen `DataSet` hinzugewinnen können.

5. `DataTable`

   Wie der Name bereits vermuten lässt, so gleicht ein `DataTable`-Objekt einer Tabelle in einer relationalen Datenbank. In solch einem Objekt können Daten, Schlüsselinformationen und Hinweise auf Verknüpfungen hinterlegt werden. Auch wenn eine `DataTable` auf den ersten Blick sehr ähnlich zum von ADO her bekannten `RecordSet` sein mag, so ist das `DataTable`-Objekt doch eine deutliche Weiterentwicklung dessen.

### 17.1.2 Zugriffsmöglichkeiten auf Datenbank-Server

Wie schon bei der Beschreibung der `Connection`-Klasse angedeutet, gibt es für unterschiedliche Datenbanksysteme speziell angepasste Namespaces, die jeweils die Basisklassen `Connection`, `Command`, `DataReader` und `DataAdapter` enthalten. Diese Unterscheidung ist auf das Konstrukt der *Managed Data Provider* zurückzuführen.

Unter einem Managed Data Provider versteht man die speziell auf eine Art der Datenbankverbindung abgestimmte Implementation des *.NET Data Providers*. Diese individuelle Anpassung mag auf den ersten Blick etwas verwirren, im Endeffekt dient das Aufteilen der unterschiedlichen Anbindung jedoch in enormem Maße der Performance. Mit dem .NET Framework 1.1 werden folgende Managed Data Provider mit ausgeliefert:

- Ein Managed Data Provider für ODBC. Dieser erlaubt es, auf alle Datenquellen zuzugreifen, die über die Standard-Verbindung ODBC angesprochen werden können. Hierzu zählen insbesondere auch strukturierte Textdateien, Excel- und MS-Access-Dateien.
- Ein Managed Data Provider für OLEDB: Mit Hilfe dieses Data Providers lassen sich schon recht performant größere Datenbanksysteme,

die diesen Standard unterstützen, ansprechen. Dazu zählen u.a. der MS-SQL Server, Oracle und Informix.

- Ein Managed Data Provider für den MS-SQL-Server: Dieser Managed Data Provider ist speziell auf den Microsoft-eigenen Datenbankserver abgestimmt und dadurch äußerst performant.
- Ein Managed Data Provider für Oracle: Genau wie für den SQL-Server steht auch für die Anbindung von Oracle-Datenbanken (Versionen 8i und höher) ein speziell optimierter Data Provider zur Verfügung. Dieser greift direkt auf Oracles SQL-Net zu und erreicht daher die beste Performance für eine Oracle-Anbindung.

Wollen Sie also in Ihrer Anwendung einen SQL-Server anbinden, so stehen Ihnen drei unterschiedliche Managed Data Provider zur Verfügung.


*Als Faustregel zur Auswahl des Managed Data Provider gilt: Gibt es für die anzubindende Datenbank einen spezifischen Managed Data Provider, so ist dieser aus Performancegründen anderen Möglichkeiten vorzuziehen. Nur wenn dieser aufgrund von speziellen Anforderungen wie z.B. der verpflichtenden Verwendung eines ODBC-Proxies nicht eingesetzt werden darf, sollten Sie zum Managed Data Provider für OLEDB oder für ODBC ausweichen.*

## 17.2 Daten lesen

Der erste Schritt beim Arbeiten mit Datenbanken ist zugleich auch der, der am häufigsten Verwendung findet: Das Lesen von Daten aus einer oder mehreren Tabellen.

Bevor Sie Daten aus einer Datenbank lesen können, müssen Sie zunächst eine Verbindung zur Datenbank aufbauen.

### Verbindungsaufbau

In unserem Beispiel verwenden wir eine kleine MS-Access Datenbank. Alternativ können Sie selbstverständlich auch andere Datenbanktypen wie den Microsoft SQL-Server oder eine Oracle Datenbank einsetzen, zum reinen Erklären der prinzipiellen Vorgehensweise genügt hier Microsofts kleinste Datenbank aber vollkommen.


*Falls Sie wie im Installationskapitel beschrieben die Quickstart Tutorials des .NET Framework SDK installiert haben, können Sie selbstverständlich auch die Datenbank-Engine MSDE einsetzen. Die ist die Datenbank-Engine des MS-SQL-Servers, hat jedoch keinerlei Benutzeroberfläche. Sofern Sie über gute SQL-Kenntnisse verfügen, können Sie sich jedoch alle gewünschten Objekte wie Tabellen, Sichten und Prozeduren selbst in der Datenbankinstanz erstellen.*

MS-Access lässt sich am einfachsten per ODBC anbinden. Um ODBC aber sinnvoll nutzen zu können, sollten Sie zunächst eine so genannte DSN einrichten. DSN bedeutet Data Source Name und ist ein Alias für Datenquellen, die Sie mit ODBC auf Ihrem Rechner ansprechen wollen. Beim Einrichten der DSN legen Sie bereits einige wichtige Parameter für den Zugriff auf eine Datenquelle fest, die dann beim Verwenden dieser Datenquelle nicht mehr genannt werden müssen. Auch der Managed Data Provider für ODBC wird über die DSN auf die Datenquelle zugreifen.

*Anlegen einer DSN*

Zum Anlegen einer DSN starten Sie über START/SYSTEMSTEUERUNG/VERWALTUNG das Programm DATENQUELLEN (ODBC) (unter Windows XP gelangen Sie über START/SYSTEMSTEUERUNG/LEISTUNG UND WARTUNG/VERWALTUNG an dieses Programm). Wechseln Sie in das Register SYSTEM-DSN. Wenn Sie jetzt hier eine DSN einrichten, dann steht diese allen Benutzern unabhängig davon zur Verfügung, wer gerade am Rechner angemeldet ist.

Das eigentliche Anlegen der DSN beginnt erst hier. Klicken Sie auf die Schaltfläche HINZUFÜGEN, um den Vorgang zu beginnen (Bild 17.2).

Wählen Sie in dem sich öffnenden Fenster zunächst den für Ihre Datenbank geeigneten Treiber aus. Klicken Sie dann auf FERTIG STELLEN.

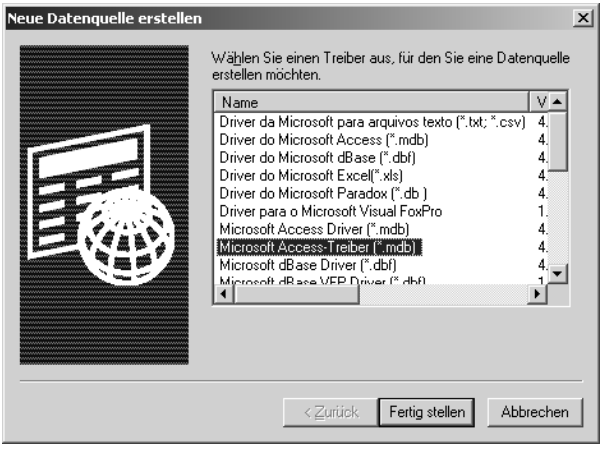



*Bild 17.2: Einrichten einer neuen DSN*

Im Fenster ODBC MICROSOFT ACCESS SETUP legen Sie nun die wichtigsten Parameter für diese Datenverbindung fest. So geben Sie einen DATENQUELLENNAME an und wählen die zu konnektierende Datenbank. Klicken Sie dazu auf die Schaltfläche AUSWÄHLEN und suchen Sie im Anschluss die zu verbindende Datenbank. Optional können Sie noch eine Beschreibung für diese Datenverbindung angeben. Komplett ausgefüllt müsste das Fenster dann etwa so aussehen:

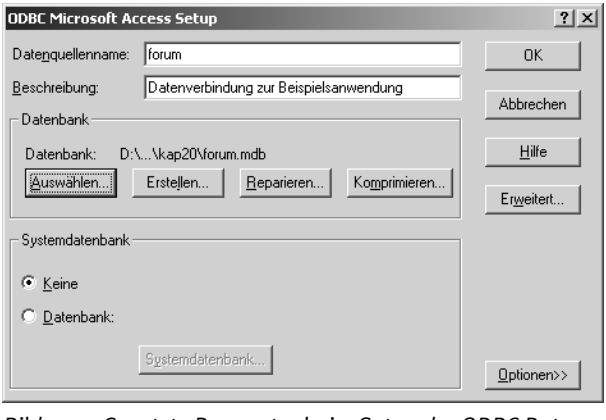



*Bild 17.3: Gesetzte Parameter beim Setup der ODBC Datenquelle*

Klicken Sie im Anschluss noch auf OK und die DSN ist fertig eingerichtet. Sie können nun auch den ODBC-DATENQUELLEN-ADMINISTRATOR wieder beenden.

*Die Datenbank von ASP.NET verbinden*

Der eigentliche Verbindungsaufbau zur Datenbank ist nun simpel. Da Sie eine Verbindung mit dem Managed Data Provider für ODBC aufbauen, muss zuerst der entsprechende Namespace eingebunden werden. Wie alle anderen Managed Data Provider, so ist auch der für ODBC unter dem `System.Data` Namespace untergeordnet.

```
<%@ Import Namespace="System.Data.Odbc" %>
```


*Das .NET Framework wurde in der Version 1.0 noch ohne den Managed Data Provider für ODBC ausgeliefert. Diesen können Sie sich bei Bedarf jedoch von der Microsoft Website herunterladen und installieren. Ab Version 1.1 des Frameworks ist auch dieser Data Provider von vornherein in das Framework integriert.*

Für den Verbindungsaufbau dient nun das `Connection`-Objekt. Dem Konstruktor dieses Objekts muss ein *ConnectionString* übergeben werden. Dieser String enthält alle zur Spezifizierung der Verbindung erforderlichen Angaben. Da Sie vorher bereits eine DSN eingerichtet haben, genügt es, auf diese zu verweisen. Der gesamte Verbindungsaufbau lautet dann wie folgt:

```
<script runat="server">
protected void Page_Load(Object Sender, EventArgs e) {
  string coStr = "DSN=forum";
  OdbcConnection objCon = new OdbcConnection(coStr);
  objCon.Open();
...
</script>
```

Mit der nun geöffneten Verbindung könnten Sie jetzt arbeiten. Leider hat die Vorgehensweise einen sehr oft wiederholten Schönheitsfehler: Schlägt die Verbindung zur Datenbank aus irgendeinem Grund fehl, wird der Besucher der Site mit einer unschönen Fehlermeldung begrüßt. Um dies zu umgehen, sollten Sie einen Fehler im Verbindungsaufbau grundsätzlich abfangen.

Zum Abfangen von Fehlern beim Verbindungsaufbau eignet sich das `try-catch`-Prinzip. In anderen Programmiersprachen war eine Ausnahmebehandlung mit `try` und `catch` schon lange möglich, jetzt können Sie so auch in Ihrer Webapplikation unerwünschte Nebeneffekte auffangen.

Der Fehler, den Sie abfangen wollen, ist eine `OdbcException`. Diese wird immer dann ausgelöst, wenn ein unerwarteter Fehler auftritt oder eine Warnung von einer ODBC-Datenquelle gesendet wird. Ein kompletter Verbindungsaufbau zur MS-Access-Datenbank lautet dann:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  string coStr = "DSN=forum";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    odbcmeldung.Text = "Verbindung erfolgreich geöffnet";
    con.Close();
  }
  catch (OdbcException ex) {
    odbcmeldung.Text = "Verbindung konnte nicht
     geöffnet werden";
  }
}
</script>
<html>
<head>
<title>Mit diesem Grundgerüst wird eine Verbindung
 zur Datenbank 'Forum' geöffnet</title>
</head>
<body>
  <asp:label id="odbcmeldung" runat="server" />
</body>
</html>
```

*Listing 17.1: Verbindungsaufbau mit Ausnahmebehandlung (datenbank_verbinden.aspx)*

### Der DataReader

Wenn es darum geht, Daten aus einer Datenbank auszulesen und in einer ASP.NET-Anwendung darzustellen, ist die Klasse `DataReader` mit ihren Eigenschaften und Methoden wie geschaffen. Um beispielsweise alle Beiträge mit deren Verfassern aus der Datenbank auszulesen, gehen Sie wie folgt vor.

Zunächst muss ein Select-Befehl an die Datenbank geschickt werden. Mit diesem Select-Befehl legen Sie fest, welche Datensätze Sie aus der Datenbank abfragen möchten:

```
SELECT ueberschrift, absender FROM beitraege
```

Wird dieser SQL-Befehl ausgeführt, so werden als Ergebnis alle Überschriften und die zugehörigen Verfasser aus der Tabelle `beitraege` von der Datenbank zurückgegeben. Um diesen Befehl ausführen zu können, müssen Sie eine Instanz eines `Command`-Objekts initiieren.

```
OdbcCommand befehl = new OdbcCommand(sqlStr, con);
```

Das über die Verbindung `con` und mit dem SQL-String `sqlStr` initiierte Objekt können Sie nun so ausführen, dass ein `DataReader`-Objekt mit dem Resultat dieses Befehls gefüllt wird:

```
OdbcDataReader reader = befehl.ExecuteReader();
```

Die Schönheit des `DataReader` liegt darin, dass Ergebnisse einer Abfrage, die in ein `DataReader`-Objekt geschrieben wurden, sehr leicht auszulesen sind. Sollen die gesamten von der Datenbank gelieferten Datensätze angezeigt werden, so können Sie diese mit einer `while`-Schleife sehr einfach in einen String schreiben:

```
while (reader.Read()) {
  daten += reader["ueberschrift"] + " : ";
  daten += reader["absender"] +"<br />";
}
```

Im nun folgenden Skript sind die gerade einzeln aufgezeigten Elemente in einen Zusammenhang gebracht. Die fertige ASP.NET-Seite stellt alle Datensätze der SQL-Abfrage dar.

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "SELECT ueberschrift, absender";
  sqlStr += " FROM beitraege";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr,
     con);
    OdbcDataReader reader = befehl.ExecuteReader();
    while (reader.Read()) {
      daten += reader["ueberschrift"] + " : ";
      daten += reader["absender"] +"<br />";
    }
    ausgabe.Text = daten;
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
<title>Einfaches Lesen von Daten</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 17.2: Ausgabe von Datensätzen mit Hilfe des DataReaders (datenbank_lesen.aspx)*

### Nur ein Ergebnis

Immer dann, wenn Sie von einer SQL-Abfrage nur ein einziges Ergebnis erwarten, können Sie einen noch einfacheren Weg als den `DataReader` zur Ausgabe wählen: die Methode `ExecuteScalar`.

`ExecuteScalar` ist eine Methode des `Command`-Objekts und liefert genau den ersten Wert der ersten Spalte einer SQL-Abfrage zurück. Das Ergebnis dieser Methode müssen Sie lediglich in einen gewünschten Datentyp umwandeln, und schon kann er ausgegeben werden.

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "SELECT COUNT(*) FROM beitraege";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    ausgabe.Text= befehl.ExecuteScalar().ToString();
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
<title>Ein einziges Ergebnis ausgeben</title>
</head>
<body>
  Die Gesamtzahl der Beiträge lautet:
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 17.3: Ein einzelnes Ergebnis einer Abfrage ausgeben (ein_ergebnis.aspx)*

## 17.3 Daten manipulieren

Natürlich können Sie mit ASP.NET nicht nur Daten ausgeben, sondern diese auch manipulieren. Ein erster Schritt der Datenmanipulation ist das Einfügen neuer Datensätze in die Datenbank.

### Daten schreiben

Daten schreiben bedeutet in unserem Beispiel einen neuen Beitrag zum Forum zu erstellen. Der passende SQL-Befehl, um einen neuen Beitrag in die Datenbank zu schreiben, lautet:

```
INSERT INTO beitraege (ueberschrift, beitrag, absender) VALUES (Wert1,
Wert2, Wert3)
```

Wenn Sie einen SQL-Befehle wie diesen ausführen, so liefert die Datenbank keinen Satz von Ergebnissen, sondern nur die Anzahl der vom Befehl betroffenen und geänderten Zeilen. Alle SQL-Befehle, die sich so verhalten, können Sie mit der `ExecuteNonQuery`-Methode eines `Command`-Objekts an die Datenbank übertragen und so ausführen.

Mit folgendem Beispiel werden über ein ASP.NET-Formular neue Beiträge in das Forum eingegeben:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void speichern(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "INSERT INTO beitraege ";
  sqlStr += "(ueberschrift, beitrag, absender) ";
  sqlStr += "VALUES ('" + ueberschrift.Text + "', ";
  sqlStr += "'" + beitrag.Text + "', ";
  sqlStr += "'" + absender.Text + "')";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    Int32 zahl = befehl.ExecuteNonQuery();
```

```
    if (zahl == 1) {
      ausgabe.Text = "Beitrag erfolgreich
       gespeichert";
    }
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
<title>Neuen Beitrag erfassen</title>
</head>
<body>
<% if (IsPostBack) { %>
  <asp:label id="ausgabe" runat="server" />
<% } else { %>
<form runat="server">
  Überschrift:<br />
  <asp:TextBox id="ueberschrift" runat="server"
   /><br />
  Ihr Beitrag:<br />
  <asp:TextBox textmode="MultiLine" id="beitrag"
   runat="server" /><br />
  Ihre Email:<br />
  <asp:TextBox id="absender" runat="server" /><br />
  <asp:Button text="Teilnehmen" onclick="speichern"
   runat="server" />
</form>
<% } %>
</body>
</html>
```

*Listing 17.4: Ein Formular zum Einfügen von Daten in eine ODBC-Datenquelle (datenbank_schreiben.aspx)*

*Eine der häufigsten Schwachstellen bei Formularen, deren Inhalte anschließend in relationalen Datenbanken abgespeichert werden sollen, ist es, dass SQL-Steuerzeichen nicht abgefangen werden. So könnte ein* `INSERT`*-Befehl leicht dadurch beendet werden, dass ein einfaches* `'` *in ein Formularfeld eingetragen wird. Anschließend kann der Besucher einer Site SQL-Befehle seiner Wahl in das Formular eingeben – und diese werden dann über den Webserver an die Datenbank übertragen und dort ausgeführt. Im Ernstfall könnte es dadurch zu katastrophalen Folgen kommen.*

*Dabei ist das Abfangen einer solchen Manipulationsmöglichkeit denkbar einfach. Wenn Sie die Werte jedes Eingabefeldes mit Hilfe einer Funktion wie dieser*

```
// bereinigt den übergebenen String vom
// SQL-Steuerzeichen '
String bereinigen(String inStr)
{
  return Regex.Replace(inStr, "'", "\"");
}
```

*von »gefährlichen« Steuerzeichen bereinigen und erst dann an Ihr SQL-Kommando übergeben, kann nichts mehr schief gehen.*

### Daten ändern

Zu einer kompletten Datenbankanwendung gehört es auch, bereits bestehende Daten ändern zu können. Prinzipiell ist die Vorgehensweise hierzu ähnlich wie beim Einfügen von Datensätzen. Der Unterschied ist lediglich, dass Sie zum Ändern eines bestimmten Datensatzes auch einen Identifikator für diesen Datensatz mit an die Datenbank übergeben müssen, da Sie sonst Gefahr laufen, dass alle Datensätze einer angesprochenen Tabelle verändert werden.

Bevor ein Benutzer einen Datensatz ändern kann, muss er zunächst entscheiden, welcher Datensatz denn überhaupt angepasst werden soll. Das folgende Script gibt alle Beiträge unseres Forums aus, und es bietet hinter jedem Beitrag einen Link, der zur Änderungsmaske dieses Beitrags führt:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "SELECT * FROM beitraege";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
    while (reader.Read()) {
      daten += "<p><b>";
      daten += reader["ueberschrift"] + "</b><br />";
      daten += reader["beitrag"] + "<br />";
      daten += reader["absender"] +"<br />";
      daten += "<a href= ' datensatz_aendern.aspx?id=";
      daten += reader["beitrag_id"] + "'>";
      daten += "Artikel ändern</a></p>";
    }
    ausgabe.Text = daten;
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet werden";
  }
}
</script>
<html>
<head>
<title>Vorbereiten der Beitragsänderung</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 17.5: Auswahl eines Artikels als Vorbereitung der Manipulation eines einzelnen Datensatzes (datensatz_selektieren.aspx)*

Ein Formular zum Ändern eines Datensatzes besteht nun aus zwei Datenbankabfragen:

Beim Laden des Formulars wird der Datensatz in seiner bestehenden Form aus der Datenbank gelesen und die einzelnen Formularelemente mit den aus der Datenbank gelesenen Werten vorgefüllt.

Wird das Formular zum Speichern abgeschickt, so kommt die eigentliche Prozedur zum Ändern der Daten zum Tragen. Mit Hilfe eines `UPDATE`-Statements wird der geänderte Datensatz in die Datenbank geschrieben. Da auch das `UPDATE`-Statement genau wie ein `INSERT`-Statement keine Datensätze zurückliefert, wird es mit der `ExecuteNonQuery`-Methode zur Ausführung gebracht.

Der gesamte Code für die Änderungsmaske lautet dann:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="server">
void Page_Load(Object Sender, EventArgs e)
{
if (!IsPostBack) {
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "SELECT * FROM beitraege";
  sqlStr += " WHERE beitrag_id=";
  sqlStr += Request.QueryString["id"];
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
    while (reader.Read()) {
      ueberschrift.Text = reader["ueberschrift"].ToString();
      beitrag.Text = reader["beitrag"].ToString();
      absender.Text = reader["absender"].ToString();
      beitrag_id.Text = reader["beitrag_id"].ToString();
    }
    ausgabe.Text = daten;
    con.Close();
  }
```

```
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
 }
}

void speichern(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String daten = "";
  String sqlStr = "UPDATE beitraege SET ";
  sqlStr += "ueberschrift = '";
  sqlStr += bereinigen(ueberschrift.Text) + "', ";
  sqlStr += "beitrag = ";
  sqlStr += "'" + bereinigen(beitrag.Text) + "', ";
  sqlStr += "absender = ";
  sqlStr += "'" + absender.Text + "' ";
  sqlStr += "WHERE beitrag_id=" + beitrag_id.Text;
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    int zahl = befehl.ExecuteNonQuery();
    if (zahl == 1) {
      ausgabe.Text = "Beitrag erfolgreich
       gespeichert";
    }
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}

String bereinigen(String inStr)
{
    return Regex.Replace(inStr, "'", "\"");
}
</script>
```

```
<html>
<head>
<title>Ändern eines Datensatzes</title>
</head>
<body>
<% if (IsPostBack) { %>
  <asp:label id="ausgabe" runat="server" />
<% } else { %>
<form runat="server">
  <asp:Label id="beitrag_id" runat="server"
   Visible="false" />
  Überschrift:<br />
  <asp:TextBox id="ueberschrift" runat="server" />
  <br /> Ihr Beitrag:<br />
  <asp:TextBox textmode="MultiLine" id="beitrag"
   runat="server" /><br />
  Ihre Email:<br />
  <asp:TextBox id="absender" runat="server" /><br />
  <asp:Button text="Teilnehmen" onclick="speichern"
   runat="server" />
</form>
<% } %>
</body>
</html>
```

*Listing 17.6: Formular zur Manipulation von Datensätzen (datensatz_aendern.aspx)*

Wie Sie sehen ist das Ändern von Datensätzen unter ASP.NET nicht sehr kompliziert. Wie bei jeder Anweisung zur Datenänderung ist zu beachten, dass ein eindeutiger Identifikator zur Angabe des zu ändernden Datensatzes mit an die Datenbank übergeben wird. Dies geschieht hier mit Hilfe des Labels `beitrag_id`. Da das Attribut `visible` für dieses Webcontrol auf den Wert `false` gesetzt ist, wird das Control nicht im Browser dargestellt und dient nur der Übergabe des Datensatzidentifikators. Die gleiche Funktionalität zur unsichtbaren Übergabe von Werten können Sie auch mit dem HTML-Formularelement `<input type = "hidden" />` erreichen.

Die Methoden, die Sie bislang im Umgang mit einer Datenbank kennen gelernt haben, sind im Vergleich zu klassischem ASP keine wirklich weit reichende Neuerung.

## 17.4 Das DataGrid

Wirklich interessante und neue Möglichkeiten bei der Arbeit mit Datenbanken bietet das *DataGrid*. Ein DataGrid (wörtlich übersetzt Datengitter) ist die Visualisierung einer Tabelle in einem `DataSet`. Sie können das DataGrid zur Darstellung und auch zur Manipulation von Datenbankinhalten benutzen, also all das verwirklichen, was Sie mit einer Tabelle eben umsetzen können.

### Einfache Ausgabe

Die Schönheit des DataGrids liegt vor allem darin, dass Sie sich beim Programmieren auf wesentliche Dinge konzentrieren können, das lästige Gestalten von HTML-Tabellen gehört der Vergangenheit an. Wenn Sie beispielsweise nur Inhalte aus der Datenbank in tabellarischer Form darstellen wollen, können Sie wie folgt vorgehen:

Nachdem Sie einen `DataReader` per `Read` mit Daten aus der Datenbank gefüllt haben, können Sie diesen als Datenquelle für eine Instanz eines DataGrids festlegen.

```
grid.DataSource = reader;
```

Im nächsten Schritt binden Sie die Daten an das grid.

```
grid.DataBind();
```

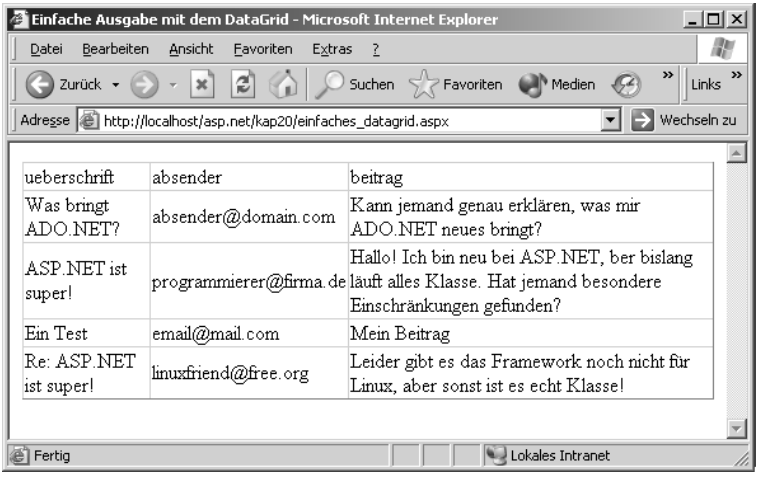
Das war's auch schon. Bereits jetzt können Sie die Daten über das `DataGrid`-Control ausgeben lassen. Der gesamte Quellcode zum tabellarischen Darstellen der Ergebnisse einer Datenbankabfrage lautet dann:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String sqlStr = "SELECT ueberschrift, absender, ";
  sqlStr += "beitrag FROM beitraege";;
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
```

```
    grid.DataSource = reader;
    grid.DataBind();
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
<title>Einfache Ausgabe mit dem DataGrid</title>
</head>
</body>
  <asp:label id="ausgabe" runat="server" />
  <asp:DataGrid id="grid" runat="server" />
</body>
</html>
```

*Listing 17.7: Einfache Ausgabe mit dem* `DataGrid` *(einfaches_datagrid.aspx)*

Das Ergebnis im Browser ist recht simpel:




| ueberschrift | absender | beitrag |
|---|---|---|
| Was bringt ADO.NET? | absender@domain.com | Kann jemand genau erklären, was mir ADO.NET neues bringt? |
| ASP.NET ist super! | programmierer@firma.de | Hallo! Ich bin neu bei ASP.NET, ber bislang läuft alles Klasse. Hat jemand besondere Einschränkungen gefunden? |
| Ein Test | email@mail.com | Mein Beitrag |
| Re: ASP.NET ist super! | linuxfriend@free.org | Leider gibt es das Framework noch nicht für Linux, aber sonst ist es echt Klasse! |

*Bild 17.4: Ein einfaches DataGrid*

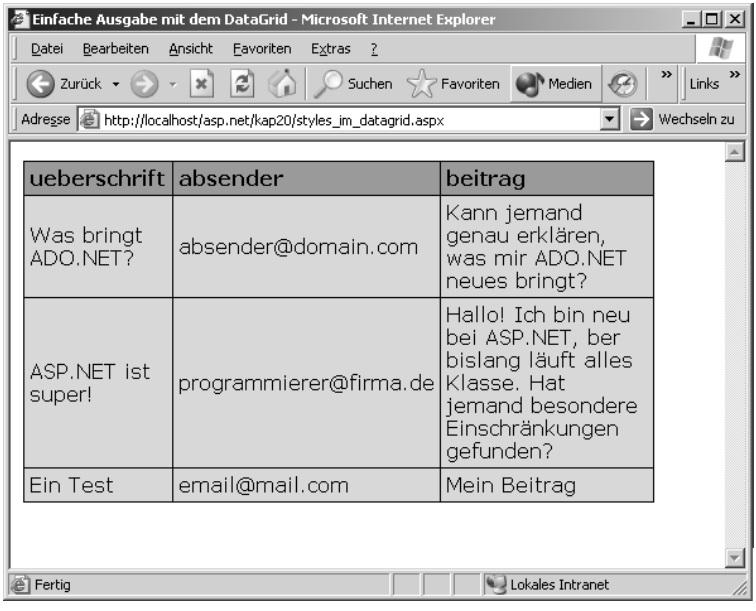
### Formatieren der Ausgabe

Um auch den gestalterischen Ansprüchen Ihrer Website zu genügen, können Sie die Ausgabe des `DataGrid` durch einige Attribute sehr einfach verschönern.

| Attribut | Parameter | Veränderung der Ausgabe |
|---|---|---|
| `BackColor` | RGB Farbcode | Ändert die Hintergrundfarbe der Tabelle |
| `BackImageUrl` | URL | Bestimmt das Hintergrundbild der Tabelle |
| `BorderColor` | RGB Farbcode | Legt die Linienfarbe der Tabelle fest |
| `BorderWidth` | Zahl | Gibt die Breite der ausgegebenen Tabellenlinien an |
| `CellSpacing`, `CellPadding` | Zahl | Verändert Abstände zwischen den Zellen (`CellSpacing`) und Zellenende und Text (`CellPadding`) |
| `Font` | Attribute des Fonts | Bestimmt die Attribute des verwendeten Fonts. So können Sie z.B. über `Font-Name` die Schriftart bestimmen und mit `Font-Italic` die Ausgabe kursiv anlegen. |
| `HeaderStyle`, `FooterStyle` | Je nach weiterem Attribut | Sie können Kopf- und Fußzeile getrennt gestalten; so legen Sie z.B. mit `HeaderStyle-Font` die Schriftart der Kopfzeile fest. |
| `HorizontalAlign` | Right, Left, Center | Gibt die Textausrichtung an |
| `ShowHeader`, `ShowFooter` | True, False | Legt fest, ob die Kopf- oder Fußzeile ausgegeben wird |
| `Width` | Zahl | Bestimmt die Breite der gerenderten Tabelle |

*Tabelle 17.1: Attribute des* `DataGrid`*-Controls*

Mit Hilfe einiger dieser Attribute sieht die Ausgabe der einfachen Abfrage schon wesentlich hübscher aus:




| ueberschrift | absender | beitrag |
|---|---|---|
| Was bringt ADO.NET? | absender@domain.com | Kann jemand genau erklären, was mir ADO.NET neues bringt? |
| ASP.NET ist super! | programmierer@firma.de | Hallo! Ich bin neu bei ASP.NET, ber bislang läuft alles Klasse. Hat jemand besondere Einschränkungen gefunden? |
| Ein Test | email@mail.com | Mein Beitrag |

*Bild 17.5: Formatierte Ausgabe des* `DataGrid`

Erzeugt wurde diese Ausgabe von folgendem Quellcode:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String sqlStr = "SELECT ueberschrift, absender, ";
  sqlStr += "beitrag FROM beitraege";;
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
```

```
    grid.DataSource = reader;
    grid.DataBind();
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
<title>Einfache Ausgabe mit dem DataGrid</title>
</head>
</body>
  <asp:label id="ausgabe" runat="server" />
  <asp:DataGrid id="grid" runat="server"
   BackColor = "#FFEE00"
   BorderColor = "black"
   CellPadding = "4"
   CellSpacing = "0"
   Font-Name = "Verdana"
   Font-Size = "12pt"
   HeaderStyle-Font-Bold = "True"
   HeaderStyle-BackColor = "#00DDDD"
   ShowFooter = "false"
   Width = "500" />
</body>
</html>
```

*Listing 17.8: Ergebnis einer Abfrage über ein formatiertes DataGrid ausgeben (styles_im_datagrid.aspx)*

## 17.5 Erweiterte Möglichkeiten des DataGrids

Das DataGrid beherrscht neben den reinen Darstellungsmöglichkeiten, die Sie bereits im vorangegangenen Abschnitt kennen gelernt haben, auch weiterführende Funktionen. So können Sie mit Hilfe des `DataGrid` verknüpfte Daten übersichtlich darstellen oder auch Datensätze ändern. Das DataGrid eignet sich daher hervorragend zum schnellen Entwickeln von Datenbankoberflächen fürs Web.

### Miteinander verknüpfte Datensätze darstellen

Eines der wesentlichen Merkmale relationaler Datenbanken ist die Verknüpfung der gespeicherten Daten miteinander. So werden beispielsweise einzelne Produkte über einen Schlüssel mit einer Produktkategorie verbunden oder Mitarbeiter einem Unternehmen zugeordnet.

In unserem Beispiel der Datenbank *forum.mdb* hängen einzelne Datensätze innerhalb der gleichen Tabelle über die Spalten `referenz_id` und `beitrag_id` zusammen. So hat ein neuer Datensatz immer den Wert `0` als `referenz_id`, sobald eine Antwort auf einen Forumsbeitrag erfolgt, wird der Wert der Spalte `beitrag_id` aus dem beantworteten Beitrag als Wert der Spalte `referenz_id` in die Antwort gesetzt. Um nun alle neuen Beiträge darzustellen, genügt ein DataGrid, dessen Datensätze über das SQL-Statement

```
SELECT * FROM beitraege WHERE referenz_id = 0
```

aus der Datenbank selektiert wurden. Um nun alle Antworten auf einen bestimmten Beitrag zu erhalten, sieht ein SQL-Befehl beispielsweise so aus:

```
SELECT * FROM beitraege WHERE referenz_id = <beitrag_id_frage>
```

In einem DataGrid können Sie mit wenig Aufwand diese Verknüpfung darstellen:

Zunächst erstellen Sie ein formatiertes DataGrid, das neben den Spalten mit für Leser relevanten Inhalten eine zusätzliche Spalte enthält. Diese verlinkt dann auf Antworten zu diesen Beiträgen. Um bei einem DataGrid nicht alle, sondern nur gewünschte Spalten ausgeben zu lassen, setzen Sie das Attribut `AutoGenerateColumns` Ihres DataGrid auf `false`. Dadurch wird zunächst keine einzige Spalte mehr ausgegeben. Zur Ausgabe können Sie diese nun wieder einzeln hinzufügen:

```
<Columns>
  <asp:BoundColumn DataField = "ueberschrift"
   HeaderText = "Überschrift" />
  ...
</Columns>
```

Mit dem Web Control `<asp:BoundColumn>` lassen sich einzelne Spalten in einem DataGrid zur Ausgabe bestimmen. Die wichtigsten Attribute dieses Controls sind `DataField`, um die Datenbankspalte anzugeben, aus

der die angezeigten Daten kommen sollen, und `HeaderText`, worüber sich eine Spaltenüberschrift festlegen lässt. Im obigen Codeausschnitt wird also eine Spalte `Überschrift` mit den Werten der Datenbankspalte `ueberschrift` ausgegeben.

Um eine Spalte mit Links zum DataGrid hinzuzufügen, verwenden Sie das Control `<asp:HyperLinkColumn>`:

```
<asp:HyperLinkColumn
  DataNavigateUrlField = "beitrag_id"
  DataNavigateUrlFormatString =
   "datagrid_verknuepft_teil2.aspx?id={0}"
  Text = "Antworten auf diesen Artikel"
  Target = "_blank" />
```

Das Control `<asp:HyperLinkColumn>` erstellt eine Spalte mit dem Text des Attributs `Text` in jeder Zeile. Interessant ist, dass der Wert `{0}` des mit dem Attribut `DataNavigateUrlFormatString` festgelegten Links durch ASP.NET automatisch durch einen für diesen Datensatz aktuellen Wert ersetzt wird. Die Spalte, aus der dieser Wert kommen soll, legen Sie mit dem Attribut `DataNavigateUrlField` fest.

Der gesamte Code zur Darstellung einer Übersicht an Basisbeiträgen des Forums lautet dann wie folgt:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  String coStr = "DSN=forum";
  String sqlStr = "SELECT * FROM beitraege ";
  sqlStr += "WHERE referenz_id=0";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
    grid.DataSource = reader;
    grid.DataBind();
    con.Close();
  }
```

```
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
}
</script>
<html>
<head>
  <title>Verknüpfte Spalten im DataGrid</title>
</head>
</body>
  <asp:label id="ausgabe" runat="server" />
  <asp:DataGrid id="grid" runat="server"
   AutoGenerateColumns = "false"
   BackColor = "#FFEE00"
   BorderColor = "black"
   CellPadding = "4"
   CellSpacing = "0"
   Font-Name = "Verdana"
   Font-Size = "12pt"
   HeaderStyle-Font-Bold = "True"
   HeaderStyle-BackColor = "#00DDDD"
   ShowFooter = "false"
   Width = "500">
     <Columns>
       <asp:BoundColumn DataField = "ueberschrift"
        HeaderText = "Überschrift" />
       <asp:BoundColumn DataField = "beitrag"
        HeaderText = "Beitrag" />
       <asp:BoundColumn DataField = "absender"
        HeaderText = "Absender" />
       <asp:HyperLinkColumn
         DataNavigateUrlField = "beitrag_id"
         DataNavigateUrlFormatString =
          "datagrid_verknuepft_teil2.aspx?id={0}"
         Text = "Antworten auf diesen Artikel"
         Target = "_blank" />
     </Columns>
   </asp:DataGrid>
</body>
</html>
```

*Listing 17.9: Ein DataGrid, in dem bestimmte Tabelleninhalte als Parameter für einen Link genutzt werden (datagrid_verknuepft_teil1.aspx)*

Die Ausgabe der Antworten auf einen Beitrag ist nun eigentlich nur noch Formsache. Aus dem Aufruf der Seite entnehmen Sie den Wert des für die WHERE-Klausel Ihres SQL-Statements relevanten Parameters und geben dadurch eingeschränkt nur die gewünschten Beiträge aus. Der Quelltext dazu lautet demnach:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e) {
  String coStr = "DSN=forum";
  String sqlStr = "SELECT * FROM beitraege ";
  sqlStr += "WHERE referenz_id=";
  sqlStr += Request.QueryString["id"];
  OdbcConnection con = new OdbcConnection(coStr);
  try
  {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
    grid.DataSource = reader;
    grid.DataBind();
    con.Close();
  }
  catch (OdbcException ex)
  {
    ausgabe.Text = "Verbindung konnte nicht geöffnet werden";
  }
}
</script>
<html>
<head>
  <title>Verknüpfte Spalten im DataGrid</title>
</head>
</body>
  <asp:label id="ausgabe" runat="server" />
  <asp:DataGrid id="grid" runat="server"
   AutoGenerateColumns = "false"
   BackColor = "#FFEE00"
   BorderColor = "black"
   CellPadding = "4"
   CellSpacing = "0"
   Font-Name = "Verdana"
   Font-Size = "12pt"
```

```
    HeaderStyle-Font-Bold = "True"
    HeaderStyle-BackColor = "#00DDDD"
    ShowFooter = "false"
    Width = "500">
      <Columns>
        <asp:BoundColumn DataField = "ueberschrift"
         HeaderText = "Überschrift" />
        <asp:BoundColumn DataField = "beitrag"
         HeaderText = "Beitrag" />
        <asp:BoundColumn DataField = "absender"
         HeaderText = "Absender" />
      </Columns>
    </asp:DataGrid>
</body>
</html>
```

*Listing 17.10: Ausgabe der Antwort auf einen Beitrag (datagrid_verknuepft_teil2.aspx)*

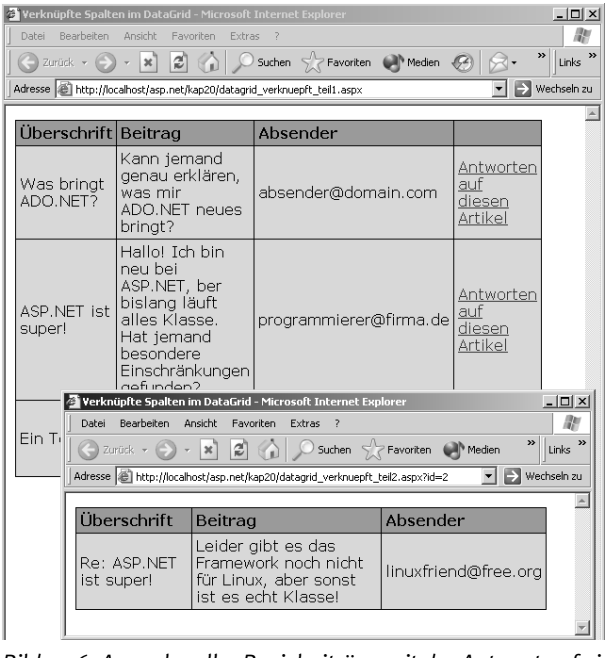



*Bild 17.6: Ausgabe aller Basisbeiträge mit der Antwort auf einen Beitrag als Darstellung eines verknüpften Datensatzes*

### Eine Zeile ändern

In einem vorangegangenen Abschnitt haben Sie gesehen, wie Sie mit Hilfe der `ExecuteNonQuery`-Methode Datensätze ändern können. Dazu war es jedoch erforderlich, einen einzelnen Datensatz zu selektieren, und dann den Schlüssel zur Identifizierung dieses Datensatzes an das eigentliche Änderungsscript zu übergeben. Viel anschaulicher wäre es doch, innerhalb der tabellarischen Darstellungsform des DataGrid einen Datensatz editieren zu können. Genau dieses ist ohne allzu viel Aufwand auch machbar.

Das DataGrid besitzt einen Editiermodus, in dem einzelne Spalten zum Editieren in Formularelementen dargestellt werden. Um in diesen Editiermodus zu wechseln, müssen Sie die Eigenschaft `EditItemIndex` des DataGrid auf die zu ändernde Spalte setzen und im Anschluss die Tabelle erneut ausgeben:

```
void editiermodus(Object Sender, DataGridCommandEventArgs E) {
  grid.EditItemIndex = (int)E.Item.ItemIndex;
  tabelle_ausgeben();
}
```

Doch wie wird diese Funktion aufgerufen? Zum einen müssen Sie dem DataGrid eine neue Spalte hinzufügen:

```
<asp:EditCommandColumn EditText = "Bearbeiten"
 CancelText = "Abbrechen"
 UpdateText="Aktualisieren"
 ItemStyle-Wrap = "false" />
```

Eine Spalte, die Sie mit diesem Control erstellen, gibt zunächst einen Link mit dem Text `Bearbeiten` an. Wenn Sie auf den Link klicken, wird ein `EditCommand` abgesetzt. Dieses Ereignis können Sie abfangen, indem Sie im DataGrid das entsprechende Attribut definieren:

```
OnEditCommand = "editiermodus"
```

Die gerade aktuelle Zeile wird durch die Funktion `editiermodus` nun in den Editiermodus versetzt, und statt des Links `Bearbeiten` stellt die `EditCommandColumn`-Spalte nun die beiden Links `Aktualisieren` und `Abbrechen` dar. Zusätzlich werden alle Elemente der aktuellen Zeile in Eingabefeldern ausgegeben. Klicken Sie nun auf einen der beiden

Links, werden die Funktionen, die hinter `OnUpdateCommand` und `OnCancelCommand` stehen, ausgeführt.

Im Zusammenhang sieht der Code zum Editieren einer Spalte im DataGrid dann so aus:

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data.Odbc" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  if ( !(IsPostBack) ) {
    tabelle_ausgeben();
  }
}

void editiermodus(Object Sender, DataGridCommandEventArgs E)
{
  grid.EditItemIndex = (int)E.Item.ItemIndex;
  tabelle_ausgeben();
}

void speichern(Object Sender, DataGridCommandEventArgs E)
{
  String coStr = "DSN=forum";
  String ueberschrift, beitrag, absender,
   beitrag_id;
  ueberschrift = ((TextBox)
   E.Item.Cells[2].Controls[0]).Text;
  beitrag = ((TextBox)
   E.Item.Cells[3].Controls[0]).Text;
  absender = ((TextBox)
   E.Item.Cells[4].Controls[0]).Text;
  beitrag_id = E.Item.Cells[1].Text;
  String sqlStr = "UPDATE beitraege SET ";
  sqlStr += "ueberschrift = '";
  sqlStr += bereinigen(ueberschrift) + "', ";
  sqlStr += "beitrag = ";
  sqlStr += "'" + bereinigen(beitrag) + "', ";
  sqlStr += "absender = ";
  sqlStr += "'" + bereinigen(absender) + "' ";
  sqlStr += "WHERE beitrag_id=" + beitrag_id;
  OdbcConnection con = new OdbcConnection(coStr);
```

```
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr,
     con);
    Int32 zahl = befehl.ExecuteNonQuery();
    if (zahl != 1) {
      ausgabe.Text = "Fehler beim Speichern";
    }
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet
     werden";
  }
  grid.EditItemIndex = -1;
  tabelle_ausgeben();
}

String bereinigen(String inStr)
{
    return Regex.Replace(inStr, "'", "\"");
}

void abbrechen(Object Sender, DataGridCommandEventArgs E)
{
  grid.EditItemIndex = -1;
  tabelle_ausgeben();
}

void tabelle_ausgeben()
{
  String coStr = "DSN=forum";
  String sqlStr = "SELECT * FROM beitraege ";
  OdbcConnection con = new OdbcConnection(coStr);
  try {
    con.Open();
    OdbcCommand befehl = new OdbcCommand(sqlStr, con);
    OdbcDataReader reader = befehl.ExecuteReader();
    grid.DataSource = reader;
    grid.DataBind();
    con.Close();
  }
  catch (OdbcException ex) {
    ausgabe.Text = "Verbindung konnte nicht geöffnet werden";
```

```
  }
}
</script>
<html>
<head>
  <title>Daten im DataGrid editieren</title>
</head>
</body>
  <asp:label id="ausgabe" runat="server" />
  <form runat="server">
  <asp:DataGrid id="grid" runat="server"
   AutoGenerateColumns = "false"
   BackColor = "#FFEE00"
   BorderColor = "black"
   CellPadding = "4"
   CellSpacing = "0"
   Font-Name = "Verdana"
   Font-Size = "12pt"
   HeaderStyle-Font-Bold = "True"
   HeaderStyle-BackColor = "#00DDDD"
   ShowFooter = "false"
   Width = "500"
   OnEditCommand = "editiermodus"
   OnCancelCommand = "abbrechen"
   OnUpdateCommand = "speichern" >
     <Columns>
       <asp:EditCommandColumn
        EditText = "Bearbeiten"
        CancelText = "Abbrechen"
        UpdateText = "Aktualisieren"
        ItemStyle-Wrap = "false" />
       <asp:BoundColumn DataField = "beitrag_id"
        HeaderText = "Beitrags- nummer"
        ReadOnly = "True" />
       <asp:BoundColumn DataField = "ueberschrift"
        HeaderText = "Überschrift" />
       <asp:BoundColumn DataField = "beitrag"
        HeaderText = "Beitrag" />
       <asp:BoundColumn DataField = "absender"
        HeaderText = "Absender" />
     </Columns>
   </asp:DataGrid>
   </form>
```

```
</body>
</html>
```

*Listing 17.11: Editieren eines Datensatzes im* `DataGrid` *(datagrid_daten_aendern.aspx)*

Prinzipiell sollten Sie den Code, der hinter all diesen Funktionen steckt, bereits aus vorangegangenen Beispielen kennen. Eines ist jedoch noch neu: der Zugriff auf die einzelnen Tabellenzellen, die sich im Editiermodus befanden.

Nun, auf Inhalte einzelner Tabellenzellen können Sie wie folgt zugreifen:

```
beitrag_id = E.Item.Cells[1].Text;
```

Befindet sich eine Zeile gerade im Editiermodus und die Spalte, die Sie ansprechen wollen, ist nicht wie die Beitragsnummer schreibgeschützt, dann müssen Sie das Control ansprechen, das gerade in der Zelle verwendet wird.

```
ueberschrift = ((TextBox) E.Item.Cells[2].Controls[0]).Text;
```


*Der Zugriff ergibt sich folgendermaßen:* `E` *ist eine Referenz auf das* `DataGrid`*,* `Item` *auf die aktuelle Zeile. Deswegen verweist* `Cells[2]` *auf die dritte Zelle in dieser Zeile,* `Controls[0]` *dann auf das erste Steuerelement in dieser Zelle (das Textfeld).*



**Web Controls im DataGrid**

Zum Abschluss dieses Kapitels wollen wir die Darstellung im Editiermodus noch etwas verbessern. Wenn Sie eine Zeile des DataGrids in den Editiermodus setzen, dann werden alle Elemente dieser Zeile in einer einzeiligen Textbox ausgegeben. Bei den Spalten Überschrift und Beitrag wäre es aber weitaus übersichtlicher, mehrzeilige Textfelder zum Ändern der Daten zu verwenden. Um so etwas zu erreichen, können Sie für jede Spalte einzeln festlegen, welches Web Control im normalen und im Editiermodus einer Zeile verwendet werden soll. Dazu benutzen Sie den Spaltentyp `TemplateColumn`:

```
<asp:TemplateColumn HeaderText = "Überschrift">
  <ItemTemplate>
    <asp:Label runat="server"
     Text='<%# DataBinder.Eval(Container.DataItem,
      "ueberschrift") %>'/>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox textmode="MultiLine" rows="5"
     wrap="true" runat="server" id="ueberschrift"
     Text='<%# DataBinder.Eval(Container.DataItem,
      "ueberschrift") %>'/>
  </EditItemTemplate>
</asp:TemplateColumn>
```

Innerhalb einer Spalte des Typs `TemplateColumn` wird mit den Tags `ItemTemplate` und `EditItemTemplate` zwischen Editier- und Normalmodus unterschieden. In beiden Bereichen wird der Zelleninhalt mittels Web Controls dargestellt. Die jeweiligen Daten werden über die Anweisung

```
<%# DataBinder.Eval(Container.DataItem, spaltenname %>
```

an das Control gebunden.

Aus Platzgründen verzichten wir darauf, den gesamten Quellcode des Beispiels *datagrid_daten_aendern_verschoenert.aspx* hier abzudrucken. Denn im Gegensatz zur einfachen Datenänderung mit einem DataGrid werden hier nur die beiden Spalten Überschrift und Beitrag mit Hilfe von `TemplateColumn`-Spalten dargestellt. Stattdessen sehen Sie im folgenden Screenshot das Ergebnis im Editiermodus:

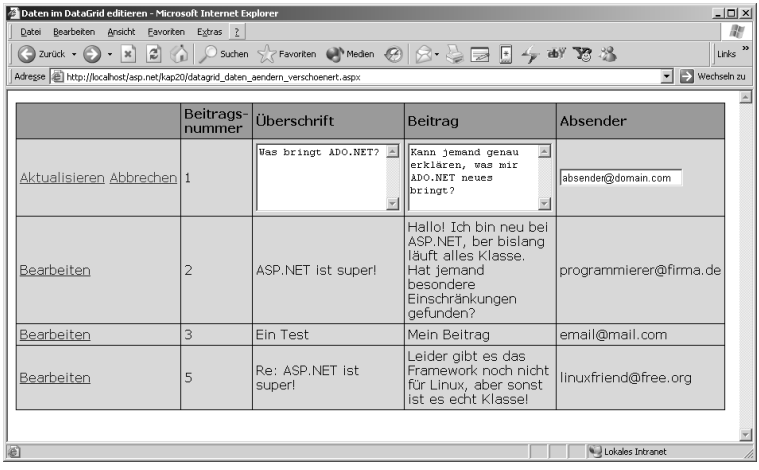



*Bild 17.7: Ein mit Web Controls verschönertes DataGrid zur Datenmanipulation (datagrid_daten_aendern_verschoenert.aspx)*

# 18 XML

Neben Datenbanken sind XML-Dateien immer häufiger die Datenquelle, die strukturierte Informationen vorhalten. Der Vorteil von XML-Dateien liegt schließlich auf der Hand: Sie können eine XML-Datei selbst erstellen und dabei deren Struktur festlegen, und da diese Datei als reine Textdatei keine besonderen Programme zum Auslesen benötigt, kann die Information in jedem Zielsystem weiterverarbeitet werden.

Natürlich unterstützt ASP.NET Sie auch hervorragend bei der Arbeit mit XML. Denn anders als bei herkömmlichen, älteren Architekturen ist XML nicht als Erweiterung zu anderen Methoden hinzugefügt worden, sondern war bereits bei der Konzeption von .NET ein elementarer Bestandteil, der als solcher mit eingeflossen ist. So sind im .NET Framework viele Klassen auf den Umgang mit XML spezialisiert. Und ganz nebenbei sind sogar die Konfigurationsdateien für das Framework und auch für ASP.NET als XML-Dateien gehalten.

In diesem Kapitel werden nun die wichtigsten Klassen und Methoden im Umgang mit XML anhand von Beispielen erklärt.

## 18.1 Lesen mit dem XmlReader

Ähnlich wie der DataReader dient der XmlReader speziell zum effektiven Lesen von Daten. Wie der Name schon vermuten lässt, unterstützt er die Klasse `XmlReader` beim Lesen von XML-basierten Daten.

Nehmen wir an, Sie wollen folgende einfache XML-Datei einlesen und im Browser ausgeben lassen:

```
<?xml version="1.0" encoding="UTF-8" ?>
<forum>
  <beitraege id="1">
    <referenz_id>0</referenz_id>
    <ueberschrift>Was bringt ADO.NET?</ueberschrift>
    <beitrag>Kann jemand genau sagen, was mir
     ADO.NET Neues bringt?</beitrag>
    <absender>absender@domain.com</absender>
  </beitraege>
  <beitraege id="2">
```

```
    <referenz_id>1</referenz_id>
    <ueberschrift>Re: Was bringt
     ADO.NET?</ueberschrift>
    <beitrag>Dynamische Websites einfach und sauber
     implementieren</beitrag>
    <absender>programmierer@domain.com</absender>
  </beitraege>
</forum>
```

*Listing 18.1: Ein XML-Beispieldokument: Die Datei forum.xml*

Die einfachste Variante, diese Aufgabe zu lösen, besteht darin, die Klasse `XmlTextReader` zu nutzen. Wenn Sie dem Konstruktor dieser Klasse eine XML-Datei übergeben, können Sie mit Hilfe einer kleinen Schleife bereits das gewünschte Ergebnis erzielen:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextReader reader = new
   XmlTextReader(Server.MapPath("forum.xml"));
  while (reader.Read()) {
    ausgabe.Text += reader.Value + "<br />";
  }
  reader.Close();
}
</script>
<html>
<head>
<title>Eine XML-Datei lesen</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 18.2: Einfaches Einlesen einer XML-Datei (xml_lesen.aspx)*

Wenn Sie dieses Script ausführen, werden Sie feststellen, dass die Ausgabe noch zu optimieren ist. So werden jetzt noch Element für Element ein Umbruch ausgegeben und zusätzlich zum eigentlichen Content auch alle Attribute der XML-Tags dargestellt.

Um die Ausgabe zu steuern, können Sie mit Hilfe der `NodeType`-Eigenschaften nur gewünschte Inhalte selektieren. Diese Eigenschaft liefert den `XmlNodeType` des gerade aktuellen Knotens zurück. Mögliche Werte sind:

| XmlNodeType | Beschreibung |
|---|---|
| `Attribute` | Attribut eines Knotens |
| `CDATA` | Ein Bereich des Typs CDATA; Inhalt, der über das `CDATA`-Flag als reiner Content klassifiziert wurde, um nicht mit Steuerzeichen in Konflikt zu geraten |
| `Comment` | Ein Kommentar |
| `Document` | Das Root-Element der XML-Datei |
| `Element` | Ein Knoten eines XML-Dokuments |
| `EndElement` | Das End-Tag des Knotens |
| `None` | Das Ende eines XML-Dokuments |
| `Notation` | Eine Anmerkung in der Definition des Dokuments |
| `Text` | Der textuelle Inhalt zwischen zwei XML-Tags |
| `XmlDeclaration` | Die XML-Deklaration (also beispielsweise `<?xml version="1.0"?>`) |

*Tabelle 18.1: Unterschiedliche Knotenbezeichnungen*

Angewandt auf unser vorheriges Beispiel können Sie so festlegen, dass die Ausgabe des XML-Dokuments den eigentlichen Dateninhalt und die dazu passenden Informationen über den aktuellen Knoten enthält:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextReader reader = new
   XmlTextReader(Server.MapPath("forum.xml"));
  while (reader.Read()) {
    if (reader.NodeType == XmlNodeType.Element) {
      ausgabe.Text += reader.Name + ": <br />";
    }
    if (reader.NodeType == XmlNodeType.Text) {
      ausgabe.Text += reader.Value + "<br />";
```

```
    }
  }
  reader.Close();
}
</script>
<html>
<head>
<title>Eine XML-Datei ausgeben - optimiert</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 18.3: Optimierte Ausgabe der XML-Datei (xml_lesen_optimiert.aspx)*

Die Ausgabe ist nun schon fast perfekt. Das Einzige, was jetzt noch fehlt, ist die Struktur der XML-Datei durch Einfügen von entsprechend vielen Leerzeichen optisch wiederzugeben.

Um festzustellen, wie tief Sie sich innerhalb eines Baumes in einer XML-Datei befinden, nutzen Sie die `Depth`-Eigenschaft. Damit wird dann eine formatierte Ausgabe der XML-Datei ermöglicht:



```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextReader reader = new
   XmlTextReader(Server.MapPath("forum.xml"));
  while (reader.Read()) {
    int i;
    if (reader.NodeType == XmlNodeType.Element) {
      for (i=1; i<=reader.Depth; i++) {
        ausgabe.Text += " ";
      }
      ausgabe.Text += "<b>" + reader.Name;
      ausgabe.Text +=  ":</b><br />";
    }
    if (reader.NodeType == XmlNodeType.Text) {
      for (i=1; i<=reader.Depth; i++) {
        ausgabe.Text += " ";
      }
```

```
      ausgabe.Text += reader.Value + "<br />";
    }
  }
  reader.Close();
}
</script>
<html>
<head>
<title>Eine XML-Datei formatiert ausgeben</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 18.4: Eine XML-Datei formatiert ausgeben (xml_lesen_formatiert.aspx)*

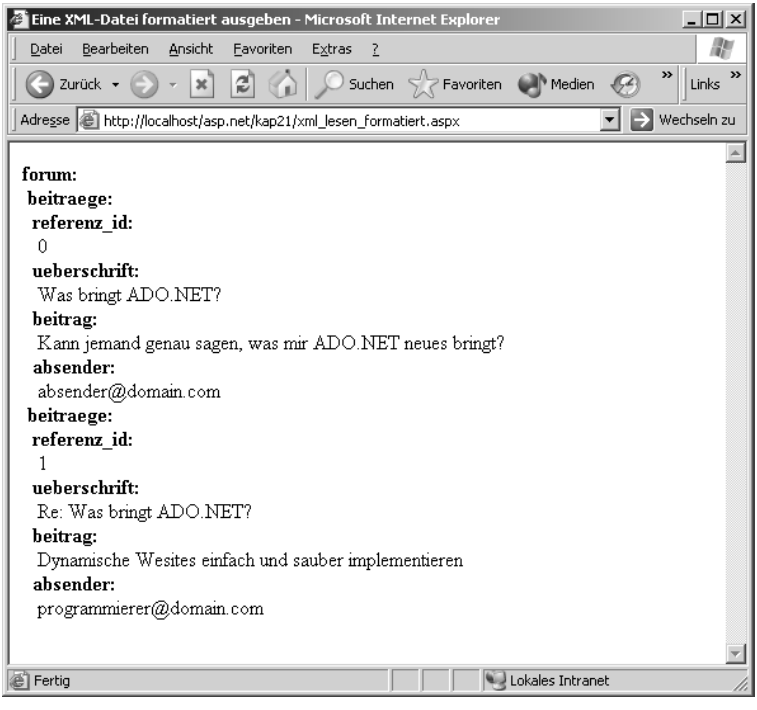



*Bild 18.1: Eine XML-Datei formatiert ausgegeben*

## 18.2 Schreiben mit dem XMLWriter

Genau wie der `XmlReader` darauf optimiert ist, einzelne XML-Dateien zu lesen, so können Sie mit dem `XmlWriter` neue XML-Dokumente erstellen. Dazu dienen eine Reihe von Methoden, wobei die wichtigsten in der folgenden Tabelle zusammengestellt sind:

| Methode | Beschreibung |
|---|---|
| `WriteStartDocument()` | Schreibt den Anfang der XML-Datei gemäß XML 1.0-Standard, also: `<?xml version="1.0"?>` |
| `WriteEndDocument()` | Schließt alle bis dahin noch offenen Elemente |
| `WriteElementString (Name, Wert)` | Schreibt den Inhalt der Variablen `Wert` zwischen ein öffnendes und ein schließendes XML-Tag `Name` |
| `WriteStartElement(Name)` | Schreibt ein XML-Tag `Name` |
| `WriteEndElement()` | Schließt das gerade aktuelle XML-Tag |
| `WriteAttribute-String(Name, Wert)` | Schreibt in das gerade aktuelle XML-Tag ein Attribut mit dem Namen `Name` und dem Wert `Wert` |
| `WriteCData(CData)` | Schreibt den Inhalt `CData` in einen CData-Bereich |
| `WriteComment(Wert)` | Schreibt den Kommentar `Wert` |
| `Flush()` | Übernimmt die im Speicherinhalt des Rechners zusammengefügten XML-Inhalte und schreibt diese in eine Datei |
| `Close()` | Beendet den Zugriff auf die Datei |

*Tabelle 18.2: Methoden der* `XMLWriter`*-Klasse*

Angenommen, Sie wollen eine XML-Datei mit diesem Inhalt erzeugen:

```
<?xml version="1.0" ?>
<projekte>
  <projekt id="1">
    <leiter>Christian Trennhaus</leiter>
    <thema>Umstellung Serversysteme</thema>
  </projekt>
  <projekt id="2">
    <leiter>Christian Wenz</leiter>
    <thema>Digitale Fotografie</thema>
  </projekt>
</projekte>
```

*Listing 18.5: Die zu erstellende XML-Datei*

Dazu genügt dann ein einfaches Script, das mit den gerade vorgestellten Methoden der `XmlWriter`-Klasse arbeitet und diese für ein `XmlTextWriter`-Objekt heranzieht:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextWriter writer = new
   XmlTextWriter(Server.MapPath("projekte.xml"),
   Encoding.UTF8);
  writer.WriteStartDocument();
  writer.WriteStartElement("projekte");
  writer.WriteStartElement("projekt");
  writer.WriteAttributeString("id", "1");
  writer.WriteElementString("leiter", "Christian Trennhaus");
  writer.WriteElementString("thema", "Umstellung Serversysteme");
  writer.WriteEndElement();
  writer.WriteStartElement("projekt");
  writer.WriteAttributeString("id", "2");
  writer.WriteElementString("leiter", "Christian Wenz");
  writer.WriteElementString("thema", "Digitale Fotografie");
  writer.WriteEndDocument();
  writer.Flush();
  writer.Close();
}
</script>
<html>
<head>
<title>Eine XML-Datei erzeugen</title>
</head>
<body>
  <a href="projekte.xml">Zur Datei</a>
</body>
</html>
```

*Listing 18.6: Erstellen einer XML-Datei (xml_schreiben.aspx)*


*Auch wenn Sie in der Praxis eine XML-Datei selten so statisch erzeugen werden wie hier, so zeigt das Beispiel doch auf, wie leicht die Methoden des* `XmlTextWriters` *anzuwenden sind. Mit diesem Wissen können Sie auf der gleichen Basis ebenso dynamische Scripte zur Erzeugung von XML-Dateien erstellen.*

Dass die erzeugte Datei wirklich unserem Ziel entspricht, sehen Sie in nachfolgendem Screenshot:

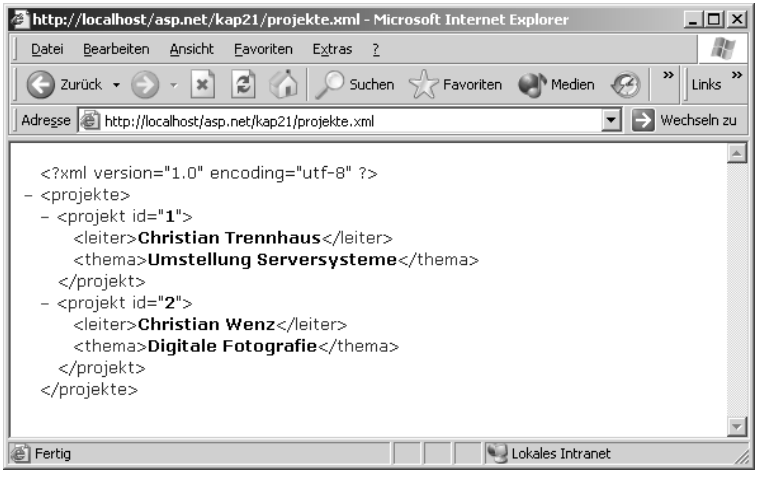



*Bild 18.2: Die zuvor erzeugte XML-Datei im Browser*

## 18.3 Die Klasse XmlDocument

Der nächste logische Schritt ist nun, bestehende XML-Dokumente zu verändern. Dabei helfen leider die Klassen `XmlReader` und `XmlWriter` nicht weiter. Stattdessen müssen Sie auf Methoden und Eigenschaften der Klasse `XmlDocument` zurückgreifen.

### 18.3.1 XML-Dateien ändern

Um beispielsweise der gerade erstellten XML-Datei *projekte.xml* einen weiteren Datensatz hinzuzufügen, gehen Sie wie folgt vor:

Zuerst müssen Sie ein `XmlDocument`-Objekt initialisieren und die XML-Datei laden:

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("projekte.xml"));
```

Um einen neuen Datensatz zu den Bestehenden hinzuzufügen, ist es am einfachsten, einen bereits Bestehenden zu kopieren und dann die Kopie zu verändern.

Suchen Sie sich also den ersten Datensatz:

```
XmlElement elem = doc.DocumentElement;
elem = (XmlElement)elem.FirstChild;
```

Der erste Datensatz ist das erste Element unterhalb des Root-Elements `DocumentElement`. Die Variable `elem` verweist nun auf den ersten Datensatz, so dass Sie diesen kopieren können.

```
elem = (XmlElement)elem.CloneNode(true);
```




*Wenn Sie beim Kopieren eines XML-Knotens wollen, dass auch alle untergeordneten Elemente mit kopiert werden, rufen Sie die* `CloneNode`*-Methode mit dem Parameter* `true` *auf.*

Im Anschluss werden die Elemente der Kopie einzelnen Variablen zugewiesen. Beachten Sie, dass Sie mit Hilfe der `NextSibling`-Methode von einem auf den nächsten Knoten springen können:

```
XmlElement leiter = (XmlElement)elem.FirstChild;
XmlElement thema = (XmlElement)leiter.NextSibling;
```

Diesen Variablen können nun neue Werte zugewiesen werden. Dies geschieht mit Hilfe der `InnerText`-Eigenschaft:

```
leiter.InnerText = pleiter.Text;
thema.InnerText = pthema.Text;
```

Nun wird es interessant: Der XML-Knoten `projekt` hat in diesem Beispiel ein Attribut `id` mit einer Zahl als Wert, die als fortlaufender Zähler fungieren soll. Also muss dieses Attribut auch in der Kopie des ersten Datensatzes, in der Sie gerade arbeiten, angepasst werden. Da der gerade bearbeitete XML-Knoten am Schluss der Datei angehängt werden soll, ist der erste Schritt nun, den Wert des entsprechenden Attributs im derzeit noch letzten Knoten zu lesen. An das letzte Element gelan-

gen Sie über die `LastChild`-Methode, die Sie auf den Root-Knoten des Dokuments anwenden:

```
XmlElement root = doc.DocumentElement;
XmlElement letztes = (XmlElement)root.LastChild;
```

Von diesem XML-Knoten können Sie nun den Wert des Attributes `id` auslesen und einen um eins erhöten Wert Ihrer Kopie zuweisen.

```
String id = letztes.GetAttribute("id");
Int16 i = Convert.ToInt16(id);
i++;
id = i.ToString();
elem.SetAttribute("id", id);
```

Den nun im Speicher vorbereiteten XML-Knoten können Sie nun an das eingelesene XML-Dokument anhängen und speichern:

```
root.AppendChild(elem);
doc.Save(Server.MapPath("projekte.xml"));
```

Das folgende Listing stellt ein Formular bereit, über das die bestehende XML-Datei *projekte.xml* verändert werden kann:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void speichern(Object Sender, EventArgs e)
{
  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("projekte.xml"));
  XmlElement elem = doc.DocumentElement;
  elem = (XmlElement)elem.FirstChild;
  elem = (XmlElement)elem.CloneNode(true);
  XmlElement leiter = (XmlElement)elem.FirstChild;
  XmlElement thema =(XmlElement)leiter.NextSibling;
  leiter.InnerText = pleiter.Text;
  thema.InnerText = pthema.Text;
  XmlElement root = doc.DocumentElement;
  XmlElement letztes = (XmlElement)root.LastChild;
  String id = letztes.GetAttribute("id");
  Int16 i = Convert.ToInt16(id);
  i++;
  id = i.ToString();
  elem.SetAttribute("id", id);
  root.AppendChild(elem);
```

```
  doc.Save(Server.MapPath("projekte.xml"));
}
</script>
<html>
<head>
  <title>Eine XML-Datei verändern</title>
</head>
<body>
<form runat="server">
  Projektleiter:
  <asp:TextBox id="pleiter" runat="server" /><br />
  Thema des Projekts:
  <asp:TextBox id="pthema" runat="server" /><br />
  <asp:Button text="Speichern" onclick="speichern"
   runat="server" /><br />
</form>
  <a href="projekte.xml">Zur Datei</a>
</body>
</html>
```

*Listing 18.7: Ändern einer XML-Datei über ein Formular (xml_aendern.aspx)*

### 18.3.2 XML-Dateien durchsuchen

Der zweite Anwendungsfall, in dem Sie Methoden der `XmlDocument`-Klasse verwenden sollten, ist das Durchsuchen von XML-Dateien.

Angenommen, Sie möchten alle Projekte eines bestimmten Projektleiters ausgeben. Zunächst müssen Sie dazu wieder ein Objekt der Klasse `XmlDocument` initialisieren und öffnen. Dann gilt es, die XML-Datei nach allen XML-Tags `leiter` zu durchsuchen:

```
XmlNodeList liste =
 doc.GetElementsByTagName("leiter");
```

Innerhalb dieser Liste können Sie nun diejenigen Datensätze herausfiltern, in denen auch der passende Projektleiter gespeichert ist:

```
foreach (XmlNode element in liste) {
  if (element.InnerText == pleiter.Text) {
```

Immer dann, wenn ein passender Datensatz gefunden wurde, merken Sie sich das zugehörige Projekt. Dieses ist als Inhalt des nachfolgenden Knotens gespeichert:

```
projektliste.Add(element.NextSibling.InnerText);
treffer++;
```

Im Weiteren gilt es nurmehr, die gewonnenen Ergebnisse zur Ausgabe aufzubereiten:

```
ausgabe.Text = pleiter.Text + " leitet derzeit ";
if (treffer > 0) {
  ausgabe.Text += "folgende Projekte:<br />";
  for (int i=0; i<treffer; i++) {
    ausgabe.Text += "- " + projektliste[i];
    ausgabe.Text += "<br />";
  }
} else {
  ausgabe.Text += " keine Projekte.";
}
```

Wie Sie sehen, ist auch das Durchsuchen von XML-Dokumenten mit Hilfe der `XmlDocument`-Klasse recht einfach. Am kompletten Quellcode erkennen Sie auch, dass die eigentliche Suche sehr kurz umzusetzen ist und lediglich von viel Code zur Gestaltung der Ausgabe umgeben wird:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void suchen(Object Sender, EventArgs e)
{
  XmlDocument doc = new XmlDocument();
  doc.Load(Server.MapPath("projekte.xml"));
  ArrayList projektliste = new ArrayList();
  int treffer = 0;
  XmlNodeList liste = doc.GetElementsByTagName("leiter");
  foreach (XmlNode element in liste) {
    if (element.InnerText == pleiter.Text) {
      projektliste.Add(element.NextSibling.InnerText);
      treffer++;
    }
  }
  ausgabe.Text = pleiter.Text + " leitet derzeit ";
  if (treffer > 0) {
    ausgabe.Text += "folgende Projekte:<br />";

    for (int i=0; i<treffer; i++) {
      ausgabe.Text += "- " + projektliste[i];
```

```
      ausgabe.Text += "<br />";
    }
  } else {
    ausgabe.Text += " keine Projekte.";
  }
}
</script>
<html>
<head>
  <title>Eine XML-Datei durchsuchen</title>
</head>
<body>
<form runat="server">
  <asp:Label id="ausgabe" runat="server" />
<br />
  Suche nach Projekten von Projektleiter:
  <asp:TextBox id="pleiter" runat="server" /><br />
  <asp:Button text="Suchen" onclick="suchen"
   runat="server" /><br />
</form>
</body>
</html>
```

*Listing 18.8: Durchsuchen einer XML-Datei (xml_suchen.aspx)*


*Bild 18.3: Durchsuchen einer XML-Datei*

## 18.4 Zusätzliche Aufgaben aus dem XML-Umfeld

Neben dem Lesen, Schreiben, Ändern und Durchsuchen gibt es bei XML-Dateien noch zwei weitere Aufgaben, die regelmäßig umgesetzt werden müssen:

- Das Validieren von XML-Dokumenten gegen ein Schema oder eine DTD-Definition
- Das Anwenden eines XSLT-Stylesheets auf eine bestehende XML-Datei

Mit diesen beiden Themen beschäftigt sich der letzte Abschnitt in diesem Kapitel.

### 18.4.1 XML-Dateien validieren

XML-Dateien sind streng strukturierte Daten-Repositories, die den von Ihnen festgelegten Regeln folgen müssen. Sobald XML-Dateien professionell eingesetzt werden sollen, ist es ratsam, diese vor Verarbeitung der XML-Datei zu validieren. Denn nur mit einer validierten XML-Datei ist sichergestellt, dass alle Eigenschaften und Methoden, mit denen Sie auf der Datei arbeiten wollen, auch ohne Fehler arbeiten können.

#### Eine XML-Datei gegen eine DTD validieren

Gerade beim Austausch von Daten ist es üblich, dass für die übertragenen Dateien eine Document Type Definition (DTD) vorliegt. Diese Datei beschreibt alle Elemente und deren Verschachtelungsmöglichkeiten. So wird in der folgenden XML-Datei bereits auf eine lokal vorhandene DTD verwiesen:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE projekte SYSTEM "projekte.dtd">
<projekte>
  <projekt id="1">
    <leiter>Christian Trennhaus</leiter>
    <thema>Umstellung Serversysteme</thema>
  </projekt>
  <projekt id="2">
    <leiter>Christian Wenz</leiter>
    <thema>Digitale Fotografie</thema>
  </projekt>
</projekte>
```

*Listing 18.9: XML-Datei mit Verweis auf eine DTD (projekte_dtd.xml)*

Die eigentliche DTD-Datei ist sehr kurz:

```
<!ELEMENT projekte (projekt)*>
<!ELEMENT projekt (leiter+, thema+)>
<!ATTLIST projekt id CDATA #REQUIRED>
<!ELEMENT leiter (#PCDATA)>
<!ELEMENT thema (#PCDATA)>
```

*Listing 18.10: Eine DTD, die die Projektliste beschreibt (projekte.dtd)*

Um nun zu überprüfen, ob obige XML-Datei den Anforderungen der DTD genügt, dient im .NET Framework die Klasse `XmlValidatingReader`. Ein Objekt dieser Klasse benötigt zum Initialisieren ein Objekt der `XmlTextReader`-Klasse:

```
XmlTextReader reader = new
 XmlTextReader(Server.MapPath("projekte_dtd.xml"));
XmlValidatingReader pruefer = new
 XmlValidatingReader(reader);
```

Über ein Objekt der `XmlValidatingReader`-Klasse können Sie gegen eine DTD genau wie gegen ein Schema prüfen. Daher sollten Sie zunächst festlegen, welche Art der Validierung Sie vornehmen möchten:

```
pruefer.ValidationType = ValidationType.DTD;
```

Nun können Sie die Validierung anstoßen. Dazu lassen Sie den Validierer einfach jede Zeile Ihres XML-Dokuments einlesen:

```
while (pruefer.Read()) {
}
```

Eine einfache Validierung eines XML-Dokuments wird also bereits mit diesem Quellcode möglich:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextReader reader = new
  XmlTextReader(Server.MapPath("projekte_dtd.xml"));
  XmlValidatingReader pruefer = new
   XmlValidatingReader(reader);
  pruefer.ValidationType = ValidationType.DTD;
```

```
  while (pruefer.Read()) {
  }
  pruefer.Close();
  reader.Close();
}
</script>
<html>
<head>
<title>Eine XML-Datei prüfen</title>
</head>
<body>
  Test erfolgreich verlaufen.
</body>
</html>
```

*Listing 18.11: Einfache Validierung von XML-Dokumenten (xml_pruefen_dtd.aspx)*

Auch wenn diese Validierung funktioniert, so hat sie noch einen entscheidenden Haken: Ist die Validierung nicht erfolgreich – das XML-Dokument genügt also nicht seiner DTD –, dann bricht auch das ASP.NET-Dokument mit einer Fehlermeldung ab. Daher werden XML-Ausnahmen im folgenden Script sauber abgefangen und ausgegeben:



```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Schema" %>
<script runat="Server">
public Boolean allesok = true;
void Page_Load(Object Sender, EventArgs e)
{
  XmlTextReader reader = new
  XmlTextReader(Server.MapPath("projekte_dtd.xml"));
  XmlValidatingReader pruefer = new XmlValidatingReader(reader);
  pruefer.ValidationType = ValidationType.DTD;
  pruefer.ValidationEventHandler += new
   ValidationEventHandler(this.Meldung);
  while (pruefer.Read()) {
  }
  if (allesok) {
    ausgabe.Text = "Test erfolgreich verlaufen.";
  }
```

```
  pruefer.Close();
  reader.Close();
}
public void Meldung (object sdr, ValidationEventArgs args) {
  allesok = false;
  ausgabe.Text += args.Message + "<br />";
}
</script>
<html>
<head>
<title>Eine XML-Datei prüfen</title>
</head>
<body>
  <asp:label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 18.12: Durch Fehlerbehandlung optimierte Prüfung eines XML-Dokuments (xml_pruefen_optimiert.aspx)*

Auch wenn die Ausnahmebehandlung in diesem Beispiel nicht sehr schön ist, so erfüllt diese doch ihren Zweck. Entfernen Sie beispielsweise bei einem Datensatz im Beispiel den Knoten `leiter`, so kommt es zu folgender Ausgabe im Browser:

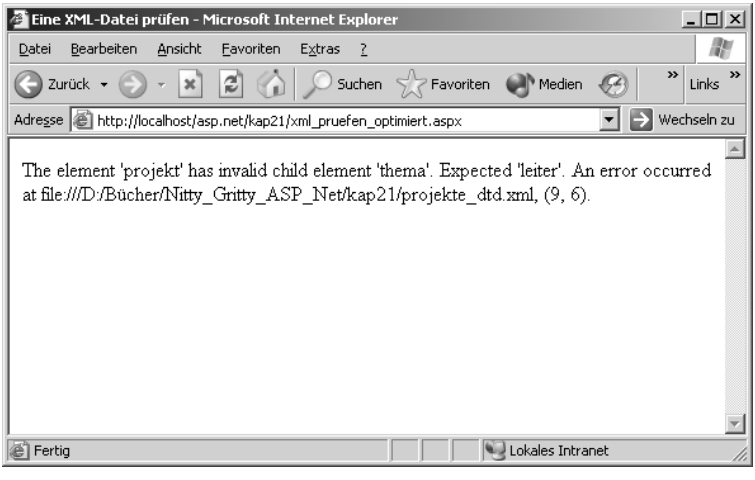



*Bild 18.4: Ausgabe des Fehlers bei einer XML-Validierung*

**Eine XML-Datei gegen ein Schema validieren**

Unter ASP.NET können Sie nicht nur gegen eine DTD, sondern auch gegen ein XML-Schema validieren. Dazu können Sie den Code aus dem vorangegangenen Beispiel (*xml_pruefen_optimiert.aspx*) fast zu hundert Prozent übernehmen. Ändern Sie lediglich

1. den Dateinamen beim Aufbau des `XmlTextReader`-Objektes, um auf die aktuell zu validierende Datei zu verweisen (`XmlTextReader reader = new XmlTextReader(Server.MapPath("projekte_dtd.xml"));`)
2. den Validierungstyp:
   `pruefer.ValidationType = ValidationType.Schema;`

Derart angepasst erzielen Sie das gleiche Ergebnis wie bei der Validierung gegen eine DTD.

### 18.4.2 Mit XSLTs arbeiten

Zum Abschluss dieses Kapitels wollen wir uns noch kurz mit XSLTs unter ASP.NET befassen. Eine XSLT-Datei enthält als Stylesheet zur Transformation von XML-Dateien in andere Beschreibungssprachen wie HTML oder PDF alle notwendigen Gestaltungselemente, um durch entsprechendes Transformieren aus der XML-Datei ein Zielformat zu erzeugen.





Ein Beispiel für eine derartige XSLT-Datei ist die Datei *projekte.xsl*:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/
 XSL/Transform" version="1.0">
  <xsl:output indent="yes" method="html" />
  <xsl:template match="/">
    <xsl:apply-templates />
  </xsl:template>
  <xsl:template match="projekte">
    <html>
      <head>
        <title>Aktuelle Projekte</title>
      </head>
    <body>
      <table width="400" border="1">
        <tr>
          <th>Projektleiter</th>
```

```
          <th>Thema</th>
        </tr>
        <xsl:apply-templates select="projekt" />
      </table>
    </body>
    </html>
  </xsl:template>
  <xsl:template match="projekt">
    <tr>
      <td><xsl:value-of select="leiter" /></td>
      <td><xsl:value-of select="thema" /></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

*Listing 18.13: Die XSLT-Datei projekte.xsl*

Dieses Beispiel ermöglicht es, durch Transformation aus der Projektliste *projekte.xml* eine ansehnliche HTML-Datei zu erzeugen. Die Transformation lässt sich in ASP.NET über Methoden der Klasse `XmlTransform` durchführen.

Das Anwenden des XSLT-Stylesheets geschieht in nur wenigen Schritten:

Laden Sie das gewünschte XSLT-Stylesheet mit der `Load`-Methode eines `XmlTransform`-Objekts:

```
XslTransform xslt = new XslTransform();
xslt.Load(Server.MapPath("projekte.xsl"));
```

Dieses XSLT-Stylesheet können Sie nun anwenden. Die `Transform`-Methode erstellt mit Hilfe des geladenen Stylesheets aus der übergebenen Quelldatei eine Zieldatei als Ergebnis der Transformation:

```
xslt.Transform(Server.MapPath("projekte.xml"),
 Server.MapPath("projekte.htm"));
```

Das war es auch schon. Den gesamten Quellcode zum Ausführen einer XSLT-Transformation sehen Sie hier:

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Xml.Xsl" %>
<script runat="Server">
void Page_Load(Object Sender, EventArgs e)
{
  XslTransform xslt = new XslTransform();
  xslt.Load(Server.MapPath("projekte.xsl"));
  xslt.Transform(Server.MapPath("projekte.xml"),
   Server.MapPath("projekte.htm"));
}
</script>
<html>
<head>
<title>Ein XSLT-Stylesheet anwenden</title>
</head>
<body>
  <a href="projekte.htm">projekte.htm</a>
</body>
</html>
```

*Listing 18.14: Eine XSLT-Transformation ausführen (xslt_anwenden.aspx)*

# 19 Web Services

Mit dem rasanten Wachstum des Internets wurden auch die Anwendungen, die den Benutzern geboten werden, immer komplexer. War es vor einigen Jahren noch etwas Besonderes, eine Website dynamisch mit Kurznachrichten zu versehen, so ist ein derartiger auf Datenbankzugriffen basierender Service heute beinahe schon selbstverständlich. Die immer aufwändiger werdenden Anwendungen stoßen aber immer häufiger auch an eine logische Grenze: Die für die Anwendung erforderlichen Daten können nicht mehr komplett vom eigenen Unternehmen bereitgestellt werden. So ist es zwar wünschenswert, auf der Homepage eines beliebten Ausflugsziels gleich noch eine knappe Wettervorhersage eingebunden zu finden, doch kann sich nicht jede Gemeinde eine kleine Wetterstation leisten. Der Bedarf nach individuell in eine Anwendung zu integrierenden Fremddaten besteht also immer häufiger – und genau in diesem Umfeld greifen Web Services an.

## 19.1 Was ist ein Web Service?

Kurz gesagt ist ein Web Service eine Schnittstelle, über die Sie anderen Funktionen und Daten bereitstellen können, über die Sie lokal auf Ihrem Server verfügen. Diese Schnittstelle können Sie natürlich auch als Benutzer verwenden, um Daten für Ihre Applikation zu laden – in diesem Falle konsumieren Sie einen Web Service.

Was ein Web Service aber genau ist und wie diese Architektur im .NET Framework unterstützt wird, zeigt der nun folgende Abschnitt.

### 19.1.1 Verteilte Anwendungen

Das Prinzip der verteilten Anwendungen ist nichts Neues. So wurden bereits zu den Hochzeiten der Großrechner Anwendungen auf mehrere Systeme verteilt und mittels proprietärer Schnittstellen erforderliche Daten on-the-fly ausgetauscht. Um sich von diesen Individuallösungen weiter zu lösen, wurden Architekturen wie *DCOM* und *RPC* entwickelt. *DCOM* steht für *Distributed Component Object Model* und bietet wie die *Remote Procedure Calls (RPC)* in der Theorie alles, was Sie zur Erstellung einer verteilten Anwendung benötigen. So können

Sie mit Hilfe dieser Techniken Parameter übertragen und auf entfernten Systemen Anfragen ausführen, die von diesen Parametern beeinflusst werden. Natürlich wird auch das Ergebnis der Abfrage an Sie zurück übertragen. Der Teufel steckt bei diesen Architekturen jedoch im Detail. So müssen zunächst alle beteiligten Serversysteme im gleichen Netzwerk ansprechbar sein, da DCOM- und RPC-Aufrufe von Firewalls blockiert werden (andernfalls wäre ja die Möglichkeit gegeben, unkontrollierbar Server und Daten von außerhalb zu manipulieren). Andererseits müssen alle beteiligten Systeme binärkompatibel sein. Dazu kommt noch, dass das Einrichten einer verteilten Anwendung mit diesen Architekturen alles andere als trivial ist. So müssen Sie systemnahe *dll*-Dateien erstellen und diese in der Registry der entsprechenden Server einrichten. All diese Umstände tragen dazu bei, dass DCOM und RPC nicht dazu geeignet sind, um unternehmensübergreifende verteilte Anwendungen zu erstellen. Der Bedarf nach einer alternativen Vorgehensweise ist also gegeben.

### 19.1.2 Web Services Architektur

Der Ansatz von Web Services musste es also sein, proprietäre Schnittstellen und Protokolle durch die Nutzung von Standards zu umgehen. Um eine möglichst allgemeingültige und universell einsetzbare Architektur zu schaffen, wurde unter Mitwirkung der großen Softwarehersteller durch die W3C ein neuer Standard ausgearbeitet.

Bei Web Services erfolgt die Übermittlung von Parametern und Abfragen genau wie die Übertragung der Ergebnisse über das HTTP-Protokoll. Zudem werden Daten im XML-Format übertragen.

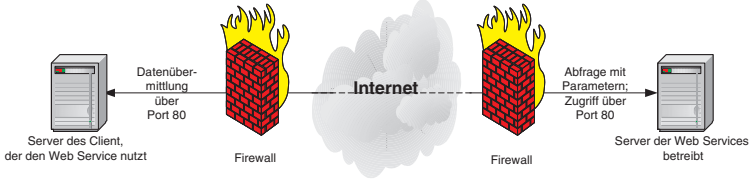



*Bild 19.1: Datenübermittlung bei Web Services*

Aus dieser Vorgehensweise ergeben sich mehrere Vorteile:

- Durch die Verwendung des HTTP-Protokolls in Verbindung mit XML zur Datenübertragung ist es gelungen, sich rein auf standardisierte Technologien zu verlassen. Da diese Basiskomponenten auf jeder Plattform verfügbar sind, ist ein systemübergreifendes Zusammenarbeiten unabhängig von einzelnen beteiligten Systemen möglich.
- Die Installation von Web Services ist im Vergleich zu *DCOM* und *RPC* trivial. Sie müssen weder dll-Dateien registrieren noch Einträge in der Registry pflegen, um einen Web Service zu nutzen oder zu betreiben. Zudem wird durch die Verwendung des Port 80 zur Datenübermittlung erreicht, dass Firewalls in den meisten Fällen nicht mehr speziell konfiguriert werden müssen – ein wichtiges Detail, wenn Sie an den Einsatz einer derartigen Architektur in größeren Unternehmen denken.

**SOAP zur Datenübertragung**

Wie bereits erwähnt, erfolgt die Datenübertragung bei Web Services unter der Verwendung von XML-Formaten. Um auch hier einem allgemeinen Standard zu folgen, findet das *Simple Object Access Protocol* (*SOAP*) zur Übermittlung der Informationen Verwendung. SOAP ist genau wie HTTP plattformübergreifend verfügbar. Dazu kommt, dass sich die gesamte SOAP-Kommunikation in HTTP-Aufrufe integrieren lässt, so dass sie zur Übermittlung aller Daten, wie gewünscht, nur eine Kommunikationsschicht einsetzen lässt. Andererseits macht es erst SOAP möglich, Daten jeglicher Art im XML-Format über HTTP zu versenden.

**WDSL**

Um eine Web Service-Anfrage starten zu können, benötigen Sie vorab einige Informationen. So müssen Sie die URL des Web Services kennen, ehe Sie diesen überhaupt ansprechen können. Diese URL können Sie entweder über ein Verzeichnis dynamisch auslesen oder manuell erfassen (dies ist die weit häufigere Variante). Als Nächstes sollten Sie darüber Kenntnis erlangen, über welche Methoden der Web Service verfügt und welche Parameter die einzelnen Methoden benötigen. Genau diese Informationen werden über die *Web Service Description Language* (*WSDL*) bereitgestellt.

Um ein sog. *WDSL*-Dokument – also ein Dokument in der *WDSL*-Sprache, das alle Informationen über den Web Service enthält – einsehen zu können, genügt es, den Parameter `?WSDL` an die URL eines Web Service anzuhängen.

Ein `WSDL`-Dokument, das einen Web Service zur Validierung von Kreditkarten beschreibt, sieht dann in etwa so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=
 "http://webservices.imacination.com/validate/
 Validate.jws" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apache-
soap="http://xml.apache.org/xml-soap" xmlns:impl="http://webser-
vices.imacination.com/
 validate/Validate.jws" xmlns:intf="http://webservices.imacina-
tion.com/
 validate/Validate.jws" xmlns:soapenc="http://schemas.xmlsoap.org/
soap/
 encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl-
soap="http://schemas.xmlsoap.org/wsdl/
 soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types />
  <wsdl:message name="validateCardRequest">
    <wsdl:part name="ccNumber" type="xsd:string"/>
    <wsdl:part name="ccDate" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="validateNumberResponse">
    <wsdl:part name="validateNumberReturn"
     type="xsd:boolean"/>
  </wsdl:message>
  <wsdl:message name="validateNumberRequest">
    <wsdl:part name="ccNumber" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="validateCardResponse">
    <wsdl:part name="validateCardReturn"
     type="xsd:boolean"/>
  </wsdl:message>
…
  </wsdl:service>
</wsdl:definitions>
```

*Listing 19.1: Auszug aus einem* `WSDL`*-Dokument (http://webservices.imacination.com/validate/Validate.jws?wsdl)*

### Proxy-Klassen

Wenn Sie auf einem Client eine Anwendung erstellen, die zur Ausführung auf einen Web Service zurückgreift, wird die Kompilierung dieser Anwendung zunächst fehlschlagen. Der Grund dafür ist einfach: Bei der Kompilierung einer Anwendung werden alle Funktionsaufrufe daraufhin überprüft, ob die Funktionen so existieren und ob übergebene und rückgelieferte Parameter den richtigen Datentyp haben. In dem Moment, in dem Sie einen Web Service einbinden, kennt der Compiler genau diese Informationen nicht und bricht daher ab.

Um diesen Konflikt zu lösen, gibt es *Proxy*-Klassen. Diese speziellen Klassen halten für den Compiler alle Methoden und Eigenschaften eines Web Services bereit. Der Compiler kann nun den Client-Code ganz genauso kompilieren, wie wenn Sie eine zusätzliche Klassen-Bibliothek hinzugeladen hätten. Beim Aufruf eines Web Services rufen Sie also eigentlich die Proxy-Klasse auf. Diese leitet dann, transparent wie ein HTTP-Proxy, Anfragen an den Server des Web Services und Ergebnisse an Ihre Anwendung weiter.

### Der Ablauf einer Web Service-Anfrage

Offensichtlich wird die Architektur von Web Services, wenn Sie sich den kompletten Ablauf einer Web Service-Anfrage veranschaulichen:

Angenommen Sie wollen einen Web Service nutzen, um eine Kreditkarte auf Gültigkeit zu prüfen. Wenn Sie einen Web Service verwenden, dann spricht man vom *Konsumieren* eines Web Services. Beim Konsumieren sprechen Sie Ihre dafür erstellte Proxy-Klasse an. Um die Proxy-Klasse überhaupt erstellen zu können, hatten Sie das WSDL-Dokument des Web Services genutzt. Die Proxy-Klasse wird zunächst eine HTTP-Anfrage an den Webserver des Dienstleisters senden. Dieser analysiert dann die im SOAP-Bereich der Anfrage enthaltenen Parameter und führt daraufhin die Operationen aus. Die berechnete Antwort wird wieder in eine SOAP-Antwort umgewandelt und über das HTTP-Protokoll an die Proxy-Klasse auf Ihrem Server zurückübertragen. Die Proxy-Klasse entnimmt die angeforderten Daten dem übermittelten SOAP-Paket und gibt sie als Rückgabewerte der aufgerufenen Proxy-Methode zu Weiterverarbeitung an Ihre Anwendung weiter.

### 19.1.3 Web Services in .NET

Gleich vorab eine gute Nachricht: .NET unterstützt Web Services hervorragend. Dazu werden in den Namespaces `System.Web.Services`, `System.Web.Services.Description` und `System.Web.Services.Protocols` eine Reihe von Methoden zur Verfügung gestellt, die das Erstellen und Konsumieren von Web Services enorm vereinfachen. So sorgen diese Klassen dafür, dass die gesamte Kommunikation mit einem Web Service für Sie automatisiert abläuft. Sie müssen sich also niemals darum kümmern, selbst SOAP-Kommunikation zu gestalten. Und beim Erstellen eines Web Services sorgt das .NET Framework dafür, dass zugehörige WSDL-Dokumente immer automatisch generiert werden. Durch diese Vorgehensweise wird zudem sichergestellt, dass angebotene Web Services, genau wie die Verarbeitung von Web Services, immer den gültigen Standards entsprechen. Wie Sie die .NET-Komponenten im Umgang mit Web Services einsetzen können, sehen Sie anhand von praktischen Beispielen in den Abschnitten »Einen Web Service erstellen« und »Web Services konsumieren« später in diesem Kapitel.

## 19.2 Pro und kontra Web Services

Ehe Sie sich blindlings auf das Erstellen und Konsumieren von Web Services stürzen, sollten Sie sich nochmals die Vor- und Nachteile dieser neuen Technologie vor Augen halten. Erst dann können Sie wirklich entscheiden, in welchen Fällen der Einsatz von Web Services für Sie Sinn macht.



**Pro Web Services**

- Web Services sind plattformunabhängig.
- Web Services sind ein anerkannter und weit verbreiteter Standard.
- Durch den Einsatz des .NET Frameworks und ASP.NET sind Web Services extrem skalierbar; Sie können Web Services, die für einen Server entwickelt wurden, einfach auf weitere Web Server kopieren.
- Der konsumierende Client und der Server, welcher einen Web Service betreibt, sind vollkommen unabhängig voneinander.
- Sie können Ihre Software und bereits bestehenden Lösungen als Web Services vermarkten.
- Durch die Verwendung von Port 80 und dem HTTP-Protokoll werden Barrieren zur Einbindung von Fremddaten umgangen.

**Kontra Web Services**

- In einem homogenen LAN sind alternative Möglichkeiten zur Einbeziehung von Fremddaten oft genauso leicht zu implementieren wie Web Services.
- Web Services sind nicht transaktionssicher.
- Wie bei allen HTTP-Aufrufen kann auch für Web Services keine Garantie für Reaktionszeiten gegeben werden.
- Web Services sind nicht für bestimmte Arten von Datentransfer optimiert, woraus eine nur mittlere Performance resultiert.

**Fazit**

Immer dann, wenn Sie eine verteilte Anwendung entwickeln, sollten Sie die Verwendung von Web Services in Ihre Planung mit einbeziehen. Denn Web Services sind leicht zu erstellen und zu konsumieren, dabei plattformunabhängig und skalierbar. Wenn Sie jedoch einen sehr hohen Anspruch an die Performance haben oder gar transaktionell arbeiten müssen, sind klassische Alternativen wie DCOM genau wie `Remoting` von .NET als alternative Lösungsmöglichkeiten in Erwägung zu ziehen.

## 19.3 Einen Web Service erstellen

Mindestens genauso wichtig wie die Theorie ist natürlich auch die praktische Umsetzung von Web Services.

**Ein einfacher Web Service**

Web Services haben unter .NET die Dateiendung *.asmx*. Ein Beispiel für einen von der Funktionalität her trivialen Web Service ist dieses:

```
<%@ WebService language="C#" Class="HalloWelt" %>
using System.Web.Services;

[WebService(Name="HalloWelt", Description="Ein
 einfaches Beispiel für einen Web Service",
 Namespace="http://domain.de/webservices/")]

public class HalloWelt : WebService {
  [WebMethod(Description="Gibt das klassische Hallo
   Welt zurück")]
```

```
  public string DerTest() {
    return "Hallo Welt";
  }
}
```

*Listing 19.2: Der Web Service hallowelt.asmx*

Die erste Neuigkeit im Vergleich zu »normalen« ASP.NET-Dateien ist bereits in der ersten Zeile zu finden. Web Services sind schon zu Beginn der Datei als solche durch

```
<%@ WebService language="C#" Class="HalloWelt" %>
```

deklariert. Neben der Angabe, dass es sich bei dieser Datei um einen Web Service handelt, sind alle öffentlichen Klassen – hier HalloWelt – benannt.

Der nächste Schritt ist die Angabe einiger Parameter, die später im automatisch zum Web Service generierten `WSDL`-Dokument wieder zu finden sind. So werden Name und Beschreibung sowie der `Namespace` des Web Services hier festgelegt:

```
[WebService(Name="HalloWelt", Description="Ein
 einfaches Beispiel für einen Web Service",
 Namespace="http://domain.de/webservices/")]
```

Ab dieser Stelle können Sie C#-Code nach Belieben einfügen. Klassen und Methoden, die als Web Service öffentlich im Zugriff sein sollen, müssen Sie jedoch entsprechend kennzeichnen. Klassen markieren Sie, indem Sie nach dem Namen der Klasse das Schlagwort WebService einfügen:

```
public class HalloWelt : WebService {
...
}
```

Innerhalb veröffentlichter Klassen können Sie nun Methoden einfügen. Jene davon, welche wiederum als Bestand des Web Services angeboten werden sollen, müssen als `WebMethod` hervorgehoben werden:

```
[WebMethod(Description="Gibt das klassische Hallo
 Welt zurück")]
public string DerTest() {
  return "Hallo Welt";
}
```

Bei der Benennung als `WebMethod` können Sie alternativ wie hier wieder eine Beschreibung mit angeben, die in das `WSDL`-Dokument übernommen werden wird.

Wenn Sie diese Datei nun mit einem Browser abrufen, erhalten Sie ein durch das .NET Framework etwas anschaulicher gestaltetes `WSDL`-Dokument.

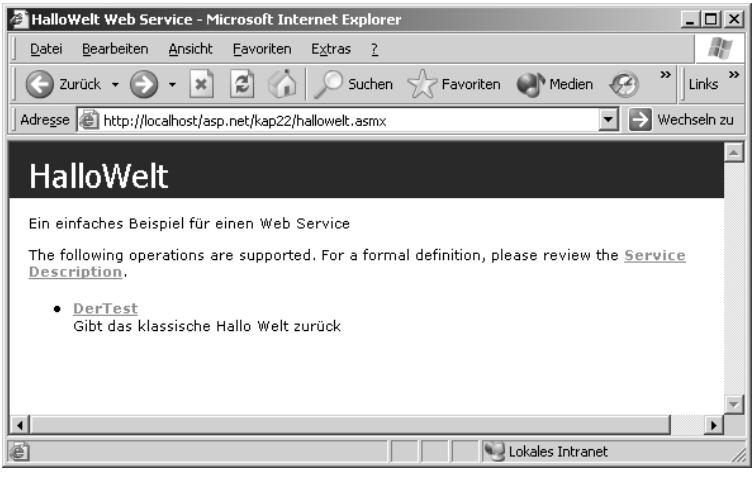



*Bild 19.2: Der Web Service HalloWelt im Browser*

Wie Sie sehen, werden Name und Beschreibung des Web Services angegeben und alle im Service enthaltenen öffentlichen Methoden mit den von Ihnen erstellten Beschreibungstexten gelistet. Wenn ein Client diesen Web Service konsumiert, dann erhält dieser wie gewünscht den String Hallo Welt als Ergebnis.

### Parameter in Web Services

Zur Vertiefung wollen wir nun noch einen weiteren Web Service erstellen. In Kapitel 18 (XML) wurde in einem Beispiel an ein XML-Dokument ein neuer Datensatz hinzugefügt. Die Daten für diesen anzuhängenden Datensatz kamen dabei aus einem Formular. Das dort verwendete Script *xml_aendern.aspx* wird hier als Grundlage wieder verwendet, um daraus einen Web Service zu erstellen. Der Web Service wird zwei Übergabeparameter erwarten:

```
public string Projekte(string pleiter,
 string pthema) {
...
}
```

Als Ergebnis werden alle Datensätze der XML-Datei als `string` an den Konsumenten übergeben. Dazu wird eine Methode angesprochen, deren Basis der Code aus dem Beispiel *xml_lesen_formatiert.aspx* (ebenfalls aus dem XML-Kapitel) liefert. Der gesamte Code für den neuen Web Service lautet dann:

```
<%@ WebService language="C#" Class="DatenAendern" %>
using System.Web.Services;
using System.Xml;
using System;

[WebService(Name="DatenAendern", Description="Aus
 bestehendem Code entwickelter Web Service",
 Namespace="http://domain.de/webservices/")]

public class DatenAendern : WebService {
  public string ausgabe;

  [WebMethod(Description="Hängt einen Datensatz an
   eine bestehende XML-Datei an und gibt diese
   anschliessend als string zurück")]
  public string Projekte(string pleiter,
   string pthema) {
    XmlDocument doc = new XmlDocument();
    doc.Load(Server.MapPath("projekte.xml"));
    XmlElement elem = doc.DocumentElement;
    elem = (XmlElement) elem.FirstChild;
    elem = (XmlElement) elem.CloneNode(true);
    XmlElement leiter = (XmlElement)elem.FirstChild;
    XmlElement thema=(XmlElement)leiter.NextSibling;
    leiter.InnerText = pleiter;
    thema.InnerText = pthema;
    XmlElement root = doc.DocumentElement;
    XmlElement letztes =(XmlElement) root.LastChild;
    string id = letztes.GetAttribute("id");
    Int16 i = Convert.ToInt16(id);
    i++;
    id = i.ToString();
    elem.SetAttribute("id", id);
    root.AppendChild(elem);
```

```
    doc.Save(Server.MapPath("projekte.xml"));
    return XmlLesen(Server.MapPath("projekte.xml"));
  }

  public string XmlLesen(string pfad) {
    XmlTextReader reader = new XmlTextReader(pfad);
    while (reader.Read()) {
      int i;
      if (reader.NodeType == XmlNodeType.Element) {
        for (i=1; i<=reader.Depth; i++) {
          ausgabe += " ";
        }
        ausgabe += "<b>" + reader.Name;
        ausgabe += ":</b><br />";
      }
      if (reader.NodeType == XmlNodeType.Text) {
        for (i=1; i<=reader.Depth; i++) {
          ausgabe += " ";
        }
        ausgabe += reader.Value + "<br />";
      }
    }
    return ausgabe;
    reader.Close();
  }
}
```

*Listing 19.3: Ein aus bestehenden ASP.NET-Seiten entwickelter Web Service (xml_aendern.asmx)*

## 19.4 Web Services konsumieren

Das Konsumieren eines Web Services in Ihrer ASP.NET-Anwendung ist etwas aufwändiger, als es das Erstellen der Web Services war. Prinzipiell läuft das Konsumieren eines Web Services immer nach folgendem Schema ab:

1. Die URL des Web Services herausfinden;
2. das WSDL-Dokument analysieren, um angebotene Methoden und verwendete Parameter kennen zu lernen;
3. eine Proxy-Klasse für den Web Service erstellen;
4. die Methoden der Proxy-Klasse in den eigentlichen Code einbinden – und das Konsumieren kann beginnen.

Der erste Schritt, das Herausfinden der URL des zu verwendenden Web Services, ist meist einfach: Sie können sie von dem Dienstleister erfragen, der den Web Service anbietet.


*Wenn Sie nach einem speziellen Service Ausschau halten, so werden Sie vielleicht in speziellen Web-Services-Verzeichnissen fündig. So werden auf Websites wie http://www.salcentral.com/ oder http://www.flash-db.com/services/ verschiedenste Web Services beschrieben, wobei die URL für jeden dieser Web Services selbstverständlich mit angegeben ist.*

**Ein WSDL-Dokument analysieren**

Im nächsten Schritt wollen wir das WSDL-Dokument des vorhin erstellten Web Services *xml_aendern.asmx* analysieren. Dieses XML-Dokument erhalten Sie, indem Sie den Web Service mit dem Parameter *?WSDL* über einen Browser aufrufen. In einer der ersten Zeilen werden alle in dem Web Service angebotenen Methoden mit Ihren Parametern genannt:

```
<s:element name="Projekte">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
       name="pleiter" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1"
       name="pthema" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
```

*Listing 19.4: Auszug aus dem WSDL-Dokument des Web Services xml_aendern.asmx*

Der Web Service, der hier konsumiert werden soll, hat nur eine Methode Projekte mit den beiden `String`-Parametern pleiter und pthema.

Handelt es sich wie hier um einen Web Service, der mit ASP.NET erstellt wurde, so können Sie diesen sogar testen. Rufen Sie dazu den Web Service im Browser ohne jegliche Parameter auf, und folgen Sie dem Link hinter einer angebotenen Methode.

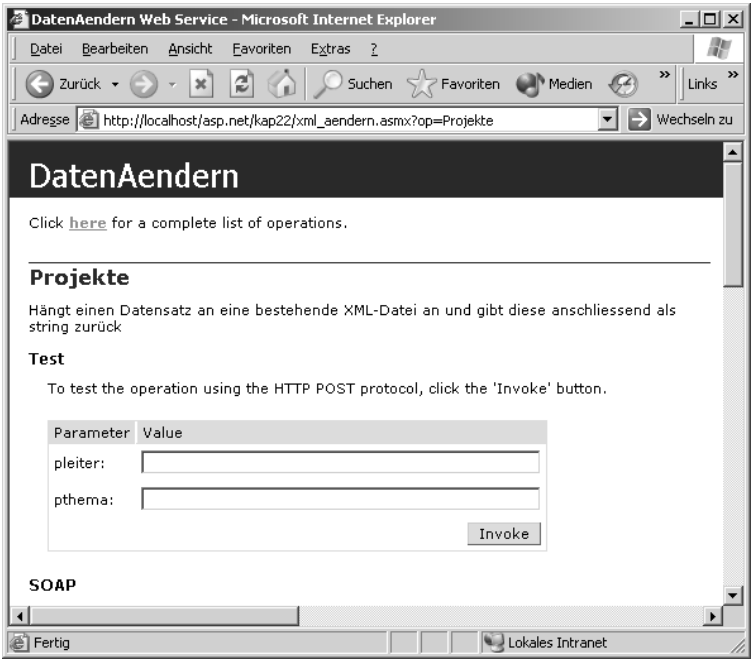



*Bild 19.3: Das durch .NET automatisch erstellte Testformular für einen Web Service*

Im nun geöffneten Formular können Sie Parameter in die entsprechenden Textfelder eintragen und an den Web Service durch einen Klick auf INVOKE senden. Das Resultat, das der Web Service zurückliefert, wird in einem neuen Browserfenster ausgegeben.

### Die Proxy-Klasse erstellen

Wie bereits im Abschnitt Web Services Architektur beschrieben, benötigen Sie eine Proxy-Klasse, um aus Ihrer Anwendung heraus einen Web Service konsumieren zu können. Die Proxy-Klasse spiegelt alle Methoden und Eigenschaften des Web Services, kann jedoch im Gegensatz zum eigentlichen Web Service beim Kompilieren Ihrer Anwendung lokal mit eingebunden werden – und erst dadurch kann Ihre Anwendung überhaupt erfolgreich kompiliert werden.

Mit dem .NET Framework wird auf Ihrem Rechner ein Dienstprogramm mit installiert, mit dem Sie Proxy-Klassen für Web Services sehr leicht erstellen können. Dieses Programm heißt *wsdl.exe* und wird von der Kommandozeile aus gestartet.

```
wsdl [optionen] {URL | Pfad}
```

Das Dienstprogramm erfordert als Parameter entweder die vollständige URL zu einem Web Service oder den Pfad zu einem abgespeicherten WSDL-Dokument. Zusätzlich gibt es einige Optionen, mit denen Sie das Programm steuern können, die wichtigsten sind hier aufgelistet:

| Option | Beschreibung |
|---|---|
| /l[anguage] | Legt die Programmiersprache fest, in der die automatisch generierte Klasse erstellt werden soll. Mögliche Werte sind vb, cs und js. |
| /n[amespace] | Sie können einen eigenen Namespace für die Proxy-Klasse angeben, den Sie dann natürlich in dem konsumierenden Programm mit einbeziehen müssen. |
| /protocol | Bestimmt, mittels welchen Protokolls mit dem Web Service kommuniziert werden soll. Mögliche Werte sind SOAP (Standard), HttpPost und HttpGet. |
| /proxy | Wenn Sie über einen HTTP-Proxy mit dem Web Service kommunizieren wollen, geben Sie diesen hier an. Ohne diese Angabe werden die Standardeinstellungen Ihres Rechners übernommen. |
| /o[ut] | Hinter diesem Flag können Sie den Namen der generierten Datei abgeben. |
| /u[sername]<br>/p[assword]<br>/d[omain] | Mit diesen Optionen können Sie Benutzernamen, Passwort und Domain zur Authentifizierung an einen Web Service übergeben. |

*Tabelle 19.1: Optionen des Kommandozeilen-Programms wsdl.exe*

Um für den Web Service *xml_aendern.asmx* eine Proxy-Klasse erstellen zu lassen, genügt beispielsweise folgender Aufruf:

```
wsdl /l:cs /n:xml_aendern /o:xml_aendern.cs
 http://localhost/asp.net/kap22/xml_aendern.asmx
```

*Listing 19.5: Aufruf des Dienstprogramms wsdl.exe zur Generierung einer Proxy-Klasse*

Der Code, den dieser Aufruf erzeugt, lautet:

```
//------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.1.4322.510
//
//     Changes to this file may cause incorrect
//     behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------

//
// This source code was auto-generated by wsdl,
// Version=1.1.4322.510.
//
namespace xml_aendern {
 using System.Diagnostics;
 using System.Xml.Serialization;
 using System;
 using System.Web.Services.Protocols;
 using System.ComponentModel;
 using System.Web.Services;


/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute ("code")]
[System.Web.Services.WebServiceBindingAttribute
 (Name="DatenAendernSoap", Namespace="http://domain.de/webservices/")]
public class DatenAendern :
System.Web.Services.Protocols.SoapHttpClientProtocol {
    /// <remarks/>
    public DatenAendern() {
        this.Url =
 "http://localhost/asp.net/kap22/xml_aendern.asmx";
    }
```

```
    /// <remarks/>
    [System.Web.Services.Protocols.
     SoapDocumentMethodAttribute
     ("http://domain.de/webservices/Projekte",
 RequestNamespace="http://domain.de/webservices/",
 ResponseNamespace="http://domain.de/webservices/",
 Use=System.Web.Services.Description.SoapBindingUse.
 Literal,
 ParameterStyle=System.Web.Services.Protocols.
 SoapParameterStyle.Wrapped)]
    public string Projekte(string pleiter,
     string pthema) {
        object[] results = this.Invoke("Projekte",
         new object[] {
                    pleiter,
                    pthema});
        return ((string)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult BeginProjekte
     (string pleiter, string pthema,
      System.AsyncCallback callback,
      object asyncState) {
        return this.BeginInvoke("Projekte",
         new object[] {
                    pleiter,
                    pthema}, callback, asyncState);
    }

    /// <remarks/>
    public string EndProjekte(System.IAsyncResult
     asyncResult) {
        object[] results =
         this.EndInvoke(asyncResult);
        return ((string)(results[0]));
    }
  }
}
```

*Listing 19.6: Durch das Dienstprogramm wsdl.exe erzeugte Proxy-Klasse xml_aendern.cs*

*Wenn Sie sich den vom Dienstprogramm erzeugten Quellcode genau ansehen, sehen Sie, was die Proxy-Klasse eigentlich macht: Alle Methoden, die der Web Service zur Verfügung stellt, werden von einzelnen SOAP-Kommandos angesprochen und die Ergebnisse an Ihren Code zurückgeliefert.*

Bevor Sie die Methoden dieser Proxy-Klasse nutzen können, müssen Sie den Code noch mit Hilfe des C#-Compilers kompilieren:

```
csc /target:library xml_aendern.cs
```

Durch dieses Kommando wird eine DLL-Datei erstellt, die in ein Verzeichnis */bin* Ihrer Webapplikation zu kopieren ist. Dieses Verzeichnis wird vom ASP.NET-Compiler automatisch nach eigenen Erweiterungen durchsucht, wodurch Ihre Proxy-Klasse ohne weiteres eingebunden werden kann.

**Die Proxy-Klasse nutzen und damit einen Web Service konsumieren**

Nun sind Sie fast am Ziel. Das eigentliche Konsumieren des Web Services geht nun leicht von der Hand:

Da die Proxy-Klasse in einem eigenen Namespace xml_aendern enthalten ist, binden Sie diesen zu Beginn des *aspx*-Scripts ein:

```
<%@ Import Namespace="xml_aendern" %>
```

*Alternativ dazu können Sie bei der Initialisierung eines Objekts des neuen Namespaces das Objekt auch mit seinem kompletten Namen ansprechen, also beispielsweise:*

```
xml_aendern.DatenAendern aendere = new
 xml_aendern.DatenAendern();
```

Jetzt können Sie ein Objekt der Klasse `DatenAendern` erstellen und die Methode Projekte nutzen, wie Sie es von lokal verfügbaren Klassen und Methoden her gewohnt sind.

Der komplette Beispielcode, bei dem der im vorherigen Abschnitt erstellte Web Service konsumiert wird, lautet:

```
<%@ Page language="C#" %>
<%@ Import Namespace="xml_aendern" %>
<script runat="Server">
void speichern(Object Sender, EventArgs e)
{
  DatenAendern aendere = new DatenAendern();
  ausgabe.Text = aendere.Projekte(pleiter.Text,
   pthema.Text);
}
</script>
<html>
<head>
  <title> Über einen Web Service eine entfernt
   gespeicherte XML-Datei verändern </title>
</head>
<body>
<form runat="server">
  Projektleiter:
  <asp:TextBox id="pleiter" runat="server" /><br />
  Thema des Projekts:
  <asp:TextBox id="pthema" runat="server" /><br />
  <asp:Button text="Speichern" onclick="speichern"
   runat="server" /><br />
</form>
  <asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 19.7: Konsumieren eines Web Services (web_service_konsumieren.aspx)*

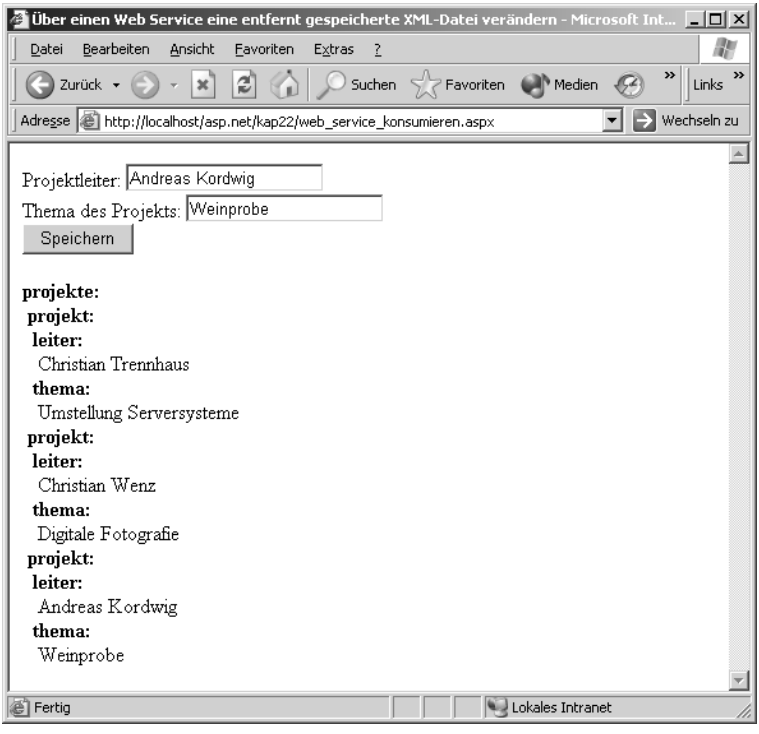



*Bild 19.4: Konsumieren eines Web Services: web_service_konsumieren.aspx*

Dass diese Klasse und die Methode eigentlich als Web Service zur Verfügung gestellt sind und als solche auch von Ihnen genutzt werden, spielt dank der transparenten Proxy-Klasse keine Rolle. Erstellen und Konsumieren von Web Services sind durch die Unterstützung im .NET Framework denkbar einfach, Sie können diese neue Architektur also gut nutzen.

# 20 E-Mail

Für den Versand von E-Mails stellt das .NET Framework einen kompletten Namespace zur Verfügung: `System.Web.Mail`. Damit können Sie direkt aus einer ASP.NET-Seite heraus E-Mails versenden. Alles, was Sie dazu benötigen, ist ein SMTP-Server (*Simple Mail Transport Protocol*), der Ihre E-Mails an die Empfänger weiterleitet.


*Unter ASP war das nicht ohne zusätzliche Bordmittel möglich. Sie mussten entweder den Microsoft SMTP Service (ein Teil von Windows NT/2000/XP) installieren oder auf eine zusätzliche Komponente wie beispielsweise JMail (http://tech.dimac.net/) setzen. Letzterer Weg ermöglichte es dann auch, beliebige SMTP-Server zu nutzen.*


*Bei den Beispielen in diesem Kapitel verwenden wir fiktive E-Mail-Adressen der Form name@xy.de. Domains, die auf .de enden, müssen mindestens drei Zeichen lang sein. Ersetzen Sie in den Beispielen unbedingt diese Adressen durch Ihre eigenen!*

## 20.1 SMTP-Server installieren

Bei den Server-Betriebssystemen von Microsoft, also Windows NT Server/2000 Server und Advanced Server/XP Professional sowie dem neuen .NET Server ist der Microsoft SMTP Service (vormals Microsoft SMTP Server) dabei. Sie können diesen auf Ihrem System installieren, indem Sie in der Systemsteuerung den Punkt Software wählen, dann Windows-Komponenten hinzufügen/entfernen. Den SMTP-Dienst können Sie dann zumeist installieren, indem Sie Internet-Informationsdienste (IIS) anklicken, Details wählen und dann die Checkbox neben SMTP-Dienst aktivieren (Bild 20.1).

Wenn Sie diesen Weg wählen (also keinen anderen geeigneten SMTP-Server in Ihrem Zugriffsbereich haben), finden Sie in der IIS-Konsole (bei den neueren Windows-Versionen in der Systemsteuerung unter Verwaltung/Internet-Informationsdienste) einen Eintrag Virtueller Standardserver für SMTP.

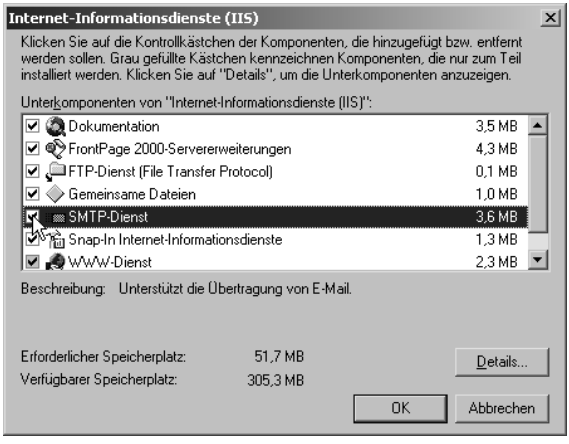



*Bild 20.1: Der SMTP-Dienst kann über die Systemsteuerung installiert werden*

Bei dessen Eigenschaften müssen Sie sicherstellen, dass Ihre Webapplikation auch wirklich Zugriff auf diesen Server hat, die Außenwelt jedoch nicht. Zunächst einmal könnten Sie unter ZUGRIFF/ZUGRIFFSKONTROLLE den anonymen Zugriff auf den Server ermöglichen; dann im selben Register beim Punkt WEITERGABEEINSCHRÄNKUNGEN die E-Mail-Weitergabe nur für Ihre IP-Adresse(n) gewähren (siehe Bild 20.2).



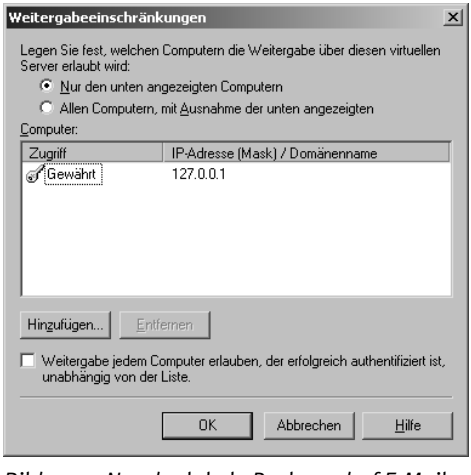



*Bild 20.2: Nur der lokale Rechner darf E-Mails versenden*

## 20.2 Eine »einfache« E-Mail

Die Klasse `SmtpMail` (innerhalb des Namespace `System.Web.Mail`) kann dazu benutzt werden, mit möglichst wenig Aufwand eine E-Mail aus einer ASP.NET-Seite heraus zu verschicken. Es steht nämlich die Methode `Send` zur Verfügung, die vier String-Parameter erwartet:

1. Die Adresse des Absenders
2. Die Adresse des Empfängers
3. Der Betreff der E-Mail
4. Der Text der E-Mail

Ein möglicher Aufruf dieser Methode könnte also wie folgt aussehen:

```
System.Web.Mail.SmtpMail.Send(
  "absender@xy.de",
  "empfaenger@xy.de",
  "Mail mit ASP.NET",
  "Das ist gar nicht so schwer!"
);
```

Im Webbrowser erhalten Sie allerdings voraussichtlich die in Bild 20.3 zu sehende Fehlermeldung.

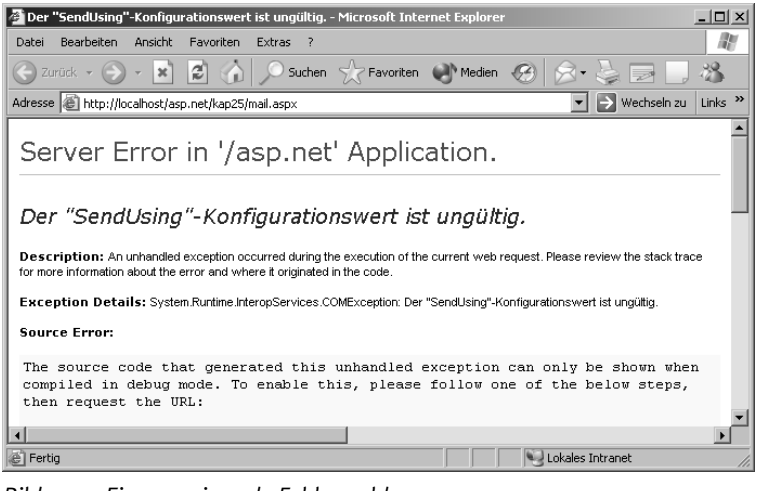



*Bild 20.3: Eine verwirrende Fehlermeldung*

Zum Glück lässt sich diese Meldung leicht beheben. Laut Dokumentation wird der lokale SMTP-Server verwendet, wenn nicht explizit ein anderer angegeben wird. Dies scheint aber nicht zu funktionieren. Sie müssen auf jeden Fall explizit den SMTP-Server angeben, und zwar in der Eigenschaft `SmtpServer` der Klasse `SmtpMail`. Nachfolgend nun ein funktionierendes Skript:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat="server">
void Page_Load()
{
  SmtpMail.SmtpServer = "localhost";
  SmtpMail.Send(
    "absender@xy.de",
    "empfaenger@xy.de",
    "Mail mit ASP.NET",
    "Das ist gar nicht so schwer!"
  );
}
</script>
```

*Listing 20.1: E-Mail-Versand mit nur zwei Anweisungen (mail.aspx)*



Sofern Ihr SMTP-Server korrekt konfiguriert ist, wird nun eine E-Mail erzeugt und auch verschickt. Wenn Sie einen anderen SMTP-Server einsetzen, müssen Sie natürlich auch die Eigenschaft `SmtpServer` auf den entsprechenden, anders lautenden Wert setzen.


*Bild 20.4: Die mit ASP.NET erzeugte E-Mail*

*Achten Sie bei anderen SMTP-Servern auf eventuelle Einschränkungen. Ein beliebter Tipp ist es beispielsweise, den SMTP-Server eines Freemailers, beispielsweise den von GMX zu verwenden. Was viele jedoch nicht beachten, ist, dass solche SMTP-Server ein Passwort benötigen und/oder nur dann funktionsfähig sind, wenn zuvor per POP3 das E-Mail-Konto abgefragt worden ist. Außerdem lassen viele E-Mail-Server nur bestimmte, bekannte Absenderadressen zu.*

## 20.3 Zusätzliche E-Mail-Optionen

Für komplexere Mails, beispielsweise solche mit CC- und BCC-Empfängern, müssen Sie einen etwas anderen Weg gehen. Dazu müssen Sie ein Objekt vom Typ `System.Web.Mail.MailMessage` instanzieren und dann dessen Eigenschaften setzen, vor allem folgende:

- `From`: Der Absender der E-Mail (Pflichtangabe)
- `Subject`: Der Betreff der E-Mail
- `Body`: Der Text der E-Mail

Versenden können Sie die E-Mail dann mit der bereits bekannten `Send`-Methode; nur müssen Sie dieses Mal als Parameter das `MailMessage`-Objekt angeben:

```
System.Web.Mail.MailMessage mail =
  new System.Web.Mail.MailMessage();
// ...
System.Web.Mail.SmtpMail.SmtpServer = "localhost";
System.Web.Mail.SmtpMail.Send(mail);
```

### 20.3.1 Empfänger

Um die jeweiligen Empfänger einer E-Mail anzugeben, verwenden Sie die folgenden Eigenschaften des `MailMessage`-Objekts:

| Eigenschaft | Beschreibung |
|---|---|
| `To` | der oder die Empfänger der E-Mail |
| `Cc` | der oder die Kopieempfänger (CC) der E-Mail |
| `Bcc` | der oder die Blindkopieempfänger (BCC) der E-Mail |

*Tabelle 20.1:* `MailMessage`*-Eigenschaften für Empfänger*

Hier ein kleines Beispiel einer E-Mail, die an insgesamt vier Empfänger geht:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat="server">
void Page_Load()
{
  MailMessage mail = new MailMessage();
  mail.From = "absender@xy.de";
  mail.To = "empfaenger1@xy.de,empfaenger2@xy.de";
  mail.Cc = "kopie@xy.de";
  mail.Bcc = "blind@xy.de";
  mail.Subject = "Mail mit ASP.NET";
  mail.Body = "Das ist gar nicht so schwer!";

  SmtpMail.SmtpServer = "localhost";
  SmtpMail.Send(mail);
}
</script>
```

*Listing 20.2: E-Mail-Versand an mehrere Empfänger (empfaenger.aspx)*


*Mehrere Empfängeradressen trennen Sie durch Kommata voneinander. Die unter anderem aus Outlook und Outlook Express bekannte Trennung durch Semikola ist eine Microsoft-Konvention, womit aber nicht alle Mailserver zurechtkommen. Also: Verwenden Sie Kommata.*


*Wenn Sie neben der E-Mail-Adresse auch noch den Namen des Absenders und/oder Empfängers angeben möchten, verwenden Sie dazu folgendes Format:*

```
mail.From = "\"Mein Name\" <ich@xy.de>";
```

### 20.3.2 Wichtigkeit

Eine eher nervige Option für E-Mails ist die so genannte Wichtigkeit oder Priorität. Eigentlich ein gut gemeintes Konzept: Der Absender kann stufenweise angeben, wenn seine E-Mail besonders wichtig oder unwichtig ist. Leider erfordert dies eine gewisse Disziplin der Absender, die heutzutage nur selten eingehalten wird. Viele Spaß-E-Mails oder nicht so dringende Anfragen werden als wichtig gekenn-

zeichnet, was dazu führt, dass wichtige Mails häufig als Letztes gelesen werden. Setzen Sie dieses Mittel also mit Bedacht ein. Wenn Sie allerdings beispielsweise automatisch eine E-Mail versenden, wenn auf Ihrer Website ein Datenbankfehler auftritt, ist eine hohe Priorität in der Tat empfehlenswert.

Die zugehörige Eigenschaft des `MailMessage`-Objekts heißt `Priority` und ist vom Typ `System.Web.Mail.MailPriority`. Diese Klasse – genauer gesagt handelt es sich um eine Enumeration – hat die folgenden Elemente:

| Element | Beschreibung |
|---|---|
| `Low` | Niedrige Priorität |
| `Normal` | Normale Priorität (Standard) |
| `High` | Hohe Priorität |

*Tabelle 20.2: Elemente in der Enumeration* `MailPriority`

Im folgenden Listing können Sie per Radiobutton auswählen, welche Priorität eine E-Mail haben soll:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat="server">
void Versand(Object o, EventArgs e)
{
  MailMessage mail = new MailMessage();
  mail.From = "absender@xy.de";
  mail.To = "empfaenger@xy.de";
  mail.Subject = "Mail mit ASP.NET";
  mail.Body = "Das ist gar nicht so schwer!";

  String p = "Normal";
  if (prio.SelectedItem != null) {
    p = prio.SelectedItem.Text;
    switch (p) {
      case "Niedrig":
        mail.Priority = MailPriority.Low; break;
      case "Hoch":
        mail.Priority = MailPriority.High; break;
      default:
        mail.Priority = MailPriority.Normal; break;
    }
```

```
  }

  SmtpMail.SmtpServer = "localhost";
  SmtpMail.Send(mail);

  ausgabe.Text = "E-Mail mit Priorität " +
                 p + " versandt!";
}
</script>
<html>
<head>
  <title>E-Mails mit ASP.NET</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
<form runat="server">
<asp:RadioButtonList id="prio" runat="server">
  <asp:ListItem Text="Niedrig" />
  <asp:ListItem Text="Normal" />
  <asp:ListItem Text="Hoch" />
</asp:RadioButtonList>
<asp:Button runat="server" Text="E-Mail versenden"
  OnClick="Versand" />
</form>
</body>
</html>
```

*Listing 20.3: E-Mail-Versand mit Priorität (prioritaet.aspx)*

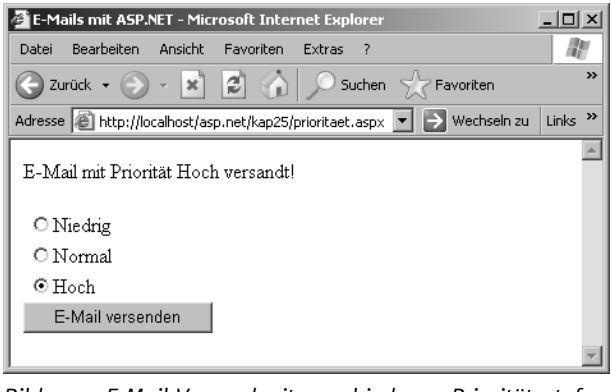



*Bild 20.5: E-Mail-Versand mit verschiedenen Prioritätsstufen*

## 20.4 Dateianhänge

Um einer E-Mail eine Datei anzuhängen, benötigen Sie eine weitere Eigenschaft der Klasse `MailMessage`: Die Rede ist von Attachments. Diese Eigenschaft kann nur gelesen, nicht geschrieben werden. Sie ist vom Typ `IList` und besitzt die Methode `Add`, mit der Sie Dateien hinzufügen können.

Als Parameter für die `Add`-Methode geben Sie einen Wert vom Typ `System.Web.Mail.MailAttachment` an. Der Konstruktor hierfür erwartet ein bis zwei Parameter:

Pflicht ist der erste Parameter: der Name der Datei (inklusive Pfad). Der zweite Parameter dagegen ist optional: Hier können Sie angeben, auf welche Art und Weise der Dateianhang kodiert wird. Dieser Parameter ist vom Typ `System.Web.Mail.MailEncoding` und hat die folgenden beiden möglichen Werte:

- `Base64` (Standard)
- `UUEncode`

Hier ein kurzer Code-Ausschnitt, in dem eine Datei angehängt und mittels UUEncode kodiert wird:

```
MailMessage mail = new MailMessage();
mail.From = "absender@xy.de";
mail.To = "empfaenger@xy.de";
mail.Subject = "Mail mit ASP.NET";
mail.Body = "Das ist gar nicht so schwer!";

MailAttachment anhang =
  new MailAttachment("c:\\test.txt",
                     MailEncoding.UUEncode);
mail.Attachments.Add(anhang);

SmtpMail.SmtpServer = "localhost";
SmtpMail.Send(mail);
```

Nachfolgend ein etwas ausführlicheres Beispiel: Ein Benutzer kann per `<input type="file">` eine Datei zum Webserver übertragen; diese wird dann als Dateianhang verschickt:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat="server">
void Versand(Object o, EventArgs e)
{
  MailMessage mail = new MailMessage();
  mail.From = "absender@xy.de";
  mail.To = "empfaenger@xy.de";
  mail.Subject = "Mail mit ASP.NET";
  mail.Body = "Das ist gar nicht so schwer!";

  if (upload.PostedFile != null) {
    System.Web.HttpPostedFile up = upload.PostedFile;
    String name =
      (new System.IO.FileInfo(up.FileName)).Name;
    up.SaveAs("c:\\temp\\" + name);
    MailAttachment anhang = new MailAttachment("c:\\temp\\" + name);
    mail.Attachments.Add(anhang);
  }

  SmtpMail.SmtpServer = "localhost";
  SmtpMail.Send(mail);

  ausgabe.Text = "E-Mail versandt!";
}
</script>
<html>
<head>
  <title>E-Mails mit ASP.NET</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
<form enctype="multipart/form-data" runat="server">
<input type="file" id="upload" runat="server" />
<asp:Button runat="server" Text="E-Mail versenden"
  OnClick="Versand" />
</form>
</body>
</html>
```

*Listing 20.4: E-Mail-Versand mit Dateianhang (anhang.aspx)*

*Dieses Beispiel funktioniert nur, wenn der temporäre Pfad für die hochgeladene Datei, c:\temp, existiert und der ASP.NET-Prozess darauf Schreibrechte besitzt! Außerdem sollten Sie dieses Verzeichnis regelmäßig leeren oder diese Funktionalität in das ASP.NET-Skript einbauen.*

## 20.5 HTML-Mails

Im letzten Teil dieses Kapitels geht es um ein besonders spannendes Thema, HTML-Mails. Mit der Verbreitung des World Wide Web wurde es möglich, nicht nur reine Textnachrichten zu verschicken, sondern gleich ganze HTML-Seiten. Da die meisten Browser mittlerweile auch ein integriertes E-Mail-Programm anbieten, war das ein nahe liegender Schritt.

Und so funktioniert's: Das ganze Geheimnis besteht darin, die Eigenschaft `BodyFormat` zu setzen. Standardmäßig hat diese den Wert `System.Web.Mail.MailFormat.Text`, Sie können aber auch `System.Web.Mail.MailFormat.Html` wählen und dann Ihre E-Mail auch im HTML-Format erstellen und versenden.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Mail" %>
<script runat="server">
void Page_Load()
{
  MailMessage mail = new MailMessage();
  mail.From = "absender@xy.de";
  mail.To = "empfaenger@xy.de";
  mail.Subject = "Mail mit ASP.NET";
  mail.BodyFormat = MailFormat.Html;
  mail.Body = "<big><b><i>Das ist gar nicht so schwer!</i></b></big>";

  SmtpMail.SmtpServer = "localhost";
  SmtpMail.Send(mail);
}
</script>
```

*Listing 20.5: E-Mail-Versand im HTML-Format (htmlmail.aspx)*

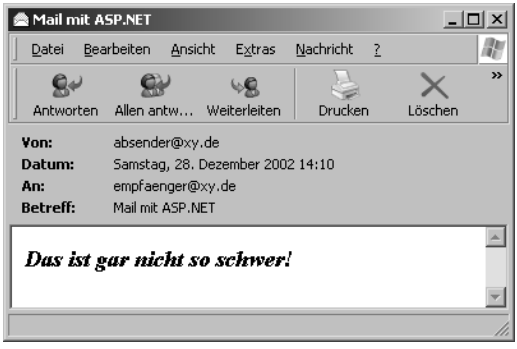



*Bild 20.6: Eine HTML-Mail, mit ASP.NET erzeugt*

Für HTML-Mails gibt es im Übrigen noch weitere Wünsche für die Praxis:

- Versand einer Hybrid-Mail, sowohl als Text als auch als HTML (um niemanden auszustoßen, der beispielsweise die HTML-Anzeige von E-Mails ausgeschaltet hat)
- Integration von Dateianhängen (z.B. Bildern) in den HTML-Code

Leider ist dies mit ASP.NET-Bordmitteln nicht trivial möglich; hierzu benötigen Sie externe Komponenten.

Zum Abschluss des Kapitels noch ein Warnhinweis. Das automatische Erstellen von E-Mails ist eine tolle Sache. Achten Sie aber penibel darauf, dass die verschickten E-Mails auch wirklich gewollt sind. Es kursieren immer wieder Spam-Mails, die über ein schlecht programmiertes Mailskript erzeugt worden sind.

# 21 HTTP-Funktionen

Eine weitere häufig verwendete Technik ist der Zugriff auf andere Websites per HTTP. Unter dem »klassischen« ASP 3.0 war das mit den mitgelieferten Bordmitteln nicht möglich, hier mussten externe Komponenten (häufig kostenpflichtig) Abhilfe schaffen.

Mit ASP.NET hat sich das grundlegend geändert. Der Namespace `System.Net` bietet alle benötigten Funktionalitäten.


*Mit* `System.Net` *können Sie auch Socketverbindungen erstellen, ein Themenblock, der in diesem Buch jedoch nur am Rande behandelt wird.*

## 21.1 HTTP-Seiten aufrufen und auslesen

Zum Aufruf und zur Auswertung von HTTP-Anfragen verwenden Sie zwei Typen aus `System.Net`:

- `WebRequest` ist das Objekt für eine HTTP-Anfrage.
- `WebResponse` ist das Objekt für die HTTP-Antwort, die einer Anfrage folgt.

### 21.1.1 Ein einfaches Beispiel

Zu Beginn ein einfaches Beispiel, bei dem das Ergebnis einer HTTP-Anfrage direkt ausgegeben wird. Hieran sehen Sie auch, wie die Ansteuerung der Klassen in `System.Net` vonstatten geht.

Zunächst einmal müssen Sie die Methode `Create()` der Klasse `HttpWebRequest` (innerhalb von `System.Net`) verwenden, in der Funktionalitäten speziell für HTTP-Anfragen gekapselt sind. Als Parameter übergeben Sie die zu testende URL:

```
WebRequest Anfrage = HttpWebRequest.Create(
  "http://localhost/asp.net/kap03/HalloWelt.aspx");
```


*Als Beispiel-URL wird die Hallo-Welt-Applikation aus Kapitel 3 des Einführungsteils verwendet.*

Die Anfrage wird allerdings dann an den betreffenden Webserver gestellt, wenn Sie die Methode `GetResponse()` des `WebRequest`-Objekts aufrufen. Der Rückgabewert ist vom Typ `WebResponse`.

```
WebResponse Antwort = Anfrage.GetResponse();
```

Das Auslesen gestaltet sich nun sehr einfach. Die Methode `GetResponseStream()` liefert einen Stream zurück, auf den Sie wie auf einen Datei-Stream zugreifen können:

```
System.IO.StreamReader sr =
  new System.IO.StreamReader(
    Antwort.GetResponseStream()
  );
String HTML = sr.ReadToEnd();
sr.Close();
```

Der resultierende HTML-Code kann dann ausgegeben werden, Sie sollten unter Umständen – je nach Anwendung – bestimmte Elemente wie beispielsweise den `<head>`-Bereich und `<body>`- und `<html>`-Tags herausfiltern. Dies ist im nachfolgenden Code bereits umgesetzt; der »eigentliche« Inhalt der Seite wird ausgegeben:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Net" %>
<script runat="server">
void Page_Load()
{
  WebRequest Anfrage = HttpWebRequest.Create(
    "http://localhost/asp.net/kap03/HalloWelt.aspx");
  WebResponse Antwort = Anfrage.GetResponse();
  System.IO.StreamReader sr = new System.IO.StreamReader(
      Antwort.GetResponseStream()
    );
  String HTML = sr.ReadToEnd();
  sr.Close();
  HTML = HTML.Remove(0, HTML.IndexOf("<body>") + 6);
  HTML = HTML.Replace("</body>", "");
  HTML = HTML.Replace("</html>", "");
  ausgabe.Text = HTML;
}
</script>
<html>
<head>
  <title>HTTP-Funktionen</title>
</head>
```

```
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 21.1: Die Hallo-Welt-Seite wird ausgelesen (auslesen.aspx)*

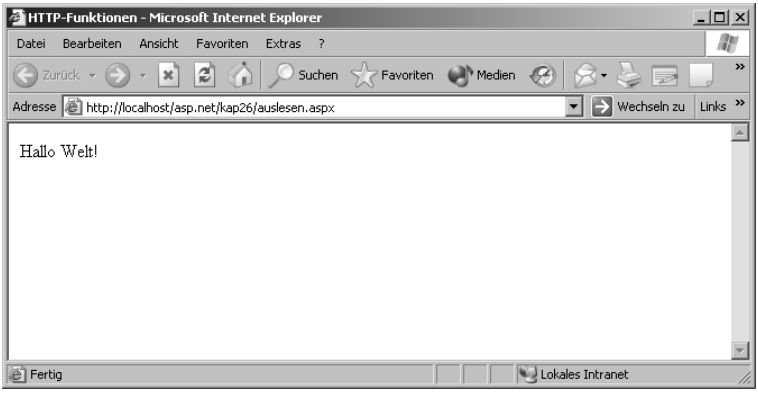



*Bild 21.1: Eine ASP.NET-Seite liest eine andere aus*

### 21.1.2 Fehlerbehandlung

Unter Umständen erhalten Sie eine Fehlermeldung, wenn Sie das vorherige Beispiel im Browser ausführen:

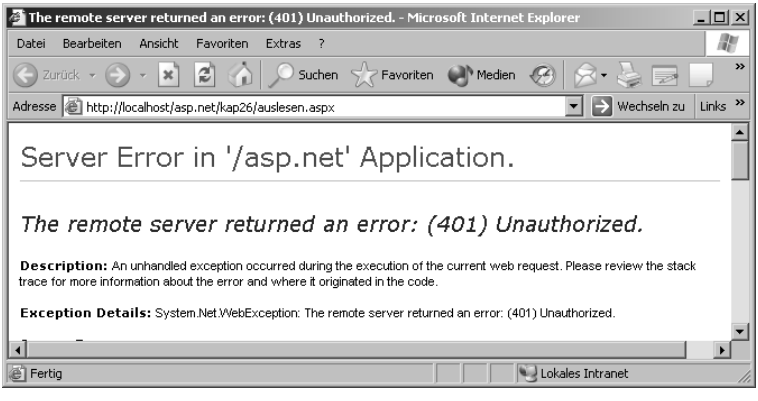



*Bild 21.2: Eine Fehlermeldung bei der Standardkonfiguration*

Wahrscheinlicher Grund: Wenn Sie ein neues virtuelles Verzeichnis anlegen (wie wir es für die Beispiele in diesem Buch ja empfohlen haben), ist hierauf unter bestimmten Betriebssystemversionen kein anonymer Zugriff möglich. Sie können die Seiten zwar direkt im Webbrowser anzeigen, da Sie als berechtigter Windows-Nutzer angemeldet sind, aber Ihre (unter dem *ASP.NET*-Konto laufende) ASP.NET-Seite darf keine andere Seite aufrufen. Um dies zu beheben, können Sie natürlich in der IIS-Konsole den anonymen Zugriff zulassen. Rufen Sie dazu die Eigenschaften des Verzeichnisses auf, wählen Sie das Register VERZEICHNISSICHERHEIT und dort die Schaltfläche BEARBEITEN und aktivieren Sie die Checkbox ANONYMER ZUGRIFF.

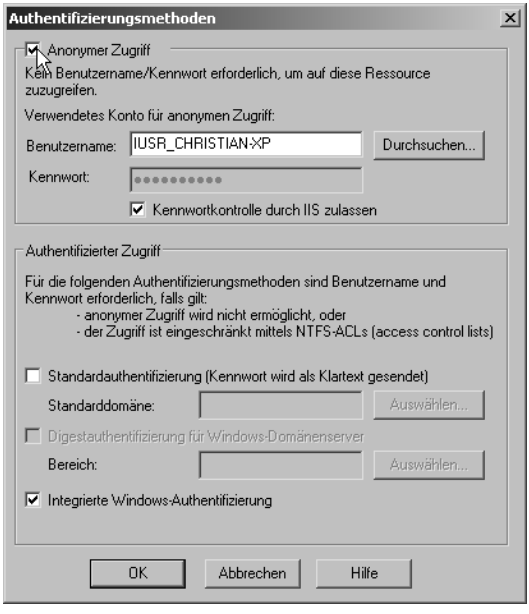



*Bild 21.3: Der anonyme Zugriff wird aktiviert*

Allerdings sollten Sie noch zusätzliche Sicherheitsvorkehrungen treffen, kann ja sein, dass die ASP.NET-Seite einmal wegen Überlastung nicht verfügbar ist. Sie sollten also auf jeden Fall bemüht sein, die Fehlermeldung mittels `try-catch` abzufangen. Wie Sie Bild 21.2 entnehmen

können, ist die Fehlermeldung vom Typ `System.Net.WebException`, der neue Code sieht also folgendermaßen aus:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Net" %>
<script runat="server">
void Page_Load()
{
  String HTML;
  WebRequest Anfrage = HttpWebRequest.Create(
    "http://localhost/asp.net/kap03/HalloWelt.aspx");
  try {
    WebResponse Antwort = Anfrage.GetResponse();
    System.IO.StreamReader sr =
      new System.IO.StreamReader(
        Antwort.GetResponseStream()
      );
    HTML = sr.ReadToEnd();
    sr.Close();
    HTML = HTML.Remove(0, HTML.IndexOf("<body>") + 6);
    HTML = HTML.Replace("</body>", "");
    HTML = HTML.Replace("</html>", "");
  } catch (System.Net.WebException ex) {
    HTML = "<b>Fehler: </b>" + ex.Message;
  }
  ausgabe.Text = HTML;
}
</script>
<html>
<head>
  <title>HTTP-Funktionen</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 21.2: HTTP-Fehler werden abgefangen (fehler-abfangen.aspx)*


*Bild 21.4: Die Fehlermeldung wird jetzt formatiert ausgegeben*

### 21.1.3 POST-Anfragen

Die bisherigen Anfragen wurden jeweils per GET durchgeführt, sprich: Alle Informationen der HTTP-Anfrage stehen in der URL. Bei Formulardaten wird jedoch häufig POST verwendet, dort werden die entsprechenden Daten als Teil des HTTP-Headers der Anforderung verschickt, können aber nicht der URL entnommen werden.

Ein Beispiel hierfür sind Login-Masken. Sie geben einen Benutzernamen und ein Passwort ein und erhalten dann Zugriff auf einen geschützten Bereich, beispielsweise Webmail. Um dies zu demonstrieren, hier eine sehr minimalistische Loginseite. Sie ist im »klassischen« ASP geschrieben, um zu zeigen, dass die Wahl der Technologie hier nicht entscheidend ist:

```
<html>
<head>
  <title>Login</title>
</head>
<body>
Willkommen,
  <% =Server.HTMLEncode(Request.Form("Login")) %>!<br />
Ich sage es nicht weiter, aber Ihr Passwort ist
  <% =Server.HTMLEncode(Request.Form("Passwort")) %> ...
</body>
</html>
```

*Listing 21.3: Die minimalistische Loginseite (login.asp)*

Wie zu sehen, erwartet diese Seite einen Benutzernamen und ein Passwort per POST (wenn Sie noch nie mit dem klassischen ASP gearbeitet haben: POST-Daten stehen in `Request.Form`). Ziel ist es nun, diese Seite per ASP.NET aufzurufen und den zurückgegebenen HTML-Code auszugeben.

Um dies zu tun, müssen einige Eigenschaften der Klasse `HttpWebRequest` verwendet werden:

- `Method` enthält die verwendete HTTP-Methode, Standard ist `"GET"`.
- `Content-Type` beinhaltet den MIME-Typ der Anfrage; beim Formularversand ist das `"application/x-www-form-urlencoded"`.
- `Content-Length` gibt bei POST an, wie viele Zeichen als POST-Daten übergeben werden.

Für das Login-Beispiel könnte der Code wie folgt aussehen:

```
String Daten = "Login=Christian&Passwort=xyz";
HttpWebRequest Anfrage = (HttpWebRequest)WebRequest.Create(
  "http://localhost/asp.net/kap26/login.asp");
Anfrage.Method = "POST";
Anfrage.ContentType="application/x-www-form-urlencoded";
Anfrage.ContentLength = Daten.Length;
```


*Die Methode* `WebRequest.Create()` *gibt standardmäßig ein Objekt vom Typ* `WebRequest` *zurück; für POST-Anfragen benötigen Sie aber unbedingt ein Objekt vom Typ* `HttpWebRequest`*, müssen also casten.*

Nun müssen Sie nur noch die POST-Daten (in der Variablen Daten) in die Anfrage einbauen. Dazu benötigen Sie ein `StreamWriter`-Objekt – richtig, Sie können auch in HTTP-Anfragen wie in Dateien hineinschreiben. Das zum `HttpWebRequest`-Objekt zugehörige `StreamWriter`-Objekt erhalten Sie mit der Methode `GetRequestStream()`:

```
System.IO.StreamWriter sw =
  new System.IO.StreamWriter(
    Anfrage.GetRequestStream());
sw.Write(Daten);
sw.Close();
```

Der Rest des Skripts ist dasselbe, inklusive `try/catch`. Nachfolgend der komplette Code, wie Sie ihn auch im Webarchiv finden:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Net" %>
<script runat="server">
void Page_Load()
{
  String HTML;
  String Daten = "Login=Christian&Passwort=xyz";
  HttpWebRequest Anfrage = (HttpWebRequest)WebRequest.Create(
    "http://localhost/asp.net/kap26/login.asp");
  Anfrage.Method = "POST";
  Anfrage.ContentType="application/x-www-form-urlencoded";
  Anfrage.ContentLength = Daten.Length;

  try {
    System.IO.StreamWriter sw =
      new System.IO.StreamWriter
        (Anfrage.GetRequestStream());
    sw.Write(Daten);
    sw.Close();

    WebResponse Antwort = Anfrage.GetResponse();
    System.IO.StreamReader sr =
      new System.IO.StreamReader
        (Antwort.GetResponseStream());
    HTML = sr.ReadToEnd();
    sr.Close();

    HTML = HTML.Remove(0, HTML.IndexOf("<body>") + 6);
    HTML = HTML.Replace("</body>", "");
    HTML = HTML.Replace("</html>", "");
  } catch (System.Net.WebException ex) {
    HTML = "<b>Fehler: </b>" + ex.Message;
  }
  ausgabe.Text = HTML;
}
</script>
<html>
<head>
  <title>HTTP-Funktionen</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 21.4: Eine HTTP-Anfrage per POST (post.aspx)*


*Bild 21.5: Die Login-Daten, per POST geschickt*

HINWEIS

*Wenn Sie sich das Debuggen vereinfachen möchten, können Sie anstelle der ASP-Loginseite auch eine ASP.NET-Seite erstellen und dort* `Debug="True"` *und* `Trace="True"` *setzen. Eine entsprechend angepasste Datei login.aspx finden Sie im Webarchiv.*

Auch wenn dieses Beispiel bewusst trivial gehalten ist, haben Sie damit das Rüstwerkzeug, um auch komplexere Anwendungen zu erstellen. Beispielsweise wäre es möglich, dass Sie sich per Benutzername und Passwort bei Ihrem Webmail-Dienst anmelden und die zurückgegebenen Daten auswerten, beispielsweise wie viele E-Mails Sie bereits erhalten haben. So können Sie im Hintergrund permanent Ihr Postfach überprüfen.

ACHTUNG

*Zwei potenzielle Stolperfallen gibt es jedoch:*

1. *Der Anbieter kann diese Art der »Zweitverwertung« untersagen oder gar versuchen Techniken einzusetzen, die einen derartigen Zugriff unterbinden.*
2. *Einige Anbieter verwenden Cookies, um einen zusätzlichen Schutz zu gewährleisten. Ohne Cookies kein Zugriff.*

### 21.1.4 Cookies lesen und setzen

Für das letzte der beiden genannten Probleme gibt es einen potenziellen Ausweg. Dazu erstellen wir zwei Hilfsdateien. Die eine Datei setzt ein Cookie, die andere liest Cookies aus. Beginnen wir mit dem Setzen des Cookies; als Wert wird einmal der Benutzername, beim zweiten Cookie das Passwort aus den POST-Daten verwendet:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  Login.Text = Request.Form["Login"];
  Passwort.Text = Request.Form["Passwort"];
  Response.Cookies.Add(
    new System.Web.HttpCookie(
      "Login",
      Request.Form["Login"]
    )
  );
  Response.Cookies.Add(
    new System.Web.HttpCookie(
      "Passwort",
      Request.Form["Passwort"]
    )
  );
}
</script>
<html>
<head>
  <title>Login</title>
</head>
<body>
Willkommen,
  <asp:Label id="Login" runat="server" />!<br />
Ich sage es nicht weiter, aber Ihr Passwort ist
  <asp:Label id="Passwort" runat="server" /> ...
</body>
</html>
```

*Listing 21.5: Auf der Login-Seite werden Cookies gesetzt (cookie-setzen.aspx)*

Eine weitere Seite gibt die Cookies wieder aus:

```
<%@ Page Language="C#" %>
<script runat="server">
void Page_Load()
{
  if (Request.Cookies["Login"] != null) {
    Login.Text = Request.Cookies["Login"].Value;
  }
  if (Request.Cookies["Passwort"] != null) {
    Passwort.Text = Request.Cookies["Passwort"].Value;
  }
}
</script>
<html>
<head>
  <title>Cookie-Infos</title>
</head>
<body>
Name laut Cookie:
  <asp:Label id="Login" runat="server" /> <br />
Passwort laut Cookie:
  <asp:Label id="Passwort" runat="server" />
</body>
</html>
```

*Listing 21.6: Die Cookies weden ausgegeben (cookie-lesen.aspx)*

Fehlt nur noch das Skript, das beide Skripte aufrufen soll, was etwas komplexer ist und deswegen schrittweise aufgebaut wird.

Zunächst einmal müssen Sie wieder eine POST-Anfrage stellen, die geht (fast) genau wie zuvor schon gesehen.

```
String HTML;
String Daten = "Login=Christian&Passwort=xyz";
HttpWebRequest Anfrage =
  (HttpWebRequest)WebRequest.Create(
    "http://localhost/asp.net/kap26/cookie-setzen.aspx");
Anfrage.Method = "POST";
Anfrage.ContentType="application/x-www-form-urlencoded";
Anfrage.ContentLength = Daten.Length;
Anfrage.CookieContainer = new CookieContainer();
```

```
try {
  System.IO.StreamWriter sw =
    new System.IO.StreamWriter(
      Anfrage.GetRequestStream());
  sw.Write(Daten);
  sw.Close();
```

Eine Zeile ist hier neu gewesen:

```
Anfrage.CookieContainer = new CookieContainer();
```

Damit übergeben Sie an die Anfrage ein leeres `CookieContainer`-Objekt, dabei handelt es sich um eine Sammlung von Cookies.


*Pikantes Detail am Rande, und deswegen extra mit einem Warnsymbol gekennzeichnet: Wenn Sie dies unterlassen, funktioniert später das Auslesen der Cookies nicht mehr, die von der ASP.NET-Seite zurückgegeben werden!*

Nun geht es ans Auswerten der Antwort vom Server. Alle Cookies, die der Server schickt, stehen in einem `CookieCollection`-Objekt. Wie Sie bereits oben gesehen haben, wird für eine erneute Anfrage allerdings ein `CookieContainer`-Objekt benötigt. Aus diesem Grund müssen Sie alle Cookies umkopieren. Das geht mit einer `foreach`-Schleife recht schnell; das `CookieContainer`-Objekt besitzt zudem eine `Add()`-Methode:

```
  HttpWebResponse Antwort =
    (HttpWebResponse)Anfrage.GetResponse();
  CookieCollection ccol = Antwort.Cookies;
  CookieContainer ccon = new CookieContainer();
  foreach (Cookie c in ccol) {
    ccon.Add(c);
  }
```

Mit diesem `CookieContainer`-Objekt können Sie nun eine neue Anfrage an das Auslesen-Skript schicken; dieses gibt dann die übermittelten Cookies aus:

```
  Anfrage = (HttpWebRequest)WebRequest.Create(
    "http://localhost/asp.net/kap26/cookie-lesen.aspx");
  Anfrage.CookieContainer = ccon;
  Antwort = (HttpWebResponse)Anfrage.GetResponse();
  System.IO.StreamReader sr =
    new System.IO.StreamReader(
      Antwort.GetResponseStream()
    );
```

Nachfolgend das komplette Skript:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Net" %>
<script runat="server">
void Page_Load()
{
  String HTML;
  String Daten = "Login=Christian&Passwort=xyz";
  HttpWebRequest Anfrage = (HttpWebRequest)WebRequest.Create(
    "http://localhost/asp.net/kap26/cookie-setzen.aspx");
  Anfrage.Method = "POST";
  Anfrage.ContentType="application/x-www-form-urlencoded";
  Anfrage.ContentLength = Daten.Length;
  Anfrage.CookieContainer = new CookieContainer();

  try {
    System.IO.StreamWriter sw =
      new System.IO.StreamWriter(
        Anfrage.GetRequestStream());
    sw.Write(Daten);
    sw.Close();

    // Cookies lesen
    HttpWebResponse Antwort =
      (HttpWebResponse)Anfrage.GetResponse();
    CookieCollection ccol = Antwort.Cookies;
    CookieContainer ccon = new CookieContainer();
    foreach (Cookie c in ccol) {
      ccon.Add(c);
    }

    // Neue Anfrage schicken, mit Cookies
    Anfrage = (HttpWebRequest)WebRequest.Create(
      "http://localhost/asp.net/kap26/cookie-lesen.aspx");
    Anfrage.CookieContainer = ccon;
    Antwort = (HttpWebResponse)Anfrage.GetResponse();
    System.IO.StreamReader sr =
      new System.IO.StreamReader(
        Antwort.GetResponseStream()
      );
    HTML = sr.ReadToEnd();
    sr.Close();
    HTML = HTML.Remove(0, HTML.IndexOf("<body>") + 6);
```

```
    HTML = HTML.Replace("</body>", "");
    HTML = HTML.Replace("</html>", "");
  } catch (System.Net.WebException ex) {
    HTML = "<b>Fehler: </b>" + ex.Message;
  }
  ausgabe.Text = HTML;
}
</script>
<html>
<head>
  <title>HTTP-Funktionen</title>
</head>
<body>
<asp:Label id="ausgabe" runat="server" />
</body>
</html>
```

*Listing 21.7: Cookies werden verschickt (post-cookies.aspx)*


*Bild 21.6: Die Cookies wurden per HTTP-Funktionen verschickt*

Mit dieser Technik können Sie nun beispielsweise Informationen von anderen Servern abrufen und auswerten – im Zweifelsfall natürlich nur von Ihren eigenen Websites.

# Stichwortverzeichnis

## Symbole



## A

## E



## F

## G



## H



## I

## P



## Q



## R



## S

## T



## U



## V



## W

# X

## Z