Klaus Löffelmann

# Visual Basic 2008 – Neue Technologien – Crashkurs

Klaus Löffelmann

# Visual Basic 2008 Neue Technologien Crashkurs

Klaus Löffelmann: Visual Basic 2008 – Neue Technologien – Crashkurs
Microsoft Press Deutschland, Konrad-Zuse-Str. 1, 85716 Unterschleißheim
Copyright © 2008 by Microsoft Press Deutschland

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor, Übersetzer und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht. Die in diesem Buch erwähnten Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Marken und unterliegen als solche den gesetzlichen Bestimmungen. Der Verlag richtet sich im Wesentlichen nach den Schreibweisen der Hersteller.

Das Werk, einschließlich aller Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
10 09 08

ISBN 978-3-86645-518-4

© Microsoft Press Deutschland
(ein Unternehmensbereich der Microsoft Deutschland GmbH)
Konrad-Zuse-Str. 1, D-85716 Unterschleißheim
Alle Rechte vorbehalten

Korrektorat: Kristin Grauthof, Lippstadt
Fachlektorat: Walter Saumweber
Satz: Silja Brands, ActiveDevelop, Lippstadt (http://www.ActiveDevelop.de)
Umschlaggestaltung: Hommer Design GmbH, Haar (www.HommerDesign.com)
Gesamtherstellung: Kösel, Krugzell (www.KoeselBuch.de)

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

*Für*

*Ada & Zoe*

*Sarah & David*

*Ute & Ruprecht*

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Inhaltsverzeichnis




"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Ein nicht ganz so ernst gemeintes
# Vorwort

Modernste Technik macht frei. Ich gönnte mir vor einiger Zeit einen Windows Vista Tablet PC und eine UMTS-Flatrate. Und das bedeutet: Ich kann arbeiten und damit Vorworte von Büchern schreiben, wo und wann ich will. Wie in diesem Beispiel: Ich schreibe die Zeilen dieses Vorworts tatsächlich in einer Turnhalle irgendwo in Ratingen-Hösel auf drei Turnkästen liegend, während ich immer wieder zweien meiner vier Lieblingskinder beim Badminton zuschaue – in der freien Zeit zwischen zwei Fuhren, in der ich (sehr, sehr gerne, übrigens) als ihr Fahrer fungiere.

Und während ich Sarah und David bewundernd zuschaue, und rätsele, wo sie ihre Reaktion hernehmen, und Bälle treffen, die ich schon mit bloßem Auge nicht mehr verfolgen kann, denke ich über eine kleine Rede ihres Papas nach. Das übrigens ist Ruprecht Dröge, SQL-Experte (für mich der Beste, für viele andere übrigens auch, selbst wenn er das immer abstreitet), ursprünglich Fachlektor meines Visual Basic Entwicklerbuches und inzwischen sehr guter Freund. Diese Rede hielt er ein paar Tage nach der Basta 2007 in seiner Küche, kurz bevor wir einmal mehr das leckerste Sushi der Welt machten. In dieser Rede ging es mal wieder um *die* immerwährende Diskussion, bzw. die Abneigung gegen dieselbe, und seine schließlich gezogene Quintessenz kann ich uneingeschränkt unterschreiben.

Gleich ihm habe ich die ewigen Diskussionen über *dieses Thema* ja so satt. Mir steht es bis hier. Sie sind kindisch, sie führen zu nichts, und wenn die C#ler in der Überzahl sind, dann haben sie mit all ihren Argumenten einfach nur Unrecht – laut brüllen ändert leider nichts an den Fakten. Um was geht's? Ganz einfach: Um die unendliche Geschichte auf Kongressen, Conventions, auf Bastas und sonstigen Events, wo wir Entwickler uns immer treffen und über das leidige Thema: »Was ist besser – Visual Basic oder C#?« endlos lange philosophieren.

Liebe C#ler, die ihr dieses Buch wahrscheinlich gar nicht wahrnehmt, es sogar schmäht und schneidet, und liebe Visual Basicler, die ihr euch nach 16 Jahren immer noch gegen die C#ler rechtfertigen müsst, weil ihr eure Anwendungen ja mit einer scheinbaren »Kinderprogrammiersprache« entwickelt, hier meine Argumentation für Visual Basic .NET. Endgültig – im Übrigen – denn ich bin unfassbar erfahren, habe die Weisheit in dieser Sache mit Löffeln verspeist und *darf* damit arrogant genug sein, dieser Diskussion ein für alle Mal ein Ende zu setzen: ;-)

1. Geschweifte Klammern lassen Programme *nicht* professioneller ausschauen. Was für ein fragwürdiges Argument soll das denn sein? (Und Leute, die mich kennen, nehmen »fragwürdig« hier als Platzhalter für andere Adjektive, die leider nichts in einem Computerbuch zu suchen haben!) Ganz im Gegenteil: Visual Basic hat den Vorteil, Strukturbefehle zur Verfügung zu stellen, bei denen man genau weiß, zu welchem Ende welcher Anfang gehört: `End If` gehört zu `If`; `Loop` gehört zu `Do`; `Next` zu `For`; `End With` zu `With` (ach ja, diese letzt genannte Schreibersparnisfunktion kennen die C#ler ja gar nicht – ups!). Bei C# hingegen sieht das Ende eines komplizierten Algorithmus folgender Maßen aus:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

1:0 für Visual Basic.

2. Wenn die Fehlerliste von Visual Basic .NET leer ist, lässt sich ein VB-Programm starten. Klar – dann können natürlich noch logische Fehler darin sein; »anlaufen« wird das Programm aber erst einmal. Das ist in C# nicht so – jedenfalls nicht, ohne Zusatztools dazu kaufen zu müssen. In C# können sich zu diesem Zeitpunkt immer noch Referenzen auf Methoden, Eigenschaften, Feldvariablen oder sonstige Elemente befinden, die der Compiler nicht auflösen konnte, weil der Entwickler schlicht vergessen hat, sie zu deklarieren, zu implementieren oder er sich einfach nur vertippt hat. Das bedeutet dann: Programm kompilieren, feststellen, ob Fehler drin sind, Fehler raus machen, Programm wieder kompilieren, wieder schauen, ob noch Fehler drin sind, wieder Fehler raus machen, usw.

   Im Übrigen ist das auch ein Vorteil von Visual Basic .NET gegenüber Visual Basic 6.0. Ein volles Kompilieren hat dort nur die erste nicht definierte Variable oder Methode gefunden; Sie mussten diese erst fixen, um anschließend wieder in die – so noch vorhanden – nächste nicht definierte Variable oder Methode zu rasseln. Deswegen sind die Turn-Around-Zeiten in Visual Basic 6.0 auch nur scheinbar kleiner als in Visual Basic .NET.

   2:0 für Visual Basic.

3. C# ist, wie C++ und Java, mal definitiv nicht für die deutsche Tastatur gemacht. Die geschweiften Klammern, die man alle 7 Sekunden benötigt, sind ein Sehnenscheidenentzündungsgarant. Mehr Tipparbeit erfordert Visual Basic .NET dennoch nicht: Das Tippen eines `If` führt automatisch zur Einfügung von `End If`, das von `Class` zu `End Class` und das von `Function` zu `End Funtion`. Dieses Verfahren ist bei allen Strukturbefehlen dasselbe.

   Also: 3:0 für Visual Basic .NET.

4. Nicht nur für Ein- und Umsteiger sind die vorhandenen Codeausschnitte eine enorme Erleichterung: Mit Strg K, Strg X gibt es in Visual Basic 2008 (und im übrigen auch schon in der 2005er Version) eine riesige Bibliothek mit kleinen Codesnippets, die wiederkehrende Coding-Aufgaben auf ein Minimum an Zeitaufwand reduzieren. Diese Snippets gibt es vom Grundsatz her auch in C# – allerdings ist hier nicht der Arbeitsaufwand sondern die Codeausschnittsbibliothek auf ein Minimum reduziert.

   Und damit: 4:0 für Visual Basic.

Fairerweise gebe ich zu, dass es einige Argumente für Visual Basic .NET gibt, die gerne in solchen Diskussionen genannt werden, aber nicht wirklich pro Visual Basic .NET sind. Beispielsweise, dass es ein Haufen von »altem« Visual Basic 6.0-Code gibt, den man einfach per *Cut/Copy/Paste* in .NET-Projekten zum Laufen bekommt – leider nicht, denn VB6 und VB.NET sind zu verschieden, als dass das sicher klappen würde; das trifft eher nur in wenigen Ausnahmefällen zu. Und auch die leichtere Erlernbarkeit von Visual Basic .NET im Gegensatz zu C# trifft nur bedingt zu. Sicherlich »fühlen« sich Visual Basic 6.0-Programmierer anfangs besser aufgehoben, da ihnen grundsätzliche Sprachelemente bekannter vorkommen; doch ehrlicherweise

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

muss ich aus meinen Schulungserfahrungen sagen, dass das spätestens beim Thema OOP aufhört. Dort ist vielleicht sogar ein sauberer Schnitt eher brauchbar, da bekannte Strukturen nicht zum alten prozeduralen Programmieren wie in VB6 verleiten.

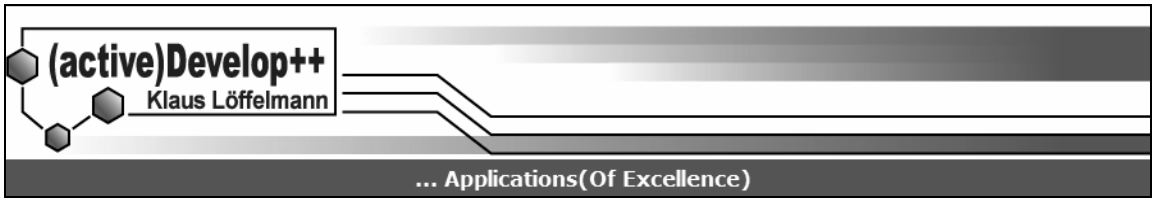
Aber dennoch. Sie sehen, dass Visual Basic .NET eindeutig die produktivere Programmiersprache ist. Und Sie sind vor allen Dingen gerüstet für die nächsten Diskussionen, aus der Sie als Sieger hervorgehen können! Und falls alles nichts hilft, dann schauen Sie Ihrem C#-Diskussionsgegner tief in die Augen, und sprechen Sie C# (eigentlich gesprochen: *ßieh schahp*) einfach »zisch ab« aus.

Lassen Sie mich aber noch zum eigentlichen Thema kommen: Vor Ihnen liegt ein Buch, das den Anspruch hat, Ihnen auf möglichst kompakte Art und Weise die Neuheiten in Visual Basic 2008 vorzuführen. Natürlich erhebt es keinen Anspruch auf Vollständigkeit: Alleine das Thema *Windows Presentation Foundation* füllt locker eigene Bücher, sogar Buchreihen. Doch sind die meisten dieser Bücher (jedenfalls im Moment noch) in der Regel für die C#ler geschrieben. Dieses Buch möchte Ihnen deswegen einen Einstieg in alle neuen 2008er Themen ermöglichen, Sie also quasi in die jeweils richtige Richtung schicken, von der aus Sie sich alles weitere dann selber erarbeiten oder mit entsprechender weiterführender Literatur weitermachen können. Ein Tipp dazu: Die Online-Hilfe ist – trotz oftmals hapernder, da mit Übersetzungsfehlern zu kämpfender Texte – gar nicht so schlecht wie ihr Ruf. Gönnen Sie ihr ruhig mehrere Blicke, wenn Sie mit der Lektüre dieses Buches durch sind, oder auch schon währenddessen!

In diesem Sinne,

*Klaus Löffelmann am 20.01.2008*

# http://www.activedevelop.de – Ein wenig Werbung in eigener Sache




Man bekommt nicht oft die Möglichkeit, für sein eigenes Business ohne größeren finanziellen Aufwand werben zu können. Umso mehr mag ich natürlich die Gelegenheit am Schopf packen, um an dieser Stelle ein wenig Öffentlichkeitsarbeit für unser eigenes Dienstleistungsunternehmen zu betreiben. So denn: Falls Sie Bedarf an

- Beratungsdienstleistungen in Sachen .NET 1.1, 2.0, 3.5, LINQ und SQL Server 2005/2008,
- Schulungen (auch inhouse) mit den Schwerpunkten »Migration von VB6-Anwendungen«, SQL Server 2005/2008 und ADO.NET 2.0, C# 2005/2008 und Visual Basic 2005/2008 sowie Einführung in neue Technologien zu Visual Studio 2008,
- Dokumentationserstellung, Übersetzungen und Softwarelokalisationen oder
- Software Development Outsourcing

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

haben, informieren Sie sich einfach auf unserer Website unter *http://www.activedevelop.de,* und setzen Sie sich mit uns in Verbindung.

Mit einem immer größer werdenden Team von Entwicklern und Designern ist ActiveDevelop in der Lage, auch große Projekte zu stemmen – und viele erfolgreiche Bücher und Referenzen von abgeschlossenen Projekten namhafter Firmen unterstreichen unsere fachliche Kompetenz und unsere Zuverlässigkeit.

# Codebeispiele

Der herunterladbare Code umfasst für die meisten Kapitel Projekte, in denen Sie auch die Codefragmente und Beispiele der einzelnen Kapitel finden. Die gesamten im Buch behandelten Codebeispiele können Sie von der Website unter der folgenden Adresse herunterladen:

*www.microsoft-press.de/support.asp?s110=518*

Ergänzende Änderungen zum Buch und eine Errata-Seite finden Sie ferner unter

*www.activedevelop.de*

# Support

Es wurden alle Anstrengungen unternommen, um die Korrektheit dieses Buchs und des Begleitinhalts sicherzustellen.

Microsoft Press stellt unter der folgenden Internetadresse eventuell notwendige Korrekturen zu veröffentlichten Büchern zur Verfügung:

*http://www.microsoft.com/germany/mspress/support*

Sollten Sie Anmerkungen, Fragen oder Ideen zu diesem Buch haben, senden Sie diese bitte an eine der folgenden Adressen von Microsoft Press:

Postanschrift:
Microsoft Press
Betrifft: Visual Basic 2008 – Neue Technologien – Crashkurs
Konrad-Zuse-Straße 1
85716 Unterschleißheim

E-Mail:
presscd@microsoft.com

Beachten Sie bitte, dass unter der oben angegebenen Adresse kein Produktsupport geleistet wird. Supportinformationen zum .NET Framework, zu Visual Basic .NET oder Visual Studio finden Sie auf der Microsoft-Produktsupportseite unter:

*http://www.microsoft.com/germany/support*

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Danksagungen

Aus vielerlei Gründen blieben mir rund 6 Wochen, dieses Buch zu schreiben – und ehrlich gesagt: das war ein sehr sportlicher Plan. Alleine hätte ich das nicht geschafft. Auf keinen Fall. Immerhin hat dieses Buch nunmehr deutlich über 350 Seiten, obwohl schon die 288 Seiten, mit denen es einst geplant war, nicht zu schaffen gewesen wären, hätte es nicht so viele helfende Hände gegeben.

Und da möchte ich an erster Stelle Bernd Marquardt nennen und mich tausendmal bei ihm bedanken, denn ohne ihn hätte es das Kapitel über die Windows Presentation Foundation nicht gegeben. Ich durfte seinen auf C# ausgerichteten WPF-Crashkurs als Vorlage nehmen, und auch wenn die WPF keine wirklich neue Technologie in Visual Studio 2008 bzw. dem .NET Framework 3.5 ist, so haben wir mit dem Adaptieren der wichtigsten Themen seines Buches viele Leserwünsche erfüllt, die um mehr Visual Basic-Beispiele bei diesem Themenkomplex baten. Deswegen, Bernd, herzlichen Dank dafür! In diesem Zusammenhang möchte ich mich auch bei Jürgen Heckhuis bedanken, der Bernds C#-Beispiele quasi nach Visual Basic 2008 übersetzt hat.

Herzlicher Dank gilt auch Andreas Belke, der sich mit großem Einsatz um das 9. und 10. Kapitel verdient gemacht hat. Wahrscheinlich war er einer der ersten Entwickler in Deutschland, der schon seit Monaten vor Erscheinen dieses Buches produktiv mit *LINQ to SQL*, *LINQ to Objects* und *LINQ to XML* in einem konkreten Projekt arbeiten durfte, und deswegen mehr als prädestiniert war, diese beiden Kapitel zu schreiben. Die Entscheidung, zu einem so frühen Zeitpunkt bereits eine neue Technologie zu verwenden, hat sich übrigens mehr als richtig herausgestellt, denn wie unsere Erfahrung zeigt: LINQ kann extrem viel Zeit sparen, und wenn das Entity-Framework mit dem SP1 zu Visual Studio 2008 erst einmal da ist, können wir Entwickler uns wirklich über ein richtig »cooles« Set an extrem arbeitserleichternden Werkzeugen freuen.

### Dann war da noch … mein persönlicher LINQ to Entities!

Und dann gibt's da noch diese liebenswerte Gang in Ratingen-Hösel, an die ich unbedingt ein paar persönliche Worte richten muss: Was zwischen uns an Beziehung aus einem einst »einfachen« Fachlektorat entstand, kann ich gar nicht in Worte fassen, und ich denke, ich muss es auch gar nicht – es ist doch viel lustiger, sich die Vorstellung anderer Menschen zu diesem Thema anzuhören, findet ihr nicht? ;-)

Eines weiß ich jedoch sicher: Wenn ich an euch alle denke und mal gerade nicht sowieso bei euch 'rumlungere, vermisse ich euch voll, nämlich, wohl. Ich finde, ihr seid was ganz Besonderes, alle miteinander, und seit Wochen mache ich mir schon Gedanken, wie ich mich einmal für all die schönen Stunden, den Spaß, den ich mit euch beim Philosophieren hatte und habe, die Sushi-Sessions, die ich mit Euch halten durfte, und von denen es hoffentlich noch hunderte geben wird, die neuen An- und Einsichten, die ich durch euch erlangt habe, die Cabrio-Touren, die wir miteinander unternommen haben, und für Vieles, Vieles mehr bedanken könnte. Und jetzt hätte ich hier gerade *zufällig* ein Buch übrig, das danach schreit, gewidmet zu werden und deswegen widme ich es euch, Ada, Zoe, Sarah, David, Ute und Ruprecht. Ihr seid meine Lieblingsentitäten und – Achtung, schlimmer Kalauer – ich danke meinem Schicksal sehr für diesen ganz persönlichen *LINQ to Entities*!

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Teil A

# Schnell zurechtfinden

**In diesem Teil:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 1

# Einführung

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Die Geschichte von .NET

Lassen Sie uns eine kurze Bestandsaufnahme machen: Wir hatten die .NET Frameworks 1.0 und 1.1, das waren aus Entwicklersicht die Anfänge von Microsofts .NET-Initiative. Für erste Versionen liefen sie erstaunlich stabil, auch wenn sie natürlich schon zum damaligen Zeitpunkt noch merklich ausbaufähig waren. In erster Linie waren sie für uns Visual Basic-Programmierer aber auf jeden Fall mal ein Schock, denn sie führten in der Version 8.0 das erste Mal echtes OOP und damit mehr als nur objektbasierende Programmierung ein – für viele von uns der eigentliche Grund, zunächst bei VB6 zu bleiben, und das mit der Schutzbehauptung »Wir warten ja mindestens auf die reifere Version 2.0«.

Dann kam die Version 2.0, und wow! Die brachte uns Generics und zu C# kompatible Operatorenüberladung, einen gut optimierten Visual Basic-Compiler, der auch geschwindigkeitsmäßig erwachsen und dem C#-Compiler ebenbürtig geworden war, den `My`-Namespace mit einer ganzen Bibliothek an simpelst aufrufbaren Miniprogrammen und -routinen für (fast) jeden Zweck. Und auch einen gründlich überarbeiteten Editor samt vieler neuer cooler Funktionen, aber der litt anfangs noch gerade bei Mehrprozessorsystemen an Schluckauf in Form von Racing Conditions zwischen ihm und dem Backgroundcompiler, und zog deswegen oft das ganze System ins Nirwana, was aber dann mit SP1 und dem kurze Zeit später erscheinendem SP1 fürs SP1 (genannt: *Update für Vista*) weitestgehend behoben wurde. Und es kamen neu die so genannten `Nullable`-Datentypen – allerdings in ihrer finalen Version vergleichsweise spät in der Produktentwicklung, um nicht zu sagen: zu spät, denn sie waren neben anderen Dingen Wegbereiter dafür, dass die Entwicklung von SQL Server 2005 und Visual Studio 2005 begann, quasi »Out of Sync» zu laufen. Der Hintergrund:

`Nullable`-Datentypen versetzen auf der Basis von Generics normale Datentypen in die Lage, den Wert `Null` (also nicht *0*, sondern `Nothing` in Visual Basic) anzunehmen. Boxt man nun einen `Null`-repräsentierenden Datentyp in ein Objekt, sollte es konsequenterweise auch `Nothing` widerspiegeln, das wäre logisch, und so sieht es auch die finale Implementierung der .NET 2.0-CLR vor – nicht aber die Version, die noch kurz vor der RTM der 2.0 CLR zum Testen unterwegs war: Dort konnte – gemäß Standardimplementierung – *natürlich* ein definierter und in ein Objekt geboxter Wertetyp *nicht* dazu führen, dass ein Objekt `Null` (`Nothing`) wurde. Die gewünschte finale Verhaltensweise konnte man deswegen nur mit einem heftigen Eingriff in der CLR selbst erreichen, und diesen Eingriff kurz vor Schluss vorzunehmen, war nicht nur gewagt, sondern hatte natürlich weitere Konsequenzen für alle darauf basierenden Entwicklungen – wie beispielsweise den SQL Server 2005, der es ja zum ersten Mal ermöglichte, dass .NET-Assemblies in seinem Kontext zur Ausführung kommen konnten, sodass er von dieser Änderung ja nicht gerade nur am Rande betroffen war. Nullables sind ja schließlich sehr essentiell, wenn es um das Speichern von Datentypen geht, die auch in der Datenbank *Null* werden. Schlussendlich ist alles in allem die Version 2.0 des .NET Frameworks eine wirklich runde, sauber laufende und bis Windows 2000 als Plattform herunterreichende Lösung, mit der es – meiner Meinung nach jedenfalls – unglaublich viel Spaß macht, zu entwickeln.

## .NET Framework 3.0

Und dann kam Windows Vista mit dem .NET Framework 3.0. Und das mit .NET Framework 3.0 ist, jedenfalls was die Versionsgebung anbelangt, so eine Sache: Denn während 1.1 und 2.0 die jeweils vorherigen Versionen CLR-technisch verbesserten, Fehler ausmerzten und teilweise leider auch mit neuen Konzepten für so

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

genannte Breaking Changes[1] sorgten, ist das .NET Framework 3.0, das übrigens das erste .NET Framework war, das werksseitig in einem Betriebssystem (nämlich Vista) vorhanden war, nur eine *Erweiterung*. Die Kernkomponenten wurden in keinster Weise angefasst, und so gab es auch keine Breaking Changes. Stattdessen gibt es ein paar zusätzliche neue Assemblies, die Bestandteile des .NET Frameworks 3.0 wurden, aber die haben es wirklich in sich:

- Die **Windows Presentation Foundation** (WPF), zu ihrer Entwicklungszeit eher bekannt unter dem Codenamen **Avalon**, stellt Klassenbibliotheken für ein neues Benutzerschnittstellenkonzept dar, deren Struktur und Aufbau prinzipiell durch eine Abwandlung von XML namens XAML[2] beschrieben werden. Auf diese Weise können – ähnlich wie beim Code-Behind-Verfahren von ASP.NET – Design und Programmlogik erstmals auch in Fat Clients voneinander getrennt werden. Obendrein gibt es zusätzliche API-Funktionen für eine leistungsfähige Animations- und 3D-Engine, die sich direkt der 3D-Computergrafikhardware der Grafikkarten sowie der Direct3D-Technologien bedient, in die sich die Benutzeroberflächen-Klassenbibliothek nahtlos eingliedert. Die Windows Presentation Foundation gibt's ab Kapitel 11 – auch wenn sie »nur« zum .NET Framework 3.0 gehört, haben wir ihr einen ganzen Buchteil (Teil C) gewidmet, da Visual Studio 2008 mit einem eigenen Designer erstmals WPF-Projekte unterstützt.
- Die **Windows Communication Foundation** (WCF), zu ihrer Entwicklungszeit eher bekannt unter dem Codenamen **Indigo**; stellt ein dienstorientiertes Nachrichtensystem dar, mit dem die Entwicklung von lokal oder entfernt miteinander kommunizierenden Komponenten möglich wird. In Ergänzung zu den schon in vorherigen .NET Framework-Versionen bekannten Web Services können auch DTS-kompatible und damit transaktionsfähige Komponenten oder Peer-To-Peer-fähige Szenarien wie Bilddateientausch-Komponenten entwickelt werden.
- Die **Windows Workflow Foundation** (WF) ermöglicht es, mithilfe von Workflows Aufgabenautomatisierung und integrierte Transaktionen zu erstellen.
- **Windows CardSpace**, während der Entwicklungszeit eher bekannt unter dem Codenamen **InfoCard,** stellt eine Softwarekomponente dar, um gesicherte digitale Identitäten beispielsweise für das Anmelden bei Webdiensten speichern zu können.

## .NET Framework 3.5

Das .NET Framework 3.5 benutzt prinzipiell ebenfalls die Version 2.0 der CLR, ist aber, anders als die 3.0 Version eben nicht nur ein purer Aufsatz bzw. eine Bibliotheksergänzung, sondern installiert das Service Pack 1 des .NET Framework 2.0, das den Wegbereiter für neue Techniken wie LINQ darstellt. Microsoft

[1] Breaking Changes in diesem Zusammenhang sind Änderungen am Framework, die unter bestimmten Umständen dafür sorgen, dass bereits für eine bestimmte Version konzipierte Programme bei einer Versionsänderung nicht mehr laufen, da sich grundlegende Verhaltensweisen geändert haben. Das hat aber nur auf den ersten Blick wirklich schlimme Auswirkungen, denn das .NET Framework-Konzept sieht es vor, dass mehrere Versionen des Frameworks parallel auf einem Rechner existieren. Im schlimmsten Fall musste also eine ältere Version des Frameworks nachinstalliert werden, um Breaking Changes durch einen Versionswechsel zu umgehen.

[2] Etwa wie die bayrische »Semmel« mit einem »G« davor ausgesprochen, also etwa »Gsämmel«

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

verspricht, dass das .NET Framework 2.0 *mit* SP1 bei der Erweiterung so behutsam vorgeht, dass keine Breaking Changes für bereits vorhandene auf .NET Framework 2.0 abzielende Software zu erwarten sind.

Die exakten Spracherweiterungen, die das .NET Framework 3.5 für Visual Basic 2008 ermöglicht, sind zentraler Bestandteil des ersten Teils dieses Buches, und eine Schnellübersicht finden Sie in und ab Kapitel 3.

## Welche Softwarevoraussetzungen benötigen Sie?

Um die Beispiele in diesem Buch nachzuvollziehen, benötigen Sie eine aktuelle Version von Visual Studio 2008 in der Professional Edition. Und es wird Sie freuen, zu hören, dass auf der beiliegenden DVD eine 90-Tage-version genau dieser Version beiliegt. »90-Tage-Testversion« bedeutet, dass Sie diese Version von Visual Studio immerhin ein viertel Jahr kostenlos und *ohne Einschränkungen* verwenden können. Nur: nach Ablauf dieser Probefrist müssen Sie sich entscheiden, wie es weitergeht. Empfehlen kann ich den Lesern unter Ihnen, die auch nach Ablauf dieser Probefrist weiterhin professionell mit Visual Studio 2008 arbeiten möchten, auf jeden Fall den Kauf der Professional Edition zumindest aber der Standardversion von Visual Studio 2008 in Erwägung zu ziehen. Haben Sie sich für die Professional Version entschieden, brauchen Sie noch nicht einmal die Installation zu wiederholen: Ein neuer Installations-Key, den Sie beim wiederholten Ausführen des Setups eingeben können, ist dann ausreichend, um die bereits installierte Version freizuschalten.

### Die kostenlose Alternative: die Express Editions

Es gibt auch eine preiswertere Alternative, die wahrscheinlich die bessere für die Hobbyisten oder die Gelegenheitsentwickler unter Ihnen darstellt. Microsoft bietet nämlich mit den so genannten Express Editions abgespeckte Versionen von Visual Studio 2008 an, die jeweils nur eine Programmiersprache beinhalten. So gibt es beispielsweise die C# Express Edition, die C++ Express Edition und natürlich auch die Visual Basic Express Edition.

Und, wie schon gesagt, diese Version können Sie kostenlos von der Microsoft Internetseite herunterladen. Für viele Beispiele dieses Buches, gerade wenn es um LINQ to SQL geht (siehe Kapitel 9), ist diese Version allerdings nicht ausreichend, um alle Funktionen zu nutzen.

# Wissenswertes zur Installation von Visual Studio 2008

Eigentlich müssen Sie nichts Außergewöhnliches beachten, wenn Sie Visual Studio auf Ihrem Computer zum Laufen bringen wollen. Die folgende Tabelle zeigt Ihnen die ideale Hardwarevoraussetzung, die ein reibungsloses Arbeiten mit Visual Studio 2008 ermöglicht.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**WICHTIG** Orientieren Sie sich bei dieser Liste lieber nicht an der angegebenen Mindestkonfiguration nach Microsoft-Spezifikation, die eher humoristischen Anforderungen genügen, sondern besser an der »Wohlfühl-Rundum-Sorglos«-Konfiguration, mit der das Arbeiten Spaß macht, und die nicht wegen langer Wartezeiten zu ständiger Nervosität, Hypertonie, Tachykardie oder Schlaflosigkeit auf Grund zu hohen Kaffeekonsums führt. Ein paar Euro in ein wenig Hardware investiert, können Ihre Turn-Around-Zeiten[3] beim Entwickeln drastisch verbessern!

Denken Sie auch daran, dass es sich bei vielen der heutigen Prozessoren, die als Standardkonfiguration von Computern verbaut werden, automatisch um so genannte Dual-Core-Prozessoren (bzw. Quad-Core-Prozessoren) handelt. Solche Prozessoren vereinen zwei (bei Quad-Core-Prozessoren sogar vier) Prozessorkerne unter einem Dach. Ganz einfach ausgedrückt bedeutet das: befindet sich ein Dual-Core-Prozessor in Ihrem Computer, dann verfügt Ihr Computer quasi über *zwei* völlig unabhängig voneinander arbeitende Prozessoren. Und bei Quad-Core-Prozessoren sind das sogar dann vier an der Zahl. Nun gilt es für die Ausstattung von Entwicklungsrechnern folgendes in Erwägung zu ziehen: Bei der Entwicklung von so genannten Multi-Threading-Anwendungen – das sind Anwendungen, bei denen verschiedene Programmteile quasi gleichzeitig ablaufen können – gibt es auf Single-Core-Prozessoren unter Umständen völlig andere Verhaltensweisen dieser parallel laufenden Programmteile als auf Dual- oder Quad-Core-Prozessoren, auf denen Programme ja tatsächlich gleichzeitig laufen können, da sie eben von zwei unabhängigen Prozessorkernen ausgeführt werden. Dem gegenüber stehen Single-Core-Computer, auf denen Multi-Threading-Anwendungen nur scheinbar gleichzeitig laufen – hier wird zwei scheinbar gleichzeitig laufenden Multi-Threading-Programmteilen Prozessorzeit im ständigen Wechsel zur Verfügung gestellt. Sie können auf einem Single-Core-Prozessor also unter Umständen gar nicht alle unterschiedlichen Aspekte einer Multi-Threading-Anwendung erfassen, nachstellen und testen. Deswegen ist es auf jeden Fall empfehlenswert, dass Ihr Entwicklungsrechner ebenfalls über mindestens zwei unabhängig arbeitende Kerne verfügt, damit die Anwendungen, die Sie entwickeln, auch später für Multi-Core-Prozessoren gerüstet sind.

| Komponente | Mindestkonfiguration laut Microsoft | Ideal-Konfiguration (erfahrungsgemäß) |
|---|---|---|
| Prozessor | 1,4 GHz | Dual Core Prozessor mit ausreichend großem Second Level Cache, zum Beispiel Intel Core 2 Duo E6600, Core 2 Quad E6600 oder AMD Athlon 64 X2 6000+.[4] |
| Ram | 256 MByte (Windows XP SP2) bzw.<br>512 MByte (Windows Vista) | 1 GByte unter Windows XP SP2 bzw. 2 GByte unter Windows Vista. |
| Festplattengröße (incl. MSDN-Hilfe) – also freier Speicherplatz.<br>**HINWEIS:** Denken Sie daran, dass gerade bei Notebooks noch ausreichend freier Speicher für Systemdateien, insbesondere für den Suspend-Modus bis zu 6 GBytes freier Speicher verbleiben müssen. | 5,4 GByte | 5,4 GByte |
| DVD-Laufwerk | Wird benötigt | Wird benötigt ▶ |

[3] Die Zeit, die man braucht, um beispielsweise nach dem Bemerken eines Fehlers und nach der entsprechenden Änderung im Code, das Programm neu zu kompilieren und wieder in den »Laufend«-Modus (*Run time mode*) zu versetzen.

[4] Wir wollen keinen der beiden großen Prozessorhersteller benachteiligen, aber zum Zeitpunkt, zu dem diese Zeilen entstehen, ist von der Firma AMD leider noch kein Quad-Core-Prozessor *wirklich* verfügbar.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Komponente | Mindestkonfiguration laut Microsoft | Ideal-Konfiguration (erfahrungsgemäß) |
|---|---|---|
| Grafikkarte | 800 x 600 256 Farben | 1280 x 1024 True Color; oder mindestens Dual-Monitor-Support mit zwei Mal 1024 x 768 bei True Color. Für den Einsatz unter Windows Vista achten Sie bitte auf eine DirectX 9.0-fähige Grafikkarte mit mindestens 128 MB Grafikkartenspeicher![5] |

**Tabelle 1.1** Die Minimal- und Idealvoraussetzungen an die Hardware für ein reibungsloses Arbeiten mit Visual Studio 2008

# Lauffähigkeit von Visual Studio und den erstellten Kompilaten unter den verschiedenen Betriebssystemen

Mit Visual Studio 2008 erstellte Programme laufen *nicht* auf Windows 95 und Windows NT, und unter Windows 98 oder Windows Millenium steht Ihnen nicht der komplette Funktionsumfang des .NET Framework 2.0 zur Verfügung (.NET Framework 2.0-Programme laufen aber grundsätzlich unter diesen beiden älteren Betriebssystemen entgegen vieler Meinungen schon). Sie benötigen allerdings *mindestens* Windows XP Home SP2, Windows XP Professional SP2, die XP-Media-Center-Edition SP2, Windows 2003 Server (natürlich auch R2), Windows 2008 Server oder eine der Windows Vista-Versionen (aber nicht die Starter-Edition! – Home Basic funktioniert hingegen), um die Entwicklungsumgebung (entweder die kostenlose Express Edition oder eine der Visual Studio-Vollversionen) installieren zu können. Die Installation der Entwicklungsumgebung unter Windows 2000 ist nicht mehr möglich.

## .NET Framework Version Targeting (Versionsbestimmung des .NET Frameworks für Kompilate)

Wichtig ist es in diesem Zusammenhang zu wissen, dass Sie mit Visual Studio 2008 bestimmen können, mit welcher .NET Framework-Version die von Ihnen entwickelte Anwendung funktioniert. Sie können also bestimmen, dass das Programm, das Sie gerade entwickeln, das .NET Framework 2.0, das .NET Framework 3.0 oder eben das .NET Framework 3.5 verwenden soll (auf Neudeutsch spricht man dabei vom sogenannten ».NET Framework Version Targeting«).

**HINWEIS** Dabei ist wiederum wichtig zu wissen, dass Programme, die noch unter Windows 2000 SP4 laufen sollen, höchstens auf das .NET Framework 2.0 abzielen können. Die Installation des Microsoft .NET Frameworks 3.0 oder 3.5 ist unter Windows 2000 nicht mehr möglich (und das ist der eigentliche Grund, weswegen Sie für dieses Betriebssystem keine Programme mehr schreiben können, die auf diese .NET Framework-Versionen abzielen).

Unter Windows XP hingegen lassen sich beide neuen .NET Framework-Versionen installieren. Voraussetzung dafür ist allerdings, dass das Service Pack 2 für Windows XP zuvor eingerichtet wurde.

Und jetzt noch eine letzte Anmerkung in diesem Zusammenhang: Windows Vista sowie Windows Server 2008 haben sowohl das .NET Framework 2.0 als auch das .NET Framework 3.0 standardmäßig an Bord. Das .NET Framework in der Version 3.5 muss hingegen auch auf diesen Betriebssystemen individuell installiert werden.

[5] Kein Budget für einen zweiten Bildschirm, aber Sie haben einen Laptop? Nutzen Sie Ihren Laptop als Zweitbildschirm. Mehr dazu gibt es unter dem IntelliLink *A0103*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Parallelinstallation verschiedener Visual Studio-Versionen

Seit Version 6 können alle Visual Studio-Versionen nebeneinander installiert werden. Das gilt allerdings nur für Windows XP SP2 als Betriebssystemplattform, denn sowohl Visual Studio 2002 als auch Visual Studio 2003 lassen sich leider nicht unter Vista zum Laufen bringen. Visual Studio 6 läuft hingegen mit Einschränkungen und auch die mit Visual Basic 6.0 entwickelten Anwendungen sind auf Vista lauffähig.

**HINWEIS** Die Wahrscheinlichkeit ist vergleichsweise groß, dass Visual Basic 6.0 und Visual Basic 6.0-Anwendungen auf zukünftigen Betriebssystemplattformen nicht mehr lauffähig sein werden! Rechtzeitiges Portieren der Anwendungen von VB6 auf entsprechende .NET-Pendants sollten Sie deswegen nicht auf die lange Bank schieben.

# Wieso fehlen noch Features? – Visual Studio 2008 Service Pack 1

Visual Studio 2008 steht noch nicht in den Regalen und schon ist die Rede vom ersten Service Pack zu Visual Studio 2008 – darf das sein?

Es muss. Sollten Sie zu Beginn dieses Kapitels die »Geschichte von .NET« gelesen haben, dann haben Sie vielleicht auch mit Interesse die kleine Anekdote zur *Nullable-Problematik* und dem *Out Of Sync*-Laufen der SQL Server- und der Visual Studio 2005-Produktentwicklung verfolgt. Solche Dinge können beim Entwickeln natürlich passieren, denn: Wenn ein Haupt-Feature eines großen Programms von einer Komponente abhängt, die selbst gerade noch in der Entwicklung ist, sind Zeitprobleme buchstäblich vorprogrammiert. So war es bei SQL Server 2005 und dem .NET Framework 2.0, und so ist es zurzeit (Februar 2008): Momentan gibt es nämlich Probleme beim Verwenden von Visual Studio 2008 mit SQL Server 2008 aus ganz ähnlichen Gründen. Und die können erst mit dem Service Pack 1 von Visual Studio 2008 final behoben werden. Die genauen Probleme finden Sie im Abschnitt »Visual Studio 2008 und SQL Server 2005 bzw. SQL Server 2008« beschrieben.

## LINQ to Entities erst mit SP1

Und dann fehlt auch noch ein großes Feature in der aktuellen Version von Visual Studio 2008: Das sogenannte Entity-Framework und damit *LINQ to Entities*. Dazu folgender Hintergrund:

LINQ dient grundsätzlich dazu, aus einer fast beliebigen Datensammlung mit einfachen, in die Sprache Basic integrierten Befehlen, Daten aus zu selektieren. Bislang, also ohne Service Pack 1, können Sie LINQ dazu verwenden, um …

- … Arrays und Auflistungen zu selektieren – das nennt sich dann *LINQ to Objects*. Praktisch jede Auflistung, die die Schnittstellen `IEnumerable` und `IEnumarable(Of Type)` implementieren, können damit selektiert werden.
- … XML-Dokumente zu selektieren – das nennt sich dann *LINQ to Xml*. *LINQ to Xml* ist prinzipiell ein Programmiermodell für im Speicher befindliche XML-Dokumente, die mit den eingebauten Abfragebefehlen selektiert werden können.
- … DataSets zu selektieren, deren Daten bereits von einem Datenprovider in ein DataSet geladen wurden – das nennt sich dann *LINQ to DataSets*. Bitte verwechseln Sie das nicht mit *LINQ to SQL* oder *LINQ to Entities*, bei denen der Umweg über ein DataSet eben *nicht* mehr erforderlich ist, und Daten aus einer Datenbank oder einer anderen Datenquelle direkt heraus selektiert werden, bevor sie in ein Businessobjekt – also in den Arbeitsspeicher Ihres Rechners gelangen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- … direkt die Daten einer Microsoft SQL-Datenbank abzufragen. Das nennt sich dann *LINQ to SQL*. Hierbei werden die LINQ-Befehle, die in die Visual Basic-Sprache integriert sind, direkt in SQL-Abfragebefehle konvertiert, die dann die gewünschten Objekte in sogenannten Businessobjekten gemaped zurückliefert.
- … direkt die Daten einer beliebigen Datenquelle abzufragen. Und das nennt sich dann *LINQ to Entities*. Das Problem: Das Entity-Framework, das das erlauben würde, ist schlicht und ergreifend noch nicht fertig, und wird erst mit dem Service Pack 1 verfügbar sein.

## Visual Studio 2008 und SQL Server 2005 bzw. SQL Server 2008

Während diese Zeilen entstehen, ist die englische Version von Visual Studio 2008 bereits »RTMt« (sprich: *er-te-emmt*), was die Abkürzung von »Release to Manufactoring« ist und soviel wie »Für die Produktion freigegeben« bedeutet. Das gilt aber leider nur für Visual Studio 2008 und nicht für SQL Server 2008 – der wird aller Voraussicht nach später im Jahr erscheinen, wahrscheinlich knapp nach dem SP1 für Visual Studio 2008.

Bis dahin gibt es für den SQL Server 2008 so genannte Community Technical Previews, CTPs (etwa: »Technische, öffentliche Vorschauversionen«), mit denen Sie sich schon im Vorfeld einen Eindruck der neuen Features verschaffen können. Das Problem dabei ist: Aufgrund von Framework Versions-Inkompatibilitäten können Sie Verbindungen nämlich nur aus bestimmten Release-Ständen von Visual Studio 2008 heraus zu bestimmten Versionen von SQL Server 2008 herstellen. Je nach Release-Stand können sich Ihre entwickelten Programme mit dem SQL Server 2008 verbinden; es gibt dann aber keine Designer-Unterstützung (was gerade für das Erlernen von *LINQ to SQL* etwas kontraproduktiv ist) – bei einigen Release-Kombinationen ist sogar gar keine Zusammenarbeit möglich.

Visual Studio 2008 funktioniert hingegen mit SQL Server 2005 problemlos, sowohl was Designerunterstützung als auch Ihre entwickelten Anwendungen anbelangt. In diesem Buch lassen wir uns deswegen gar nicht erst auf irgendwelche Beta-Experimente ein: Alle Beispiele und Beispieldatenbanken aus den entsprechenden *LINQ to SQL*-Beispielen arbeiten grundsätzlich mit der Kombination Visual Studio 2008/SQL Server 2005.

# Die weiteren Kapitel in diesem Teil

Möglicherweise fallen Ihnen beim Durchlesen des Inhaltsverzeichnisses oder beim Durchblättern dieses Teils drei Kapitel ins Auge, die Themen behandeln, welche Ihnen nicht gerade brandneu erscheinen werden (was auch zutrifft):

- Klassen (Kapitel 4) hat es sicherlich schon in der ersten .NET-Version gegeben.
- Das Gleiche gilt für Auflistungen (Collections, Kapitel 5).
- Generics – oder Generika, wie es auf deutsch heißt – (ab Kapitel 6) gibt es dann wiederum zwar erst ab der 2005er Version, aber sicherlich kann und will ich Ihnen dieses Thema nicht als neu verkaufen.

Leider sind das die drei großen Themenkomplexe, die Sie beherrschen sollten, um alle fachlichen Voraussetzungen für das Thema LINQ parat zu haben, das im zweiten Teil (B) dieses Buches behandelt wird. Ohne

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Klassen, keine Auflistungen. Ohne Auflistungen im Allgemeinen gibt es natürlich auch keine *generischen* Auflistungen, mit denen Ihnen aber ebenfalls die Grundvoraussetzungen für LINQ, das wiederum zum großen Teil auf generischen Auflistungen basiert, fehlen würden.

Aus diesem Grund haben wir (Verlag und Autor) uns nach einigem Hin und Her dazu entschlossen, einige Seiten für die LINQ-Grundlagen zu ergänzen, denn noch heute ist es so, dass gerade Entwicklerteams mit einer langjährigen Visual Basic 6.0-Historie Visual Basic .NET vielfach rein prozedural einsetzen. Und das ist auch nicht notwendigerweise falsch – so antwortete ein Geschäftspartner eines guten Freundes von mir auf die Frage, wie er es geschafft habe, mit seinem Team ein komplettes ERP-System innerhalb nur eines Jahres realisieren zu können: »Wir haben konsequent auf objektorientiertes Programmieren verzichtet!«. Nun gut – hätte es LINQ damals schon gegeben, vielleicht hätten sie es dann in nur 10 Monaten geschafft, aber darum geht es ja gerade gar nicht.

Tatsache ist: Um LINQ richtig verstehen und anwenden zu können, sollten Sie diese drei Themenkomplexe beherrschen. Das Thema Klassen müssen Sie dabei nicht komplett verinnerlicht haben – in diesem Buch wird es deswegen gerade so ausführlich behandelt, wie es zum späteren Verständnis von LINQ notwendig ist.

Wer mehr in Sachen Klassen und OOP lernen möchte, kann sich meines Buches *Visual Basic 2005 – Das Entwicklerbuch* bedienen, das Sie komplett mit Beispielen unter *www.activedevelop.de* herunterladen können. Dessen Nachfolger, *Visual Basic 2008 – Das Entwicklerbuch*, erscheint im Übrigen erst im 3. Quartal 2008, damit es auch auf die erst mit SP1 kommenden Features eingehen kann.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 2

# Das ist neu in der IDE von Visual Studio 2008

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Der erste Start von Visual Studio

Auf den ersten Blick sind Unterschiede zum Vorgänger von Visual Studio 2008 kaum sichtbar. Den ersten Start vollziehen Sie genau so, wie Sie es beim Vorgänger gemacht haben. Visual Studio zeigt Ihnen einen Dialog, in dem Sie die Voreinstellungen der Benutzeroberfläche bestimmen können, etwa wie in Abbildung 2.1 zu sehen:


**Abbildung 2.1** Beim ersten Start des Programms bestimmen Sie, wie die Entwicklungsumgebung voreingestellt werden soll

In diesem Dialog bestimmen Sie, mit welchen Voreinstellungen die Entwicklungsumgebung konfiguriert werden soll. Wenn Sie es gewohnt sind, mit Visual Studio 2002, 2003 und 2005 in der Standardeinstellung zu arbeiten, wählen Sie an dieser Stelle *Allgemeine Entwicklungseinstellungen* aus.

**TIPP** *Allgemeine Entwicklungseinstellungen* ist die Einstellung, mit der die meisten Visual Studio 2005- und 2008-Installationen voreingestellt werden. Visual Basic-Entwicklungseinstellungen enthalten spezielle Anpassungen, bei denen Fensterlayout, Befehlsmenüs und Tastenkombinationen mehr auf das schnelle Erreichen spezieller Visual Basic-Befehle angepasst sein sollen. Der Dialog zum Anlegen eines neuen Projektes ist beispielsweise nur auf Visual Basic-Projekte angepasst, um nicht zu sagen: beschnitten. Einige Optionen, wie das automatische Anlegen einer Projektmappe sind beim Anlegen eines neuen Projektes ausgeblendet – Sie haben die Möglichkeit, Projekte namenlos zu erstellen und erst am Ende der Entwicklungssitzung unter einem bestimmten Namen zu speichern. Das gilt auch für Befehle, die Sie über Dropdown-Menüs abrufen können. Im Ergebnis sind unter den Visual Basic-Einstellungen viele Funktionen, die auch Visual Basic-Projekte betreffen könnten, nicht verfügbar.

Probieren Sie die für Sie am besten geeignete Voreinstellung aus. Falls Sie mit der hier gewählten später nicht mehr zufrieden sind, beherzigen Sie das im folgenden Absatz Gesagte.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Zurücksetzen der IDE-Einstellungen

Wenn Sie die gesamte IDE in ihren Ausgangszustand zurückversetzen möchten, wählen Sie aus dem Menü *Extras* den Menüpunkt *Einstellungen importieren und exportieren*. Visual Studio zeigt Ihnen anschließend den Dialog, den Sie auch in Abbildung 2.2 sehen können.

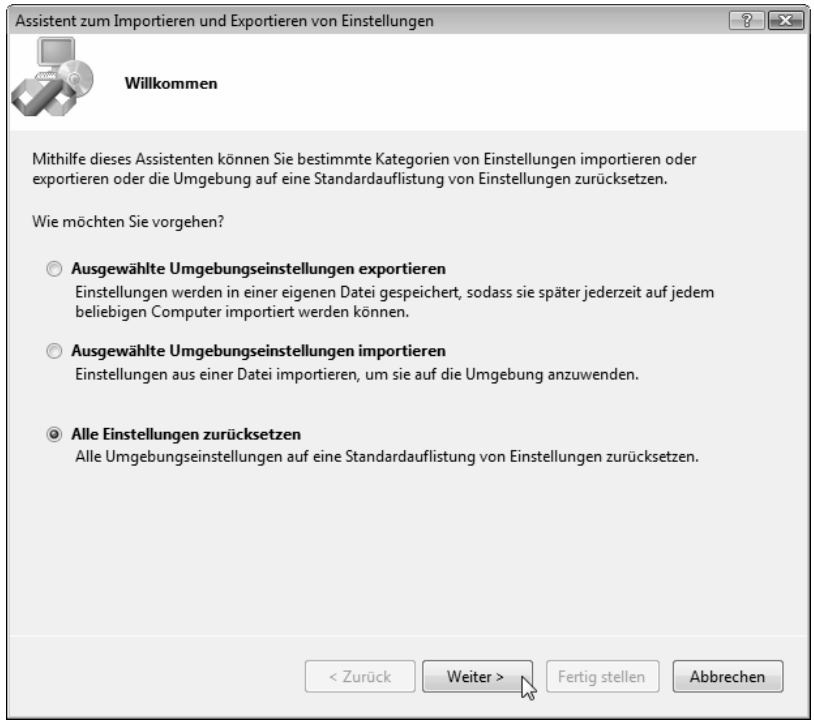



**Abbildung 2.2** Um mit diesem Dialog die IDE-Einstellungen in den Ursprungszustand zurückzuversetzen, wählen Sie aus dem Menü *Extras* den Menüpunkt *Importieren und Exportieren von Einstellungen*

**HINWEIS** Beachten Sie, dass der Vorgang, den Sie hier durchführen, die gesamte Entwicklungsumgebung in den Ausgangszustand zurücksetzt. Das bedeutet auch, dass bestimmte Schriftartenzuordnungen oder Tastaturkürzel, die Sie umdefiniert haben, dabei verloren gehen. Falls Sie nur das Fensterlayout von Visual Studio in den Originalzustand zurücksetzen wollen, wählen Sie aus dem Menü *Fenster* den Menüpunkt *Fensterlayout zurücksetzen*.

Klicken Sie auf *Weiter* und im Dialog, der jetzt erscheint, wählen Sie *Nein, nur Einstellungen zurücksetzen und die aktuellen Einstellungen überschreiben*. Der Dialog, der anschließend zu sehen ist, entspricht wieder weitestgehend dem in Abbildung 2.1. Wählen Sie Ihre bevorzugten Einstellungen und bestätigen Sie den Dialog mit *Fertig stellen*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Verbesserungen an der integrierten Entwicklungsumgebung (IDE)

Die IDE wurde an vielen Punkten überarbeitet und modernisiert.

## Einfacheres Positionieren und Andocken von Toolfenstern

Das Verschieben und Andocken von Toolfenstern ist leichter geworden; Das Verschieben und Andocken von Toolfenstern ist leichter geworden. Beim Ziehen mit der Maus wird ein so genanntes Diamant-Führungssymbol angezeigt. Indem Sie bei gedrückter Maustaste den Mauszeiger auf eines der Pfeilsymbole führen und dann loslassen, können Sie bequem die Zielposition bestimmen (siehe Abbildung 2.3).

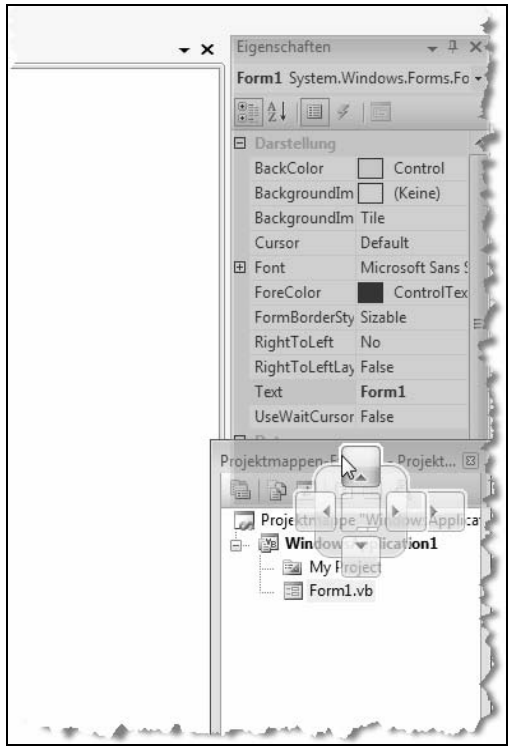



**Abbildung 2.3** Mithilfe des Diamant-Führungssymbols lassen sich Fenster schnell und bequem positionieren

## Navigieren durch Dateien und Toolfenster mit dem IDE-Navigator

Um im Codeeditor zu einer bestimmten geöffneten Datei zu navigieren, egal wann diese zuletzt verwendet wurde, können Sie den IDE-Navigator verwenden. Die Funktionsweise des IDE-Navigators ist in etwa dieselbe, wie die des Task-Switchers in Windows selbst, den Sie mit der Tastenkombination Alt ⇆

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

bedienen. Der IDE-Navigator ist nicht über Menüs verfügbar. Sie können ihn über die Tastenkombinationen Strg ↹ bzw. Strg ⇧ ↹ bedienen – je nachdem, in welcher Reihenfolge Sie durch die Dateien navigieren möchten (siehe Abbildung 2.4).

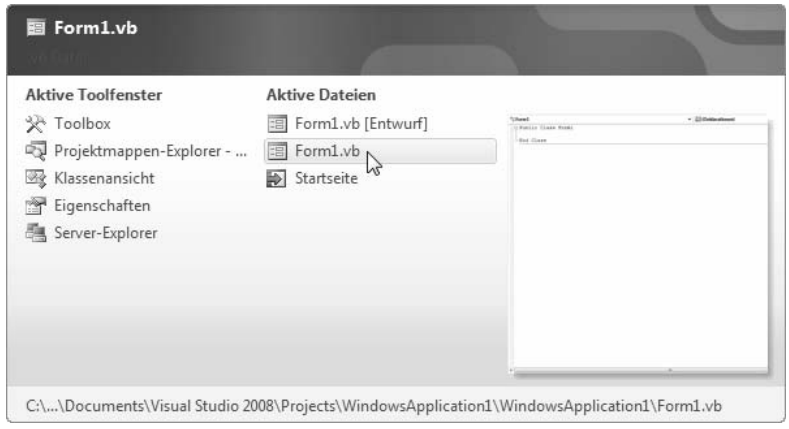



**Abbildung 2.4** Den IDE-Navigator rufen Sie mit Strg ↹ oder Strg ⇧ ↹ ins Leben

Drücken Sie bei gehaltener Strg-Taste – für die umgekehrte Richtung halten Sie zusätzlich die ⇧-Taste gedrückt) – die Tabulatortaste so oft ↹, bis die gewünschte Datei ausgewählt ist oder klicken Sie diese direkt mit der Maus an.

**TIPP** Alternativ klicken Sie in der oberen rechten Ecke des Editors auf die Schaltfläche *Aktive Dateien* neben der Schaltfläche *Schließen*, und wählen Sie die gewünschte Datei aus der Liste aus.

Mit dem IDE-Navigator können Sie auch zwischen den Toolfenstern navigieren, die in der IDE geöffnet sind. Je nachdem, in welcher Reihenfolge Sie navigieren möchten, können Sie die Tastenkombinationen Alt F7 oder Alt ⇧ F7 verwenden.

## Umgebungsschriftart definieren

Für nicht näher bestimmte IDE-Elemente konnte man bislang keine Schriftart definieren, was insbesondere bei der Entwicklung auf Monitoren größer als 24 Zoll Probleme machte. Wählen Sie aus dem Menü *Extras* den Menüpunkt *Optionen*, erweitern Sie den Knoten *Umgebung und wählen Sie Schriftarten und Farben*. In der Klappliste unter *Einstellungen* anzeigen für: wählen Sie die Option *Umgebungsschriftart*. Anschließend bestimmten Sie eine neue Schriftart und den entsprechenden Schriftgrad.

Betroffen von der Umgebungsschriftart sind alle Schriften, die Sie nicht extra festlegen können, also Dropdown-Menüschriften, Dialogschriften, Projektmappen-Explorer, Toolbox, Registerkartenbeschriftungen, wie beispielsweise in Abbildung 2.5 zu sehen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

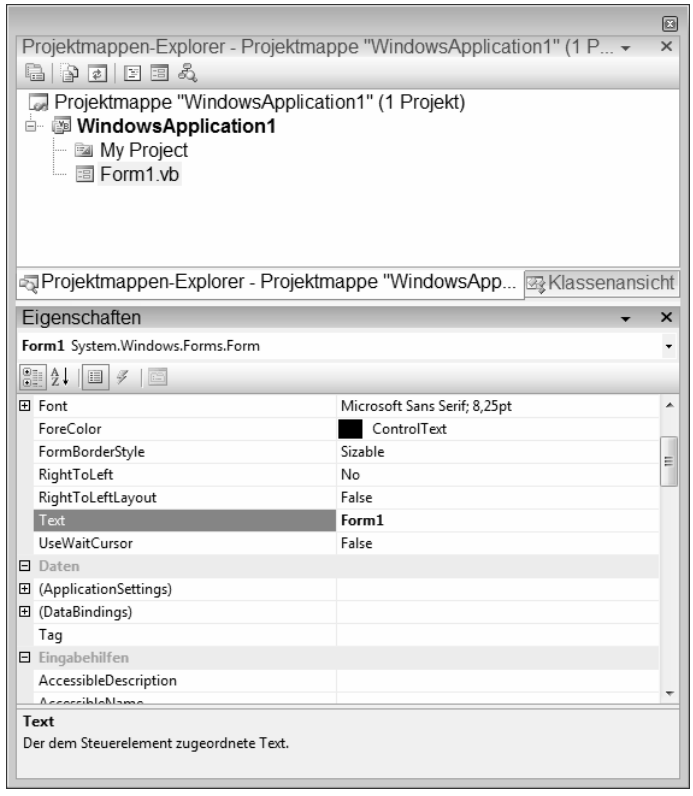



**Abbildung 2.5** Mit der Umgebungsschriftart lässt sich die Basisschriftart der IDE verändern; das hat Auswirkungen auf verschiedene Toolfenster, Dialoge und Dropdown-Menüs – wie hier im Bild am Beispiel des im Projektmappen-Explorers gezeigt

# Übernehmen von Visual Studio 2005-Einstellungen

Wenn Visual Studio 2005 und Visual Studio 2008 auf demselben Computer installiert sind, können Sie beim ersten Starten von Visual Studio 2008 die meisten Einstellungen von Visual Studio 2005 übernehmen. Codeausschnitte (auch bekannt unter dem Namen Code Snippets) können hingegen nicht *automatisch* übernommen werden und müssen für die Verwendung in Visual Studio 2008 manuell neu installiert werden. Wenn Visual Studio 2005 und Visual Studio 2008 nicht auf demselben Computer installiert sind, können Sie Ihre Visual Studio 2005-Einstellungen immer noch manuell für die Verwendung in Visual Studio 2008 migrieren. In diesem Fall exportieren Sie die Einstellungen in eine Datei, indem Sie aus dem Menü *Extras* den Menüpunkt *Einstellungen importieren und exportieren* wählen. Visual Studio zeigt Ihnen wieder den Dialog, den Sie in Abbildung 2.2 sehen können. Wählen Sie die Option *Ausgewählte Umgebungseinstellungen importieren* und im Dialog, der anschließend erscheint, die Einstellungsdatei aus und folgen Sie den weiteren Anweisungen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Visual Studio 2005-Projekte nach Visual Studio 2008 migrieren

Grundsätzlich hat Visual Studio 2008 einen anderen Projektdateien- und Projektmappendateiaufbau als Visual Studio 2005. Das heißt erstmal: Ein Projekt, dass Sie mit Visual Basic 2005 (oder einer noch früheren Version) erstellt haben, können Sie nicht direkt in Visual Studio 2008 öffnen. Visual Basic 2005 und Visual Studio 2008 können aber auf ein und demselben Rechner problemlos koexistieren, und wenn Visual Studio 2005 zusammen mit der 2008er Version auf einem Rechner installiert ist, haben Sie folgende Optionen:

- Sobald Sie eine Projekt- oder Projektmappendatei im Explorer doppelklicken, starten Sie damit automatisch die Instanz der »richtigen« (also der dem Projekt zugehörigen) Version von Visual Studio. Geregelt wird das dadurch, dass die entsprechende Dateiendung nicht mehr einer bestimmten Visual Studio-Version sondern einem so genannten *Visual Studio Version Selector* zugewiesen ist, der sich die aufzurufende Projektdatei anschaut, analysiert und dann entscheidet, welche Version von Visual Studio gestartet werden soll.


**Abbildung 2.6** Nach dem Öffnen eines Visual Basic 2005-Projektes startet sofort der Migrationsassistent, mit dem Sie das Projekt in das neue Projektformat konvertieren können

- Wenn Sie eine Projekt- oder eine Projektmappendatei in Visual Studio 2008 öffnen, die in Visual Studio 2005 erstellt wurde, wird der Migrationsassistent aktiv, der es Ihnen ermöglicht, Ihr vorhandenes Projekt in das neue Format zu migrieren (siehe Abbildung 2.6).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Nach der Konvertierung eines Projektes zielt das in das Visual Basic 2008-Format konvertierte Projekt auf das .NET Framework 2.0 – Sie haben hier also weiterhin »nur« die Möglichkeit, alte .NET 2.0-Funktionalität in Ihrem Projekt weiterzuverwenden, aber dafür ist das Kompilat Ihres Projektes auch in der Lage, ältere Windows 2000-Clients[1] zu bedienen, die mit den neueren .NET Framework-Versionen nicht ausgestattet werden können.

Visual Basic 2008 erlaubt es jedoch, auch gegen unterschiedliche .NET Framework-Versionen zu entwickeln – der nächste Absatz hält dazu genauere Informationen bereit. Möchten Sie Ihr konvertiertes Projekt umstellen, sodass Sie auch die Funktionalitäten der neueren 3.0 bzw. 3.5 .NET Framework-Versionen nutzen können, ändern Sie die Projekteinstellungen, wie im ersten Abschnitt des nächsten Kapitels beschrieben.

# Designer für Windows Presentation Foundation-Projekte

Mit dem Windows Presentation Foundation (WPF)-Designer können Sie WPF-Anwendungen und benutzerdefinierte Steuerelemente in der IDE erstellen. Der WPF-Designer kombiniert die Echtzeitbearbeitung der XAML (Extended Application Markup Language) mit einer verbesserten grafischen Entwurfszeiterfahrung. Ein neues WPF-Projekt erstellen Sie, indem Sie

1. aus dem Menü *Datei* den Befehl *Neu*/*Projekt* auswählen
2. im Dialog, der jetzt erscheint, unter *Projekttypen* den Eintrag *Windows* wählen und unter *Vorlagen* den Eintrag *EPF-Anwendung*.
3. unter *Name* einen neuen Namen für Ihr Projekt bestimmen, ebenso den Speicherort und anschließend auf *OK* klicken, um das neue WPF-Projekt bearbeiten zu können.

Der WPF-Designer unterscheidet sich vom Windows-Forms-Designer grundlegend. Anders als beim Windows-Forms-Designer kann der Aufbau eines WPF-Forms in einer so genannten XAML-Datei gespeichert werden – der Aufbau eines WPF-Forms nur durch reinen Code wie beim Windows-Forms-Designer ist natürlich ebenfalls möglich; die Trennung zwischen Code und Aufbaubeschreibung macht es jedoch möglich, dass ein beauftragter Designer unabhängig vom Entwickler der Formularlogik das Formular designen kann.

Dem Thema WPF ist ein eigenes Kapitel in diesem Buch gewidmet (siehe Kapitel 11). Die hier beschriebenen Features und Vorgehensweisen sollen nur einen kurzen und groben Überblick über die Neuheiten aus IDE-Sicht bieten.

Die folgenden Features sind im WPF-Designer neu.

- Mithilfe der SplitView-Funktionalität können Sie Objekte im grafischen Designer anpassen und die Änderungen des zugrunde liegenden XAML-Codes direkt anzeigen lassen. Entsprechend werden Änderungen am XAML-Code sofort im grafischen Designer umgesetzt. Abbildung 2.7 vermittelt Ihnen einen Eindruck davon.

[1] Oder, mit eingeschränkter Funktionalität, sogar alte Windows 98-Clients – doch das sei wirklich nur der Vollständigkeit halber erwähnt, denn schon aus Sicherheitsaspekten sollten Sie Ihren Kunden gegenüber fairerweise gar nicht mehr erwähnen, dass Ihre Software theoretisch auch unter Windows 98 lauffähig ist.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Da der WPF-Designer lange nicht über den gleichen Funktionsumfang wie der Windows Forms-Designer verfügt, ist dieses erste Feature auch gleichzeitig sein bestes: In vielen Fällen ist es leichter, den für den Formularaufbau erforderlichen XAML-Code zu erstellen, als zu versuchen, beispielsweise Steuerelementekombinationen mit dem Designer interaktiv zusammenzuklicken. Aus diesem Grund werden wir später, ab dem 11. Kapitel, das Augenmerk auch mehr auf XAML an sich als auf den Umgang mit dem Designer legen.

Eine viel bessere Alternative zum eingebauten WPF-Designer ist das Tool *Expression Blend*, mit dem Sie sogar WPF-konform interaktiv Animationen erstellen können – ein Feature, das der eingebaute Designer überhaupt nicht unterstützt. Mehr zum Thema Expression Blend erfahren Sie unter *http://www.microsoft.com/expression/*.

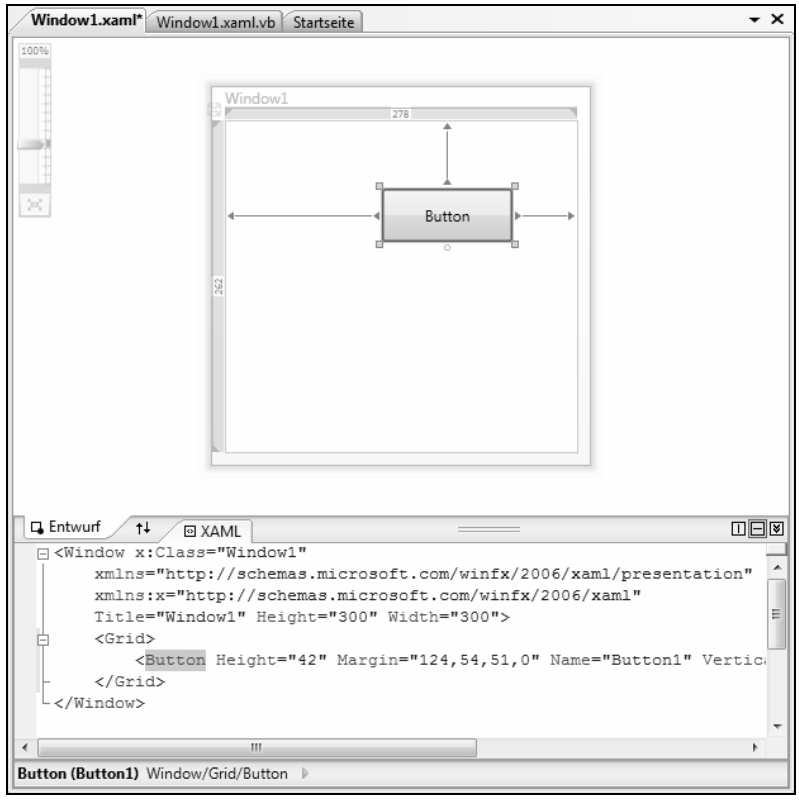



**Abbildung 2.7** Der WPF-Designer bietet Split-View-Funktionalität: Grafische Änderungen werden direkt in XAML-Code umgesetzt, Änderungen im XAML-Code spiegeln sich direkt in der grafischen Repräsentation wieder.

- Im Fenster *Dokumentgliederung* können Sie den XAML-Code bei vollständiger Auswahlsynchronisierung zwischen Designer, Dokumentgliederung, XAML-Editor und Eigenschaftenfenster anzeigen lassen und darin navigieren.
- IntelliSense im XAML-Editor ermöglicht den schnellen Codeeintrag. IntelliSense unterstützt jetzt selbstdefinierte Typen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

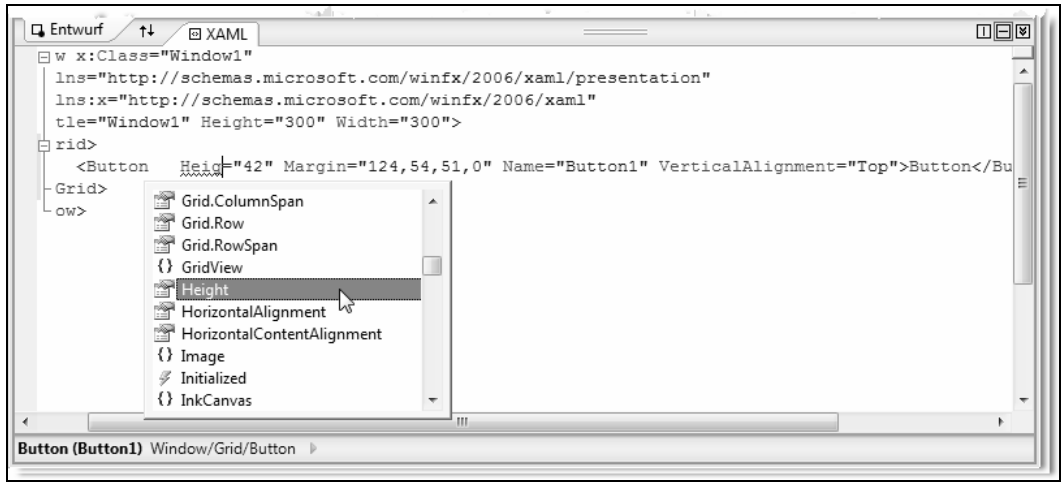



**Abbildung 2.8** Der XAML-Editor verfügt über volle IntelliSense-Unterstützung

- Rasterlinien können den Rastern im Designer hinzugefügt werden, um die einfache rasterbasierte Platzierung von Steuerelementen zu ermöglichen.
- Steuerelemente und Text können leicht an Ausrichtungslinien ausgerichtet werden.
- Der Designer unterstützt jetzt das Laden der von Ihnen definierten Typen. Dazu gehören benutzerdefinierte Steuerelemente und Benutzersteuerelemente.
- Sie können das Laden großer XAML-Dateien abbrechen.
- Die Entwurfszeiterweiterung unterstützt Entwurfsmodus und Eigenschaften-Editoren.

Dem Thema WPF ist ein eigener Buchteil gewidmet, der sich auch ein wenig näher mit dem WPF-Designer auseinandersetzt. Ab Kapitel 11 finden Sie Näheres zu diesem Thema.

# IntelliSense-Verbesserungen

Das Visual Basic-Team hat die Unterstützung des Entwicklers beim Coden durch IntelliSense in Visual Studio 2008 in verschiedenen Punkten verbessert.

## Syntax-Tooltipps

Am auffälligsten ist die Einführung sogenannter Syntax-Tooltipps, wie Sie sie vielleicht schon kennen, sollten Sie nicht nur in Visual Basic sondern auch schon in C# 2005 entwickelt haben. In Visual Basic 2008 zeigt die Entwicklungsumgebung das IntelliSense-Fenster bereits an, wenn Sie die ersten Buchstaben gedrückt haben (siehe folgende Abbildungen).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 2.9** IntelliSense wird bereits aktiv, wenn Sie den ersten Buchstaben in das Code-Fenster eingeben und unterstützt Sie nicht nur mit der Auflistung bestimmter Schlüsselworte ...

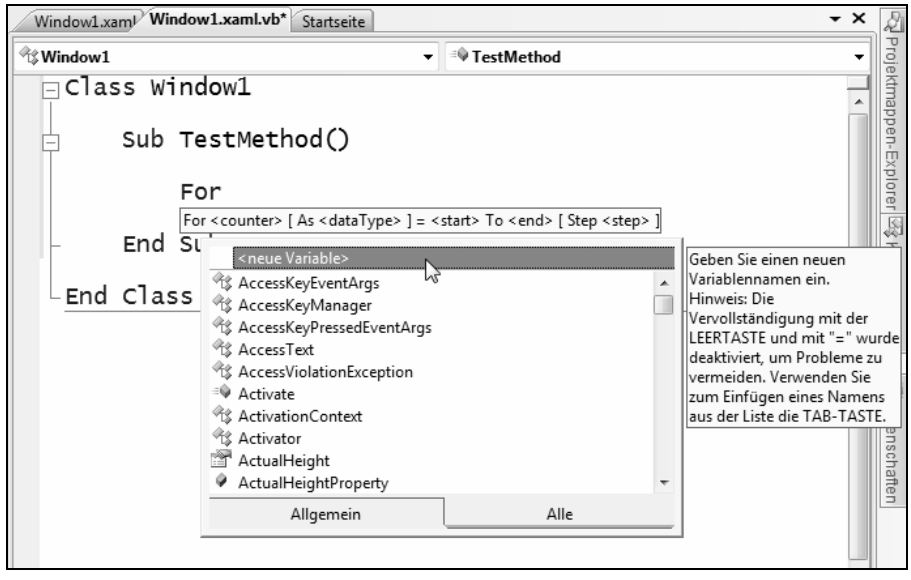



**Abbildung 2.10** ... sondern zeigt in ausführlichen Tooltipps auch die genaue Syntax und Funktionsweise von Strukturbefehlen an

**TIPP** In vorherigen Versionen störten die IntelliSense-Listen und Tooltipps ab und an, da man nicht durch sie hindurch sehen konnte. Das wird mit Visual Studio 2008 anders: Drücken Sie einfach Strg, während die IntelliSense-Elemente angezeigt werden, und sie geben die Sicht auf den darunterliegenden Code frei!

## Filtern beim Tippen

In vorherigen Versionen wurde man oftmals durch die langen IntelliSense-Listen erschlagen. In der 2008er Version werden die Inhalte der IntelliSense-Listen beim Tippen gefiltert, wie in den folgenden beiden Abbildungen zu sehen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

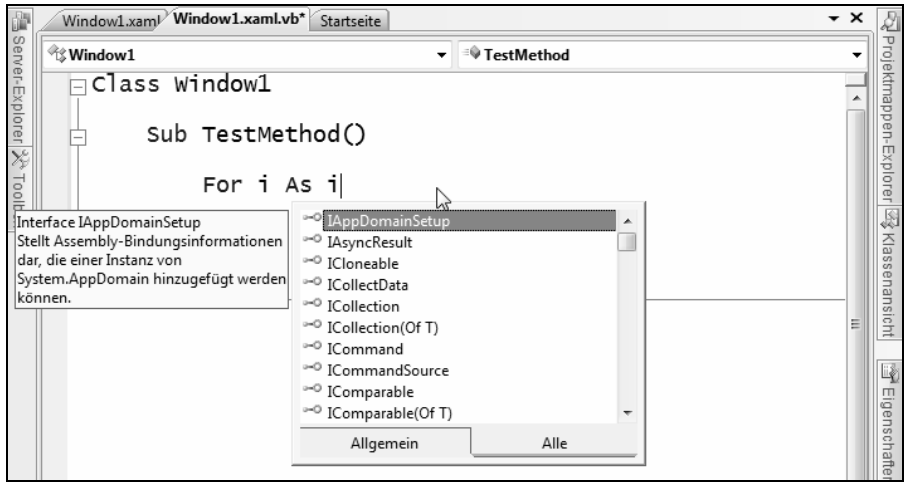



**Abbildung 2.11** Die IntelliSense-Auswahlliste wird durch die bislang eingegebenen Zeichen gefiltert. Mit jedem weiteren getippten Buchstaben ...

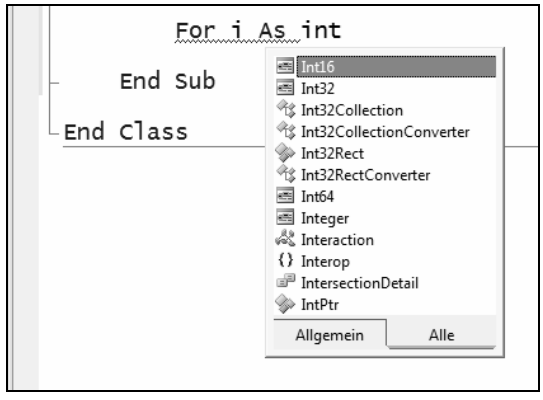



**Abbildung 2.12** ...verkleinert sich die Liste auf die Elemente, die dem bislang Getippten entsprechen.

# Inhalte von System- oder externen Typen in Vorschaufenstern fürs Debuggen konfigurieren

Beim Debuggen von Anwendungen war die vollständige Konfiguration der Wertentsprechungen von Systemtypen oder von Typen, zu denen Sie den Quellcode nicht besitzen, in Variablen-Vorschaufenstern (anders als beispielsweise in C#) nicht implementiert. In Visual Basic 2008 wurde dieses Manko ausgemerzt und die Unterstützung für `DebuggerDisplayAttribute` komplettiert.

Worum geht's genau? Ganz vereinfacht ausgedrückt um die Darstellung des eigentlichen Wertes eines Typs beim Debuggen in einem Vorschaufenster, etwa wie in der folgenden Abbildung zu sehen:

**BEGLEITDATEIEN** Unter *.\Samples\Chapter02\DebugExtTypes* finden Sie das Beispielprojekt dieses Abschnittes.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

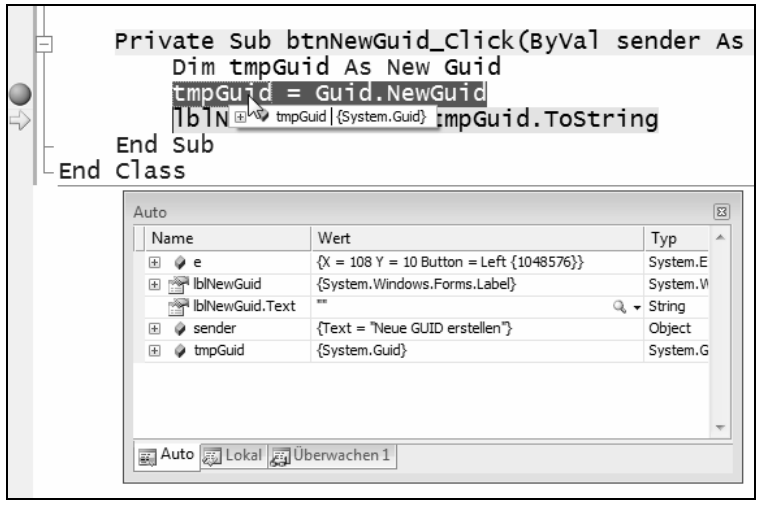



**Abbildung 2.13** Bei bestimmten Typen ist keine Entsprechung bei der Ausgabe seines Inhalts als Text definiert – stattdessen wird, wie hier bei Guid, der nicht sehr informative Typname ausgegeben

- Um dem Abhilfe zu leisten, definieren Sie eine neue Assembly in Form einer Klassenbibliothek. Wählen Sie dazu aus dem Kontextmenü des Projektmappen-Explorers über die Projektmappe (nicht das Projekt!) den Menüpunkt *Hinzufügen/Neues Projekt*.
- Wählen Sie unter Projekttypen *Visual Basic/Windows* und unter Vorlagen *Klassenbibliothek* aus, bestimmen Sie Projektnamen (beispielsweise *CustomDebugDisplays*) und Speicherort, und klicken Sie auf *OK*. Falls nur das Projekt, nicht aber die Projektmappe im Projektmappen-Explorer zu sehen ist, wählen Sie aus dem Menü *Datei* den Befehl *Neu/Projekt* aus, um an den entsprechenden Dialog zu gelangen.
- Klicken Sie im Projektmappen-Explorer das neu hinzugefügte Projekt an, und wählen Sie das Projektmappen-Explorer-Symbol mit der Tooltipp-Beschreibung *Alle Dateien anzeigen* aus. Öffnen Sie anschließend den Zweig *My Projekt*.
- Doppelklicken Sie auf *AssemblyInfo.vb*, um die Assembly-Infos im Editor anzeigen zu lassen.
- Fügen Sie über das `Assembly`-Attribut die entsprechenden `DebuggerDisplay`-Attribute hinzu, etwa:

```
<Assembly: DebuggerDisplay("{ToString}", Target:=GetType(Guid))>
```

- Mit dieser Anweisung bestimmen Sie die `ToString`-Methode (erstes Argument) zur Anzeige des `Guid`-Typs (zweites Argument).
- Erstellen Sie die Projektmappe neu, und kopieren Sie die neue Assembly in das Verzeichnis *Visual Studio 2008/Visualizers*, dass Sie im *Eigene Dokumente*-Verzeichnis Ihres Windows-Laufwerkes finden. Sie finden die Assembly im Unterverzeichnis *.\bin\debug* des Projektverzeichnisses.

Beim erneuten Debuggen finden Sie die korrekte Darstellung des `Guid`-Typs in allen Vorschaufenstern:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

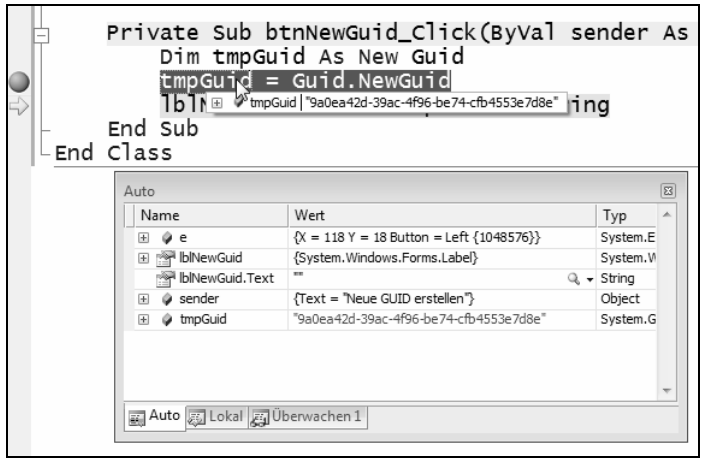



**Abbildung 2.14** Nachdem die neue Assembly in *Visual Basic 2008/Visualizers* unterhalb des *Eigene Dokumente*-Ordners kopiert wurde, wird der Inhalt einer Guid-Instanz in den Vorschaufenstern korrekt dargestellt.

# Zugriff auf den .NET Framework-Quellcode beim Debuggen

Dieser Abschnitt beschäftigt sich mit einem Thema, dessen Ankündigung in der Presse schon für eine kleine Sensation sorgte. Die simple und dazu umgekehrt proportional Aufmerksamkeit erregende Meldung lautete: »Microsoft legt .NET-Quelltexte offen[2]«. Das gelesen, stellten sich viele enthusiastische C#-Entwickler dann vor, »Oh wie cool – dann lade ich die Quelltexte herunter und bastele mir mein eigenes .NET Framework[3]«. Doch ganz so einfach hat es uns Microsoft nicht gemacht, an die begehrten Quellen des Frameworks zu kommen.

Zunächst einmal benötigen Sie mindestens Visual Studio 2008 in der Standardversion. Ältere Visual Studio-Versionen und auch sämtliche Express-Versionen bleiben außen vor. Und dann können Sie die Quelltexte der jeweiligen Komponenten, die Sie interessieren, nur dann bekommen, wenn Sie innerhalb von Visual Studio in diese hinein debuggen – und zu nichts anderem sind die Quellcodes bzw. die Freigabe derselben auch gedacht: Nämlich um Sie beim Debuggen innerhalb Ihrer eigenen Projekte zu unterstützen.

Und für uns Visual Basic-Entwickler gibt es eventuell noch eine kleine Einschränkung, und ich möchte mich hier bewusst nicht weiter auf das Vorwort dieses Buches beziehen: Der verwaltete Teil des .NET Frameworks ist natürlich zu ganz erheblichen Teilen in C# geschrieben worden. Zu welchen genau entzieht sich meiner Kenntnis, aber müsste ich schätzen, dann würde ich sagen: 95% sind in C#, der Rest (beispielsweise die *Microsoft.VisualBasic.dll*-Assembly) in Visual Basic. Das wiederum bedeutet: Sie sehen den Quelltext natürlich auch nur in der Sprache, in der er entwickelt wurde, und das ist eben in den meisten Fällen C#.

---

[2] So der Heise-Ticker unter *www.heise.de* am 4.10.2007 um 11:15. Unter *http://www.heise.de/newsticker/meldung/96909* können Sie diese Meldung direkt abrufen (Stand: 18.1.2008).

[3] Kleine Anekdote am Rande: Hier bei ActiveDevelop sorgte die Meldung auch für lustiges Tohuwabohu, nachdem die *schon* ein wenig sarkastische Diskussion über den Nutzen des Frameworks »entglitt« und unser Chefentwickler sich der Realisierung seiner Weltübernahmepläne mithilfe einer eigenen Framework-Version einen Schritt näher wähnte… ;-)

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Doch auch bevor das geschehen kann, müssen Sie – Stand 18.01.2008 – noch ein paar Vorbereitungen treffen, um in den Genuss des NET Framework-Debuggings zu gelangen. Wie es genau geht, zeigt die folgende Schritt-für-Schritt-Anleitung.

- Zunächst benötigen Sie ein Projekt, das Sie debuggen möchten. In guter alter Fernsehköchemanier haben wir da mal was vorbereitet – und was könnte es anderes sein, als die 132-gazillionste Version von »Hello World«!

**BEGLEITDATEIEN** Das Projekt, mit dem wir das Debuggen des .NET Framework-Quellcodes demonstrieren wollen, befindet sich in *.\Samples\Chapter02 - NeuInIde\FrameworkDebugging*

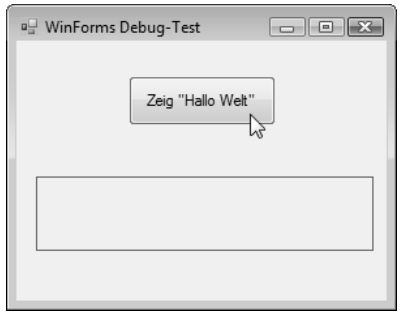



**Abbildung 2.15** Diese spektakuläre Windows-Anwendung wird das Debuggen von .NET Framework-Code demonstrieren: Entdecken Sie, was wirklich passiert, wenn Sie die Schaltfläche drücken!

- Und bevor wir uns daran machen können, herauszufinden, was wirklich passiert, wenn wir einem Label-Steuerelement eine neue Zeichenkette zuweisen …

```
'Mehr iss nich!
Public Class Form1

    Private Sub btnShowHelloWorld_Click(ByVal sender As System.Object, _
                                        ByVal e As System.EventArgs) _
                                        Handles btnShowHelloWorld.Click

        'Text der Schaltfläche setzen
        lblHelloWorld.Text = "Hallo Welt!"

    End Sub
End Class
```

- … müssen wir noch einige Einstellungen an der Benutzeroberfläche von Visual Studio vornehmen.[4] Dazu wählen Sie in der IDE von Visual Studio aus dem Menü *Extras* den Menüpunkt *Optionen* aus.
- Im Dialog, der anschließend erscheint, wählen Sie in der linken Liste den Eintrag *Debugging*.

[4] Shawn Burke blogt den Tipp, zuvor einen Patch zu installieren. Zu finden war dieser Patch – Windows-Live ID vorausgesetzt – unter *https://connect.microsoft.com/VisualStudio/Downloads/DownloadDetails.aspx?DownloadID=10443&wa=wsignin1.0*. Shawn Burke ist Entwickler bei Microsoft; sein Blog gibt's unter *http://blogs.msdn.com/sburke/default.aspx* (Stand: 18.1.2008)

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

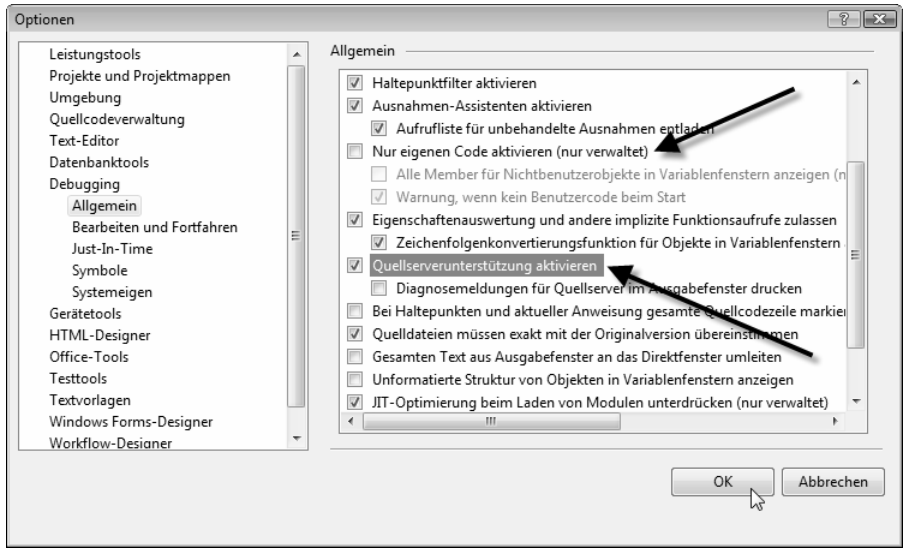



**Abbildung 2.16** Deselektieren Sie die Option *Nur eigenen Code aktivieren (nur verwaltet)* und aktivieren Sie die Option *Quellserverunterstützung aktivieren*

- Auf der rechten Seite des Dialogs werden nun die allgemeinen Debugging-Einstellungen angezeigt. Hier deselektieren Sie die Option *Nur eigenen Code aktivieren (nur verwaltet)* und aktivieren die Option *Quellserverunterstützung aktivieren*.
- Wechseln Sie in der linken Spalte desselben Dialogs auf den *Debugging*-Untereintrag *Symbole*. Dort klicken Sie auf die *Neu*-Schaltfläche (das Symbol, auf das die Maus in Abbildung 2.17 zeigt), und geben Sie den Microsoft-Symboldateienserver an: *http://referencesource.microsoft.com/symbols*.[5]
- Unter dem Punkt *Symbole vom Symbolserver in diesem Verzeichnis zwischenspeichern* geben Sie anschließend einen Speicherort auf Ihrer Festplatte an, an dem die Quellcodedateien zwischengespeichert werden. Das Zwischenspeichern hat den Vorteil, dass die Dateien nur beim ersten Mal vom Microsoft-Server aus dem Internet geladen werden müssen – schon beim zweiten Quellcode-Debuggen fällt dieser Schritt weg, und die Quellcode-Dateien werden viel schneller aus diesem Cache-Speicher entnommen.
- Wenn Sie den Dialog nun mit *OK* bestätigen, sehen Sie einen EULA, den Sie ebenfalls nach natürlich aufmerksamem Studium mit *OK* bestätigen müssen – und dann dauert es ein kleines Weilchen, bis das Nächste passiert, da Visual Studio sofort anfängt, benötigte Symboldateien für das Beispielprojekt herunterzuladen. In der Statuszeile von Visual Studio können Sie beobachten, was passiert.

[5] Stand: 18.01.2008 – dieser Eintrag könnte sich, wenn auch mit nicht großer Wahrscheinlichkeit, ändern – das sollte dann aber ausreichend früh von Microsoft kommuniziert werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

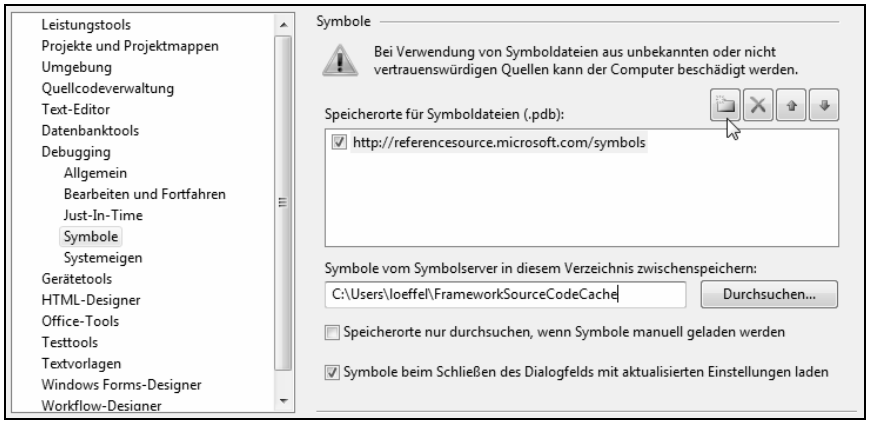



**Abbildung 2.17** Im Symbole-Dialog geben Sie die Quelle der Symboldateien an, die auch den eigentlichen Quellcode des .NET Frameworks enthalten sowie ein Verzeichnis, in dem die Quellcode-Dateien zwischengespeichert werden können

- Den anschließenden Sicherheitshinweis bestätigen Sie ebenfalls mit *Ja* (natürlich auch nicht, bevor Sie ihn aufmerksam gelesen haben!).
- Und damit haben Sie quasi die Voraussetzungen für das Debuggen gelegt. Um das Debuggen nun zu starten, setzten Sie mit F9 einen Haltepunkt – am besten an der Stelle, an der im Programmcode die Textzuweisung von »Hallo Welt!« an das Label erfolgt:

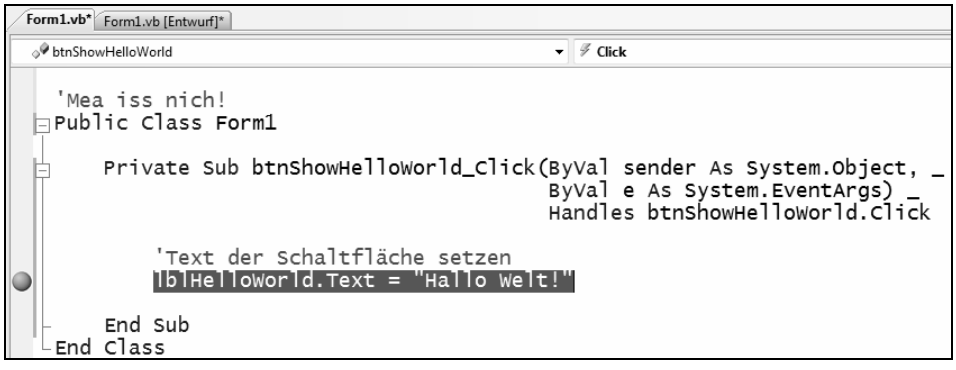



**Abbildung 2.18** Setzen Sie in der Zeile den Haltepunkt, die die Ausgangsbasis für den Sprung ins .NET Framework sein soll

- Starten Sie anschließend Debuggen mit F5.
- Wenn das Programm gestartet ist, klicken Sie auf die Schaltfläche *Zeig "Hallo Welt"*.
- Das Programm trifft auf die entsprechende Haltepunktzeile, und die Programmausführung wird an dieser Stelle unterbrochen.
- Wählen Sie jetzt aus dem Menü *Debuggen*, den Menüpunkt *Fenster* und weiter *Aufrufliste*. Alternativ drücken Sie Strg Alt C.
- Sollten die Einträge für die Assembly *System.Windows.Forms*, anders als in der folgenden Abbildung zu sehen, ausgegraut sein, öffnen Sie das Kontext-Menü mit der rechten Maustaste und wählen den Menüpunkt *Symbole laden*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

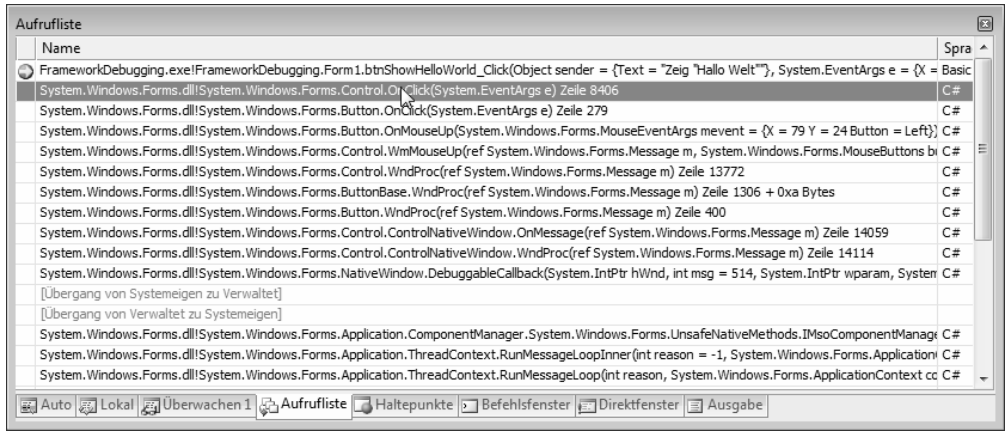



**Abbildung 2.19** Im Unterschied zu »sonst«, sollten die .NET Framework-Assembly-Methoden und Eigenschaften in der Aufrufliste nicht ausgegraut sein, als Zeichen dafür, dass Sie in sie hineindebuggen können. Falls doch: Symbole über das Kontextmenü laden!

- Wenn Sie anschließend mit F11 in die Routine hineinsteppen, landen Sie tatsächlich im .NET Framework-Quell-Code, wie Abbildung 2.20 zeigt.


**Abbildung 2.20** Voila! – Der Source-Code des .NET Frameworks, natürlich nur ein ganz, ganz kleiner Ausschnitt (hier: Label.cs – der Sourcecode des Label-Controls)

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 3

# Das ist neu im Compiler von Visual Basic 2008

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# First and definitely not least – die Performance des Visual Basic 2008-Compilers

Es gibt eine ganze Reihe an Kleinigkeiten, die das Visual Basic 2008-Team in das neue Visual Basic 2008 integriert hat. Es gibt auch Dinge, die man nicht sieht, die aber dennoch von Wichtigkeit sind – und das ist die Performance des Compilers, mal ganz davon zu schweigen, dass er reibungslos funktionieren und nicht hier und da »mal« abstürzen sollte.

Letzteres war nämlich vor dem Service Pack 1 beim Visual Basic 2005-Compiler gerade auf Mehrprozessor- oder Multi-Core-Prozessor-Systemen bei größeren Projekten sehr häufig der Fall. Mitunter mussten Projekte sogar liegen bleiben, bis die ersten Patches für diese »Race-Condition«, die sich zwischen Editor und Background-Compiler wohl entwickelte, vom Visual Basic-Team »geklärt« war.

Doch das ist glücklicherweise Schnee von gestern. Heute geht es nicht darum, dass der Compiler richtig funktioniert (denn das tut er!), sondern wie schnell. Hier ist in Sachen Produktivität einiges an Steigerungspotential herauszuholen, und die Jungs und Mädels aus Redmond haben sich das richtig zu Herzen genommen; im Falle von Visual Basic 2008 übrigens eher sogar die Mädels.

Im Folgenden finden Sie eine Tabelle[1], die die Performance-Unterschiede zum Visual Basic 2005-Compiler dokumentiert – diese Tabelle stammt übrigens ursprünglich aus einem Blog von Lisa Feigenbaum, einer der Entwicklerinnen im Visual Basic 2008-Team.

Verstehen Sie die Tabelle bitte nicht falsch: Es geht dabei nicht um die Geschwindigkeit, mit der die späteren Programme laufen, sondern wie schnell Background-Compiler und Main-Compiler arbeiten und damit wie groß (oder: klein) die Turn-Around-Zeiten beim Kompilierungsprozess sind.

| Szenario | VB2005 (ms) | VB2008 (ms) | VB2008 ist x-mal schneller als VB2005 | VB2008 braucht x% der Zeit von VB2005 |
|---|---|---|---|---|
| Build eines umfangreichen Projektes (bei Verwendung von Hintergrund-Kompilierung mit dem Background-Compiler). | 222206.25 | 1352.88 | 164.25 | 0.61% |
| Build einer großen Projektmappe mit mehreren Projekten (explizite Build-Erstellung). | 1618604.75 | 57542.75 | 28.13 | 3.56% |
| Build einer großen Projektmappe mit mehreren Projekten (bei Verwendung von Hintergrund-Kompilierung mit dem Background-Compiler). | 222925.50 | 19861.88 | 11.22 | 8.91% |
| Reaktionszeit nach dem Hinzufügen eines Members zu einer Klasse. | 327.00 | 36.50 | 8.96 | 11.16% |

[1] Quelle Lisa Feigenbaum, *http://blogs.msdn.com/vbteam/archive/2008/01/04/vb2008-outperforms-vb2005-lisa-feigenbaum.aspx*, Stand 18.1.2008.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Szenario | VB2005 (ms) | VB2008 (ms) | VB2008 ist x-mal schneller als VB2005 | VB2008 braucht x% der Zeit von VB2005 |
|---|---|---|---|---|
| Reaktionszeit nach dem Öffnen eines Projekts. | 255551.25 | 38769.38 | 6.59 | 15.17% |
| IntelliSense aufrufen, um die Typenliste anzeigen zu lassen (erster Aufruf). | 1192.50 | 530.5 | 2.25 | 44.49% |
| Edit-and-Continue in einer Projektmappe, die XML-Kommentare enthält (erstes Mal). | 441.25 | 210.5 | 2.10 | 47.71% |
| Reaktionszeit nach dem Ändern eines Methoden-Statements. | 390.25 | 236.38 | 1.65 | 60.57% |
| 10 Schritte im Debugger (kummulierte Zeiten). | 1850.75 | 1167.13 | 1.59 | 63.06% |
| IntelliSense aufrufen, um eine Typenliste zu bearbeiten (Folgezeiten). | 79.25 | 51.5 | 1.54 | 64.98% |
| Ausführen nach F5 wenn die Projektmappe bereits vollständig erstellt wurde. | 385.20 | 278.7 | 1.38 | 72.35% |
| Zeit, nachdem ein Fehler der Fehlerliste hinzugefügt wird. | 531.25 | 394.5 | 1.35 | 74.26% |
| 10 Schritte im Debugger (zum ersten Mal). | 1336.50 | 1150 | 1.16 | 86.05% |
| Reaktionsverhalten, während der Hintergrund-compiler eine offene Projektmappe verarbeitet. | 4803.00 | 4284.75 | 1.12 | 89.21% |
| Laden einer umfangreichen Projektmappe. | 13667.5 | 12407.25 | 1.10 | 90.78% |
| Laden einer umfangreichen Projektmappe (erstmalig).<br>**HINWEIS:** Entspricht der Verbesserung auf Windows XP. Unter Vista ist der Geschwindig-keitsvorteil noch mal doppelt so schnell. | 19946.25 | 18222 | 1.09 | 91.36% |

# NET Framework Version Targeting (.NET Framework-Versionszielwahl) bei Projekten

Visual Studio 2008 bzw. Visual Basic 2008 erlaubt, dass Sie das .NET Framework für ihre Anwendung, die Sie entwickeln, bestimmen können. Was bedeutet das genau?

Wenn Sie mit Visual Studio 2002 gearbeitet haben, haben Sie, weil es nichts anderes gab, mit dem .NET Framework 1.0 entwickelt. Mit Visual Studio 2003 kam dann das .NET Framework 1.1 auf den Markt. Mit dieser Visual Studio-Version haben Sie zwangsläufig Anwendungen für dieses .NET Framework entwickelt. Genauso verhielt es sich mit Visual Studio 2005 und dem .NET Framework 2.0.

Schon zu Visual Studio 2005-Zeiten wäre es wünschenswert gewesen, zwar die damals neue Entwicklungs-umgebung zu verwenden, aber dennoch, aus Abwärtskompatibilitätsgründen, auch gegen ältere .NET Framework-Versionen entwickeln zu können.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Mit Visual Studio 2008 ist das möglich. Sie können zwar nicht mehr für das 1.0er oder das 1.1er-.NET Framework entwickeln, aber für alle neueren Versionen – 2.0, 3.0 und 3.5 – ist die Auswahl beim Erstellen eines Projektes möglich. Bedenken Sie dabei Folgendes:

- Wenn Sie gegen das .NET Framework 2.0 entwickeln, können Sie auch Computer mit älteren Betriebssystemen bedienen, wie beispielsweise Windows 2000 (nur mit SP4!), und eingeschränkt sogar noch Windows 98. Sie müssen dann allerdings auf LINQ-Unterstützung (dafür ist 3.5 erforderlich) und auch auf die Windows Presentation Foundation (WPF), die Windows Communication Foundation (WCF) und WF (Windows Workflow) verzichten (dafür ist 3.0 erforderlich).
- Wenn Sie gegen das .NET Framework 3.0 entwickeln, können Sie auf die .NET Framework-Bibliothek für die WPF, WCF und WF zurückgreifen; auf älteren Windows-Versionen läuft Ihre Anwendung dann allerdings nicht mehr: Sie benötigen mindestens Windows Server 2003 bzw. Windows XP mit Service Pack 2. Für Windows Vista ist keine .NET Framework-Installation erforderlich, da das .NET Framework bereits Bestandteil dieses Betriebssystems ist.
- Entwickeln Sie gegen das .NET Framework 3.5, sind Sie ebenfalls mit Windows XP SP2 und Windows Server 2003 dabei – dieses .NET Framework muss allerdings gezielt auf dem System, auf dem Sie Ihre Anwendung laufen lassen wollen, mit installiert werden. Mit dem .NET Framework 3.5 steht Ihnen zusätzlich die komplette LINQ-Funktionalität auf dem Zielsystem zur Verfügung.

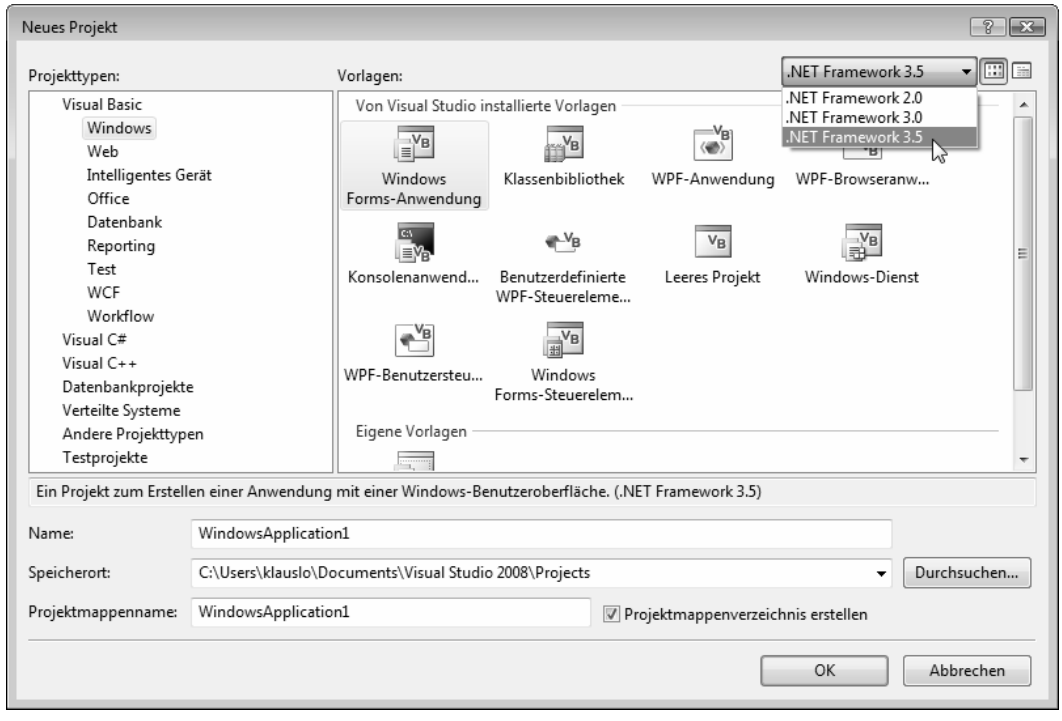



**Abbildung 3.1** Beim Erstellen eines neuen Projekts wählen Sie aus der Aufklappliste am rechten, oberen Dialograndgegen welche .NET Framework-Version Ihr neues Projekt abzielen soll

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Beim Erstellen eines neuen Projektes, wählen Sie aus dem Menü *Datei* den Menüpunkt *Neu* und weiter *Projekt*. Visual Studio zeigt Ihnen anschließend einen Dialog (siehe Abbildung 3.1), mit dem Sie nicht nur wie gewohnt ein neues Projekt erstellen können, sondern ebenfalls aus der rechts oben zu findenden Aufklappliste die .NET Framework-Version auswählen können, gegen die ihr Projekt abzielen soll.

Sollten Sie hingegen bereits ein Projekt angelegt haben, und sich anschließend noch für eine andere .NET Framework-Version entscheiden, dann verfahren Sie wie folgt:

1. Klicken Sie das entsprechende Projekt mit der *rechten* Maustaste im Projektmappen-Explorer an.
2. Im Kontextmenü, das jetzt aufklappt, wählen Sie *Eigenschaften*.
3. Wählen Sie die Registerkarte *Kompilieren* im Eigenschaftendialog.
4. Klicken Sie auf *Erweiterte Kompilierungsoptionen*. Bei einer kleineren Bildschirmauflösung müssen Sie ggf. das Fenster nach unten scrollen, um die Schaltfläche sehen zu können.
5. Aus der Aufklappliste *Zielframework (alle Konfigurationen)* wählen Sie die .NET Framework-Version, gegen die Sie entwickeln wollen (siehe auch Abbildung 3.2).
6. Bestätigen Sie das anschließend erscheinende Meldungsfeld mit *OK*. Danach schließt Visual Studio die geöffneten Dateien des Projekts, um sie kaum merklich sofort wieder zu öffnen, und Sie entwickeln ab jetzt gegen die ausgewählte .NET Framework-Version.

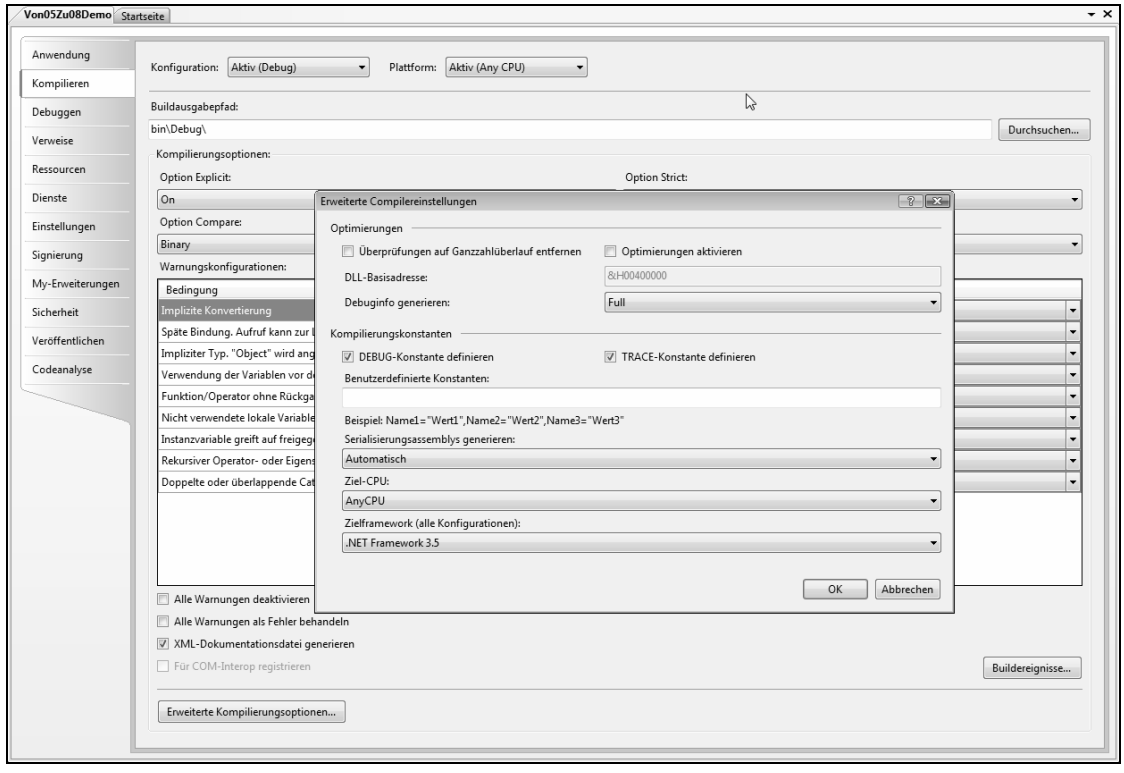



**Abbildung 3.2** Falls Sie die Version des Zielframeworks Ihres Projektes ändern möchten, wählen Sie in den Projekteigenschaften das Register *Kompilieren*, klicken *Erweiterte Kompilierungsoptionen* und ändern die .NET Framework-Version in der Klappliste *Zielframework*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Lokaler Typrückschluss

Visual Basic 2008 erlaubt, dass Typen auch aufgrund ihrer ersten Zuweisung festgelegt werden. Ganz eindeutig wird das beispielsweise an der Zuweisung

```
Dim blnValue = True
```

Wenn Sie einer primitiven Variable den Wert `True` zuweisen und dazu noch Typsicherheit definiert ist, dann muss es sich bei der Variablen einfach um den boolschen Datentyp handeln. Genau so ist das bei

```
Dim strText = "Eine Zeichenkette."
```

`strText` *muss* eine Zeichenkette sein – das bestimmt die Zuweisung. Anders ist es bei numerischen Variablen. Hier muss man wissen, dass durch Zuweisung einer ganzen Zahl an eine bislang noch nicht typbestimmte Variable, der `Integer`-Typ definiert wird, durch Zuweisung einer Fließkommazahl der `Double`-Typ. Doch diese Standardtypen von Konstanten gab es vorher schon – letzten Endes bestimmen die Konstanten mit ihren Typliteralen, welchen Typ sie darstellen.

```
Dim einInteger = 100 ' Integer, ganze Zahl definiert Integerkonstante
Dim einShort = 101S  ' Short, weil das Typliteral S eine Short-Konstante bestimmt
Dim einSingle = 101.5F  ' Single, weil das Typliteral F eine Single-Konstante bestimmt
```

**WICHTIG** Lokaler Typrückschluss funktioniert übrigens nur auf Prozedurebene, nicht auf Klassenebene (deswegen auch die Bezeichnung »lokaler« Typrückschluss).

Gesteuert wird der lokale Rückschluss übrigens durch `Option Infer`, die als Parameter `Off` oder `On` übernimmt (von engl. *Inference*, etwa: *der Rückschluss*). Standardmäßig ist der lokale Typrückschluss eingeschaltet.

Sie können sie also durch die entsprechende Anweisung

```
Option Infer Off
```

direkt am Anfang der Codedatei eben nur für die Klassen und Module dieser Codedatei oder aber global für das ganze Projekt ausschalten (oder eben anschalten).

## Typrückschluss für Typparameter bei Generics

Im Rahmen von Generics kann Typrückschluss dazu verwendet werden, den konkreten Typ einer generischen Klasse durch den Typ eines übergebenen Parameters zu bestimmen. Der eigentliche Typ der generischen Klasse muss dann nicht gesondert angegeben werden, weil er sich eben aus den übergebenen Argumenten ergibt. Mehr über Typrückschluss für Typparameter bei Generics erfahren Sie in Kapitel 7.

## Generelles Einstellen von Option Infer, Strict, Explicit und Compare

Um Einstellungen von `Option Infer`, aber auch für `Option Strict` (Typsicherheit), `Option Explicit` (Deklarationszwang von Variablen) und `Option Compare` (Vergleichsverhalten) für ein ganzes Projekt global durchzuführen, öffnen Sie das Kontextmenü des Projektes (nicht der Projektmappe!) im Projektmappen-Explorer und wählen den Menüpunkt *Eigenschaften* aus. Auf der Registerkarte *Kompilieren* finden Sie Aufklapplisten für jede der genannten Optionen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

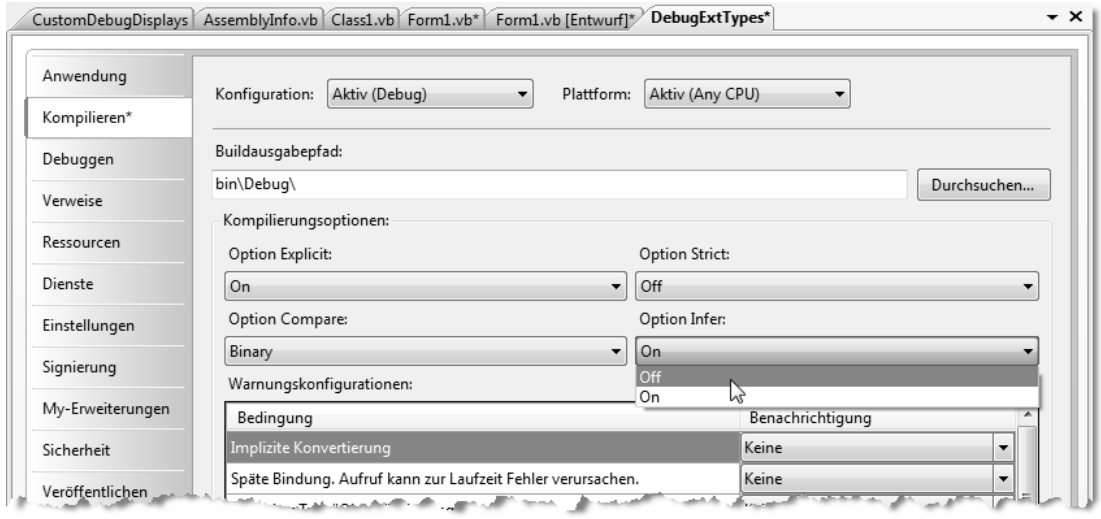



**Abbildung 3.3** Im Eigenschaftendialog des Projektes stellen Sie das `Option XXX`-Verhalten auf der Registerkarte *Kompilieren* projektglobal ein

# If-Operator vs. IIf-Funktion

Einfacher ist auch der Umgang mit `IIf` geworden. Bislang mussten Sie, wenn Sie die `IIf`-Funktion (mit zwei »i«) verwendet haben, das Ergebnis in den Typ casten, der der Zuweisung entsprach, da die `IIf`-Funktion nur `Object` zurücklieferte, also beispielsweise:

```
Dim c As Integer
'Liefert 10 zurück
c = CInt(IIf(True, 10, 20))
```

Das geht jetzt einfacher, denn das Schlüsselwort `If` (mit einem »i«) ist für den Gebrauch als Operator erweitert worden:

```
Dim c As Integer
'Liefert 20 zurück
c = If(False, 10, 20)
```

Noch weniger Schreibaufwand verursacht das, wenn Sie den `If`-Operator mit lokalem Typrückschluss kombinieren:

```
'Liefert 20 zurück
Dim c = If(False, 10, 20)
```

Das Mischen von verschiedenen Typen bei der Typrückgabe bringt den Compiler allerdings ins Straucheln, wie in der folgenden Abbildung zu sehen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

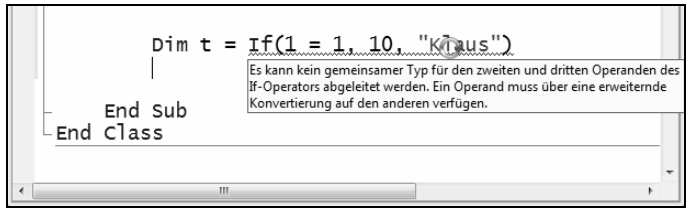



**Abbildung 3.4** If als Ersatz für IIf funktioniert nur dann, wenn Sie dem Compiler die Chance geben, die entsprechenden Typen eindeutig zu ermitteln

# Festlegen der Projekteinstellungen für die Benutzerkontensteuerung (Windows Vista)

Wenn Sie es wünschen, können Sie für ausführbare Programme die Regeln für die Benutzerkontensteuerung für Betriebssysteme ab Windows Vista oder Windows Server 2008 festlegen. Wenn nichts anderes gesagt wird, starten Ihre Anwendungen im Kontext des angemeldeten Benutzers (sofern dessen Rechte es überhaupt erlauben, die Anwendung zu starten). Sie können allerdings bestimmen, dass höhere Rechte als die vorhandenen des Benutzers erforderlich sein sollen.

- Dazu klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf das Projekt (nicht auf die Projektmappe!), und wählen aus dem Kontextmenü *Eigenschaften*.
- Klicken Sie im Register Anwendung auf *Einstellungen für die Benutzerkontensteuerung anzeigen*.
- Visual Studio zeigt Ihnen jetzt die Manifest-Datei Ihrer Anwendung an.
- Ändern Sie den Eintrag *level* für *requestedExecutionLevel* auf die gewünschte Einstellung. Möglich sind dabei *asInvoker* (als Aufrufer – dies ist die Standardeinstellung), *requireAdministrator* (Administrator erforderlich) sowie *highestAvailable* (höchst möglich).


**Abbildung 3.5** In der Manifest-Datei Ihrer Anwendung bestimmen Sie die erforderlichen Einstellungen für die Benutzerkontensteuerung unter Windows Vista bzw. Windows Server 2008 (oder höher).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Nullable-Typen

Nullable-Typen gab es zwar schon in Visual Basic 2005, aber sie waren nur – sagen wir einmal – halbherzig in Visual Basic implementiert. Zwar erfuhren sie schon eine (notwendige!) Sonderbehandlung durch die CLR (siehe Abschnitt »Besonderheiten bei Nullable beim Boxen« ab Seite 41), aber anders als in C# waren sie noch nicht mit einem eigenen Typliteral implementiert. Das hat sich geändert.

`Nullable` ist ein generischer Datentyp mit einer Einschränkung auf Wertetypen. Er ermöglicht, dass ein beliebiger Wertetyp neben seiner eigentlichen Werteart einen weiteren Zustand »speichern« kann – nämlich `Nothing`.

**HINWEIS** Generics gibt es zwar schon seit der Version 2005, jedoch sind sie vielen Entwicklern noch nicht geläufig. Diesem Thema ist deswegen ein eigenes Kapitel, nämlich das Kapitel 6 gewidmet.

Ist das wichtig? Oh ja! Beispielsweise in der Datenbankprogrammierung. Wenn Sie bereits Erfahrungen in der Datenbankprogrammierung haben, wissen Sie auch sicherlich, dass Ihre Datenbanktabellen über Datenfelder verfügen können, die den »Wert« `Null` »speichern« können – als Zeichen dafür, dass eben *nichts* (auch nicht die Zahl 0) in diesem Feld gespeichert wurde.

Ein anderes Beispiel sind `CheckBox`-Steuerelemente in Windows Forms-Anwendungen: Sie verfügen über einen Zwischenzustand, der den Zustand »nicht definiert« anzeigen soll. Eine einfache boolesche Variable könnte alle möglichen Zustände nicht aufnehmen – `True` und `False` sind dafür einfach zu wenig. Anders ist es, wenn Sie eine Variable vom Typ `Boolean` definieren könnten, die auch den »Nichts-ist-gespeichert«-Wert widerspiegeln könnte.

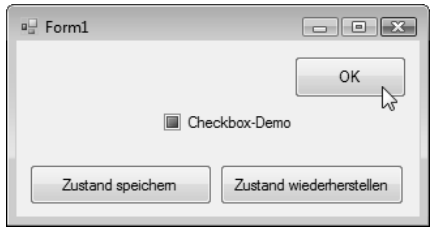



**Abbildung 3.6** Ein `Boolean` eignet sich auch dazu, Zwischenzustände eines `CheckBox`-Steuerelements zu speichern

Und das geht: In Visual Basic 2008 definiert man eine primitive Variable als Nullable-Datentyp, indem man ihrem Bezeichner oder dem Bezeichner für den Datentyp ein Fragezeichen anhängt. Das sieht entweder so …

```
Private myCheckBoxZustand As Boolean?
```

… oder so aus:

```
Private myCheckBoxZustand? As Boolean
```

In Visual Basic 2005 war es notwendig, Nullables auf folgende Weise zu definieren:

```
Private myCheckBoxZustand As Nullable(Of Boolean)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Aber keine Angst: Sie müssen sich jetzt nicht durch hunderte, vielleicht tausende Zeilen Code hangeln und für Visual Basic 2008 die entsprechenden Änderungen mit dem Fragezeichen vornehmen. Die umständlichere ältere Variante behält natürlich nach wie vor ihre Gültigkeit.

**BEGLEITDATEIEN** Unter *.\Samples\Chapter03 - NeuInCompiler\NullableUndCheckbox* finden Sie das folgende Beispielprojekt.

Dieses Beispiel demonstriert, wie alle Zustände eines `CheckBox`-Steuerelements, dessen `ThreeState`-Eigenschaft zur Anzeige aller *drei* Zustände auf `True` gesetzt wurde, in einer Member-Variablen vom Typ `Boolean?` gespeichert werden können. Klicken Sie beim Ausführen der Anwendung auf *Zustand speichern*, um den Zustand des `CheckBox`-Steuerelements in der Member-Variablen zu sichern, verändern Sie anschließend den Zustand, und stellen Sie den ursprünglichen Zustand des `CheckBox`-Steuerelements mit der entsprechenden Schaltfläche wieder her.

Der entsprechende Code dazu lautet folgendermaßen:

```
Public Class Form1

    Private myCheckBoxZustand As Boolean?

    'Das ginge auch:
    'Private myCheckBoxZustand? As Boolean

    'Und das auch:
    'Private myCheckBoxZustand As Nullable(Of Boolean)

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
                        btnOK.Click
        Me.Close()
    End Sub

    Private Sub btnSpeichern_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
                        btnSpeichern.Click

        If chkDemo.CheckState = CheckState.Indeterminate Then
            myCheckBoxZustand = Nothing
        Else
            myCheckBoxZustand = chkDemo.Checked
        End If

    End Sub

    Private Sub btnWiederherstellen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
                        Handles btnWiederherstellen.Click
        If Not myCheckBoxZustand.HasValue Then
            chkDemo.CheckState = CheckState.Indeterminate
        Else
            If myCheckBoxZustand.Value Then
                chkDemo.CheckState = CheckState.Checked
            Else
                chkDemo.CheckState = CheckState.Unchecked
            End If
        End If
    End Sub
End Class
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Zeilen, in denen die Member-Variable `Boolean?` zum Einsatz kommt, sind im Listing fett markiert. Dabei fällt Folgendes auf:

- **Wertzuweisung:** Wenn Sie einen Wert des zugrunde liegenden Typs an `type?` zuweisen wollen, können Sie die implizite Konvertierung verwenden, den entsprechenden Wert also direkt zuweisen, etwa wie in der Zeile

```
myCheckBoxZustand = chkDemo.Checked
```

  zu sehen.

- **Auf `Nothing` zurücksetzen:** Möchten Sie eine Nullable-Instanz auf `Nothing` zurücksetzen, weisen Sie ihr einfach den »Wert« `Nothing` zu – wie im Listing an dieser Stelle zu sehen:

```
myCheckBoxZustand = Nothing
```

- **Auf Wert prüfen:** Möchten Sie wissen, ob eine `Nullable`-Instanz einen Wert oder `Nothing` enthält, verwenden Sie deren Eigenschaft `HasValue`. Auch dafür gibt es ein Beispiel im Listing:

```
If Not myCheckBoxZustand.HasValue Then
    chkDemo.CheckState = CheckState.Indeterminate
Else
.
.
.
```

- **Wert abrufen:** Und schließlich müssen Sie natürlich auch den Wert, den eine `Nullable`-Instanz trägt, wenn sie nicht `Nothing` ist, ermitteln können. Dazu dient die Eigenschaft `Value`. Ein Beispiel dafür:

```
If myCheckBoxZustand.Value Then
    chkDemo.CheckState = CheckState.Checked
Else
    chkDemo.CheckState = CheckState.Unchecked
End If
.
.
.
```

**HINWEIS** Erst in diesem Beispiel fiel auf, dass man offensichtlich ein `CheckBox`-Steuerelement, dessen `ThreeState`-Eigenschaft gesetzt ist und das momentan den *Intermediate*-Zustand trägt, nicht mit seiner `Checked`-Eigenschaft in einen anderen Zustand versetzen kann (`Checked` oder `Unchecked`). Sie können in diesem Fall nur die `CheckState`-Eigenschaft verwenden, um das `CheckBox`-Steuerelement programmgesteuert wieder aus dem *Intermediate*-Zustand herauszuholen!

## Besonderheiten bei Nullable beim Boxen

Der Datentyp Nullable (`type?`) ist das, was man in Visual Basic als Struktur programmieren würde, also ein Wertetyp. Doch Sie könnten diesen Wertetyp nicht 1:1 nachprogrammieren, denn er erfährt durch die Common Language Runtime eine besondere Behandlung – und das ist auch gut so.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**BEGLEITDATEIEN** Unter .*Samples**Chapter03 - NeuInCompiler**NullableDemo* finden Sie das folgende Beispielprojekt.

Wenn Sie eine Instanz einer beliebigen Struktur – also eines beliebigen Wertetyps – verarbeiten, kommt irgendwann der Zeitpunkt, an dem Sie diesen Wertetyp in einer Objektvariablen boxen müssen – beispielsweise wenn Sie ihn als Bestandteil eines Arrays oder einer Auflistung (Collection) speichern.

Wann immer Sie einen definierten Wertetyp in einem Objekt boxen, kann dieses Objekt logischerweise nicht `Nothing` sein, ganz egal, welchen »Wert« diese Struktur hat. Im Falle des Nullable-Typs ist das anders, wie das folgende Beispiel zeigt:

```
Module NullableDemo

    Sub Main()
        Dim locObj As Object
        Dim locNullOfInt As Integer? = Nothing

        'Es gibt natürlich eine verwendbare Instanz, denn
        'Integer? ist ein Wertetyp!
        Console.WriteLine("Hat locNullOfInt einen Wert: " & locNullOfInt.HasValue)

        'Und dennoch ergibt das folgende Konstrukt True,
        'als würde locObj keine Referenz haben!
        locObj = locNullOfInt
        Console.WriteLine("Ist locObj Nothing? " & (locObj Is Nothing).ToString)
        Console.WriteLine()

        'Und auch das "Entboxen" geht!
        'Es gibt keine Null-Exception!
        locNullOfInt = DirectCast(locObj, Integer?)

        'Und geht das dann auch? - Natürlich!
        locNullOfInt = DirectCast(Nothing, Integer?)

        'Und noch weiter. Wir boxen einen Integer?
        locNullOfInt = 10
        '
        locObj = locNullOfInt
        Dim locInt As Integer = DirectCast(locObj, Integer)

        Console.WriteLine("Taste drücken zum Beenden!")
        Console.ReadKey()

        'Das geht übrigens nicht, obwohl Nullable die
        'Contraints-Einschränkung im  Grunde genommen erfüllt!
        'Dim locNullOfInt As Nullable(Of Integer?)
    End Sub
End Module
```

Wenn Sie dieses Beispiel ausführen, gibt es die folgenden Zeilen aus:

```
Hat locNullOfInt einen Wert: False
Ist locObj Nothing? True

Taste drücken zum Beenden!
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Das, was hier passiert, ist beileibe keine Selbstverständlichkeit – aber dennoch sauberes Design der CLR, denn: Zwar wird `locNullOfInt` nicht initialisiert (oder, um es in diesem Beispiel deutlich zu machen, mit `Nothing` – aber das kommt auf dasselbe raus), aber natürlich existiert dennoch eine Instanz der Struktur. Sie spiegelt eben nur den Wert `Nothing` wider. Gemäß den bekannten Regeln müsste das anschließende Boxen in der Variablen `locObj` auch ergeben, dass `locObj` einen Zeiger auf eine Instanz der `Nothing` widerspiegelnden `locNullOfInt` enthält und keinen Null-Zeiger. Doch das ist nicht der Fall, denn die anschließende Ausgabe von

```
Console.WriteLine("Ist locObj Nothing?" & (locObj Is Nothing).ToString)
```

zeigt

```
Ist locObj Nothing? True
```

auf dem Bildschirm an.

Das »zurückcasten« von `Nothing` in einen Nullable ist damit natürlich genauso gestattet, wie ebenfalls im Listing zu sehen.

Und noch eine Unregelmäßigkeit erfahren Nullables, nämlich wenn es darum geht, einen geboxten Typ (vorausgesetzt er ist eben nicht `Nothing`) in seinen Grundtyp zurückzucasten, wie der folgende Codeausschnitt zeigt:

```
        'Und noch weiter. Wir boxen einen Integer?
        locNullOfInt = 10
        '
        locObj = locNullOfInt
        Dim locInt As Integer = DirectCast(locObj, Integer)
```

Hier wird ein Nullable-Datentyp in einem Objekt geboxt, aber später zurück in seinen *Grunddatentypen* gewandelt. Ein, wie ich finde, logisches Design, was allerdings dem »normalen« Vorgehen beim Boxen von Wertetypen in Objekten völlig widerspricht.

**HINWEIS** Kleine Anekdote am Rande: Dieses Verhalten ist erst zu einem sehr, sehr späten Zeitpunkt beim Entwickeln von Visual Studio 2005 und dem .NET Framework in die CLR eingebaut worden und hat für erhebliche Mehrarbeit bei allen Entwicklerteams und viel zusätzlichen Testaufwand gesorgt. Dass sich Microsoft dennoch für das nunmehr implementierte Verhalten entschieden hat, geht nicht zuletzt auf das Drängen von Kunden und Betatestern zurück, die das Design mit der ursprünglichen, »normalen« CLR-Behandlung von Nullables nicht akzeptieren konnten und als falsch erachteten.

# Anonyme Typen

Mit anonymen Typen können eine Reihe schreibgeschützter Eigenschaften in einem einzelnen Objekt gekapselt werden, ohne dafür zuerst einen konkreten Typ – also eine Klasse oder eine Struktur – explizit definieren zu müssen. Natürlich entsteht durch den Compiler ein .NET-konformer Typ bei diesem Vorgang, doch der Typname wird vom Compiler generiert, und er ist nicht auf Quellcodeebene verfügbar. Der Typ der Eigenschaften wird vom Compiler abgeleitet.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der Sinn von anonymen Typen wird nur im Rahmen von LINQ deutlich: Sie werden normalerweise in der `Select`-Klausel eines Abfrageausdrucks verwendet, um einen Typen zu erstellen, der ein untergeordnetes Set der Eigenschaften enthält, die in jedem Objekt, das in der Abfrage berücksichtigt wird, quasi »vorkommen«.

Wie Sie anonyme Typen genau verwenden, erfahren Sie sinnvollerweise im richtigen Kontext und deswegen im LINQ-Teil dieses Buches – an dieser Stelle seien sie nur der Vollständigkeit halber erwähnt.

Kapitel 7 zeigt Ihnen im Einführungsteil von LINQ, wie Sie anonyme Typen im Zusammenang mit den LINQ-Erweiterungsmethoden anwenden können.

# Lambda-Ausdrücke

Bei Lambda-Ausdrücken handelt es sich um Funktionen, die nur im Kontext definiert werden, die Ausdrücke und Anweisungen enthalten und die für die Erstellung von Delegaten oder Ausdrucksbaumstrukturen verwendet werden können. Man nennt sie auch anonyme Funktionen, weil sie Funktionen bilden, die aber selbst keinen Namen haben. Gerade wieder mit Schwerpunkt auf LINQ ist es oft notwendig, an bestimmten Stellen Delegaten – also Funktionszeiger – einzusetzen, die aber ausschließlich bei einem konkreten Aufruf und mit einer Minimalausstattung an Code zurechtkommen.

Der Visual Basic-Compiler kennt Lambda-Ausdrücke seit der Version 2008 – sie seien an dieser Stelle aber nur der Vollständigkeit halber erwähnt. Im nächsten Kapitel finden Sie die ausführliche Erklärung von Lambda-Ausdrücken im richtigen Kontext von Delegaten und Klassen und mit anschaulichen Praxisbeispielen erklärt.

# Abfrageausdrücke mit LINQ

LINQ (*L*anguage *in*tegrated *Q*uery, etwa: *sprachintegrierte Abfrage*) ist eine mächtige Erweiterung des VB-Compilers, und LINQ ermöglicht es, Datenabfragen, Datensortierungen und Datenselektionen durchzuführen, etwa von Business-Objekten, die in Auflistungen gespeichert sind, aber auch von Daten die in SQL-Server-Datenbanken gespeichert sind.

Mit dem Service Pack 1 von Visual Basic 2008 wird es darüber hinaus möglich, LINQ-Abfragen für alle Daten-Provider durchzuführen, für die eine ADO.NET-Implementierung vorhanden ist (*LINQ to Entities*); auch die Implementierung eigener LINQ-Provider wird dann möglich sein.

Dem Thema LINQ ist ein eigener Buchteil gewidmet; da LINQ jedoch den größten Aufwand bei der Entwicklung des Visual Basic 2008-Compilers darstellte, sei das Thema im Rahmen dieses Kapitels zumindest der Vollständigkeit halber erwähnt.

# Erweiterungsmethoden

Prinzipiell gab es schon immer Möglichkeiten, Klassen um neue Methoden oder Eigenschaften zu erweitern. Sie vererbten sie, und fügten ihnen anschließend im vererbten Klassencode neue Methoden oder Eigenschaften hinzu. Das funktionierte solange, wie es sich um Typen handelte, die …

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- … nicht mit dem `NotInheritable`-Modifizierer (etwa: *nicht vererbbar*; der Vollständigkeit halber: `Sealed` in C#, etwa: *versiegelt*) gekennzeichnet waren, oder
- … keine Wertetypen waren – denn mit `Structure` erstellte Typen sind automatisch Wertetypen und die sind implizit nicht vererbbar.

Für eine bessere Strukturierung Ihres Codes wurden daher in Visual Basic 2008 sogenannte Erweiterungsmethoden eingeführt, mit deren Hilfe Entwickler bereits definierten Datentypen benutzerdefinierte Funktionen hinzufügen können, ohne eben einen neuen, vererbten Typ zu erstellen. Erweiterungsmethoden sind eine besondere Art von statischen Methoden, die Sie jedoch wie Instanzmethoden für den erweiterten Typ aufrufen können. Für in Visual Basic (und natürlich auch C#) geschriebenen Clientcode gibt es keinen sichtbaren Unterschied zwischen dem Aufrufen einer Erweiterungsmethode und den Methoden, die in einem Typ tatsächlich definiert sind.

Allerdings gibt es dabei Einschränkungen bzw. Konventionen, was das Erweitern von Klassen auf diese Weise anbelangt:

- Erweiterungsmethoden müssen mit einem Attribut besonders gekennzeichnet werden. Dazu dient das `ExtensionAttribut`, das Sie im Namespace `System.Runtime.CompilerServices` finden.
- Bei einer Erweiterungsmethode kann es sich ausschließlich um eine `Sub` oder eine `Function` handeln. Sie können eine Klasse durch Erweiterungsmethoden *nicht* um Eigenschaften, neue Member-Variablen (Felder) oder Ereignisse erweitern.
- Eine Erweiterungsmethode muss sich in einem gesonderten Modul befinden.
- Im Bedarfsfall muss das Modul, sollte es sich in einem anderen Namespace befinden als die Instanz, die es verwendet, wie jede andere Klasse auch, entsprechend importiert werden.
- Eine Erweiterungsmethode weist mindestens einen Parameter auf, der den Typ (die Klasse, die Struktur) bestimmt, die sie erweitert. Möchten Sie beispielsweise eine Erweiterungsmethode für den `String`-Datentyp erstellen, muss der erste Parameter, den die Erweiterungsmethode entgegennimmt, vom Typ `String` sein. Aus diesem Grund kann ein `Optional`-Parameter oder ein `ParamArray`-Parameter nicht der erste Parameter in der Parameterliste sein, da diese zur Laufzeit variieren, dem Compiler aber bereits zur Entwurfszeit bekannt sein müssen.

## Der eigentliche Zweck von Erweiterungsmethoden

Sie werden Erweiterungsmethoden in Ihren eigenen Klassen und Assemblies (Klassenbibliotheken) so gut wie immer benötigen, denn sie haben den Quellcode, und können ihn – natürlich immer versionskompatibel(!) – so erweitern, wie Sie es wünschen.

Der eigentliche Zweck von Erweiterungsmethoden ist es, eine Infrastruktur für LINQ schaffen zu können, mit denen sich die Verwendung allgemeingültiger, generischer und statischer Funktionen typgerecht »anfühlt«. Für das tiefere Verständnis brauchen Sie allerdings gute Kenntnisse von Schnittstellen, den Abschnitt über Lambda-Ausdrücke sollten Sie ebenfalls bereits durchgearbeitet haben.

Die Entwickler des .NET Frameworks 3.5 standen vor der Aufgabe, in vorhandene Typen quasi eine zusätzliche Funktionalität einzubauen, ohne aber den vorhandenen Code dieser Typen – insbesondere waren dabei alle generischen Auflistungsklassen betroffen – in irgendeiner Form auch nur um ein einzelnes Byte zu verändern. Das hätte nämlich zwangsläufig zu so genannten *Breaking Changes* (Änderungen mit unbe-

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

rechenbaren Auswirkungen auf vorhandene Entwicklungen) geführt. So war es prinzipiell nur möglich, zusätzliche Funktionalität in statischem, generischem Code unterzubringen, der sich allerdings sehr »uncool« angefühlt hätte.

Erweiterungsmethoden sind im Grunde genommen nur eine Mogelpackung, denn sie bleiben das, was sie sind: Simple statische Methoden, die sich, wie gesagt, lediglich wie Member-Methoden der entsprechenden Typen anfühlen. Aber, jetzt kommt der geniale Trick: Da Erweiterungsmethoden alle Typen als ersten Parameter aufweisen können, können ihnen auch Schnittstellen als Typ übergeben werden. Das führt aber zwangsläufig dazu, dass alle Typen, die diese Schnittstelle implementieren, auch mit den entsprechenden scheinbar zusätzlichen Methoden ausgestattet sind. In Kombination mit Generics und Lambda-Ausdrücken ist das natürlich eine geniale Sache, da die eigentliche Funktionalität durch – je nach Ausbaustufe der Überladungsversionen der statischen Erweiterungsmethoden – einen oder mehrere Lambda-Ausdrücke gesteuert wird und der eigentlich zu verarbeitende Typ eben durch die Verwendung von Generics erst beim späteren Typisieren (Typ einsetzen) ans Tageslicht kommt. Mit diesem Kunstgriff schafft man im Handumdrehen eine komplette, typsichere Infrastruktur beispielsweise für das Gruppieren, Sortieren, Filtern, Ermitteln oder sich gegenseitige Ausschließen von Elementen aller Auflistungen, die eine bestimmte, um Erweiterungsmethoden ergänzte Schnittstelle implementieren – beispielsweise bei `IEnumerable(T)` um die statischen Methoden der Klasse `Enumerable`. Oder kurz zusammengefasst: *Alle* Auflistungen, die `IEnumerable(T)` einbinden, werden scheinbar um Member-Methoden ergänzt, die in `Enumerable` zu finden sind, und das sind genau diejenigen, welche für LINQ benötigt werden, um Selektionen, Sortierungen, Gruppierungen und weitere Funktionalitäten von Daten durchzuführen. Es ist aber nur scheinbar so – die ganze Funktionalität von LINQ wird letzten Endes in eine rein prozedural aufgebaute Klasse mit einer ganzen Menge statischer Funktionen delegiert.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 4

# Klassen und Objekte

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Klassen sind der Hauptstützpfeiler der objektorientierten Programmierung. Spricht man von Klassen und Klassenkonzepten, dann sind Technologien wie Schnittstellen, Klassenmodelle, das Ableiten bzw. Vererben von Klassen sowie Polymorphie mit dem gezielten Überschreiben von Klassenmethoden und –eigenschaften nicht weit. Doch darum soll es vordergründig in diesem Kapitel gar nicht gehen.

Klassen sind, mit Hinblick auf Auflistungen und letzten Endes auf das Datenabfragekonzept *LINQ to Objects*, in erster Linie mal Schablonen, die ihre Daten reglementieren und von der Außenwelt schützen – und auch nur unter diesem Gesichtspunkt sollen sie in diesem Kapitel erklärt werden.

Polymorphie im Rahmen von Klassenvererbung, insbesondere was das Überschreiben von Methoden und Eigenschaften anbelangt, das Erstellen von abstrakten Klassen für Klassenvorlagen und Schnittstellen sollen nicht Thema dieses Kapitels sein, da es für die weiteren Neuerungen, insbesondere LINQ nicht von so entscheidender Bedeutung ist (insgesamt aber natürlich schon – es würde nur einfach den Rahmen dieses Buches sprengen).

**TIPP** Wer unter Ihnen nach der Lektüre dieses Kapitels hungrig auf mehr geworden ist, der kann das komplette Buch *Visual Basic 2005 – das Entwicklerbuch* direkt von der ActiveDevelop-Homepage herunterladen. Sie erreichen unseren Internetauftritt unter *http://www.activedevelop.de*.

# Einführung

»Eine Klasse schafft die Strukturen für das Speichern von Daten und beinhaltet gleichzeitig Programmcode, der diese Daten reglementiert.« Diese Erklärung finden Sie nicht nur oftmals als *die* Erklärung für das Konzept von Klassen, sie ist auch kurz, knackig, absolut zutreffend und so abstrakt, dass jemand, der sich zum ersten Mal mit dieser Materie beschäftigt, damit überhaupt nichts anfangen kann.

Deswegen wollen wir im Folgenden das Pferd bewusst von der anderen Seite aufzäumen und ein Beispiel bemühen, das zeigt, wie ein Programm Daten speichert und organisiert, das *ohne* Klassen zurechtkommen muss.

## Miniadresso – die prozedurale Variante

Also lassen Sie uns mal einen Blick auf die »So-besser-Nicht«-Variante werfen – hochtrabend *Miniadresso-Prozedu* genannt.

**HINWEIS** Bei den Beispielprogrammen der folgenden Abschnitte handelt es sich nicht um Windows-, sondern um so genannte Konsolenanwendungen – um Anwendungen also, die sich ausschließlich unter der Windows-Eingabeaufforderung verwenden lassen.

### Konsolenanwendung in VB.NET

Im Gegensatz zu Visual Basic 6.0 können Sie in Visual Basic .NET auch ohne größeren Aufwand Konsolenanwendungen entwerfen. Das sind Programme, die über keine grafische Oberfläche verfügen, sondern nur unter der Windows-Eingabeaufforderung laufen und mit dem Anwender über reine Textein- und -ausgabe kommunizieren. Konsolenanwendungen sind für Programme sehr gut geeignet, die bei der reinen Stapelverarbeitung eingesetzt werden sollen und wenig oder überhaupt keine Kommunikation mit dem Anwender erfordern. Aber gerade auch beim Debuggen – zum Beispiel, um ohne großen Aufwand neue Typen (Klassen, Strukturen) zu testen – leisten sie sehr gute Dienste. ▶

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Möchten Sie selber eine neue Kommandozeilenanwendung erstellen, wählen Sie in der Visual Studio-IDE aus dem Menü *Datei* den Menüpunkt *Neu/Projekt* und im Dialog, den Visual Studio anschließend zeigt, aus dem Zweig *Visual Basic* und dem Bereich *Windows* die Vorlage *Konsolenanwendung*.

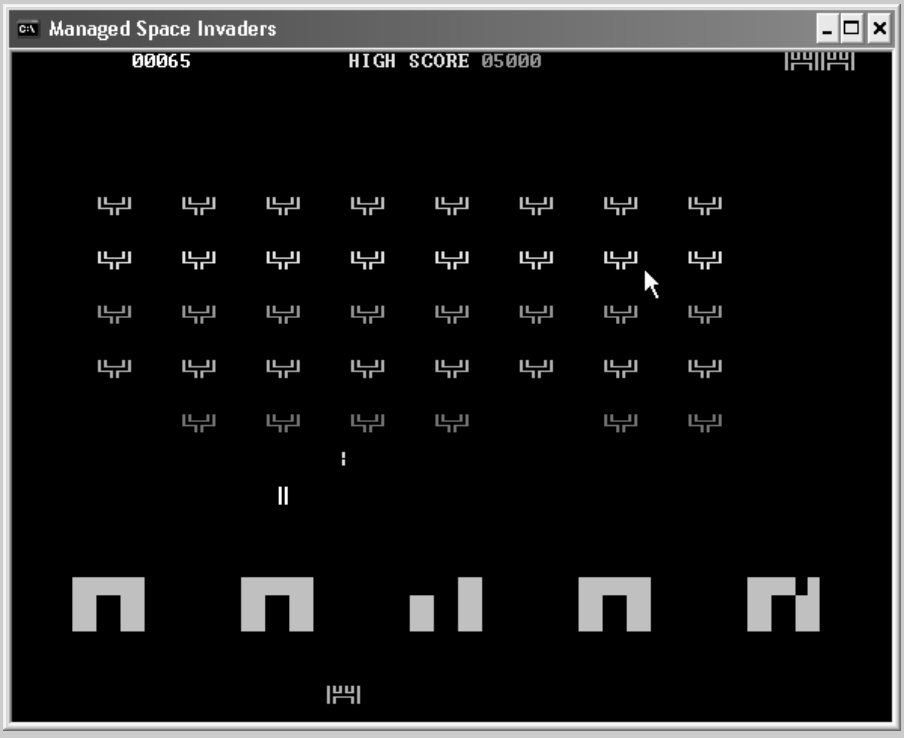



**Abbildung 4.1** Kid George vom BCL-Team demonstrierte zum Launch von Visual Studio 2005 die Möglichkeiten von Konsolenanwendungen eindrucksvoll mit seiner Implementierung von Space Invaders!

Seit Visual Studio 2005, bzw. dem .NET Framework 2.0, gibt es übrigens einige Erweiterungen in der Kommandozeilenunterstützung aus .NET heraus. So haben Sie direkt über die `Console`-Klasse Einfluss auf die Farbgebung, auf die direkte Positionierung der jeweils nächsten Ausgabe und Sie können sogar ganze Bildschirmbereiche mit einfachen Befehlen verschieben, und damit Scrolling in jede Richtung oder sogar einfache bewegte Bildschirmgrafiken realisieren.

**BEGLEITDATEIEN** Unter *.\Samples\Chapter04\MiniAdressoProzedu* finden Sie das Beispielprojekt dieses Abschnittes.

Wenn Sie das Programm starten, sehen Sie ein Konsolenfenster, das die Programmausgabe beinhaltet, wie Sie sie in etwa auch in Abbildung 4.2 betrachten können.

**TIPP** Bei den vergleichsweise großen Auflösungen der 22 und 24 Zoll-Monitore, die heutzutage schon für kleines Geld zu haben sind, stellen Sie den Font der Konsolenfenster am besten auf die größte Schriftart (10 × 18), die Ihnen das Systemmenü über den Menüpunkt *Standardwerte* (Windows Vista) bzw. *Eigenschaften* (Windows XP) und im Dialog, der jetzt erscheint, über die Registerkarte *Schriftart* anbietet.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

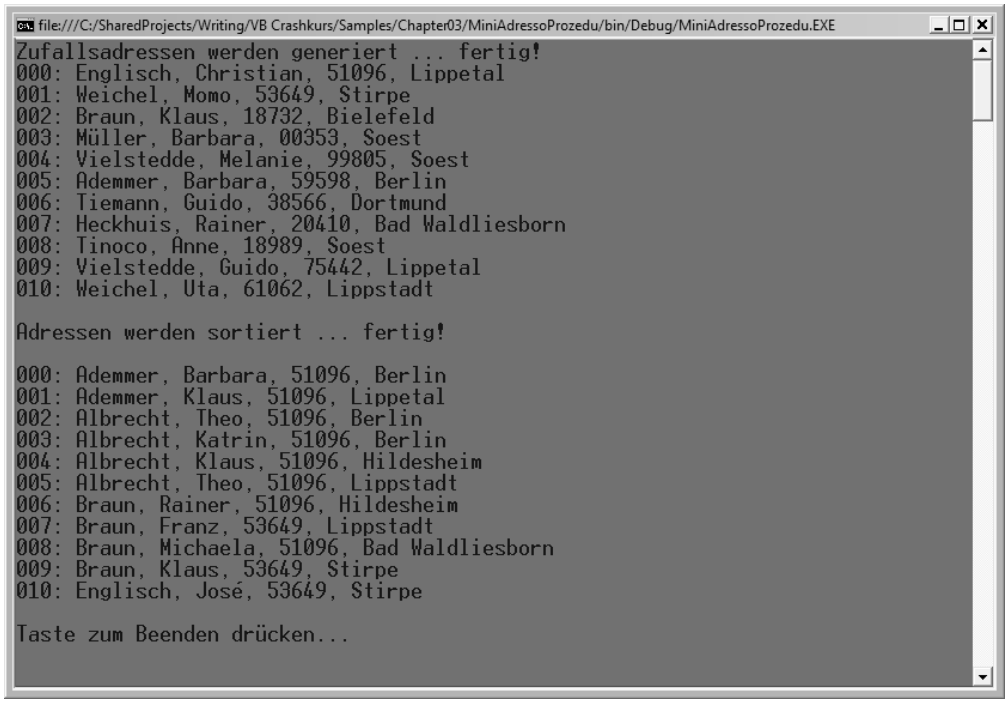



**Abbildung 4.2** Die erste Version des Programms macht scheinbar Dienst nach Vorschrift – doch durch das prozedurale Konzept hat sich leider ein schwerer Fehler eingeschlichen, der nicht gleich ersichtlich ist.

OK – hier haben wir also die erste Version unseres kleines Beispielprogramms, das nichts weiter macht, als sich ein paar Kontaktadressen auszudenken, deren Einzeldaten in dafür vorgesehene Arrays abzulegen, die Kontakte nach Nachnamen zu sortieren und sie anschließend auszugeben. Schauen wir uns an, wie das Programm generell mit seinen Daten umgeht:

```
Module Hauptmodul

    Dim Nachname(0 To 100) As String
    Dim Vorname(0 To 100) As String
    Dim PLZ(0 To 100) As String
    Dim Ort(0 To 100) As String

    Sub Main()
```

Es benötigt für jede Kontaktadresse jeweils vier Einzelelemente, die es in insgesamt vier Arrays ablegt, und genau das macht es so schwierig für den Entwickler, die Daten einer einzigen Kontaktadresse auch wirklich zusammen zu halten. Ohne eine *zusammenhängende* Entität zu schaffen, die die Daten eines Kontakts auch programmtechnisch mehr oder weniger automatisch bzw. durch ihr Konzept bedingt zusammenhält, sind Fehler buchstäblich vorprogrammiert – und genau ein solcher Fehler hat sich dann auch in dieser Programmversion eingeschlichen (OK, natürlich habe ich ihn zu Demozwecken einschleichen lassen, aber dieses prozedurale Programmierbeispiel ist typisch für eine interne Datenverwaltung, genau so typisch eben wie die Fehler, die dabei auftreten können).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Sub AdressenSortieren()

        'Beispiel für lokalen Typrückschluss
        Dim anzahlElemente = 101
        Dim delt = 1

        Dim aeussererZaehler As Integer
        Dim innererZaehler As Integer
```

Bei einem Blick auf die Sortierroutine fällt zunächst etwas Besonderes auf, was es neu in Visual Basic 2008 gibt und unter dem Namen *lokaler Typrückschluss* bezeichnet wird. Dabei werden die Typen von Variablen nur durch reine Zuweisung von Konstanten festgelegt. Kapitel 3 erklärt, wie der lokale Typrückschluss genau funktioniert.

Doch zurück zum eigentlichen Ausgangsproblem: Die Sortierroutine des Programms ist in diesem Beispiel der Übeltäter: Gerade sie müsste durch extrem sorgfältige Ausführung bei der Programmierung dafür sorgen, dass ein Adressdatensatz logisch zusammengehörig bleibt, damit die Integrität des gesamten Datenaufkommens nicht verletzt wird. Schauen wir uns diese Sortierroutine jedoch genauer an …

```
        Dim tempVorname, tempNachname, tempPLZ, tempOrt As String

        'Größten Wert der Distanzfolge ermitteln
        Do
            delta = 3 * delta + 1
        Loop Until delta > anzahlElemente

        Do
            'Späteres Abbruchkriterium - entsprechend kleiner werden lassen
            delta \= 3

            'Shellsort's Kernalgorithmus
            For aeussererZaehler = delta To anzahlElemente – 1
                tempVorname = Vorname(aeussererZaehler)
                tempNachname = Nachname(aeussererZaehler)
                tempPLZ = PLZ(aeussererZaehler)
                tempOrt = Ort(aeussererZaehler)

                innererZaehler = aeussererZaehler
                Do
                    If tempNachname >= Nachname(innererZaehler - delta) Then Exit Do
                    Vorname(innererZaehler) = Vorname(innererZaehler - delta)
                    Nachname(innererZaehler) = Nachname(innererZaehler - delta)
                    PLZ(innererZaehler) = PLZ(innererZaehler - delta)

                    innererZaehler = innererZaehler – delta
                    If (innererZaehler <= delta) Then Exit Do
                Loop
                Vorname(innererZaehler) = tempVorname
                Nachname(innererZaehler) = tempNachname
                Ort(innererZaehler) = tempOrt
            Next
        Loop Until delta = 0
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

… so stellen wir bei näherem Hinsehen fest, dass sich, was die Datenintegrität anbelangt, ein dicker Fehler in den Algorithmus eingeschlichen hat: Beim Dreieckstausch eines Kontaktelements nämlich, werden nicht wirklich alle Felder getauscht. Beim »Hintauschen« bleibt der Ort, beim »Zurücktauschen« die Postleitzahl auf der Strecke; für ein kommerzielles Programm wäre das natürlich eine Katastrophe, zumal selbst bei einer Liste von nur 100 Kontaktadressen dieser Fehler nicht sofort ins Auge sticht.

# Was ist eine Klasse?

Versetzen Sie sich in Ihre Kindheit zurück. Und zwar so weit, dass Sie sich vorstellen können, im Sandkasten zu sitzen und mit Förmchen und nassem Sand zu spielen. Sie werden es nicht glauben, aber genau zu diesem Zeitpunkt haben Sie bereits das Klassenkonzept angewendet. Ein Förmchen ist im Grunde genommen nämlich nichts anderes als eine Klasse. Das Förmchen gibt vor, wie Objekte ausschauen sollen, die aus ihm entstehen werden, aber selbst ist es noch kein Objekt, sondern nur eine Vorlage. Wenn Sie aus einer Klasse (einem Förmchen) einen Sandkuchen (ein Objekt) machen wollen, dann müssen Sie ein Objekt dieser Klasse instanziieren. Um bei der Analogie zu bleiben: Sie instanziieren einen Sandkuchen aus einem Förmchen, indem Sie nassen Sand in die Form hineingeben, das ganze Ding umdrehen und die Form abziehen.

Klassen im .NET Framework sind natürlich etwas abstrakter, aber Sie machen sich daran auch nicht die Hände schmutzig. Sie bestehen erst einmal aus einer Reihe von sogenannten Member-Variablen – auch Felder oder Feldvariablen genannt – und der Klassenrumpf-Definition, die diese Variablen einschließt und der Klasse ihren Namen gibt.

Damit existiert dann bereits die einfachste Version einer Klasse, die, wie wir gleich sehen werden, ihre Daten logisch kapselt. Immerhin erlaubt sie uns schon, das Problem unseres ersten Beispielprogramms im Ansatz zu ersticken – die zweite Version dieses Demoprogramms stellt das unter Beweis. Hier gibt es nun eine Klasse namens `Kontakt`, die aus vier Feldern besteht – die vier Datenfelder, die eine einzige Adresse ausmachen. In der Folge benötigen wir nun nicht mehr vier, eigentlich völlig zusammenhanglose Arrays, sondern nur noch ein einziges Array[1], das aus Elementen besteht, die jeweils eine komplette Adresse speichern.

Um das zu tun, fügen wir dem bestehenden Programm einfach eine neue Codedatei hinzu – eine Klassendatei – namens *Kontakt.vb*. In diese Klassen-Codedatei schreiben wir dann folgende Zeilen:

```
Public Class Kontakt
    Public Nachname As String
    Public Vorname As String
    Public PLZ As String
    Public Ort As String
End Class
```

Diese Klasse namens `Kontakt` stellt die einfachste Form einer Klasse dar. Sie enthält nur vier öffentlich zugängliche Felder, aber die Klasse fasst eben diese vier Felder logisch zu einem Datensatz zusammen.

[1] Mehr zu Arrays und Auflistungen erfahren Sie im nächsten Kapitel.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## Klassen mit New instanziieren

Um eine so genannte Instanz dieser Klasse zu schaffen, bedient man sich des Schlüsselwortes `New` – intern wird damit der entsprechende Speicherplatz geschaffen, den man benötigt, um die Daten der Member-Variablen der Klasse im Arbeitsspeicher[2] abzulegen. Eine Objektvariable, die man zuvor als Typ dieser Klasse definiert, dient dann dazu, auf die Elemente der Klasseninstanz zuzugreifen. Die Anwendung dieser einfachen Beispielklasse würde also wie folgt aussehen:

```
Dim adr As Kontakt
adr = New Kontakt
```

Die erste Codezeile legt eine Objektvariable vom Typ `Kontakt` an und die zweite Codezeile weist ihr eine neue Instanz zu. Sie können diese beiden Zeilen auch in einer Zeile zusammenfassen:

```
Dim adr As New Kontakt
```

Anschließend verwenden Sie die Instanz dieser Klasse mithilfe der definierten Objektvariablen, um auf die einzelnen Felder zuzugreifen:

```
adr.Nachname = "Dröge"
adr.Vorname = "Ute"
adr.PLZ = "59555"
adr.Ort = "Lippstadt"
```

Um nochmals auf unsere Analogie mit dem Sandkuchen zurückzukommen: Die Objektinstanz entspricht hier einem Sandkuchen, der aus einem Förmchen hervorgegangen ist. Die Klasse hingegen ist das Förmchen. Die Objektvariable benennt wiederum den Sandkuchen, der aus dem Förmchen hervorgegangen ist, und natürlich können Sie, wie aus dem Förmchen, beliebig viele Sandkucheninstanzen erstellen. Beliebig viele? Nicht ganz: Natürlich nur so viele, bis der »verwaltete Sandberg« zuneige geht, aus dem Sie die Sandkuchen machen, der in der Analogie dem Managed Heap entspricht, aus dem Sie den Speicherplatz für Ihre Objektinstanzen beziehen.

Ein gern gemachter Fehler am Anfang ist es übrigens, die Klasse nicht zu instanziieren, sondern direkt verwenden zu wollen:

```
'So geht's natürlich nicht!
Kontakt.Nachname = "Dröge"
Kontakt.Vorname = "Ute"
Kontakt.PLZ = "59555"
Kontakt.Ort = "Lippstadt"
```

Warum? – Denken Sie mal drüber nach. In Analogie zu einer primitiven Variablen würden Sie praktisch statt

[2] Genau genommen in einem bestimmten Teil des Arbeitsspeichers, der durch das .NET Framework verwaltet, d.h. ständig aufgeräumt wird. Sein Name: *Managed Heap*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Dim i as Integer
i = 5
```

einfach

```
Integer = 5
```

schreiben, quasi den Bezeichner für den Datentyp als Variablennamen verwenden. Das geht natürlich nicht!

### Öffentliche Felder oder Eigenschaften beim Instanziieren initialisieren

Visual Basic erlaubt neu in der 2008er Version auch eine kürzere Variante beim Vorbelegen der Felder bzw. Eigenschaften mit Werten: Sie können öffentliche Felder oder Eigenschaften direkt beim Instanziieren mithilfe des `With`-Schlüsselwortes definieren, wie im folgenden Beispiel zu sehen:

```
Dim adr As New Kontakt With {.Nachname = "Dröge", _
                        .Vorname = "Ute", .PLZ = "59555", .Ort = "Lippstadt"}
```

## New oder nicht New – wieso es sich bei Objekten um Verweistypen handelt

Zum besseren Verständnis für alle späteren Kapitel ist es überaus sinnvoll, ein paar Worte zur Speicherung von Objekten, also Instanzen von Klassen zu verlieren. Objektvariablen und Objekte sind nämlich nicht *so* miteinander verbunden, wie man es sich zunächst vielleicht vorstellt. Im Gegenteil: Der Verbund einer Objektvariablen mit den eigentlichen Daten des Objektes hält bestenfalls so gut, wie eine amerikanische Prominentenehe: Was auf den ersten Blick so innig und für immer geschaffen zu sein scheint, ist in der nächsten Sekunde auch schon wieder sauber getrennter Schnee von gestern.

Die Wahrheit ist nämlich: Eine Objektvariable speichert im Grunde genommen nur *einen Zeiger* auf die eigentlichen Daten im Managed Heap, in den die Daten der Objekte gelangen, die mithilfe von `New` aus Klassen instanziiert werden.

Und wie wir (die Älteren unter uns jedenfalls) aus unseren Anfängertagen mit 64ern, Atari STs und Maschinensprache noch alle wissen, untergliedert sich der Arbeitsspeicher eines Computers in bestimmte Speicherstellen, die alle bestimmte »Hausnummern« (die Speicheradressen) besitzen.

Wenn Sie nun ein Objekt aus einer Klasse erstellen – zum Beispiel indem Sie einen `Kontakt`-Datentyp[3] mit `New` in die Objektvariable `objVarKontakt` instanziieren – dann legt das .NET Framework die Daten für diese Objektinstanz beispielsweise an Speicheradresse 460.386 auf dem Managed Heap ab, und die Objektvariable wird intern sozusagen zu einer `Integer`-Variablen (oder auf 64-Bit-Systemen zu einer `Long`-Variablen), die diese Adresse trägt. Bildlich sieht das folgendermaßen aus:

[3] Für diese Erklärung wurde übrigens auch der Datentypname `Kontakt` dem Namen `Adresse` vorgezogen, um nicht zu viel Konfusion mit dem Wort Speicher*adresse* entstehen zu lassen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

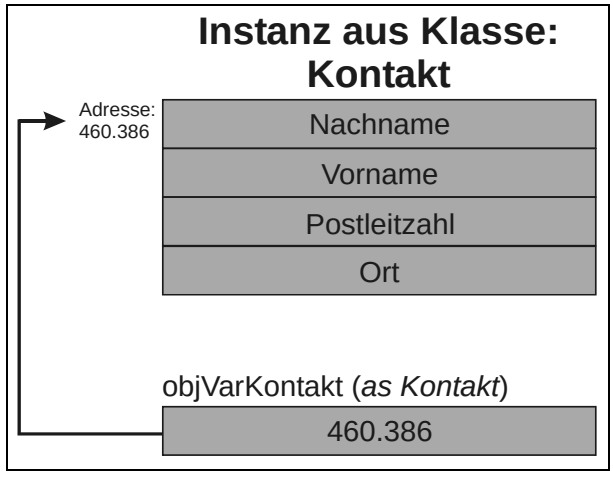



**Abbildung 4.3** Objektvariablen speichern im Grunde genommen nur die Speicheradressen auf die eigentlichen Daten, die das .NET Framework im Managed Heap ablegt

Diese Tatsache hat aber entscheidende Folgen, wie das folgende Beispiel gleich zeigen wird.

**BEGLEITDATEIEN** Sie finden dieses Beispielprojekt unter *.\Samples\ Chapter04\KlassenObjekteSpeicher*.

```
Module Module1

    Sub Main()

        'Instanziieren mit New und dadurch
        'Speicher für das Kontakt-Objekt
        'auf dem Managed Heap anlegen.
        'Instanz gleichzeitig mit Daten initialisieren.
        Dim objVarKontakt As New Kontakt With {.Nachname = "Pfeiffer", _
                .Vorname = "Ute", .PLZ = "59555", .Ort = "Lippstadt"}

        'Nur Objektvariable anlegen,
        'es wird aber kein Speicher reserviert!
        Dim objVarKontakt2 As Kontakt

        'objVarKontakt2 "zeigt" ab jetzt auf
        'dieselbe Instanz wie objVarKontakt
        objVarKontakt2 = objVarKontakt

        'Und das kann man auch beweisen:
        'Das Ändern der Instanz geschieht...
        objVarKontakt2.Nachname = "Dröge"

        'durch beide Objektvariablen, die natürlich
        'auch dasselbe widerspiegeln.
        Console.WriteLine(objVarKontakt.Nachname)

        Console.WriteLine()
        Console.WriteLine("Zum Beenden Taste drücken")
        Console.ReadKey()
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
End Module

Public Class Kontakt
    Public Vorname As String
    Public Nachname As String
    Public PLZ As String
    Public Ort As String
End Class
```

Wenn Sie dieses Beispiel laufen lassen, erhalten Sie die Ausgabe

```
Dröge

Zum Beenden Taste drücken
```

Beachten Sie, dass es in diesem Beispiel zwar nur eine einzige Dateninstanz der Klasse `Kontakt` gibt, die jedoch durch zwei Objektvariablen angesprochen und damit auch widergespiegelt wird: Beim Instanziieren wird die Adresse des Speichers, an dem die Daten der Instanz abgelegt werden, in der Objektvariablen `objVarKontakt` gespeichert. Diese Adresse wird in die Variable `objVarKontakt2` übertragen, und wichtig, hierbei wird nicht etwa eine Kopie der gesamten Instanz im Managed Heap angelegt!

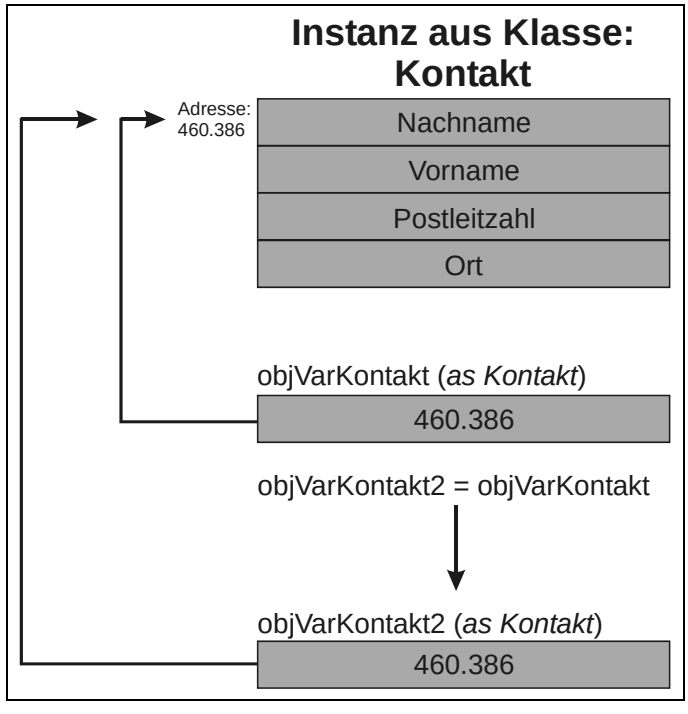



**Abbildung 4.4** Das Kopieren einer Objektvariablen in eine andere Objektvariable kopiert nur den Zeiger auf die Instanz – die dann durch beide Variablen manipulierbar und abrufbar wird

Eine Objektvariable »zeigt« also quasi auf die Daten, die sie verwaltet. In anderen Programmiersprachen wie C++ gibt es zu diesem Zweck besondere Variablen, die auch als *Zeiger* bezeichnet werden. In Visual Basic wird eine Objektvariable, die die Instanz einer Klasse verwaltet, automatisch zu dem, was man in anderen Programmiersprachen als Zeiger bezeichnet. In Form einer Grafik sieht das Ganze dann aus wie in Abbildung 4.4.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Dieses Verhältnis zwischen Objektvariable und eigentlichem Objekt (eigentlicher Instanz) sollten Sie sich gut einprägen, da sie einerseits Quelle schwer zu findender Fehler ist – schließlich kann es auch versehentlich passieren, dass, wenn Sie nicht aufpassen, eine Objektvariable die Instanz eines Objektes verändert, die eigentlich und ausschließlich durch eine ganz andere Objektvariable angesprochen werden sollte. Andererseits kann Ihnen dieses Verhältnis auch zum Vorteil gereichen, nämlich wenn es darum geht, nicht Objekte zu kopieren, sondern nur die Zeiger auf diese – beispielsweise wenn es beim Sortieren großer Objektmengen auf Geschwindigkeit ankommt und deswegen keine ganzen Speicherblöcke mit den eigentlichen Daten sondern nur die Zeiger auf die Objektinstanzen kopiert werden sollen.

## Nothing

Mit dem Wissen des vorherigen Abschnittes können Sie sich auch erklären, wieso eine Objektvariable den Wert `Nothing` (`null` in C# - nur der Vollständigkeit halber erwähnt) aufweisen kann. Wenn Sie mit `New` eine Instanz einer Klasse erstellen, weisen Sie diese Instanz (genauer: die Adresse dieser Instanz) der Objektvariablen zu. Erst dann können Sie mithilfe der Objektvariablen auf die öffentlichen Felder, Eigenschaften und Methoden der Klasseninstanz (des Objektes) zugreifen.

Wenn Sie versuchen mit einer nicht initialisierten Objektvariablen auf Klassen-Member zuzugreifen, wird zur Laufzeit eine entsprechende Ausnahme ausgelöst. In vielen Fällen kann der Hintergrundcompiler von Visual Basic diesen Zustand voraussehen und zeigt eine entsprechende Warnmeldung an (siehe Abbildung 4.5).

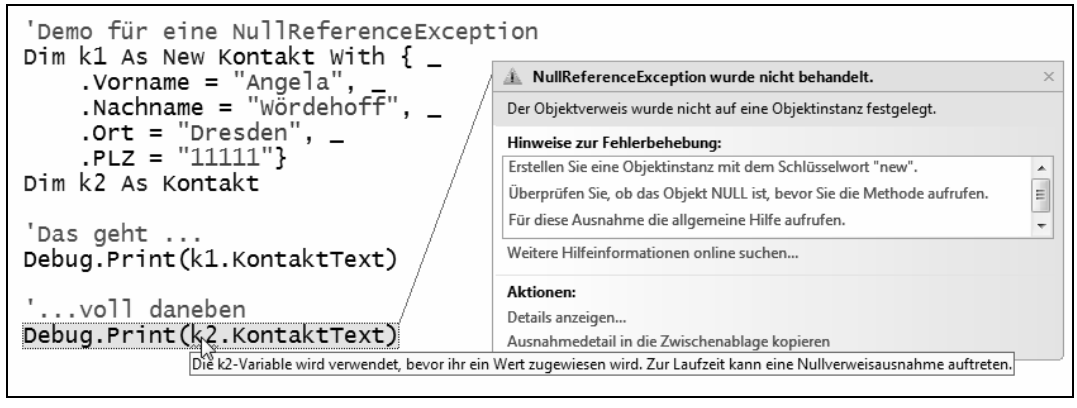



**Abbildung 4.5** Wenn Sie mit einer Objektvariablen, der keine Klasseninstanz zugeordnet ist, auf Klassenelemente (Felder, Eigenschaften, Methoden) zugreifen, wird eine *NullReferenceException* ausgeöst

# Klassen anwenden

Natürlich handelt es sich bei Klassen zwar um ganz neue Datentypen, die Sie aber genauso handhaben können, wie alle anderen Datentypen des .NET Frameworks. Und damit können einzelne Instanzen einer Klasse genauso in Arrays abgelegt werden, wie das bei ganz normalen primitiven Datentypen der Fall ist – wir sind damit beim ersten richtigen Anwendungsbeispiel für Klassen, nämlich der zweiten Version unseres Adressenbeispiels.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Mit all den Infos aus den vorherigen Abschnitten können wir das anfängliche Beispielprogramm nun so umschreiben, dass es »nur« noch mit einem Array auskommt, in dem Elemente vom gerade neu geschaffenen Typ `Kontakt` vorhanden sind, und dazu sind nicht einmal viele Änderungen notwendig:

**BEGLEITDATEIEN** Sie finden dieses Beispielprojekt unter *.\Samples\ Chapter04\MiniAdressoClass*.

Die erste betrifft das Einrichten der Datenstruktur am Anfang des Programms:

```
Module Hauptmodul

    Dim Kontakte(0 To 100) As Kontakt
```

Im Gegensatz zur ursprünglichen Version kommen wir jetzt mit nur noch einem Array aus, das Elemente enthält, die die Felder eines kompletten Kontaktes kapseln. Viel aufgeräumter sieht nun auch der Sortieralgorithmus aus (Änderungen an ihm sind fett hervorgehoben).

```
    Sub AdressenSortieren()

        Dim anzahlElemente As Integer = 101

        Dim aeussererZaehler As Integer
        Dim innererZaehler As Integer
        Dim delta As Integer

        Dim tempKontakt As Kontakt

        delta = 1

        'Größten Wert der Distanzfolge ermitteln
        Do
            delta = 3 * delta + 1
        Loop Until delta > anzahlElemente

        Do
            'Späteres Abbruchkriterium - wieder kleiner werden lassen
            delta = delta \ 3

            'Shellsort's Kernalgorithmus
            For aeussererZaehler = delta To anzahlElemente - 1
                tempKontakt = Kontakte(aeussererZaehler)

                innererZaehler = aeussererZaehler
                Do
                    If tempKontakt.Nachname >= Kontakte(innererZaehler - delta).Nachname Then Exit Do
                    Kontakte(innererZaehler) = Kontakte(innererZaehler - delta)

                    innererZaehler = innererZaehler - delta
                    If (innererZaehler <= delta) Then Exit Do
                Loop
                Kontakte(innererZaehler) = tempKontakt
            Next
        Loop Until delta = 0
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Da der neue Datentyp `Kontakt` die Daten einer kompletten Kontaktadresse kapselt, kann der Fehler aus der letzten Programmversion gar nicht mehr passieren. Der Dreieckstausch vollzieht sich jetzt zwangsläufig an einem vollständigen Kontakt. Somit ist die Gefahr ausgeschlossen, dass einzelne Felder auf der Strecke bleiben. Die fettgedruckten Zeilen im vorherigen Listing dokumentieren das »Aufräumen«.

# Klassencode

Betrachten wir nochmals die anfängliche Definition von Klassen: »Eine Klasse schafft die Strukturen für das Speichern von Daten und beinhaltet gleichzeitig Programmcode, der diese Daten reglementiert.« Bislang haben wir erst die eine Komponente einer Klasse kennen gelernt: so genannte Member-Variablen, die die Möglichkeit schaffen, bestimmte Dinge in einer Klasseninstanz zu speichern.

Der zweite Teil, der den Zugriff auf die Daten reglementieren soll, fehlt bislang. Das Klassenkonzept sieht es vor, dass datenreglementierender Code in Form von zwei Prozedurtypen in einer Klasse vorhanden sein kann: In Methoden (`Sub`, `Function`) und in Eigenschaftenprozeduren (`Property`).

## Eigenschaftenprozeduren (Property-Prozeduren)

Die bislang einzige Beispielklasse dieses Kapitels bestand aus nur vier Member-Variablen, auf die man, da sie mit dem Schlüsselwort `public` (also als öffentlich zugänglich) definiert wurden – in jeder Instanz zugreifen konnte. Das hat mit der Kapselung von Daten natürlich denkbar wenig zu tun; dieses Beispiel diente bislang lediglich dazu, verschiedene Daten zu einer logischen Dateneinheit zusammenzufassen – von wirklicher Kapselung der Daten, bei der das Rein und Raus der Daten durch irgendeine Klassenpolizei geregelt wurde, konnte man dabei allerdings noch nicht sprechen.

Jetzt stellen Sie sich vor, Sie möchten, dass die letzten drei Buchstaben des Ortes, falls er mehr als 30 Buchstaben hat, durch drei Auslassungspunkte »…« ersetzt wird.

In der prozeduralen Welt würden Sie dazu eine Funktion entwickeln, die überprüft, ob eine Zeichenfolge mehr als dreißig Zeichen aufweist. Wäre das der Fall, würden Sie sie jedes Zeichen nach dem 30. Zeichen abschneiden sowie die letzten drei Zeichen der Zeichenkette durch die drei Auslassungspunkte ersetzen lassen.

Sie würden dann nicht direkt auf die Feldvariable zugreifen, sondern, wie im folgenden Beispiel, jeden Zugriff auf das Feld *Ort* in einen Funktionsaufruf einbauen. Das könnte dann im Ergebnis folgendermaßen ausschauen:

Wir haben also zunächst unsere »AuslassungszeichenAnText«-Routine …

```
Function EllipseString(ByVal text As String, ByVal MaxLength As Integer) As String
    Dim tmpText As String

    If text.Length > MaxLength Then
        tmpText = text.Substring(0, MaxLength - 3) + "..."
    Else
        tmpText = text
    End If
    Return tmpText
End Function
```

… und diese Routine rufen Sie immer dann auf, wenn Sie auf das kritisch zu behandelnde Feld `Ort` einer Instanz der Klasse `Kontakt` zugreifen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Sub KlasseVerarbeiten()
    .
    .
    .
        Dim tmpKontakt As New Kontakt
    .
    .
    .
        tmpKontakt.Ort = EllipseString(Console.ReadLine, 30)
    End Sub
```

Es gibt aber noch eine viel elegantere und vor allen Dingen *sichere* Möglichkeit, dafür zu sorgen, dass die Member-Variable `Ort` der Kontakt-Klasse niemals eine zu lange Zeichenkette zugewiesen bekommt: So genannte *Eigenschaften-Prozeduren*.

Wenn Sie schon längere Zeit mit Visual Basic (egal, ob mit .NET oder 6.0) programmieren, haben Sie Eigenschaften natürlich längst kennen gelernt. Mithilfe von Eigenschaften können Sie in der Regel bestimmte Zustände von Objekten abfragen *und* verändern, und von außen »fühlt« sich das dann so an, also würden Sie direkt ein öffentliches Feld einer Klasse manipulieren oder abfragen. Möchten Sie beispielsweise wissen, ob die Schaltfläche eines Formulars anwählbar ist, verwenden Sie die Eigenschaft in Abfrageform, etwa wie hier:

```
If Schaltfläche.Enabled Then TuWas
```

Oder Sie legen die Eigenschaft eines Objektes fest, etwa wie mit der folgenden Zeile:

```
Schaltfläche.Enabled = false    ' Abschließen, kommt keiner mehr 'ran
```

Sie wissen nun aber aus Erfahrung, dass Sie beim Setzen einer Schaltfläche nicht nur den Zustand einer Variablen dieser Button-Instanz verändern, sondern dass das Setzen dieser Eigenschaft auch noch etwas Weiteres bewirkt – im Fallbeispiel nämlich, dass die Schaltfläche auch sichtbar auf dem Formular ausgegraut wird (oder eben wieder – beim Zuweisen von `True` – den visuellen Einschaltzustand bekommt). Der Code, der dafür sorgt, muss innerhalb der Klasse natürlich irgendwo platziert werden, und so lautet die viel interessantere Frage: Wie statten Sie Ihre eigenen Klassen mit Eigenschaften aus?

Visual Basic stellt Ihnen zu diesem Zweck, wie schon erwähnt, Eigenschaftenprozeduren zur Verfügung. Eine Eigenschaft wird in Visual Basic innerhalb einer Klasse folgendermaßen definiert:

```
Property EineEigenschaft() As Datentyp

    Get
        Return DatentypObjektvariable
    End Get

    Set(ByVal Value As Datentyp)
        DatentypObjektvariable = Value
    End Set

End Property
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wenn Sie diese Eigenschaft in einer Klasse implementieren, können Sie sie bei instanziierten Objekten dieser Klasse auf folgende Weise verwenden:

### Zuweisen von Eigenschaften

Mit der Anweisung

```
Object.EineEigenschaft = Irgendetwas
```

weisen Sie der Eigenschaft `EineEigenschaft` des Objektes einen Wert zu. Sie können im *Set*-Accessor[4] (`Set(ByVal Value as Datentyp)`) der Eigenschaftenprozedur mit `Value` auf das Objekt zugreifen, das sich in `Irgendetwas` befindet. Nur der *Set*-Teil der Eigenschaftenprozedur wird in diesem Fall ausgeführt.

### Ermitteln von Eigenschaften

Umgekehrt können Sie mit der Anweisung

```
Irgendetwas = Object.EineEigenschaft
```

den Inhalt der Eigenschaft wieder auslesen. In diesem Fall wird nur der *Get*-Accessor der Eigenschaftenprozedur ausgeführt, die das Ergebnis mit `Return` zurückliefert und dieses der Objektvariablen `Irgendetwas` zuweißt, die natürlich vom gleichen Typ, wie die Eigenschaft sein muss.

Mit diesem Wissen können wir unsere Klasse jetzt folgendermaßen umschreiben:

**BEGLEITDATEIEN** Sie finden dieses Beispielprojekt unter *.\Samples\ Chapter04\MiniAdressoClass V2*.

```
Public Class Kontakt
    Private myVorname As String
    Private myNachname As String
    Private myPLZ As String
    Private myOrt As String

    Public Property Vorname() As String
        Get
            Return myVorname
        End Get
        Set(ByVal value As String)
            myVorname = value
        End Set
    End Property

    Public Property Nachname() As String
        Get
            Return myNachname
        End Get
        Set(ByVal value As String)
            myNachname = value
```

[4] Etwa: »Zugreifer«.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        End Set
    End Property

    Public Property PLZ() As String
        Get
            Return myPLZ
        End Get
        Set(ByVal value As String)
            myPLZ = value
        End Set
    End Property

    Public Property Ort() As String
        Get
            Return myOrt
        End Get
        Set(ByVal value As String)
            myOrt = EllipseString(value, 30)
        End Set
    End Property

    Private Function EllipseString(ByVal text As String, ByVal MaxLength As Integer) As String

        Dim tmpText As String

        If text.Length > MaxLength Then
            tmpText = text.Substring(0, MaxLength - 3) + "..."
        Else
            tmpText = text
        End If
        Return tmpText
    End Function

End Class
```

Sie sehen an diesem einfachen Beispiel, wie der Schutz der eigentlichen Datenstruktur durch Eigenschaftenprozeduren vonstatten geht: Die Daten, die in einer Klasseninstanz gespeichert werden, weisen private Zugriffsmodifizierer auf (mehr zu Zugriffsmodifizierern finden Sie im Abschnitt »Zugriffsmodifizierer für Klassen, Methoden und Eigenschaften« ab Seite 67), und sie können damit nicht mehr direkt von außen, sondern nur noch durch den Klassencode selbst verändert werden. Die Member-Variablen heißen auch nicht mehr wie zuvor, sondern tragen den Zusatz bzw. das Präfix »my« als rein textlichen Hinweis darauf, dass es sich dabei um »meine« Variablen aus Klassen- bzw. Klasseninstanzsicht handelt. Dies ist eine durchaus übliche, aber nicht vorgeschriebene Konvention – Sie können die Benennung der Member-Variablen nach eigenem Belieben vornehmen; Sie sollten allerdings darauf achten, dass Sie die Member-Variablen anders als Ihre von außen zugreifbaren Eigenschaften nennen, denn das würde zu einem Kompilierungsfehler führen.

Die Eigenschaftenprozeduren schließlich heißen so, wie die Namen der ursprünglich öffentlichen Member-Variablen. Aus Entwicklersicht hat sich damit nichts an der Handhabung geändert – im Ergebnis allerdings schon, denn durch die Eigenschaftenroutinen wird nun der Zugriff auf die Member-Variablen exklusiv geregelt. In diesem kleinen Beispiel sehen Sie, dass der Ort durch dieses Verfahren niemals mehr als 30 Zeichen aufweisen wird. Bei mehr als 30 Zeichen wird durch das Einbinden der `EllipseString`-Methode die Anpas-

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

sung der Zeichenkette mit den Auslassungszeichen vorgenommen und auf den eigentlichen Datenspeicher des Orts – die Member-Variable `myOrt` – kann von außen niemand mehr zugreifen, da ihr Zugriff durch Verwendung des Zugriffsmodifizierers `private` für diese (wie auch für alle anderen Member-Variablen) geschützt ist. Genau das ist es, was eines der Hauptanliegen einer Klasse ist, nämlich *Daten zu kapseln und den Zugriff auf sie durch Klassencode zu reglementieren*.

## Öffentliche Variablen oder Eigenschaften – eine Glaubensfrage?

Jetzt haben Sie schon so viel über Eigenschaften erfahren – vielleicht fragen Sie sich, wieso man sie anstelle von einfachen öffentlichen Member-Variablen einsetzen sollte.

Solange, wie Sie Eigenschaften in einer Klasse nur benötigen, um irgendwelche Werte zu speichern, aber beim Abfragen oder Setzen dieser Werte nichts Weiteres passieren muss, wären öffentliche Variablen eigentlich ausreichend.

Die Klasse

```
Class MitEinerEigenschaft
    Public DieEigenschaft As Integer
End Class
```

erfüllt zunächst nämlich den gleichen Zweck wie die folgende Klasse,

```
Class AuchMitEinerEigenschaft

    Private myDieEigenschaft As Integer

    Public Property DieEigenschaft() As Integer
        Get
            Return myDieEigenschaft
        End Get
        Set(ByVal Value As Integer)
            myDieEigenschaft = Value
        End Set
    End Property
End Class
```

die natürlich sehr viel mehr Schreibarbeit erfordert. Prinzipiell ist das richtig. Und dennoch ist die zweite Methode der ersten Methode vorzuziehen, aus folgenden Gründen, die Ihnen teilweise schon bekannt sind:

- Eigenschaften kapseln und reglementieren den Zugriff auf Daten, wie Sie im vorangegangenen Beispiel bereits gesehen haben. Auch wenn Sie eine Eigenschaft nur komplett an eine Member-Variable durchreichen – für mögliche Erweiterungen, was Reglementierungsalgorithmen anbelangt, sind Sie auf jeden Fall schon mal vorbereitet.
- Felder lassen sich grundsätzlich nicht für die Datenbindung verwenden. Vielfach werden nicht nur Quellen wie Datenbanken, sondern auch Objekte, wie beispielsweise Auflistungen (Collections) als Datenquellen verwendet. Öffentliche Felder (also öffentliche Member-Variablen) können dabei nicht als Datenquelle dienen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Ein weiteres wichtigeres Argument ist das Ersetzen von Eigenschaften durch die so genannte Polymorphie beim Vererben von Klassen, auf das wir im Rahmen dieses Crashkurses leider nicht eingehen können. Dennoch soviel: Eine einmal als öffentlich deklarierte Variable bleibt für alle Zeiten öffentlich. Sie können in vererbten Klassen keine zusätzliche Steuerung hinzufügen. Haben Sie hingegen Ihre Daten nur durch Eigenschaftenprozeduren nach außen offen gelegt, können Sie zu einem späteren Zeitpunkt noch zusätzliche Regeln (Bereichsabfragen, Fehler abfangen) hinzufügen. Sie brauchen dazu die ursprüngliche Klasse kein bisschen zu verändern.

Eigenschaften sind also öffentlichen Feldern in jedem Fall vorzuziehen. Damit das Erstellen von Eigenschaften nicht zu viel Tipparbeit verschlingt, gibt es in Form der Visual Basic-Codeausschnittsbibliothek eine Eingabehilfe, die der folgende Kasten beschreibt.

### Zeit sparen beim Erstellen von Eigenschaftenprozeduren mit Code-Ausschnitten

An der Beispielklasse des vorherigen Abschnittes lässt sich eindrucksvoll aufzeigen, dass das Verwalten, Regeln und auch Schützen der Member-Variablen einer Klasse durch entsprechende Eigenschaftenprozeduren durchaus sinnvoll ist, Ihnen jedoch auch eine Menge Tipparbeit abverlangt.

Mithilfe von Codeausschnitten können Sie sich gerade beim »Properties kloppen«, wie es umgangssprachlich in Entwicklerkreisen genannt wird, eine Menge an Zeit sparen. Möchten Sie eine neue Eigenschaft in Ihrer Klasse einführen, die auf einer Member-Variablen basiert, verfahren Sie am besten wie folgt:

- Schreiben Sie die Zeichenfolge **Pro** an die Stelle, an der Sie die neue Eigenschaftenprozedur erstellen wollen. Sie sehen anschließend einen Dialog, etwa wie auch in Abbildung 4.6 zu sehen.
- Drücken Sie zweimal ⇥, um den Property-Codeausschnitt einzufügen. Sie sehen anschließend ein Szenario, etwa wie in Abbildung 4.6 zu sehen.

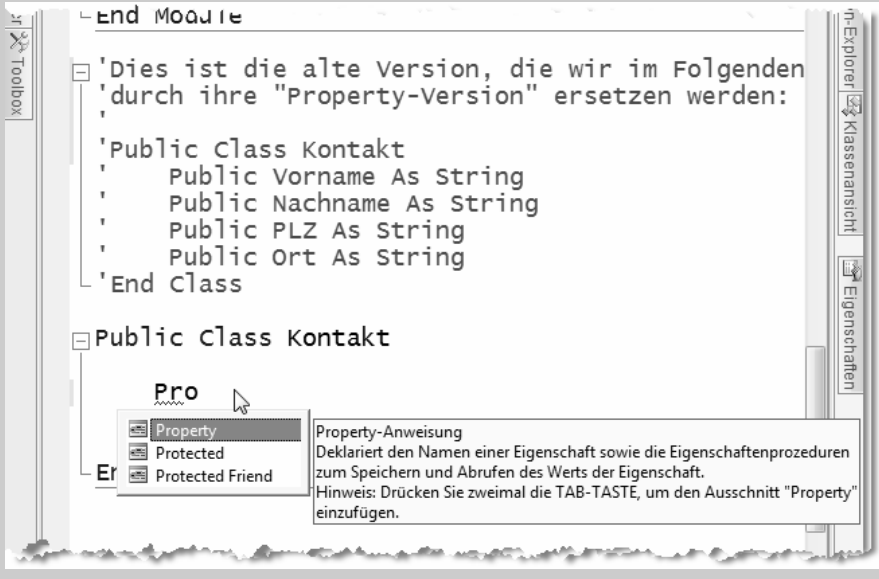



**Abbildung 4.6** Um eine Eigenschaftenprozedur durch Codeausschnitte einzufügen, schreiben Sie die Zeichenfolge »Pro«. Drücken Sie, wie im Hinweis der IntelliSense-Liste zu sehen, zweimal ⇥, um den Property-Codeausschnitt einzufügen, ...

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Editieren Sie anschließend die Member-Variable, die zunächst mit dem Vorgabenamen `newPropertyValue` eingefügt wurde, in den Namen, den Sie für die Member-Variable vorsehen.

```
Public Class Kontakt


    Private newPropertyValue As String
    Public Property NewProperty() As String
        Get
            Return newPropertyValue
        End Get
        Set(ByVal value As String)
            newPropertyValue = value
        End Set
    End Property


End Class
```

**Abbildung 4.7** ... um dann anschließend die selektierte Member-Variable zu editieren und mit ⇆ den passenden Datentyp hinzuzufügen ...

- Sobald Sie ⇆ gedrückt haben, ändern sich alle Referenzen der Member-Variablen im Codeblock in den neu benannten Namen. Gleichzeitig gelangen Sie zum nächsten editierbaren Feld des Codeausschnitts, dem Typen der Eigenschaftenprozedur. Ändern Sie diesen, und drücken Sie abermals die Taste ⇆.

```
Public Class Kontakt


    Private myVorname As String
    Public Property NewProperty() As String
        Get
            Return myVorname
        End Get
        Set(ByVal value As String)
            myVorname = value
        End Set
    End Property


End Class
```

**Abbildung 4.8** ... den Sie anschließend im Bedarfsfall ebenfalls editieren, um schließlich mit einem weiteren ⇆ das Erstellen der Eigenschaftenprozedur mit dem Editieren des eigentlichen Eigenschaftennamens abzuschließen.

- Sobald Sie ⇆ gedrückt haben, ändern sich alle Referenzen der Member-Variablen im Codeblock in den neuen Namen. Gleichzeitig gelangen Sie zum nächsten editierbaren Feld des Codeausschnitts, dem Typen der Eigenschaftenprozedur. ▶

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Ändern Sie diesen, und drücken Sie abermals die Taste ⇆. Damit ändern Sie die Typen sowohl für die Eigenschaftenprozedur (und deren `value`-Variablen mit dem im `Set`-Accessor der Zuweisungswert an die Eigenschaft übergeben wird) als auch für die korrelierende Member-Variable, die in der Eigenschaftenprozedur gesetzt wird.
- Schließlich ändern Sie noch den Namen der Eigenschaft. Damit sind mit der Definition der Eigenschaftenroutine fertig.

Im Ergebnis stehen alle Eigenschaften und deren entsprechende Member-Variablen jeweils als Pärchen zusammen. Viele Entwickler mögen das nicht in dieser Form; sie bevorzugen, dass Member-Variablen am Anfang der Klassencodedatei definiert werden – Sie können die Member-Variablen natürlich anordnen, wie Sie möchten; das tut der Funktionalität der Klasse keinen Abbruch.

## Klassenmethoden mit Sub und Function

Klassencode wird nicht nur in Eigenschaftenprozeduren ausgeführt. Auch Methoden, die in Visual Basic durch `Sub` oder `Function` definiert werden, gehören dazu. Methoden nicht statischer Natur (solche also, die nicht, wie in »Statische Methoden« ab Seite 71 beschrieben) sind dabei Methoden, die mit den Member-Variablen einer Klasse arbeiten sollten.

Ein Beispiel dafür ist das folgende: Eine Funktion wie beispielsweise `KontaktText` könnte für das Zurückliefern der textrepräsentativen Ausgabe der Klasse sorgen – und die entsprechenden Routinen dafür würden dann folgendermaßen aussehen:

```
Public Function KontaktText() As String

    Return Me.Nachname & ", " & Me.Vorname & ", " & _
           Me.PLZ & ", " & Me.Ort

End Function
```

Diese Member-Methode der Klasse bedient sich direkt der Instanzvariablen, und gibt sie verkettet als einen zusammenhängenden String aus. Das bedeutet, dass wir entsprechende Änderungen auch bei der Ausgaberoutine im Modul machen können, die die einzelnen Instanzen verwendet:

```
Sub AdressenAusgeben(ByVal von As Integer, ByVal bis As Integer)

    For c As Integer = von To bis
        Console.WriteLine(c.ToString("000") & ": " & Kontakte(c).KontaktText())
    Next

End Sub
```

Die Methode `KontaktText` wird hier für jede auszugebenden Kontakt-Instanz aufgerufen und liefert natürlich auch für jede Instanz ein unterschiedliches Ergebnis – hieran sieht man deutlich, dass es sich um eine Member-Methode handelt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Zugriffsmodifizierer für Klassen, Methoden und Eigenschaften

Zugriffsmodifizierer sind Schlüsselworte, mit denen Sie in Visual Basic bestimmen können, von wo aus der Zugriff auf ein Element gestattet ist und von wo aus nicht. Die Zugriffsmodifizierer `Private` und `Public` haben Sie schon kennen gelernt. Sie bestimmen, ob auf ein Element nur innerhalb eines bestimmten Gültigkeitsbereiches zugegriffen werden kann (`Private`) oder von überall aus (`Public`). Welche weiteren Zugriffsmodifizierer es für Objekte, Klassen und Funktionen/Eigenschaften gibt, zeigen die folgenden Tabellen:

## Zugriffsmodifizierer bei Klassen

**HINWEIS** Wenn nichts anderes gesagt wird, werden Klassen standardmäßig als `Friend` deklariert.

| Zugriffsmodifizierer | CTS-Bezeichnung | Beschreibung |
|---|---|---|
| Private | Private | Als Private können Klassen nur dann definiert werden, wenn sie geschachtelt in einer anderen Klasse definiert sind. Beispiel:<br>`Public Class A`<br>`  Private Class B`<br>`  End Class`<br>`End Class`<br><br>`Public Class C`<br>`  'Zugriff verweigert, Class B ist Private!`<br>`  Dim b as A.B`<br>`End Class` |
| Public | Public | Sie können auf die Klasse uneingeschränkt von außen zugreifen, auch aus anderen Assemblies heraus. |
| Friend | Assembly | Sie können innerhalb der Assembly auf die Klasse zugreifen, aber nicht aus einer anderen Assembly heraus. |
| Protected | Family | Es gilt das für Private Gesagte. Zusätzlich gilt: Auch aus der Klasse abgeleitete Klassen können auf die mit Protected gekennzeichneten und geschachtelten »inneren« Klassen zugreifen. |
| Protected Friend | FamilyOrAssembly | Der Zugriff auf die geschachtelte Klasse ist in abgeleiteten und von Klassen der gleichen Assembly aus möglich. |

**Tabelle 4.1** Mögliche Zugriffsmodifizierer für Klassen in Visual Basic .NET

## Zugriffsmodifizierer bei Prozeduren (Subs, Functions, Properties)

**HINWEIS** Wenn nichts anderes gesagt wird, werden `Subs`, `Functions` und `Properties` standardmäßig als `Public` deklariert. Sie sollten gerade bei diesen Elementen aber auf jeden Fall einen Zugriffsmodifizierer verwenden, damit beim Blättern durch den Quellcode schnell deutlich wird, welchen Zugriffsmodus ein Element innehat.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Zugriffsmodifizierer | CTS-Bezeichnung | Beschreibung |
|---|---|---|
| Private | Private | Nur innerhalb einer Klasse kann auf die Prozedur zugegriffen werden. |
| Public | Public | Sie können auf die Prozedur uneingeschränkt von außen zugreifen, auch aus anderen Assemblies heraus. |
| Friend | Assembly | Sie können innerhalb der Assembly auf die Prozedur zugreifen, aber nicht aus einer anderen Assembly heraus. |
| Protected | Family | Nur innerhalb der Klasse oder einer abgeleiteten Klasse kann auf die Prozedur zugegriffen werden. |
| Protected Friend | FamilyOrAssembly | Nur innerhalb der Klasse, einer abgeleiteten Klasse oder innerhalb der Assembly kann auf die Prozedur zugegriffen werden. |

**Tabelle 4.2** Mögliche Zugriffsmodifizierer für Prozeduren in Visual Basic .NET

## Zugriffsmodifizierer bei Variablen

Variablen, die auf Klassenebene nur mit `Dim` deklariert werden, gelten als `Private`, also nur von der Klasse aus zugreifbar. Variablen, die innerhalb eines Codeblocks oder auf Prozedurebene deklariert werden, gelten nur für den entsprechenden Codeblock. Innerhalb eines Codeblocks können Sie nur die `Dim`-Anweisung und keine anderen Zugriffsmodifizierer verwenden. Auf Prozedurebene können Sie eine Variable zusätzlich als `Static` deklarieren. Mehr über den `Static`-Modifizierer erfahren Sie im Abschnitt »Statische Komponenten«.

| Zugriffsmodifizierer | CTS-Bezeichnung | Beschreibung |
|---|---|---|
| Private | Private | Nur innerhalb einer Klasse kann auf die Variable zugegriffen werden. Variablen innerhalb von Prozeduren oder noch kleineren Gültigkeitsbereichen können nicht explizit als Private definiert werden, sind es aber standardmäßig. |
| Public | Public | Von außen kann auf die Klassenvariable uneingeschränkt zugegriffen werden. Sie sollten Variablen aber bestenfalls als Protected deklarieren und sie nur durch Eigenschaften nach außen offen legen. Mehr zu diesem Thema erfahren Sie im Abschnitt »Öffentliche Variablen oder Eigenschaften – eine Glaubensfrage?« auf Seite 63. |
| Friend | Assembly | Sie können innerhalb der Assembly auf die Klassenvariable zugreifen, aber nicht aus einer anderen Assembly heraus. Es gilt das für Public Gesagte. |
| Protected | Family | Nur innerhalb derselben oder einer abgeleiteten Klasse kann auf die Variable zugegriffen werden. Variablen sollten in Klassen, bei denen Sie davon ausgehen, dass sie später des Öfteren vererbt werden, als Protected definiert werden, damit abgeleitete Klassen ebenfalls darauf zugreifen können. |
| Protected Friend | FamilyOrAssembly | Nur innerhalb der Klasse, einer abgeleiteten Klasse oder innerhalb der Assembly kann auf die Klassenvariable zugegriffen werden. Von dieser Kombination sollten Sie absehen. |
| Static | - - - | Sonderfall in Visual Basic. Lesen Sie dazu bitte die Ausführungen im Abschnitt »Statische Komponenten«. |

**Tabelle 4.3** Mögliche Zugriffsmodifizierer für Variablen in Visual Basic .NET

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Diese Tabellen sollen Ihnen kompakt und auf einen Blick die Zugriffsmodifizierer verdeutlichen. Die CTS-Bezeichnungen der Zugriffsmodifizierer benötigen Sie, wenn Sie sich den IML-Code einer Klasse anschauen, um zu erkennen, welchen Zugriffsmodus beispielsweise eine Methode hat.

## Unterschiedliche Zugriffsmodifizierer für Eigenschaften-Accessors

Seit Visual Studio 2005 ist es möglich, dass die `Get`- und `Set`-Accessors unterschiedliche Zugriffsmodifizierer aufweisen. So haben Sie beispielsweise die Möglichkeit, zu bestimmen, dass zwar eine bestimmte Eigenschaft von jedem Ort aus gelesen werden kann (`Public`), aber Eigenschaften nur von derselben Klasse aus geschrieben werden dürfen (`Private`). Im Code würde eine solche Eigenschaft folgendermaßen ausschauen:

```
Module Module1

    Sub Main()
        Dim locEigenschaftenTest As New EigenschaftenTest("Text für Eigenschaft")

        'Das Auslesen der Eigenschaft ist problemlos möglich
        Console.WriteLine("Eigenschaft enthält: " & locEigenschaftenTest.EineEigenschaft)

        'FEHLER: Der Set-Zugriffsmodifizierer verbietet aber das Schreiben,
        'weil er 'private' ist.
        locEigenschaftenTest.EineEigenschaft = "Neuer Text"
    End Sub

End Module

Public Class EigenschaftenTest

    Private myEineEigenschaft As String

    Sub New(ByVal textFürEigenschaft As String)
        'Ist erlaubt - die Klasse darf die
        'Eigenschaft beschreiben!
        EineEigenschaft = textFürEigenschaft
    End Sub

    Public Property EineEigenschaft() As String
        Get
            Return myEineEigenschaft
        End Get
        Private Set(ByVal value As String)
            myEineEigenschaft = value
        End Set
    End Property

End Class
```

Dieses kleine Beispiel besteht aus zwei Einheiten – einem Modul und einer Klasse. Die Klasse `EigenschaftenTest` enthält einen parametrisierten Konstruktor sowie eine Eigenschaft, die aber Accessoren mit unterschiedlichen Zugriffsmodifizierern hat.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Innerhalb des Konstruktors (`Sub New`) ist der Schreibzugriff auf die Eigenschaft problemlos möglich, da aus der Klasse selbst heraus auch auf Elemente zugegriffen werden kann, deren Zugriff mit `Private` eingeschränkt wurde. Innerhalb des Moduls funktioniert der Zugriff allerdings nicht mehr, da sich das Programm zum Zeitpunkt des Zugriffs außerhalb der Klasse befindet, und wegen `Private` ist von diesem Punkt aus kein Herankommen an die Eigenschaft möglich.

Doch wozu brauchen Sie unterschiedliche Zugriffsmodifizierer in Eigenschaften während der Entwicklung Ihrer Software? Denken Sie beispielsweise an das zur Verfügung stellen von Assemblies (Klassenbibliotheken) für anderer Entwickler: Sie möchten beispielsweise in der Lage sein, bestimmte Eigenschaften Ihrer Klassen von jedem Punkt Ihrer Assembly aus zu manipulieren; Sie möchten aber gleichzeitig verhindern, dass ein Entwickler, der Ihre Assembly verwendet, dazu auch in der Lage ist. In diesem Fall definieren Sie den `Get`-Accessor als `Public` – das Lesen der Eigenschaft stellt schließlich kein Risiko dar und kann von überall aus erfolgen – aber den `Set`-Accessor als `Friend`. Vom gesamten Programmcode, der sich in Ihrer Klassenbibliothek befindet, können Sie dann die Eigenschaft der betreffenden Klasse manipulieren; Entwickler, die die Assembly einbinden, also von außerhalb Ihrer Assembly zugreifen, können Sie aber nicht mehr direkt manipulieren. Solcherlei Eigenschaften sind dann wichtig, wenn es sich bei ihnen um Quasi-Konstanten handeln soll. Ihre Assembly definiert die Eigenschaft wann und wie sie will auf Grund bestimmter Zustände. Die Assemblies, die sie konsumieren, müssen aber mit dieser Einstellung leben. Klassen, die beispielsweise Einstellungen aus der Windows-Registry widerspiegeln, können davon Gebrauch machen. Denkbar wäre auch, eine Verbindungszeichenfolge zu einem SQL Server auf diese Weise freizulegen – Sie hätten zwar aus Ihrer Assembly heraus Manipulationsspielraum für die Verbindungszeichenfolge; eine Assembly könnte aber immer nur die Verbindungszeichenfolge mit der Eigenschaft auslesen, die Sie innerhalb Ihrer Assembly vorgeben.

# Vererbung, Polymorphie, Abstrakte Klassen und Schnittstellen

Um es ganz kurz zu machen: Die Schlagworte, die Sie in dieser Überschrift finden, würden den Rahmen dieses Buches sprengen. Sie finden in diesem Kapitel, wie bereits angekündigt, ausschließlich neue Features und solche Dinge erklärt, die Sie als Grundlagen für weitere neue Features, insbesondere für *LINQ to Objects* benötigen.

Falls Sie an weiterem, ausführlichem Lehrmaterial zur objektorientierten Programmierung interessiert sind, empfehle ich Ihnen das Buch *Visual Basic 2005 – Das Entwicklerbuch*, das Sie auf der ActiveDevelop-Internetseite *http://www.activedevelop.de* inklusive aller Beispiele kostenlos herunterladen können.

Kurz zusammengefasst handelt es sich bei den einzelnen Themen um Folgendes:

- **Klassenvererbung** erfolgt durch das `Inherits`-Schlüsselwort. Damit haben Sie die Möglichkeit, vorhandene Klassen – ganz gleich, ob das Ihre eigenen oder Klassen aus bereits vorhandenen Klassenbibliotheken (und damit auch des .NET Frameworks) sind – zu erweitern.
- **Polymorphie** kann durch das Überschreiben von besonders gekennzeichneten Eigenschaften und Methoden realisiert werden. Dadurch haben Sie die Möglichkeit, Verhaltensweisen von Klassen beim Vererben nicht nur zu übernehmen, sondern auch gezielt abzuändern oder zu erweitern.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- **Abstrakte Klassen** sind Klassen, die nur zur Implementierung von Grundfunktionalitäten für das Vererben in andere Klassen vorgesehen sind – etwa wie eine Dokumentenvorlage für weitere Word-Dokumente. Abstrakte Klassen können selbst nicht instanziiert werden.
- **Schnittstellen (Interfaces)** schreiben vor, welche Elemente (Methoden, Eigenschaften, Ereignisse, etc.) eine Klasse zu implementieren hat. Sie dienen ebenfalls dazu, eine Klasse für bestimmte Aufgaben oder Möglichkeiten zu kennzeichnen.

# Statische Komponenten

Bislang haben Sie im Rahmen von Klassen nur Member-Variablen (Felder), Member-Eigenschaften und Member-Methoden kennengelernt, also solche Methoden, die zum sauberen Funktionieren auf die Instanzvariablen zugreifen mussten, wofür sie in jedem Fall eine Instanz der Klasse benötigten – anderenfalls hätten Sie eine `NullReferenceException` ausgelöst, wie im Abschnitt »Nothing« ab Seite 57 zu lesen.

Sie haben aber auch die Möglichkeit, statische Komponenten zu erstellen, die eben nicht auf Member-Variablen einer Klasse zurückgreifen, und damit auch ohne Instanziierung aufrufbar sind. Die folgenden Abschnitte zeigen, wie's geht:

**Nicht durch die Schlüsselwörter Static und Shared in Visual Basic verwirren lassen!**

`Static` im Gegensatz zu `Shared` bewirkt, dass eine Variable, die nur innerhalb einer Prozedur (`Sub` oder `Function`) verwendet wird, ihren Inhalt auch nach Verlassen der Unterroutine nicht verliert. Dennoch können Sie auf diese Variable nur innerhalb der Unterroutine zugreifen, in der sie definiert wird. Im Prinzip ist eine mit `Static` definierte Variable eine, die als `Shared`-Member für die Klasse definiert und mit einem internen Attribut versehen wurde, das die Verwendung auf den Gültigkeitsbereich reglementiert, in dem sie deklariert wurde.

Vorhanden sind `Static`-Variablen übrigens nur aus Gründen der Kompatibilität zum alten Visual Basic 6.0 (und vorherigen Versionen).

## Statische Methoden

Statische Methoden sind Methoden (`Sub`, `Function`) die mit dem Schlüsselwort `Shared` (nicht `Static`! – Siehe vorheriger Kasten!) gekennzeichnet und damit direkt über die Klasse und nicht über die Klasseninstanz definiert sind. Das bedeutet: Anders als bei Instanzmethoden können Sie eine statische Methode direkt über den Klassennamen aufrufen und brauchen eben keine Instanz dafür.

Ein gutes Beispiel für eine statische Methode ist die `Parse`-Methode aller numerischen Datentypen. Möchten Sie beispielsweise eine Zeichenfolge in einen `Integer`-Wert konvertieren, bedienen Sie sich genau dieser Funktion:

```
Dim intVar as Integer = Integer.Parse("1234")
```

Hier sehen Sie, dass Sie sich für den Funktionsaufruf keiner Instanzmethode bedienen, da auch keine Member-Variablen der Instanz für den Konvertierungsprozess benötigt werden. Im umgekehrten Fall ist das anders. Möchten Sie einen `Integer`-Wert in eine Zeichenkette umwandeln, bedienen Sie sich der Instanzmethode `ToString`, die natürlich auf die interne Member-Variable des `Integer`-Datentyps zurückgreift:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Dim intVar as Integer = 5
Debug.Print(intVar.ToString())
```

**HINWEIS** Im Grunde genommen müsste die `EllipseString`-Methode unseres Klassenbeispiels ebenfalls statisch und damit mit `Shared` definiert werden, da sie ebenfalls nur einen Algorithmus kapselt, der auf *keine* Member-Variable der Klasse direkt zugreift. Konsequenterweise sollte diese Methode dann auch als öffentlich definiert werden, sodass ihre korrekte Signatur folgendermaßen aussehen sollte:

```
Public Shared Function EllipseString(ByVal text As String, ByVal MaxLength As Integer) As String
.
.
.
End Function
```

Die richtige Funktionsweise dieser Routine wäre dann:

```
Dim s as String = Kontakt.EllipseString("Dies ist die abzuschneidende Zeichenkette", 20)
```

Sollten Sie eine Objektvariable auf eine instanziierte Kontakt-Instanz haben, könnten Sie rein theoretisch auch über diese Objektvariable auf die statische Methode zugreifen. Doch Sie sollten das nicht machen, da es für Verwirrung sorgt. Die Verwendung von Objektvariablen impliziert immer, dass Sie etwas mit der Klasseninstanz (also dem Dateninhalt der Klasse) anstellen wollen, was Sie in diesem Fall gar nicht machen, da, wie gerade gesagt, statische Methoden keine Member-Variablen nutzen (und auch gar nicht nutzen können!). Dieser Meinung ist auch der Background-Compiler von Visual Basic, denn einen Versuch, eine statische Methode über eine Objektvariable aufzurufen, quittiert er, wie in Abbildung 4.9 zu sehen, mit einer Warnung.

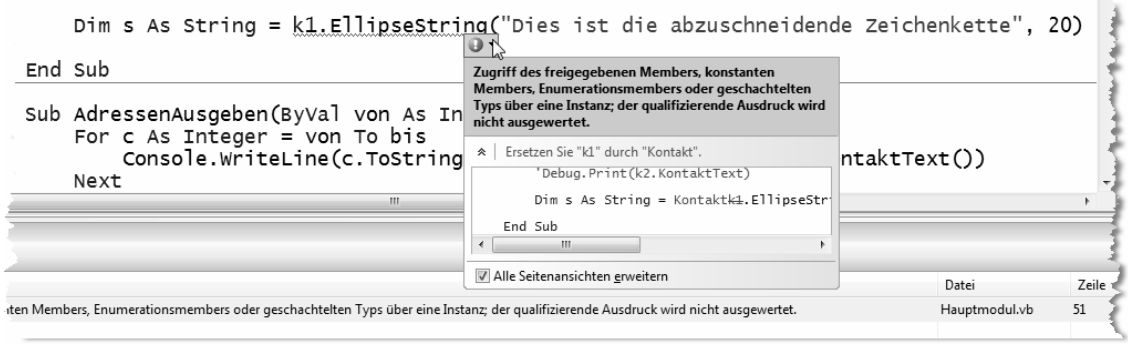



**Abbildung 4.9** Einen Versuch, eine statische Methode über eine Objektvariable aufzurufen, quittiert der Background-Compiler von Visual Basic mit einer Warnung. Die Fehlerkorrekturoptionen helfen Ihnen übrigens beim Berichtigen.

## Statische Eigenschaften

Visual Basic sieht auch statische Eigenschaften vor. Genau wie bei statischen Funktionen sind statische Eigenschaften direkt über die Klasse und nicht über die Klasseninstanz definiert. Das bedeutet, dass statische Eigenschaften Funktionalitäten bereitstellen sollten, die für alle Objekte dieser Klasse gelten und nicht für eine bestimmte Objektinstanz.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Das beste Beispiel für eine statische Eigenschaft ist die `Now`-Eigenschaft von `DateTime`. Sie liefert die aktuelle Uhrzeit zurück und ist natürlich von den Eigenschaften einzelner `DateTime`-Instanzen völlig unabhängig. Statische Eigenschaften werden, ebenfalls wie statische Funktionen, mit dem `Shared`-Schlüsselwort deklariert. Ein Beispiel für eine statische Eigenschaft finden Sie im folgenden Abschnitt.

## Module in Visual Basic – automatisch statische Klassen erstellen

Module gibt es bei allen bislang existierenden .NET-Programmiersprachen nur in Visual Basic. Und auch hier ist ein Modul nichts weiter als eine Mogelpackung, denn das, was als Modul bezeichnet wird, ist im Grunde genommen nichts anderes als eine abstrakte Klasse (eine, die also nicht instanziierbar ist) mit ausschließlich statischen Methoden, statischen Eigenschaften und statischen öffentlichen Feldern.

Halten wir fest:

- Ein Modul ist nicht instanziierbar. Eine abstrakte Klasse auch nicht.
- Ein Modul kann keine überschreibbaren Prozeduren zur Verfügung stellen. Die statischen Prozeduren einer Klasse können das auch nicht.
- Ein Modul kann nur Prozeduren zur Verfügung stellen, auf die aber nur ohne Instanzobjekt direkt zugegriffen werden kann. Das gleiche gilt für die statischen Prozeduren einer abstrakten Basisklasse.

Es gibt aber auch feine Unterschiede: So können Module beispielsweise keine Schnittstellen implementieren; das können zwar abstrakte Klassen, aber Sie können keine statischen Schnittstellenelemente definieren. Insofern ist dieser scheinbare Unterschied in Wirklichkeit gar keiner. Ein Modul kann auch nur auf oberster Ebene definiert und nicht in einem anderen geschachtelt werden.

Module setzen Sie bei der OOP vorschlagsweise so wenig wie möglich ein, denn sie widersprechen dem Anspruch von .NET, möglichst wieder verwendbaren Code zu schaffen.

Hier im Buch finden Sie aus diesem Grund Module nur, wenn es um »Quick-And-Dirty«-Projekte geht, bei denen beispielsweise eine Konsolen-Anwendung Tests durchzuführen hat oder »mal eben« etwas demonstrieren soll, genauso, wie Sie es in den vergangenen Kapiteln bereits erlebt haben.

# Delegaten und Lambda-Ausdrücke

Lambda-Ausdrücke sind quasi anonyme Funktionen, die Ausdrücke und Anweisungen enthalten und für die Erstellung von Delegaten oder Ausdrucksbaumstrukturen verwendet werden können.

Gerade wieder mit Schwerpunkt auf LINQ ist es oft notwendig, an bestimmten Stellen Delegaten – also Funktionszeiger – einzusetzen, die aber ausschließlich bei einem konkreten Aufruf und mit einer Minimalausstattung an Code zurechtkommen.

Erinnern wir uns: Delegaten sind Typen, die Adressen speichern – ähnlich wie Objektvariablen Zeiger auf Instanzen von Klassen. Allerdings speichern Delegaten keine Speicheradressen auf Objektdaten im Managed Heap, sondern die Speicheradressen von Methoden, die dann, wie eine `Sub` oder `Function` selbst, über die Delegaten aufgerufen werden können. Das praktische daran: Eine Delegatenvariable kann mal auf die eine oder mal auf die andere Methode zeigen – je nachdem, welche Methode eben im entsprechenden Kontext Sinn machen. Wichtig nur: Die Methoden müssen dabei die gleiche *Signatur* besitzen, also die gleichen Typen von Parametern in der gleichen Reihenfolge übernehmen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Umgang mit Delegaten am Beispiel

Schauen wir uns eine weitere Version unseres bisherigen Beispiels an, zunächst um ein Gefühl für die Verwendung von Delegaten zu bekommen. Wir bauen dessen Sortierroutine dort so um, dass der Sortierkriteriumsausdruck, der bislang fest im Programm mit dem Nachnamen des Kontakts verdrahtet war, durch einen Delegaten ersetzt wird. Der Delegat wiederum erwartet zwei Parameter, nämlich die Kontakte, die miteinander verglichen werden sollen. Den Rückgabewert liefert er vom Typ `Integer` zurück, mit folgendem Hintergrund. Ergibt das Funktionsergebnis 0, sind beide Kontakte gleich, bei 1, ist der erste größer, bei -1 ist der erste kleiner. Was letzten Endes verglichen wird, regelt dann eine Prozedur, auf die die Delegatenvariable zeigen soll. Die Änderungen dazu sehen zunächst einmal folgendermaßen aus:

```
Public Delegate Function ComparerDelegate(ByVal k1 As Kontakt, ByVal k2 As Kontakt) As Integer

Sub AdressenSortieren(ByVal cDel As ComparerDelegate)

    Dim anzahlElemente As Integer = 101

    Dim aeussererZaehler As Integer
    Dim innererZaehler As Integer
    Dim delta As Integer

    Dim tempKontakt As Kontakt

    delta = 1

    'Größten Wert der Distanzfolge ermitteln
    Do
        delta = 3 * delta + 1
    Loop Until delta > anzahlElemente

    Do
        'Ist Abbruchkriterium - deswegen herunterzählen
        delta \= 3

        'Shellsort's Kernalgorithmus
        For aeussererZaehler = delta To anzahlElemente - 1
            tempKontakt = Kontakte(aeussererZaehler)

            innererZaehler = aeussererZaehler
            Do
                If cDel.Invoke(tempKontakt, Kontakte(innererZaehler - delta)) = 1 OrElse _
                    cDel.Invoke(tempKontakt, Kontakte(innererZaehler - delta)) = 0 Then Exit Do
                Kontakte(innererZaehler) = Kontakte(innererZaehler - delta)

                innererZaehler = innererZaehler - delta
                If (innererZaehler <= delta) Then Exit Do
            Loop
            Kontakte(innererZaehler) = tempKontakt
        Next
    Loop Until delta = 0
End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

An den zuletzt in Fettschrift formatierten Zeilen dieses Listings können Sie erkennen, wie der eigentliche Aufruf einer Methode erfolgt, die durch eine Delegatenvariable referenziert wird – nämlich mit der `Invoke`-Methode, der genau die Parameter übergeben werden, die sie normalerweise der Methode übergeben würde.

Und jetzt schauen wir uns an, wie wir uns diesen Umbau für eine flexiblere Nutzung der Sortierroutine zu Nutze machen können:

```
    Sub Main()
.
.
.
        'Zufallsadressen generieren
        Console.Write("Zufallsadressen werden generiert ... ")
        ZufallsAdressenGenerieren()
        Console.WriteLine("fertig!")

        'Die ersten 10 Zufallsadressen ausgeben
        AdressenAusgeben(0, 10)

        'Die Adressen nach Nachnamen sortieren
        Console.WriteLine()
        Console.Write("Adressen werden nach Nachnamen sortiert ... ")
        Dim compDelegate As ComparerDelegate = AddressOf KontaktNachnamenVergleich
        AdressenSortieren(compDelegate)
        Console.WriteLine("fertig!")
        Console.WriteLine()

        'Die ersten 10 Zufallsadressen ausgeben
        AdressenAusgeben(0, 10)

        'Die Adressen nach Ortsnamen sortieren
        Console.WriteLine()
        Console.Write("Adressen werden nach Ortsnamen sortiert ... ")
        compDelegate = AddressOf KontaktOrtVergleich
        AdressenSortieren(compDelegate)
        Console.WriteLine("fertig!")
        Console.WriteLine()
.
.
.
    End Sub
```

Hier werden jetzt an zwei Stellen der Sortierroutine die Vergleichsdelegaten übergeben, mit denen dann die eigentliche Sortierung durchgeführt wird. Diese beiden Methoden schauen folgendermaßen aus:

```
    Function KontaktNachnamenVergleich(ByVal k1 As Kontakt, ByVal k2 As Kontakt) As Integer
        Return String.Compare(k1.Nachname, k2.Nachname)
    End Function

    Function KontaktOrtVergleich(ByVal k1 As Kontakt, ByVal k2 As Kontakt) As Integer
        Return String.Compare(k1.Ort, k2.Ort)
    End Function
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Soweit, so gut. Und was passiert jetzt genau, wenn das Programm gestartet wird? Die folgende Schritt-für-Schritt-Erklärung macht es deutlich:

1. Nachdem die Zufallsadressen erstellt wurden, definiert die Anwendung die Delegatenvariable `compDelegate` und weist dieser gleichzeitig die Adresse der `KontaktNachnamenVergleich`-Methode zu.
2. Sie ruft die Methode `AdressenSortieren` auf und übergibt ihr gleichzeitig die Delegatenvariable.
3. `AdresseSortieren` wiederum führt `Invoke` auf die Delegatenvariable aus und ruft dabei für den Elementvergleich die Methode `KontaktNachnamenVergleich` auf. Die Kontaktliste wird damit nach Nachnamen sortiert.
4. Wieder zurück im Hauptmodul wird der Delegatenvariable `compDelegate` nach der Sortierung die Adresse der `KontaktOrtVergleich`-Methode zugewiesen.
5. Die Anwendung ruft abermals die Methode `AdressenSortieren` auf und übergibt ihr gleichzeitig die Delegatenvariable, die jetzt allerdings auf eine andere Methode als zuvor zeigt.
6. `AdresseSortieren` führt `Invoke` auf die Delegatenvariable aus und ruft aber dieses Mal dabei für den Elementvergleich die Methode `KontaktOrtVergleich` auf. Die Kontaktliste wird damit beim zweiten Durchlauf nach dem Ortsnamen sortiert.

## Lambda-Ausdrücke

Bei einem Lambda-Ausdruck handelt es sich um eine quasi namenlose Funktion, die ein einzelnes Ergebnis zurückliefert. Lambda-Ausdrücke können an allen Stellen verwendet werden, an denen ein Delegattyp gültig ist.

Das folgende Beispiel stellt einen Lambda-Ausdruck dar, der das ihm übergebene Argument um eins erhöht und den Wert als Funktionsergebnis zurückliefert.

```
Function (num As Integer) num + 1
```

So für sich steht dieser Lambda-Ausdruck recht »lose im Raum«. Zu seiner wiederholten Verwendung kann in vereinfachter Form durch lokalen Typenrückschluss (siehe Kapitel 3) eine Delegatenvariable definiert werden, die die Adresse dieses Lambda-Ausdrucks aufnimmt:

```
Dim add1 = Function(num As Integer) num + 1
```

Im Weiteren können Sie dann für den Aufruf diese Variable verwenden, wobei Sie den Wert in gewohnter Weise als Parameter übergeben.

```
'Die folgende Codezeile bewirkt, dass 6 ausgegeben wird:
Console.WriteLine(add1(5))
```

Alternativ kann die Funktion gleichzeitig deklariert und ausgeführt werden.

```
Console.WriteLine((Function(num As Integer) num + 1)(5))
```

Ein Lambda-Ausdruck kann als Wert eines Funktionsaufrufs zurückgegeben oder als Argument an einen Delegatenparameter übergeben werden. Im folgenden Beispiel werden boolesche Lambda-Ausdrücke als Argument an die `testResult`-Methode übergeben. Mit dieser Methode wird der boolesche Test auf ein Ganzzahlenargument, `value`, angewendet. Wenn der Lambda-Ausdruck `True` zurückgibt, wird »Erfolgreich« angezeigt, falls er `False` zurückgibt, wird »Fehlgeschlagen« angezeigt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Module Module2

    Sub Main()
        ' Die folgende Zeile gibt 'Erfolgreich' aus, da 4 gerade ist.
        testResult(4, Function(num) num Mod 2 = 0)
        ' Die folgende Zeile gibt 'Fehlgeschlagen' aus, da 5 nicht > 10 entspricht.
        testResult(5, Function(num) num > 10)
    End Sub

    Sub testResult(ByVal value As Integer, ByVal fun As Func(Of Integer, Boolean))
        If fun(value) Then
            Console.WriteLine("Erfolgreich")
        Else
            Console.WriteLine("Fehl geschlagen")
        End If
    End Sub

End Module
```

## Lambda-Ausdrücke am Beispiel

Unser uns permanent begleitendes Beispiel können wir mithilfe von Lambda-Ausdrücken stark vereinfachen, da wir den Zwischenschritt der Delegatenvariablendefinition nicht mehr benötigen, sondern der Sortierroutine direkt den Sortierkriteriumsausdruck als Lambda-Ausdruck übergeben können.

```
    Sub Main()
.
.
.
        'Die Adressen nach Nachnamen sortieren
        Console.WriteLine()
        Console.Write("Adressen werden nach Nachnamen sortiert ... ")
        AdressenSortieren((Function(k1 As Kontakt, k2 As Kontakt) _
                           String.Compare(k1.Nachname, k2.Nachname)))
        Console.WriteLine("fertig!")
        Console.WriteLine()

        'Die ersten 10 Zufallsadressen ausgeben
        AdressenAusgeben(0, 10)

        'Die Adressen nach Ortsnamen sortieren
        Console.WriteLine()
        Console.Write("Adressen werden nach Ortsnamen sortiert ... ")
        AdressenSortieren((Function(k1 As Kontakt, k2 As Kontakt) _
                           String.Compare(k1.Ort, k2.Ort)))
        Console.WriteLine("fertig!")
        Console.WriteLine()
.
.
.
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Sortierroutine kann diesen Lambda-Ausdruck ebenfalls direkt als solchen entgegennehmen und braucht ebenfalls nicht mehr den »Umweg« über einen definierten Delegatentyp zu gehen:

```
'Lambda-Funktion nimmt zwei Kontakt-Objekte entgegen und liefert Integer zurück
    Sub AdressenSortieren(ByVal cDel As Func(Of Kontakt, Kontakt, Integer))

        Dim anzahlElemente As Integer = 101

        Dim aeussererZaehler As Integer
        Dim innererZaehler As Integer
        Dim delta As Integer

        Dim tempKontakt As Kontakt

        delta = 1

        'Größten Wert der Distanzfolge ermitteln
        Do
            delta = 3 * delta + 1
        Loop Until delta > anzahlElemente

        Do
            'Ist Abbruchkriterium - deswegen herunterzählen
            delta \= 3

            'Shellsort's Kernalgorithmus
            For aeussererZaehler = delta To anzahlElemente - 1
                tempKontakt = Kontakte(aeussererZaehler)

                innererZaehler = aeussererZaehler
                Do
                    'Invoke ist optional! - Es geht auch ohne:
                    If cDel(tempKontakt, Kontakte(innererZaehler - delta)) = 1 OrElse _
                        cDel(tempKontakt, Kontakte(innererZaehler - delta)) = 0 Then Exit Do
                    Kontakte(innererZaehler) = Kontakte(innererZaehler - delta)

                    innererZaehler = innererZaehler - delta
                    If (innererZaehler <= delta) Then Exit Do
                Loop
                Kontakte(innererZaehler) = tempKontakt
            Next
        Loop Until delta = 0
    End Sub
```

**HINWEIS** Der Einsatz von Lambda-Ausdrücken macht insbesondere dort Sinn, wo Sie es mit so genannten `Predicate`-, `Comparison`- oder `Action`-Delegaten zu tun haben, und das ist sowohl bei der nichtgenerischen `ArrayList` als auch bei der generischen `List(Of )`-Auflistung der Fall. Kapitel 6 hält mehr darüber im letzten Abschnitt bereit. Dort finden Sie auch im Rahmen eines etwas ausführlicheren `List(Of )`-Beispiels entsprechenden Beispielcode dazu.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 5

# Arrays und Auflistungen

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Arrays kennt fast jedes Basic-Derivat seit Jahrzehnten – und natürlich können Sie auch in Visual Basic .NET auf diese Datenfelder (so der deutsche Ausdruck) zurückgreifen. Doch das .NET Framework wäre nicht das .NET Framework, wenn nicht auch Arrays viel mehr Möglichkeiten bieten würden, als nur auf Daten indiziert zuzugreifen.

Das bedeutet, dass die Leistung von Arrays weit über das reine Zur-Verfügung-Stellen von Containern für die Speicherung verschiedener Elemente eines Datentyps hinausreicht. Da Arrays auf `Object` basieren und damit eine eigene Klasse darstellen (`System.Array` nämlich), bietet das .NET Framework über `Array`-Objekte weit-reichende Funktionen zur Verwaltung ihrer Elemente an.

So können Arrays beispielsweise quasi auf Knopfdruck sortiert werden. Liegen sie in sortierter Form vor, können Sie auch binär nach ihren Elementen suchen und viele weitere Dinge mit ihnen anstellen, ohne selbst den entsprechenden Code dafür entwickeln zu müssen.

Zu guter Letzt bildet der Typ `System.Array` aber auch die Basis für weitere Datentypen – zum Beispiel `ArrayList` –, die Datenelemente in einer bestimmten Form verwalten können, aber auch die Grundlage für viele der generischen Auflistungstypen, die wiederum die Basis für *LINQ to Objects* bilden.

Dieses Kapitel zeigt Ihnen, was Sie mit Arrays und den von ihnen abgeleiteten Klassen alles anstellen können, und bereitet Sie nicht zuletzt damit auf den Umgang mit *LINQ to Objects* vor.

# Grundsätzliches zu Arrays

Arrays im ursprünglichen Basic-Sinne dienen dazu, *mehrere* Elemente desselben Datentyps unter einem bestimmten Namen verfügbar zu machen. Um die einzelnen Elemente zu unterscheiden, bedient man sich eines Indexes (ganz einfach ausgedrückt: einer Nummerierung der Elemente), damit man auf die verschiedenen Array-Elemente zugreifen kann.

**HINWEIS** Viele der größeren nun folgenden Beispiele sind im Projekt *Arrays* in verschiedenen Methoden zusammengefasst. Das Projekt befindet sich im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\Arrays*. Sie können dieses Projekt verwenden, um die Beispiele an Ihrem eigenen Rechner nachzuvollziehen oder um eigene Experimente mit Arrays durchzuführen.

Ein Beispiel:

```
Sub Beispiel1()

    'Array mit 10 Elementen fest deklarieren.
    'Wichtig: Anders als in C# oder C++ wird der Index
    'des letzten Elementes, nicht die Anzahl der Elemente
    'festgelegt! Elementzählung beginnt bei 0.
    'Die folgende Anweisung definiert also 10 Elemente:
    Dim locIntArray(9) As Integer

    'Zufallsgenerator initialisieren,
    Dim locRandom As New Random(Now.Millisecond)

    For count As Integer = 0 To 9
        locIntArray(count) = locRandom.Next
    Next
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        For count As Integer = 0 To 9
            Console.WriteLine("Element Nr. {0} hat den Wert {1}", count, locIntArray(count))
        Next

        Console.ReadLine()

    End Sub
```

Wenn Sie dieses Beispiel ausführen, erscheint im Konsolenfenster eine Liste mit Werten, etwa wie im nachstehenden Bildschirmauszug zu sehen (die Werte sollten sich natürlich von Ihren unterscheiden, da es zufällige sind).[1]

```
Element Nr. 0 hat den Wert 1074554181
Element Nr. 1 hat den Wert 632329388
Element Nr. 2 hat den Wert 1312197477
Element Nr. 3 hat den Wert 458430355
Element Nr. 4 hat den Wert 1970029554
Element Nr. 5 hat den Wert 503465071
Element Nr. 6 hat den Wert 112607304
Element Nr. 7 hat den Wert 1507772275
Element Nr. 8 hat den Wert 1111627006
Element Nr. 9 hat den Wert 213729371
```

Dieses Beispiel demonstriert die grundsätzliche Anwendung von Arrays. Natürlich sind Sie bei der Definition des Elementtyps nicht auf `Integer` festgelegt. Es gilt der Grundsatz: Jedes Objekt in .NET kann Element eines Arrays sein.

## Änderung der Array-Dimensionen zur Laufzeit

Das Definieren der Array-Größe mit variablen Werten macht Arrays – wie im Beispiel des letzten Abschnittes gezeigt – zu einem sehr flexiblen Werkzeug bei der Verarbeitung von großen Datenmengen. Doch Arrays sind noch flexibler: Mit der `ReDim`-Anweisung gibt Ihnen Visual Basic die Möglichkeit, ein Array noch nach seiner ersten Deklaration neu zu dimensionieren. Damit werden selbst einfache Arrays zu dynamischen Datencontainern. Auch hier soll ein Beispiel den Umgang verdeutlichen:

```
    Sub Beispiel3()

        Dim locAnzahlStrings As Integer = 15
        Dim locStringArray As String()
        locStringArray = GeneriereStrings(locAnzahlStrings, 30)
        Console.WriteLine("Ausgangsgröße: {0} Elemente. Es folgt der Inhalt:", locStringArray.Length)
        Console.WriteLine(New String("="c, 40))
        DruckeStrings(locStringArray)
```

[1] Kleine Anmerkung am Rande: Ganz so zufällig sind die Werte nicht – `Random` stellt nur sicher, dass generierte Zahlenfolgen zufällig *verteilt* sind. Bei gleicher Ausgangsbasis (definiert durch den Parameter `Seed`, den Sie der `Random`-Klasse beim Instanziieren übergeben) produziert `Random` auch gleiche Zahlenfolgen. Da wir hier die Millisekunde als Basis übergeben, und eine Sekunde aus 1000 Millisekunden besteht, gibt es eine Wahrscheinlichkeit von 1:1000, dass Sie dieselbe wie die hier abgedruckte Zahlenfolge generieren lassen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Wir brauchen 10 weitere, die alten sollen aber erhalten bleiben!
        ReDim Preserve locStringArray(locStringArray.Length + 9)

        'Bleiben die alten wirklich erhalten?
        Console.WriteLine()
        Console.WriteLine("Inhaltsüberprüfung:", locStringArray.Length)
        Console.WriteLine(New String("="c, 40))
        DruckeStrings(locStringArray)

        '10 weitere Elemente generieren.
        Dim locTempStrings(9) As String

        '10 Zeichen mehr pro Element, sodass wir die neuen leicht erkennen können.
        locTempStrings = GeneriereStrings(10, 40)

        'In das "alte" Array kopieren, aber ab Index 15,
        locTempStrings.CopyTo(locStringArray, 15)

        'und nachschauen, was nun wirklich drinsteht!
        Console.WriteLine()
        Console.WriteLine("Inhaltsüberprüfung:", locStringArray.Length)
        Console.WriteLine(New String("="c, 40))
        DruckeStrings(locStringArray)

        Console.ReadLine()
    End Sub
```

Dieses Beispiel macht sich die Möglichkeit zunutze (direkt in der ersten Codezeile), die Dimensionierung und Deklaration eines Arrays zeitlich voneinander trennen zu können. Das Array `locStringArray` wird zunächst nur als Array deklariert – wie groß es sein soll, wird zu diesem Zeitpunkt noch nicht bestimmt. Dabei spielt es in Visual Basic übrigens keine Rolle, ob Sie eine Variable in diesem

```
Dim locStringArray As String()
```

oder diesem

```
Dim locStringArray() As String
```

Stil als Array deklarieren.

Die Größe des Arrays wird das erste Mal von der Prozedur `GeneriereString` festgelegt. Hier erfolgt zwar die Dimensionierung eines zunächst völlig anderen Arrays (`locStrings`); da dieses Array aber als Rückgabewert der aufrufenden Instanz überlassen wird, lebt der hier erstellte Array-Inhalt unter anderem Namen (`locStringArray`) weiter. Das durch beide Objektvariablen referenzierte Array ist dasselbe (im ursprünglichen Sinne des Wortes).

Übrigens: Diese Vorgehensweise entspricht eigentlich schon einem typsicheren Neudimensionieren eines Arrays. Wie Sie beim ersten Array-Beispiel gesehen haben, spielt es natürlich keine Rolle, ob Sie eine Array-Variable zur Aufnahme eines Array-Rückgabeparameters verwenden, die zuvor mit einer festen Anzahl an Elementen oder ohne die Angabe der Array-Größe dimensioniert wurde. Allerdings: Sie verlieren bei dieser Vorgehensweise den Inhalt des ursprünglichen Arrays, denn die Unterroutine erstellt ein neues Array, und mit der Zuweisung an die Objektvariable `locStringArray` wird intern nur ein Zeiger umgebogen. Der Speicherbereich der alten Elemente ist nicht mehr referenzierbar.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**WICHTIG** Diese Tatsache hat Konsequenzen, denn: Anders, als es bei Visual Basic 6.0 noch der Fall war, werden Arrays bei einer Zuweisung an eine andere Objektvariable *nicht* automatisch kopiert. Array-Variablen verhalten sich so wie jeder andere Referenztyp auch unter .NET: Ein Zuweisen einer Array-Variablen an eine andere biegt nur ihren Zeiger auf den Speicherbereich der eigentlichen Daten im Managed Heap um. Die Elemente bleiben an ihrem Platz im Managed Heap, und es wird kein neuer Speicherbereich mit einer Kopie der Array-Elemente für die neue Objektvariable erzeugt!

Das Redimensionieren kann nicht nur über Zuweisungen, sondern – wie im Beispielcode zu sehen – auch mit der `ReDim`-Anweisung erfolgen. Mit dem zusätzlichen Schlüsselwort `Preserve` haben Sie darüber hinaus die Möglichkeit zu bestimmen, dass die alten Elemente dabei erhalten bleiben. Man möchte meinen, dass diese Verhaltensweise die Regel sein sollte, doch mit dem Wissen im Hinterkopf, was beim Neudimensionieren mit `ReDim` genau passiert, sieht die Sache schon anders aus:

- Wird `ReDim` aufgerufen, wird ein komplett neuer Speicherbereich dafür reserviert.
- Der Zeiger für die Objektvariable auf den Bereich für die zuvor zugeordneten Array-Elemente wird auf den neuen Speicherbereich umgebogen.
- Der Speicherbereich, der die alten Array-Elemente enthielt, fällt dem nächsten Garbage-Collector-Durchlauf zum Opfer.
- Wenn `Preserve` hinter der `ReDim`-Anweisung platziert wird, bleiben die alten Array-Elemente erhalten. Doch das entspricht nicht der vollständigen Erklärung des Vorgangs. In Wirklichkeit wird auch hier ein neuer Speicherbereich erstellt, der den Platz für die neu angegebene Anzahl an Array-Elementen bereithält. Auch bei `Preserve` wird der Zeiger auf den Speicherbereich mit den alten Elementen für die betroffene Objektvariable auf den neuen Speicherbereich umgebogen. Doch bevor der alte Speicherbereich freigegeben wird und sich der Garbage Collector über die alten Elemente hermachen kann, werden die vorhandenen Elemente in den neuen Bereich kopiert.

Aus diesem Grund können Sie mit `Preserve` nur Elemente retten, die in eindimensionalen Arrays gespeichert sind oder die durch die letzte Dimension eines mehrdimensionalen Arrays angesprochen werden.

## Wertevorbelegung von Array-Elementen im Code

Alle Arrays, die in den vorangegangenen Beispielen verwendet wurden, sind zur Laufzeit mit Daten gefüllt worden. In vielen Fällen möchten Sie aber Arrays erstellen, die Sie automatisch mit Daten vorbelegen, die das Programm fest vorgibt:

```
Sub Beispiel5()

    'Deklaration und Definition von Elementen mit Double-Werten
    Dim locDoubleArray As Double() = {123.45F, 5435.45F, 3.14159274F}

    'Deklaration und spätere Definition von Elementen mit Integer-Werten
    Dim locIntegerArray As Integer()
    locIntegerArray = New Integer() {1I, 2I, 3I, 3I, 4I}
    'Deklaration und spätere Definition von Elementen mit Date-Werten
    Dim locDateArray As Date()
    locDateArray = New Date() {#12/24/2005#, #12/31/2005#, #3/31/2006#}
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Deklaration und Definition von Elementen im Char-Array:
        Dim locCharArray As Char() = {"V"c, "B"c, "."c, "N"c, "E"c, "T"c, " "c, _
                                      "r"c, "u"c, "l"c, "e"c, "s"c, "!"c}

        'Zweidimensionales Array
        Dim locZweiDimensional As Integer(,)
        locZweiDimensional = New Integer(,) {{10, 10}, {20, 20}, {30, 30}}

        'Oder verschachtelt (das ist nicht Zwei-Dimensional)!
        Dim locVerschachtelt As Date()()
        locVerschachtelt = New Date()() {New Date() {#12/24/2004#, #12/31/2004#}, _
                                         New Date() {#12/24/2005#, #12/31/2005#}}
End Sub
```

Häufigste Fehlerquelle bei dieser Vorgehensweise: Der Zuweisungsoperator wird falsch gesetzt. Bei der kombinierten Deklaration/Definition wird er benötigt; *definieren* Sie nur neu, lassen Sie ihn weg. Beachten Sie auch den Unterschied zwischen mehrdimensionalen und verschachtelten Arrays, auf den ich im nächsten Abschnitt genauer eingehen möchte.

# Mehrdimensionale und verschachtelte Arrays

Bei der Definition von Arrays sind Sie nicht auf eine Dimension beschränkt – das ist ein Feature, das schon bei jahrzehntealten Basic-Dialekten zu finden ist. Sie können ein mehrdimensionales Array erstellen, indem Sie bei der Deklaration die Anzahl der Elemente für jede Dimension durch Komma getrennt angeben:

```
Dim DreiDimensional(5, 10, 3) As Integer
```

Möchten Sie die Anzahl der zu verwaltenden Elemente bei der Deklaration des Arrays nicht festlegen, verwenden Sie die folgende Deklarationsanweisung:

```
Dim AuchDreiDimensional As Integer(,,)
```

Mit `ReDim` oder dem Aufruf von Funktionen, die ein entsprechend dimensioniertes Array zurückliefern, können Sie anschließend das Array definieren.

## Verschachtelte Arrays

Verschachtelte Arrays sind etwas anders konzipiert als mehrdimensionale Arrays. Bei verschachtelten Arrays ist ein Array-Element selbst ein Array (welches auch wieder Arrays beinhalten kann usw.). Anders als bei mehrdimensionalen Arrays können die einzelnen Elemente unterschiedlich dimensionierte Arrays enthalten, und diese Zuordnung lässt sich auch im Nachhinein noch ändern.

Verschachtelte Arrays definieren Sie, indem Sie die Klammerpaare mit der entsprechenden Array-Dimension hintereinander schreiben – anders als bei mehrdimensionalen Arrays, bei denen die Dimensionen in einem Klammerpaar mit Komma getrennt angegeben werden.

Die folgenden Beispiel-Codezeilen (aus der `Sub Beispiel6`) zeigen, wie Sie verschachtelte Arrays definieren, deklarieren und ihre einzelnen Elemente abrufen können:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
'Einfach verschachtelt; Tiefe wird nicht definiert.
Dim EinfachVerschachtelt(10)() As Integer

'Erstes Element hält ein Integer-Array mit drei Elementen.
EinfachVerschachtelt(0) = New Integer() {10, 20, 30}

'Zweites Element hält ein Integer-Array mit acht Elementen.
EinfachVerschachtelt(1) = New Integer() {10, 20, 30, 40, 50, 60, 70, 80}

'Druckt das dritte Element des zweiten Elementes (30) des Arrays.
Console.WriteLine(EinfachVerschachtelt(1)(2))

'In einem Rutsch alles neu zuweisen.
EinfachVerschachtelt = New Integer()() {New Integer() {30, 20, 10}, _
                                        New Integer() {80, 70, 60, 50, 40, 30, 20, 10}}

'Druckt das dritte Element des zweiten Elementes (jetzt 60) des Arrays.
Console.WriteLine(EinfachVerschachtelt(1)(2))
Console.ReadLine()
```

# Die wichtigsten Eigenschaften und Methoden von Arrays

In den vorangegangenen Beispielprogrammen ließ es sich nicht vermeiden, die eine oder andere Eigenschaft oder Methode des Array-Objektes bereits anzuwenden. Dieser Abschnitt soll sich ein wenig genauer mit den zusätzlichen Möglichkeiten dieses Objektes beschäftigen – denn sie sind mächtig und können Ihnen, richtig angewendet, eine Menge Entwicklungszeit ersparen.

## Anzahl der Elemente eines Arrays ermitteln mit Length

Wenn Sie wissen wollen, wie viele Elemente ein Array beherbergt, bedienen Sie sich seiner `Length`-Eigenschaft. Bei zwei- und mehrdimensionalen Arrays ermittelt `Length` ebenfalls die Anzahl aller Elemente.

Aufgepasst bei verschachtelten Arrays: Hier ermittelt `Length` nämlich nur die Elementanzahl des umgebenden Arrays. Sie können die Array-Länge eines Elementes des umgebenden Arrays etwa so ermitteln:

```
'Verschachtelte Arrays
Dim locVerschachtelt As Date()()
locVerschachtelt = New Date()() {New Date() {#12/24/2004#, #12/31/2004#}, _
                                 New Date() {#12/24/2005#, #12/31/2005#}}

Console.WriteLine("Äußeres Array hat {0} Elemente.", locVerschachtelt.Length)
Console.WriteLine("Array des 1. Elements hat {0} Elemente.", locVerschachtelt(0).Length)
```

## Sortieren von Arrays mit Array.Sort

Arrays lassen sich durch eine ganz simple Methode sortieren. Das folgende Beispiel soll demonstrieren, wie es geht:

```
   Sub Beispiel7()

        'Array-Erstellen:
        Dim locNamen As String() = {"Jürgen", "Martina", "Hanna", "Gaby", "Michaela", "Miriam", "Ute", _
                            "Leonie-Gundula", "Melanie", "Uwe", "Andrea", "Klaus", "Anja", _
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
                            "Myriam", "Daja", "Thomas", "José", "Kricke", "Flori", "Katrin", "Momo", _
                            "Gareth", "Anne"}
        System.Array.Sort(locNamen)
        DruckeStrings(locNamen)
        Console.ReadLine()
    End Sub
```

Wenn Sie dieses Beispiel starten, produziert es folgendes Ergebnis im Konsolenfenster:

```
Andrea
Anja
Anne
Daja
Flori
Gaby
Gareth
Hanna
José
Jürgen
Katrin
Klaus
Kricke
Leonie-Gundula
Martina
Melanie
Michaela
Miriam
Momo
Myriam
Thomas
Ute
Uwe
```

Die `Sort`-Methode ist eine statische Methode von `System.Array` und sie kann noch eine ganze Menge mehr. So haben Sie beispielsweise die Möglichkeit, einen korrelierenden Index mitsortieren zu lassen, oder Sie können bestimmen, zwischen welchen Indizes eines Arrays die Sortierung stattfinden soll. Die Online-Hilfe zum .NET Framework verrät Ihnen, welche Überladungen die `Sort`-Methode genau anbietet.

### Umdrehen der Array-Anordnung mit Array.Reverse

Wichtig in diesem Zusammenhang ist eine weitere statische Methode von `System.Array` namens `Reverse`, die die Reihenfolge der einzelnen Array-Elemente umkehrt. Im Zusammenhang mit der `Sort`-Methode erreichen Sie durch den anschließenden Einsatz von `Reverse` die Sortierung eines Arrays in absteigender Reihenfolge. Wenn Sie das vorherige Beispiel um diese Zeilen

```
        Console.WriteLine()
        Console.WriteLine("Absteigend sortiert:")
        Array.Reverse(locNamen)
        DruckeStrings(locNamen)
        Console.ReadLine()
```

ergänzen, sehen Sie schließlich die Namen in umgekehrter Reihenfolge im Konsolenfenster, etwa:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Absteigend sortiert:
Uwe
Ute
Thomas
.
.
.
Anja
Andrea
```

### Durchsuchen eines sortierten Arrays mit Array.BinarySearch

Auch bei der Suche nach bestimmten Elementen eines Arrays ist Ihnen das .NET Framework behilflich. Dazu stellt die Array-Klasse die statische Funktion `BinarySearch` zur Verfügung.

**WICHTIG** Damit eine binäre Suche in einem Array durchgeführt werden kann, muss das Array in sortierter Form vorliegen – ansonsten wird die Funktion höchstwahrscheinlich falsche Ergebnisse zurückliefern. Wirklich brauchbar ist die Funktion überdies nur dann, wenn Sie sicherstellen, dass es keine Dubletten in den Elementen gibt. Da eine `binäre` Suche erfolgt, ist nicht gewährleistet, ob die Funktion das `erste` zutreffende Objekt findet oder ein beliebiges, das dem Gesuchten entsprach.

Beispiel:

```
Sub Beispiel8()

    'Array-Erstellen:
    Dim locNamen As String() = {"Jürgen", "Martina", "Hanna", "Gaby", "Michaela", "Miriam", "Ute", _
                          "Leonie-Gundula", "Melanie", "Uwe", "Andrea", "Klaus", "Anja", _
                          "Myriam", "Daja", "Thomas", "José", "Kricke", "Flori", "Katrin", "Momo", _
                                "Gareth", "Anne", "Jürgen", "Gaby"}
    System.Array.Sort(locNamen)
    Console.WriteLine("Jürgen wurde gefunden an Position {0}", _
            System.Array.BinarySearch(locNamen, "Jürgen"))
    Console.ReadLine()
End Sub
```

Wie `Sort` ist auch `BinarySearch` eine überladene Funktion und bietet weitere Optionen, die die Suche beispielsweise auf bestimmte Indexbereiche beschränkt. IntelliSense und die Online-Hilfe geben hier genauere Hinweise für die Verwendung.

## Implementierung von Sort und BinarySearch für eigene Klassen

Das .NET Framework erlaubt es, Arrays von beliebigen Typen zu erstellen, auch von solchen, die Sie selbst erstellt haben. Bei der Erstellung einer Klasse, die als Element eines Arrays fungieren soll, brauchen Sie dabei nichts Besonderes zu beachten.

Anders wird das allerdings, wenn Sie eine Funktion auf das Array anwenden wollen, die einen Elementvergleich erfordert, oder wenn Sie sogar steuern wollen, nach welchen Kriterien Ihr Array beispielsweise sortiert werden soll oder auch nach welchem Kriterium Sie es mit `BinarySearch` durchsuchen lassen möchten. Dazu ein Beispiel:

Sie haben eine Klasse entwickelt, die die Adressen einer Kontaktdatenbank speichert. Der nachfolgend gezeigte Code demonstriert, wie sie funktioniert und wie man sie anwendet:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**BEGLEITDATEIEN** Im Folgenden sehen Sie zunächst den Klassencode, der einen Adresseneintrag verwaltet – dieses Projekt finden Sie im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\IComparer01*

```
Public Class Adresse

    Protected myName As String
    Protected myVorname As String
    Protected myPLZ As String
    Protected myOrt As String

    Sub New(ByVal Name As String, ByVal Vorname As String, ByVal Plz As String, ByVal Ort As String)
        myName = Name
        myVorname = Vorname
        myPLZ = Plz
        myOrt = Ort
    End Sub

    Public Property Name() As String
        Get
            Return myName
        End Get
        Set(ByVal Value As String)
            myName = Value
        End Set
    End Property

    Public Property Vorname() As String
        Get
            Return myVorname
        End Get
        Set(ByVal Value As String)
            myVorname = Value
        End Set
    End Property

    Public Property PLZ() As String
        Get
            Return myPLZ
        End Get
        Set(ByVal Value As String)
            myPLZ = Value
        End Set
    End Property

    Public Property Ort() As String
        Get
            Return myOrt
        End Get
        Set(ByVal Value As String)
            myOrt = Value
        End Set
    End Property
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Public Overrides Function ToString() As String
        Return Name + ", " + Vorname + ", " + PLZ + " " + Ort
    End Function
End Class
```

Sie sehen selbst: einfachstes Visual Basic. Die Klasse stellt ein paar Eigenschaften für die von ihr verwalteten Daten zur Verfügung, und sie überschreibt die `ToString`-Methode der Basis-Klasse, damit sie eine Adresse als kompletten String ausgeben kann.[2]

Das folgende kleine Beispielprogramm definiert ein Array aus dieser Klasse, richtet ein paar Adressen zum Experimentieren ein und druckt diese anschließend in einer eigenen Unterroutine aus:

```
Module ComparerBeispiel

    Sub Main()
        Dim locAdressen(5) As Adresse

        locAdressen(0) = New Adresse("Löffelmann", "Klaus", "11111", "Soest")
        locAdressen(1) = New Adresse("Heckhuis", "Jürgen", "99999", "Gut Uhlenbusch")
        locAdressen(2) = New Adresse("Sonntag", "Miriam", "22222", "Dortmund")
        locAdressen(3) = New Adresse("Sonntag", "Christian", "33333", "Wuppertal")
        locAdressen(4) = New Adresse("Ademmer", "Uta", "55555", "Bad Waldholz")
        locAdressen(5) = New Adresse("Kaiser", "Wilhelm", "12121", "Ostenwesten")

        Console.WriteLine("Adressenliste:")
        Console.WriteLine(New String("="c, 40))
        DruckeAdressen(locAdressen)
        'Array.Sort(locAdressen)
        Console.ReadLine()
    End Sub
    Sub DruckeAdressen(ByVal Adressen As Adresse())
        For Each locString As Adresse In Adressen
            Console.WriteLine(locString)
        Next
    End Sub

End Module
```

Auch hier werden Sie erkennen: Es passiert nichts wirklich Aufregendes. Wenn Sie das Programm starten, produziert es, wie zu erwarten, folgende Ausgabe im Konsolenfenster:

```
Adressenliste:
======================================
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
```

[2] Überschreiben bedeutet, dass ein Element der Basisklasse, von der aus eine Klasse abgeleitet wurde, durch ein neues ersetzt wird. Da jede Klasse, die Sie neu erstellen, implizit von `Object` erbt, verfügt jede Klasse auch automatisch über die Funktionen, die `Object` kennt – die `ToString`-Methode, die eine Objektbeschreibung als Zeichenkette ermittelt, gehört dazu. Um eine Methode zu implementieren, die den Inhalt eines Objektes als String ausgibt, ergibt es in jedem Fall Sinn, dazu die immer vorhandene `ToString`-Methode durch eine neue zu ersetzen und diese dann weiter zu verwenden. Mehr zum Thema Vererbung, Polymorphie und Überschreiben von Methoden finden Sie im Buch *Visual Basic 2005 – das Entwicklerbuch*, das sie unter *www.activedevelop.de* kostenlos herunterladen können.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten
```

Die Liste, wie wir Sie anzeigen lassen, ist allerdings ein ziemliches Durcheinander. Beobachten Sie, was passiert, wenn wir das Programm um eine `Sort`-Anweisung ergänzen und wie die Liste anschließend ausschaut. Dazu nehmen Sie die Auskommentierung der `Sort`-Methode im Listing einfach zurück.

Nach dem Start des Programms warten Sie vergeblich auf die zweite, sortierte Ausgabe der Liste. Stattdessen löst das .NET Framework eine Ausnahme aus, etwa wie in Abbildung 5.1 zu sehen.

Der Grund dafür: Die `Sort`-Methode der Array-Klasse versucht, die einzelnen Elemente des Arrays miteinander zu vergleichen. Dazu benötigt es einen so genannten *Comparer* (etwa: *Vergleicher*). Da wir nicht explizit angeben, dass wir einen speziellen Comparer verwenden wollen (Welchen auch? – Wir haben noch keinen!), erzeugt es einen Standard-Comparer, der aber wiederum die Einbindung einer bestimmten Schnittstelle in der Klasse verlangt, deren Elemente er miteinander vergleichen soll. Diese Schnittstelle nennt sich `IComparable`. Leider haben wir auch diese Schnittstelle nicht implementiert, und die Ausnahme ist die Folge.

Wir haben nun drei Möglichkeiten. Wir binden die `IComparable`-Schnittstelle ein, dann können Elemente unserer Klasse auch ohne die Nennung eines expliziten Comparer verglichen und im Endeffekt sortiert werden.

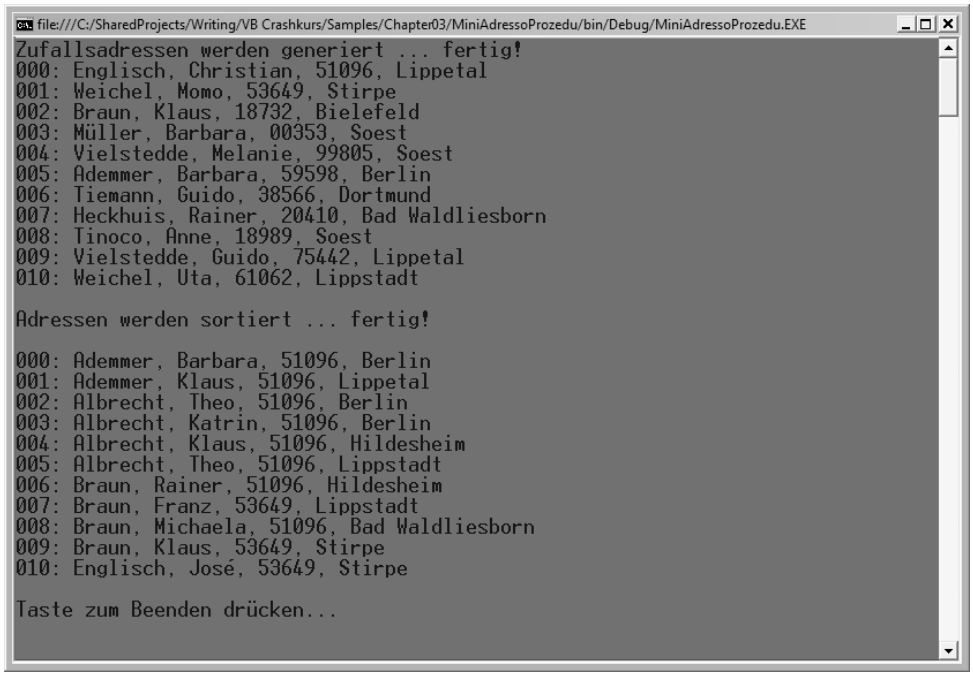



**Abbildung 5.1** Wenn Sie das Programm starten, löst es eine Ausnahme aus, sobald die Sort-Methode der Array-Klasse erreicht ist – in der Detailbeschreibung zur Ausnahme sehen Sie, was falsch läuft!

Oder wir stellen der Klasse einen *expliziten* Comparer zur Verfügung; in diesem Fall müssen wir ihn beim Einsatz von `Sort` (oder auch `BinarySearch`) benennen. Dieser Comparer hätte den Vorteil, dass sich durch ihn steuern ließe, nach welchem Kriterium die Klasse durchsucht bzw. sortiert werden soll.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die dritte Möglichkeit: Wir machen beides. Damit wird unsere Klasse universell nutzbar und läuft nicht Gefahr, eine Ausnahme auslösen zu können. Und genau das werden wir in den nächsten beiden Abschnitten in die Tat umsetzen.

### Implementieren der Vergleichsfähigkeit einer Klasse durch IComparable

Die Implementierung der `IComparable`-Schnittstelle, damit Instanzen unserer Klasse miteinander verglichen werden können, ist vergleichsweise simpel.

Erinnern wir uns: Damit bestimmter Code allgemeingültig verwendbar sein kann, kann man sich des Konzeptes von Schnittstellen (engl. *Interfaces*) bedienen. Das Einbinden einer Schnittstelle in eine Klasse oder eine Struktur erzwingt, dass in der Klasse die Elemente (Eigenschaften, Methoden, Ereignisse) vorhanden sein müssen, die die Schnittstelle vorschreibt. Gleichzeitig reicht es aus, beispielsweise eine Methode, die eine Schnittstelle für eine Klasse vorschreibt, über eine Schnittstellenvariable aufzurufen. Damit spielt es für die aufrufende Instanz keine Rolle mehr, mit was sie es genau zu tun hat – ob es beispielsweise ein Integer oder ein Decimal-Datentyp ist. Die aufrufende Instanz arbeitet lediglich mit einer Schnittstellenvariablen, und ruft über diese die gewünschte Methode auf. Bei einem Integer-Datentyp, der diese Schnittstelle einbindet, werden so beispielsweise Integer-Werte verglichen, bei einem String-Datentyp, der die gleiche Schnittstelle einbindet, eben Zeichenketten. Doch das merkt und interessiert die aufrufende Instanz gar nicht (in diesem Fall die Array-Klasse des .NET Frameworks). Sie ist nur am Ergebnis interessiert.

Das Implementieren der Schnittstelle in unserer Beispielklasse erfordert übrigens zusätzlich lediglich das Vorhandensein einer `CompareTo`-Methode, die die aktuelle Instanz der Klasse mit einer weiteren vergleicht, und das Anzeigen, dass es sich bei der Methode um die von der Schnittstelle vorgeschriebene Methode handelt.

Da durch den Einsatz von `IComparable` keine Möglichkeit besteht festzulegen, welches Datenfeld das Vergleichskriterium sein soll, wird unser Kriterium eine Zeichenkette sein, die aus Namen, Vornamen, Postleitzahl und Ort besteht – genau die Zeichenkette also, die `ToString` in der jetzigen Version bereits zurückliefert. Deswegen brauchen wir den eigentlichen Vergleich noch nicht einmal selbst durchzuführen, sondern können ihn an die `CompareTo`-Funktion des `String`-Objektes, das wir von `ToString` zurückerhalten, weiterreichen. Die Modifizierungen an der Klasse sind also denkbar gering (aus Platzgründen nur gekürzter Code, der die Änderungen widerspiegelt):

```
Public Class Adresse
    Implements IComparable
    .
    .
    .
    Public Function CompareTo(ByVal obj As Object) As Integer Implements System.IComparable.CompareTo

     Dim locAdresse As Adresse

     Try
         locAdresse = DirectCast(obj, Adresse)
     Catch ex As InvalidCastException
      Dim up As New InvalidCastException("'CompareTo' der Klasse 'Adresse' kann keine Vergleiche " + _
                                         "mit Objekten anderen Typs durchführen!")
         Throw up
     End Try
     Return ToString.CompareTo(locAdresse.ToString)
    End Function
End Class
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

An dieser Stelle vielleicht erwähnenswert ist der fett gekennzeichnete mittlere Bereich im Programm. Auf den ersten Blick mag es unsinnig erscheinen, einen möglichen Fehler abzufangen und ihn anschließend mehr oder weniger unverändert wieder auszulösen. Aber: Der Fehlertext ist entscheidend, und Sie tun sich selbst einen Gefallen, wenn Sie den Fehlertext einer Ausnahme, die Sie generieren, so formulieren, dass Sie eindeutig wissen, wer oder was sie ausgelöst hat. Wenn Sie das Programm anschließend starten, liefert es das gewünschte Ergebnis, wie im Folgenden zu sehen:

```
Adressenliste:
======================================
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten

Adressenliste (sortiert):
======================================
Ademmer, Uta, 55555 Bad Waldholz
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Kaiser, Wilhelm, 12121 Ostenwesten
Löffelmann, Klaus, 11111 Soest
Sonntag, Christian, 33333 Wuppertal
Sonntag, Miriam, 22222 Dortmund
```

### Implementieren einer gesteuerten Vergleichsfähigkeit durch IComparer

So weit, so gut – unser Beispielprogramm läuft immerhin schon, und die Elemente des Arrays lassen sich sortieren. Aber: Das Programm sortiert stumpf nach dem Nachnamen – und diese Verhaltensweise können wir ihm ohne weitere Maßnahmen auch nicht abgewöhnen.

Was die `Adresse`-Klasse braucht, ist eine Art Steuerungseinheit, die durch Setzen bestimmter Eigenschaften festlegt, wie Vergleiche stattfinden sollen. Wenn Sie sich zum Beispiel die `Sort`-Funktion von `System.Array` ein wenig genauer anschauen, werden Sie feststellen, dass sie als optionalen Parameter eine Variable vom Typ `IComparer` entgegennimmt.

Eine Klasse, die `IComparer` einbindet, macht genau das Verlangte. Sie kann den Vergleichsvorgang beeinflussen. Wenn Sie `IComparer` in einer Klasse implementieren, müssen Sie auch die Funktion `Compare` in dieser Klasse zur Verfügung stellen. Dieser Funktion werden zwei Objekte übergeben, die die Funktion miteinander vergleichen soll. Ist eines der Objekte größer (was auch immer das heißt, denn es ist bis dahin ein abstraktes Attribut), liefert sie den Wert 1, ist es kleiner, den Wert –1, und ist es gleich, liefert sie den Wert 0 zurück.

Ein Comparer steht in keinem direkten Verhältnis zu den Klassen, die er vergleichen soll; er wird also nicht durch weitere Schnittstellen reglementiert. Aus diesem Grund muss der Comparer selber dafür Sorge tragen, dass ihm nur die Objekte zum Vergleichen angeliefert werden, die er verarbeiten will. Bekommt er andere, muss er eine Ausnahme auslösen.

In unserem Fall muss der Comparer noch ein bisschen mehr können. Er muss die Funktionalität zur Verfügung stellen, durch die der Entwickler steuern kann, nach *welchem* Kriterium der *Adresse*-Klasse er vergleichen will. Aus diesen Gründen ergibt sich folgender Code für einen Comparer unseres Beispiels:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**BEGLEITDATEIEN** Sie finden das Projekt im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\IComparer02*

```
'Nur für den einfachen Umgang mit der Klasse.
Public Enum ZuVergleichen
    Name
    PLZ
    Ort
End Enum

Public Class AdressenVergleicher
    Implements IComparer

    'Speichert die Einstellung, nach welchem Kriterium verglichen wird.
    Protected myZuVergleichenMit As ZuVergleichen

    Sub New(ByVal ZuVergleichenMit As ZuVergleichen)
        myZuVergleichenMit = ZuVergleichenMit
    End Sub

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
                                        Implements System.Collections.IComparer.Compare
        'Nur erlaubte Typen durchlassen:
        If (Not (TypeOf x Is Adresse)) Or (Not (TypeOf y Is Adresse)) Then
            Dim up As New InvalidCastException("'Compare' der Klasse 'Adressenvergleicher' kann nur " +
                                               "Vergleiche vom Typ 'Adresse' durchführen!")
_
            Throw up
        End If

        'Beide Objekte in den richtigen Typ casten, damit das Handling einfacher wird:
        Dim locAdr1 As Adresse = DirectCast(x, Adresse)
        Dim locAdr2 As Adresse = DirectCast(y, Adresse)

        'Hier passiert die eigentliche Steuerung,
        'nach welchem Kriterium verglichen wird:
        If myZuVergleichenMit = ZuVergleichen.Name Then
            Return locAdr1.Name.CompareTo(locAdr2.Name)
        ElseIf myZuVergleichenMit = ZuVergleichen.Ort Then
            Return locAdr1.Ort.CompareTo(locAdr2.Ort)
        Else
            Return locAdr1.PLZ.CompareTo(locAdr2.PLZ)
        End If

    End Function

    'Legt die Vergleichseinstellung offen.
    Public Property ZuVergleichenMit() As ZuVergleichen
        Get
            Return myZuVergleichenMit
        End Get
        Set(ByVal Value As ZuVergleichen)
            myZuVergleichenMit = Value
        End Set
    End Property

End Class
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Zum Beweis, dass alles wie gewünscht läuft, ergänzen Sie das Hauptprogramm um folgende Zeilen (fettgedruckt im folgenden Listing):

```
Module ComparerBeispiel

    Sub Main()
        Dim locAdressen(5) As Adresse

        locAdressen(0) = New Adresse("Löffelmann", "Klaus", "11111", "Soest")
        locAdressen(1) = New Adresse("Heckhuis", "Jürgen", "99999", "Gut Uhlenbusch")
        locAdressen(2) = New Adresse("Sonntag", "Miriam", "22222", "Dortmund")
        locAdressen(3) = New Adresse("Sonntag", "Christian", "33333", "Wuppertal")
        locAdressen(4) = New Adresse("Ademmer", "Uta", "55555", "Bad Waldholz")
        locAdressen(5) = New Adresse("Kaiser", "Wilhelm", "12121", "Ostenwesten")

        Console.WriteLine("Adressenliste:")
        Console.WriteLine(New String("="c, 40))
        DruckeAdressen(locAdressen)

        Console.WriteLine()
        Console.WriteLine("Adressenliste (sortiert nach Postleitzahl):")
        Console.WriteLine(New String("="c, 40))
        Array.Sort(locAdressen, New AdressenVergleicher(ZuVergleichen.PLZ))
        DruckeAdressen(locAdressen)
        Console.ReadLine()
    End Sub
    .
    .
    .
End Module
```

Wenn Sie das Programm nun starten, erhalten Sie folgende Ausgabe auf dem Bildschirm:

```
Adressenliste:
======================================
Löffelmann, Klaus, 11111 Soest
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Kaiser, Wilhelm, 12121 Ostenwesten

Adressenliste (sortiert nach Postleitzahl):
======================================
Löffelmann, Klaus, 11111 Soest
Kaiser, Wilhelm, 12121 Ostenwesten
Sonntag, Miriam, 22222 Dortmund
Sonntag, Christian, 33333 Wuppertal
Ademmer, Uta, 55555 Bad Waldholz
Heckhuis, Jürgen, 99999 Gut Uhlenbusch
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Enumeratoren

Wenn Sie Arrays oder – wie Sie später sehen werden – Auflistungen für die Speicherung von Daten verwenden, dann müssen Sie in der Lage sein, diese Daten abzurufen. Mit *Indexern* gibt Ihnen das .NET Framework eine einfache – *die* einfachste – Möglichkeit: Sie verwenden eine Objektvariable und versehen sie mit einem Index, der auch durch eine andere Variable repräsentiert werden kann. Ändern Sie diese Variable, die als Index dient, können Sie dadurch programmtechnisch bestimmen, welches Element eines Arrays Sie gerade verarbeiten wollen. Typische Zählschleifen, mit denen Sie durch die Elemente eines Arrays iterieren, sind die Folge – etwa im folgenden Stil:

```
For count As Integer = 0 To Array.Length - 1
    TuWasMit(Array(count))
Next
```

**HINWEIS** Enumeratoren haben nichts mit Enums zu tun. Lediglich die Namen sind sich etwas ähnlich. Enums sind aufgezählte Benennungen von bestimmten Werten im Programmlisting, während Enumeratoren die Unterstützung von `For/Each` zur Verfügung stellen!

Wenn ein Objekt allerdings die Schnittstelle `IEnumarable` implementiert, gibt es eine elegantere Methode, die verschiedenen Elemente, die das Objekt zur Verfügung stellt, zu durchlaufen. Glücklicherweise implementiert `System.Array` die Schnittstelle `IEnumarable`, sodass Sie auf Array-Elemente mit dieser eleganten Methode – namentlich mit `For/Each` – zugreifen können. Ein Beispiel:

```
        'Deklaration und Definition von Elementen im Char-Array
        Dim locCharArray As Char() = {"V"c, "B"c, "."c, "N"c, "E"c, "T"c, " "c, _
                                      "r"c, "u"c, "l"c, "e"c, "s"c, "!"c}
        For Each c As Char In locCharArray
            Console.Write(c)
        Next
        Console.WriteLine()
        Console.ReadLine()
```

Wenn Sie dieses Beispiel laufen lassen, sehen Sie im Konsolenfenster den folgenden Text:

```
VB.NET rules!
```

Enumeratoren werden von allen typdefinierten Arrays unterstützt und ebenso von den meisten Auflistungen. Enumeratoren können Sie aber auch in Ihren eigenen Klassen einsetzen, wenn Sie erlauben möchten, dass der Entwickler, der mit Ihrer Klasse arbeitet, durch Elemente mit `For/Each` iterieren kann.

**WICHTIG** Enumeratoren sind lebenswichtig für LINQ (siehe ab Kapitel 7). Nur eine Auflistungsklasse, die `IEnumerable(Of t)` implementiert, kann als Abfrageausdruck in LINQ verwendet werden. Denken Sie daran, wenn Sie eigene Auflistungsklassen entwerfen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## Benutzerdefinierte Enumeratoren durch Implementieren von IEnumerable

Enumeratoren können allerdings nicht nur für die Aufzählung von gespeicherten Elementen in Arrays oder Auflistungen eingesetzt werden. Sie können Enumeratoren auch dann einsetzen, wenn eine Klasse ihre Enumerations-Elemente durch Algorithmen zurückliefert.

Als Beispiel dafür möchte ich Ihnen zunächst einen Codeausschnitt zeigen, der nicht funktioniert – bei dem es aber in einigen Fällen wünschenswert wäre, wenn er funktionierte:

```
'Das würde nicht funktionieren:
for d as Date=#24/12/2004# to #31/12/2004#
    Console.WriteLine("Datum in Aufzählung: {0}", d)
Next d
```

Dennoch könnte es für bestimmte Anwendungen sinnvoll sein, tageweise einen bestimmten Datumsbereich zu durchlaufen. Etwa wenn Ihre Anwendung herausfinden muss, wie viele Mitarbeiter, deren Daten Ihre Anwendung speichert, in einem bestimmten Monat Geburtstag haben.

Wenn das, was wir vorhaben, nicht mit `For/Next` funktioniert, vielleicht können wir dann aber eine Klasse schaffen, die einen Enumerator zur Verfügung stellt, sodass das Vorhaben mit `For/Each` gelingt. Diese Klasse sollte beim Instanziieren Parameter übernehmen, mit denen wir bestimmen können, welcher Datumsbereich in welcher Schrittweite durchlaufen werden soll. Damit Sie mit `For/Each` durch die Elemente einer Klasse iterieren können, muss die Klasse die Schnittstelle `IEnumarable` einbinden.

Das kann sie nur, wenn sie gleichzeitig eine Funktion `GetEnumerator` zur Verfügung stellt, die erst das Objekt mit dem eigentlichen Enumerator liefert. Doch eines nach dem anderen.

**BEGLEITDATEIEN** Schauen wir uns zunächst die Basisklasse an, die die Grundfunktionalität zur Verfügung stellt – Sie finden das Projekt im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\Enumerators*). Starten Sie das Programm mit Strg F5.

```
Public Class Datumsaufzählung
    Implements IEnumerable
    Dim locDatumsaufzähler As Datumsaufzähler

    Sub New(ByVal StartDatum As Date, ByVal EndDatum As Date, ByVal Schrittweite As TimeSpan)
        locDatumsaufzähler = New Datumsaufzähler(StartDatum, EndDatum, Schrittweite)
    End Sub
    Public Function GetEnumerator() As System.Collections.IEnumerator _
                    Implements System.Collections.IEnumerable.GetEnumerator
        Return locDatumsaufzähler
    End Function
End Class
```

Sie sehen, dass diese Klasse selbst nichts Großartiges macht – sie schafft durch die ihr übergebenen Parameter lediglich die Rahmenbedingungen und stellt die von `IEnumarable` verlangte Funktion `GetEnumerator` zur Verfügung. Die eigentliche Aufgabe wird von der Klasse `Datumsaufzähler` erledigt, die auch als Rückgabewert von `GetEnumerator` zurückgegeben wird.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Eine Instanz dieser Klasse wird bei der Instanziierung von `Datumsaufzählung` erstellt. Was in dieser Klasse genau passiert, zeigt der folgende Code:

```
Public Class Datumsaufzähler
    Implements IEnumerator

    Private myStartDatum As Date
    Private myEndDatum As Date
    Private mySchrittweite As TimeSpan
    Private myAktuellesDatum As Date

    Sub New(ByVal StartDatum As Date, ByVal EndDatum As Date, ByVal Schrittweite As TimeSpan)
        myStartDatum = StartDatum
        myAktuellesDatum = StartDatum
        myEndDatum = EndDatum
        mySchrittweite = Schrittweite
    End Sub

    Public Property StartDatum() As Date
        Get
            Return myStartDatum
        End Get
        Set(ByVal Value As Date)
            myStartDatum = Value
        End Set
    End Property

    Public Property EndDatum() As Date
        Get
            Return myEndDatum
        End Get
        Set(ByVal Value As Date)
            myEndDatum = Value
        End Set
    End Property

    Public Property Schrittweite() As TimeSpan
        Get
            Return mySchrittweite
        End Get
        Set(ByVal Value As TimeSpan)
            mySchrittweite = Value
        End Set
    End Property

    Public ReadOnly Property Current() As Object Implements System.Collections.IEnumerator.Current
        Get
            Return myAktuellesDatum
        End Get
    End Property

    Public Function MoveNext() As Boolean Implements System.Collections.IEnumerator.MoveNext
        myAktuellesDatum = myAktuellesDatum.Add(Schrittweite)
        If myAktuellesDatum > myEndDatum Then
            Return False
        Else
            Return True
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        End If
    End Function

    Public Sub Reset() Implements System.Collections.IEnumerator.Reset
        myAktuellesDatum = myStartDatum
    End Sub
End Class
```

Der Konstruktor und die beiden Eigenschaftsprozeduren sollen hier nicht so sehr interessieren. Vielmehr von Interesse ist, dass diese Klasse eine weitere Schnittstelle namens `IEnumerator` einbindet, und sie stellt die eigentliche Aufzählungsfunktionalität zur Verfügung. Sie muss dazu die Eigenschaft `Current`, die Funktion `MoveNext` sowie die Methode `Reset` implementieren.

Wenn eine `For/Each`-Schleife durchlaufen wird, dann wird das aktuell bearbeitete Objekt der Klasse durch die `Current`-Eigenschaft des eigentlichen Enumerators ermittelt. Anschließend zeigt `For/Each` mit dem Aufruf der Funktion `MoveNext` dem Enumerator an, dass es auf das nächste Objekt zugreifen möchte. Erst wenn `MoveNext` mit `False` als Rückgabewert anzeigt, dass es keine weiteren Objekte mehr zur Verfügung stellen kann (oder will), ist die umgebende `For/Each`-Schleife beendet.

In unserem Beispiel müssen wir bei `MoveNext` nur dafür sorgen, dass unsere interne Datums-Zähl-Variable um den Wert erhöht wird, den wir bei ihrer Instanziierung als Schrittweite bestimmt haben. Hat die Addition der Schrittweite auf diesen Datumszähler den Endwert noch nicht überschritten, liefern wir `True` als Funktionsergebnis zurück – `For/Each` darf mit seiner Arbeit fortfahren. Ist das Datum allerdings größer als der Datums-Endwert, wird die Schleife abgebrochen – der Vorgang wird beendet.

Das Programm, das von dieser Klasse Gebrauch machen kann, lässt sich nun sehr elegant einsetzen, wie die folgenden Beispielcodezeilen zeigen:

```
Module Enumerators
    Sub Main()
        Dim locDatumsaufzählung As New Datumsaufzählung(#12/24/2004#, _
                                                        #12/31/2004#, _
                                                        New TimeSpan(1, 0, 0, 0))
        For Each d As Date In locDatumsaufzählung
            Console.WriteLine("Datum in Aufzählung: {0}", d)
        Next

    End Sub
End Module
```

Das Ergebnis sehen Sie anschließend in Form einer Datumsfolge im Konsolenfenster.

# Grundsätzliches zu Auflistungen (Collections)

Arrays haben im .NET Framework einen entscheidenden Nachteil. Sie können zwar dynamisch zur Laufzeit vergrößert oder verkleinert werden, aber der Programmieraufwand dazu ist doch eigentlich recht aufwendig. Wenn Sie sich schon früher mit Visual Basic beschäftigt haben, dann ist Ihnen »Auflistung« sicherlich ein Begriff.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Auflistungen erlauben es dem Entwickler, Elemente genau wie Arrays zu verwalten. Im Unterschied zu Arrays wachsen Auflistungen jedoch mit Ihren Speicherbedürfnissen.

Damit ist aber auch klar, dass das Indizieren mit Nummern zum Abrufen der Elemente nur bedingt funktionieren kann. Wenn ein Array 20 Elemente hat, und Sie möchten das 21. Element hinzufügen, dann können Sie das dem Array nicht einfach so mitteilen. Ganz anders bei Auflistungen: Hier fügen Sie ein Element mit der `Add`-Methode hinzu.

Intern werden (fast alle) Auflistungen ebenfalls wie (oder besser: als) Arrays verwaltet. Rufen Sie eine neue Auflistung ins Leben, dann hat dieses Array, wenn nichts anderes gesagt wird, eine Größe von 16 Elementen. Wenn die Auflistungsklasse später, sobald Ihr Programm richtig »in Action« ist, »merkt«, dass ihr die Puste mengentechnisch ausgeht, dann legt sie ein Array nunmehr mit 32 Elementen an, kopiert die vorhandenen Elemente in das neue Array, arbeitet fortan mit dem neuen Array und tut ansonsten so, als wäre nichts gewesen.

Dieser anfängliche Load-Faktor erhöht sich bei jedem Neuanlegen des Arrays, um die Kopiervorgänge zu minimieren. Man geht einfach davon aus, dass, wenn der anfängliche Load-Faktor von 16 Elementen nicht ausreicht, 32 beim nächsten Mal auch zu wenig sind – und gerade in unserem Beispiel ist das ja auch richtig. So sind es beim zweiten Mal bereits 32 Elemente, die dazukommen, beim nächsten Mal 64 usw. bis der maximale Load-Faktor mit 2048 Elementen erreicht ist.

In etwa entspricht also die Grundfunktionsweise einer Auflistung stark vereinfacht der folgenden Klasse:

```
Class DynamicList
    Implements IEnumerable

    Protected myStepIncreaser As Integer
    Protected myCurrentArraySize As Integer
    Protected myCurrentCounter As Integer
    Protected myArray() As Object

    Sub New()
        MyClass.New(16)
    End Sub

    Sub New(ByVal StepIncreaser As Integer)
        myStepIncreaser = StepIncreaser
        myCurrentArraySize = myStepIncreaser
        ReDim myArray(myCurrentArraySize)
    End Sub

    Sub Add(ByVal Item As Object)

        'Prüfen, ob aktuelle Arraygrenze erreicht wurde
        If myCurrentCounter = myCurrentArraySize - 1 Then
            'Neues Array mit mehr Speicher anlegen,
            'und Elemente hinüberkopieren. Dazu:

            'Neues Array wird größer:
            myCurrentArraySize += myStepIncreaser

            'temporäres Array erstellen
            Dim locTempArray(myCurrentArraySize - 1) As Object
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            'Elemente kopieren
            'Wichtig: Um das Kopieren müssen Sie sich,
            'anders als bei VB6, selber kümmern!
            Array.Copy(myArray, locTempArray, myArray.Length)

            'temporäres Array dem Memberarray zuweisen
            myArray = locTempArray

            'Beim nächsten Mal werden mehr Elemente reserviert!
            myStepIncreaser *= 2
        End If

        'Element im Array speichern
        myArray(myCurrentCounter) = Item

        'Zeiger auf nächstes Element erhöhen
        myCurrentCounter += 1

    End Sub

    'Liefert die Anzahl der vorhandenen Elemente zurück
    Public Overridable ReadOnly Property Count() As Integer
        Get
            Return myCurrentCounter
        End Get
    End Property

    'Erlaubt das Zuweisen und Abfragen
    Default Public Overridable Property Item(ByVal Index As Integer) As Object
        Get
            Return myArray(Index)
        End Get

        Set(ByVal Value As Object)
            myArray(Index) = Value
        End Set
    End Property

    'Liefert den Enumerator der Basis (dem Array) zurück
    Public Function GetEnumerator() As System.Collections.IEnumerator Implements
System.Collections.IEnumerable.GetEnumerator
        Return myArray.GetEnumerator
    End Function
End Class
```

Nun könnte man meinen, diese Vorgehensweise könnte sich zu einer Leistungsproblematik entwickeln, da alle paar Elemente der komplette Array-Inhalt kopiert werden muss.

**BEGLEITDATEIEN** Im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\DynamicList* finden Sie das Projekt *DynamicList*, das Ihnen das Gegenteil beweist:

Wie lange, glauben Sie, dauert es, ein Array mit 200.000 Zufallszahlen (mit Nachkommastellen) auf diese Weise anzulegen? 3 Sekunden? 2 Sekunden? Finden Sie es selbst heraus, indem Sie das Programm starten:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Anlegen von 200000 zufälligen Double-Elementen...
...in 21 Millisekunden!
```

Gerade mal 21 Millisekunden benötigt das Programm für diese Operation[3] – beeindruckend, wie schnell Visual Basic dieser Tage ist, finden Sie nicht?

Nun muss ich Ihnen leider verraten: Die Klasse `DynamicList` werden Sie nie benötigen. Bestandteil des .NET Frameworks ist nämlich eine Klasse, die das schon kann. Sogar noch ein kleines bisschen schneller. Und: Sie hat zusätzlich noch andere Möglichkeiten, die Ihnen die selbst gestrickte Klasse nicht bietet.

Im Beispielprojekt finden Sie eine `Sub Beispiel2`, die Ihnen die gleiche Prozedur mit der Klasse `ArrayList` demonstriert:

```
Sub Beispiel2()
    Dim locZeitmesser As New HighSpeedTimeGauge
    Dim locAnzahlElemente As Integer = 200000
    Dim locDynamicList As New ArrayList
    Dim locRandom As New Random(Now.Millisecond)

    Console.WriteLine("Anlegen von {0} zufälligen Double-Elementen...", locAnzahlElemente)
    locZeitmesser.Start()
    For count As Integer = 1 To locAnzahlElemente
        locDynamicList.Add(locRandom.NextDouble * locRandom.Next)
    Next
    locZeitmesser.Stop()
    Console.WriteLine("...in {0:#,##0} Millisekunden!", locZeitmesser.DurationInMilliSeconds)
    Console.ReadLine()

End Sub
```

Dessen Geschwindigkeit ist auch nicht von schlechten Eltern:

```
Anlegen von 200000 zufälligen Double-Elementen...
...in 19 Millisekunden!
```

Im Prinzip arbeitet `ArrayList` nach dem gleichen Verfahren, das Sie in `DynamicList` kennengelernt haben. `ArrayList` verfährt auch mit dem gleichen Trick, um möglichst viel Leistung herauszuholen. Es verdoppelt die jeweils nächste Größe des neuen Arrays im Vergleich zu der vorherigen Größe des Arrays. Damit reduziert sich der Gesamtaufwand des Kopierens erheblich. Da die Methode aber Bestandteil des .NET Frameworks ist, muss sie nicht zur Laufzeit »geJITted« werden, was der Geschwindigkeit zusätzlich zugute kommt.

[3] Auf einem Intel Core 2 Quad Q6600 übrigens. Lesern des Buches *Visual Studio 2005 – das Entwicklerbuch* (wie mehrfach schon erwähnt unter *www.activedevelop.de* kostenlos herunterladbar!) wird auffallen, dass das Ergebnis sogar eine Millisekunde langsamer ist, als auf dem Rechner, auf dem ich vor 3 Jahren das Visual Basic 2005-Buch schrieb. Sehr schön ist dabei zu sehen, dass neuere Rechner ihre Leistung kaum noch »vertikal«, über die Taktfrequenz, sondern nur noch horizontal über mehrere Prozessoren skalieren können – beim Laufenlassen dieses Beispiels wird nur einer von vier Prozessorkernen genutzt – ¾ des Prozessors liegen ungenutzt (schlimmer: in diesem Beispiel sogar unnutzbar!) brach. Um nicht nur alles sondern überhaupt mehr an Leistung aus modernen Prozessoren herauszukitzeln ist daher Multithreading-Entwicklung sehr entscheidend und sollte bei Programmportierungen – beispielsweise wenn sie ältere Borland-, MFC- oder VB6-Anwendungen auf .NET portieren wollen oder müssen – *unbedingt* berücksichtigt werden!

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Im Übrigen werden die Daten der einzelnen Elemente ja nicht wirklich bewegt. Lediglich die Zeiger auf die Daten werden kopiert – vorhandene Elemente bleiben im Managed Heap an ihrem Platz. Kapitel 4 erklärt Ihnen im Abschnitt »New oder nicht New – Wieso es sich bei Objekten um Verweistypen handelt « übrigens mehr zu diesem Thema.

# Wichtige Auflistungen der Base Class Library

Die BCL des .NET Frameworks enthält eine ganze Reihe von Auflistungs-Typen, von denen Sie einen der wichtigsten – `ArrayList` – schon im Einsatz gesehen haben. In diesem Abschnitt möchte ich Ihnen die wichtigsten dieser Auflistungen kurz vorstellen und darauf hinweisen, für welchen Einsatz sie am besten geeignet sind oder welche Besonderheiten Sie bei ihrem Gebrauch beachten sollten. Für eine genauere Beschreibung ihrer Eigenschaften und Methoden verwenden Sie bitte die Online-Hilfe von Visual Studio.

## ArrayList – universelle Ablage für Objekte

`ArrayList` können Sie als Container für Objekte aller Art verwenden. Sie instanziieren ein `ArrayList`-Objekt und weisen ihm mithilfe seiner `Add`-Funktion das jeweils nächste Element zu. Mit der `Default`-Eigenschaft `Item` können Sie schon vorhandene Elemente abrufen oder neu definieren. `AddRange` erlaubt Ihnen, die Elemente einer vorhandenen `ArrayList` einer anderen `ArrayList` hinzuzufügen.

Mit der `Count`-Eigenschaft eines `ArrayList`-Objektes finden Sie heraus, wie viele Elemente es beherbergt.

`Clear` löscht alle Elemente einer `ArrayList`. Mit `Remove` löschen Sie ein Objekt aus der `Arraylist`, das Sie als Parameter übergeben. Wenn mehrere gleiche Objekte (die `Equals`-Methode jedes Objektes wird dabei verwendet) existieren, wird das erste gefundene Objekt gelöscht. Mit `RemoveAt` löschen Sie ein Element an einer bestimmten Position. `RemoveRange` erlaubt Ihnen schließlich, einen ganzen Bereich von Array-Elementen ab einer bestimmten Position im `ArrayList`-Objekt zu löschen.

`ArrayList`-Objekte können in einfache Arrays umgewandelt werden. Dabei ist jedoch einiges zu beachten: Die Elemente der `ArrayList` müssen ausnahmslos alle dem Typ des Arrays entsprechen, in den sie umgewandelt werden sollen. Die Konvertierung nehmen Sie mit der `ToArray`-Methode des entsprechenden `ArrayList`-Objektes vor. Dabei bestimmen Sie, wenn Sie in ein typdefiniertes Array (wie `Integer()` oder `String()`) umwandeln, den Grundtyp (nicht den Arraytyp!) als zusätzlichen Parameter. Wenn Sie ein Array in eine `ArrayList` umwandeln wollen, verwenden Sie den entsprechenden Konstruktor der `ArrayList` – die entsprechende Konstruktorroutine nimmt anschließend die Konvertierung in eine `ArrayList` vor.

**HINWEIS** Bitte schauen Sie sich dazu auch das weiter unten gezeigte Listing an, insbesondere was die Konvertierungshinweise von `ArrayList`-Objekten in Arrays betrifft.

`ArrayList` implementiert die Schnittstelle `IEnumerable`. Aus diesem Grund stellt die Klasse einen Enumerator zur Verfügung, mit dem Sie mithilfe von `For/Each` durch die Elemente der `ArrayList` iterieren können. Beachten Sie dabei, den richtigen Typ für die Schleifenvariable zu verwenden. `ArrayList`-Elemente sind nicht typsicher, und eine Typ-Verletzung ist nur dann ausgeschlossen, wenn Sie genau wissen, welche Typen gespeichert sind (das nachfolgende Beispiel demonstriert diesen Fehler recht anschaulich):

Hier die Beispiele, die das gerade Gesagte näher erläutern und den Code dazu dokumentieren:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**BEGLEITDATEIEN** Sie finden die im Folgenden besprochenen Codeausschnitte im Verzeichnis *.\Samples\Chapter05 - ArraysCollections\CollectionsDemo*

```
Sub ArrayListDemo()
    Dim locMännerNamen As String() = {"Jürgen", "Uwe", "Klaus", "Christian", "José"}
    Dim locFrauenNamen As New ArrayList
    Dim locNamen As ArrayList

    'ArrayList aus vorhandenem Array erstellen.
    locNamen = New ArrayList(locMännerNamen)

    'ArrayList mit Add auffüllen.
    locFrauenNamen.Add("Ute") : locFrauenNamen.Add("Miriam")
    locFrauenNamen.Add("Melanie") : locFrauenNamen.Add("Anja")
    locFrauenNamen.Add("Stephanie") : locFrauenNamen.Add("Heidrun")

    'Arraylist einer anderen Arraylist hinzufügen:
    locNamen.AddRange(locFrauenNamen)

    'Arraylist in eine Arraylist einfügen.
    Dim locHundenamen As String() = {"Hasso", "Bello", "Wauzi", "Wuffi", "Basko", "Franz"}
    'Einfügen *vor* dem 6. Element
    locNamen.InsertRange(5, locHundenamen)

    'ArrayList in ein Array zurückwandeln.
    Dim locAlleNamen As String()

    'Vorsicht: Fehler!
    'locAlleNamen = DirectCast(locNamen.ToArray, String())

    'Vorsicht: Ebenfalls Fehler!
    'locAlleNamen = DirectCast(locNamen.ToArray(GetType(String())), String())

    'So ist es richtig.
    locAlleNamen = DirectCast(locNamen.ToArray(GetType(String)), String())

    'Repeat legt eine Arraylist aus wiederholten Items an.
    locNamen.AddRange(ArrayList.Repeat("Dublettenname", 10))

    'Ein Element im Array ändern.
    locNamen(10) = "Fiffi"
    'Mit der Item-Eigenschaft geht es auch:
    locNamen.Item(13) = "Miriam"

    'Löschen des ersten zutreffenden Elementes aus der Liste.
    locNamen.Remove("Basko")

    'Löschen eines Elementes an einer bestimmten Position.
    locNamen.RemoveAt(4)

    'Löschen eines bestimmten Bereichs aus der ArrayList mit RemoveRange.
    'Count ermittelt die Anzahl der Elemente in der ArrayList.
    locNamen.RemoveRange(locNamen.Count - 6, 5)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Ausgeben der Elemente über die Default-Eigenschaft der ArrayList (Item).
        For i As Integer = 0 To locNamen.Count - 1
            Console.WriteLine("Der Name Nr. {0} lautet {1}", i, locNamen(i).ToString)
        Next

        'Anderes als ein String-Objekt der ArrayList hinzufügen,
        'um den folgenden Fehler "vorzubereiten".
        locNamen.Add(New FileInfo("C:\TEST.TXT"))

        'Diese Schleife kann nicht bis zum Ende ausgeführt werden,
        'da ein Objekt nicht vom Typ String mit von der Partie ist!
        For Each einString As String In locNamen
            'Hier passiert irgendetwas mit dem String.
            'nicht von Interesse, deswegen kein Rückgabewert.
            einString.EndsWith("Peter")
        Next
        Console.ReadLine()
    End Sub
```

Wenn Sie dieses Beispiel laufen lassen, dann sehen Sie zunächst die erwarteten Ausgaben auf dem Bildschirm. Doch der Programmcode erreicht nie die Anweisung `Console.ReadLine`, um auf Ihre letzte Bestätigung zu warten. Stattdessen löst er eine Ausnahme aus, etwa wie in Abbildung 5.2 zu sehen.

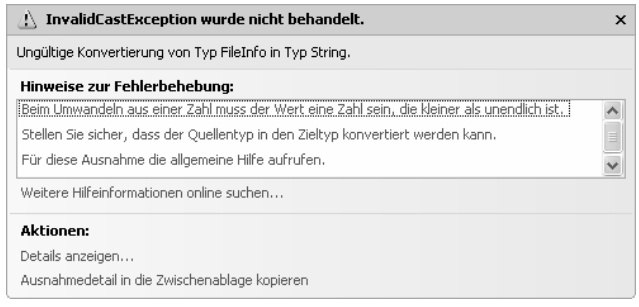



**Abbildung 5.2** Achten Sie beim Iterieren mit For/Each durch eine Auflistung darauf, dass die Elemente dem Schleifenvariablentyp entsprechen, um solche Ausnahmen zu vermeiden!

Der Grund dafür: Das letzte Element in der `ArrayList` ist kein String. Aus diesem Grund wird die Ausnahme ausgelöst. Achten Sie deshalb stets darauf, dass die Schleifenvariable eines `For/Each`-Konstrukts immer den Typen entspricht, die in einer Auflistung gespeichert sind.

Sie können diesem Problem entgehen, indem Sie eine Auflistung mithilfe von *Generics* (*Generika*, auf Microsoft-Deutsch) dazu zwingen, homogen zu sein (also nur Elemente des gleichen Typs zu verarbeiten). Für den späteren Einsatz von *LINQ to Objects* (siehe Kapitel 8) ist das sogar eine Notwendigkeit.

### Hashtables – für das Nachschlagen von Objekten

`Hashtable`-Objekte sind das ideale Werkzeug, wenn Sie eine Datensammlung aufbauen wollen, aber die einzelnen Objekte nicht durch einen numerischen Index, sondern durch einen Schlüssel abrufen wollen. Ein Beispiel soll das verdeutlichen.

Angenommen, Sie haben eine Adressenverwaltung programmiert, bei der Sie einzelne Adressen durch eine Art Matchcode abrufen wollen (Kundennummer, Lieferantennummer, was auch immer). Bei der Verwendung einer `ArrayList` müssten Sie schon einigen Aufwand betreiben, um an ein Array-Element auf Basis des Matchcode-Namens zu gelangen: Sie müssten zum Finden eines Elements in der Liste für die verwendete Adressklasse eine `CompareTo`-Methode implementieren, damit die Liste mittels `Sort` sortiert werden könnte.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Anschließend könnten Sie mit `BinarySearch` das Element finden – vorausgesetzt, die `CompareTo`-Methode würde eine Instanz der Adressklasse über ihren Matchcode-Wert vergleichen.

`Hashtable`-Objekte vereinfachen ein solches Szenario ungemein: Wenn Sie einer Hashtable ein Objekt hinzufügen, dann nimmt dessen `Add`-Methode nicht nur das zu speichernde Objekt (den hinzuzufügenden Wert) entgegen, sondern auch ein weiteres. Dieses zusätzliche Objekt (das genau genommen als erster Parameter übergeben wird) stellt den Schlüssel – den Key – zum Wiederauffinden des Objektes dar. Sie rufen ein Objekt aus einer Hashtable anschließend nicht wie bei der `ArrayList` mit

```
Element = eineArrayList(5)
```

ab, sondern mit dem entsprechenden Schlüssel, etwa:

```
Element = eineHashtable("ElementKey")
```

Voraussetzung bei diesem Beispiel ist natürlich, dass der Schlüssel zuvor ein entsprechender String gewesen ist.

### Queue – Warteschlangen im FIFO-Prinzip

»First in first out« (als erstes rein, als erstes raus) – nach diesem Muster arbeitet die `Queue`-Klasse der BCL. Angewendet haben Sie dieses Prinzip selbst schon sicherlich einige Male in der Praxis – und zwar immer dann, wenn Sie unter Windows mehrere Dokumente hintereinander gedruckt haben. Das Drucken unter Windows funktioniert gemäß dem Warteschlangenprinzip. Das Dokument, das als Erstes in die Warteschlange eingereiht (*enqueue* – einreihen) wurde, wird als Erstes verarbeitet und anschließend wieder aus ihr entfernt (*dequeue* – ausreihen). Aus diesem Grund verwenden Sie die Methoden `Enqueue`, um Elemente der Queue hinzuzufügen und `Dequeue`, um sie zurückzubekommen und gleichzeitig aus der Warteschlange zu entfernen.

**BEGLEITDATEIEN** Falls Sie mit der `Queue`-Klasse experimentieren möchten, verwenden Sie dazu am besten nochmals das *CollectionsDemo*-Projekt, das Sie unter *.\Samples\Chapter05 - ArraysCollections\CollectionsDemo* finden. Verändern Sie das Programm so, dass es die `Sub QueueDemo` aufruft, um das folgende Beispiel nachzuvollziehen:

```
Sub QueueDemo()

    Dim locQueue As New Queue
    Dim locString As String

    locQueue.Enqueue("Erstes Element")
    locQueue.Enqueue("Zweites Element")
    locQueue.Enqueue("Drittes Element")
    locQueue.Enqueue("Viertes Element")

    'Nachschauen, was am Anfang steht, ohne es zu entfernen.
    Console.WriteLine("Element am Queue-Anfang:" + locQueue.Peek().ToString)
    Console.WriteLine()

    'Iterieren funktioniert auch.
    For Each locString In locQueue
        Console.WriteLine(locString)
    Next
    Console.WriteLine()
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Alle Elemente aus Queue entfernen und Ergebnis im Konsolenfenster anzeigen.
        Do
            locString = CStr(locQueue.Dequeue)
            Console.WriteLine(locString)
        Loop Until locQueue.Count = 0
        Console.ReadLine()
    End Sub
```

Wenn Sie dieses Programm ablaufen lassen, produziert es folgende Ausgabe im Konsolenfenster:

```
Element am Queue-Anfang:Erstes Element

Erstes Element
Zweites Element
Drittes Element
Viertes Element

Erstes Element
Zweites Element
Drittes Element
Viertes Element
```

## Stack – Stapelverarbeitung im LIFO-Prinzip

Die `Stack`-Klasse arbeitet nach dem Prinzip »Last in first out« (»als letztes rein, als erstes raus«), arbeitet also genau umgekehrt zum FIFO-Prinzip der `Queue`-Klasse. Mit der `Push`-Methode schieben Sie ein Element auf den Stapel, mit `Pull` ziehen Sie es wieder herunter und erhalten es damit zurück. Das Element, das Sie zuletzt auf den Stapel geschoben haben, wird also mit `Pull` auch als Erstes wieder entfernt.

**BEGLEITDATEIEN** Falls Sie mit der `Stack`-Klasse experimentieren möchten, verwenden Sie das *CollectionsDemo*-Projekt, das Sie unter *.\Samples\Chapter05 - ArraysCollections\CollectionsDemo* finden. Verändern Sie das Programm so, dass es die `Sub StackDemo` aufruft, um das folgende Beispiel nachzuvollziehen:

```
    Sub StackDemo()
        Dim locStack As New Stack
        Dim locString As String

        locStack.Push("Erstes Element")
        locStack.Push("Zweites Element")
        locStack.Push("Drittes Element")
        locStack.Push("Viertes Element")

        'Nachschauen, was oben auf dem Stapel liegt, ohne das Element zu entfernen.
        Console.WriteLine("Element zu oberst auf dem Stapel: " + locStack.Peek.ToString)
        Console.WriteLine()

        'Iterieren funktioniert auch.
        For Each locString In locStack
            Console.WriteLine(locString)
        Next
        Console.WriteLine()
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Alle Elemente vom Stack ziehen und Ergebnis im Konsolenfenster anzeigen.
        Do
            locString = CStr(locStack.Pop)
            Console.WriteLine(locString)
        Loop Until locStack.Count = 0
        Console.ReadLine()
    End Sub
```

Wenn Sie dieses Programm ablaufen lassen, produziert es folgende Ausgabe im Konsolenfenster:

```
Element zu oberst auf dem Stapel: Viertes Element

Viertes Element
Drittes Element
Zweites Element
Erstes Element

Viertes Element
Drittes Element
Zweites Element
Erstes Element
```

### SortedList – Elemente ständig sortiert halten

Wenn Sie Elemente schon direkt nach dem Einfügen in der richtigen Reihenfolge in einer Auflistung halten wollen, dann ist die `SortedList`-Klasse das richtige Werkzeug für diesen Zweck. Allerdings sollten Sie beachten: Von allen Auflistungsklassen ist die `SortedList`-Klasse diejenige, die die meisten Ressourcen verschlingt. Für zeitkritische Applikationen sollten Sie überlegen, ob Sie Ihre Daten auch anders organisieren und stattdessen lieber auf eine unsortierte Hashtable oder gar auf `ArrayList` zurückgreifen können.

Der Vorteil von `SortedList` ist, dass sie quasi aus einer Mischung von `ArrayList`- und `Hashtable`-Funktionen besteht (obwohl sie algorithmisch gesehen, überhaupt nichts mit `Hashtable` zu tun hat). Sie können auf der einen Seite über einen Schlüssel, auf der anderen Seite aber auch über einen Index auf die Elemente von `SortedList` zugreifen.

Das folgende erste Beispiel zeigt den generellen Umgang mit `SortedList`.

**BEGLEITDATEIEN** Sie finden dieses Projekt unter *.\Samples\Chapter05 - ArraysCollections\SortedListDemo*. Es besteht aus drei Codedateien. In der Datei *Daten.vb* finden Sie die schon bekannte `Adresse`-Klasse (bekannt, falls Sie die vorherigen Abschnitte ebenfalls durchgearbeitet haben) – allerdings in leicht veränderter Form. Der Matchcode der Zufallsadressen beginnt in dieser Version mit einer laufenden Nummer und endet mit der Buchstabenkombination des Nach- und Vornamens. Damit wird vermieden, dass eine Sortierung des Matchcodes grob auch die Adressen nach Namen und Vornamen sortiert und etwaige Nachweise eines bestimmten Programmverhaltens nicht geführt werden können.

```
Module SortedListDemo
    Sub Main()
        Dim locZufallsAdressen As ArrayList = Adresse.ZufallsAdressen(6)
        Dim locAdressen As New SortedList
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Console.WriteLine("Ursprungsanordnung:")
        For Each locAdresse As Adresse In locZufallsAdressen
            Console.WriteLine(locAdresse)
            locAdressen.Add(locAdresse.Matchcode, locAdresse)
        Next

        'Zugriff per Index:
        Console.WriteLine()
        Console.WriteLine("Zugriff per Index:")
        For i As Integer = 0 To locAdressen.Count - 1
            Console.WriteLine(locAdressen.GetByIndex(i).ToString)
        Next

        Console.WriteLine()
        Console.WriteLine("Zugriff per Index:")
        'Zugriff per Enumerator
        For Each locDE As DictionaryEntry In locAdressen
            Console.WriteLine(locDE.Value.ToString)
        Next
        Console.ReadLine()
    End Sub
End Module
```

Wenn Sie dieses Programm starten, generiert es in etwa die folgenden Ausgaben im Konsolenfenster (die Adressen werden zufällig generiert, deswegen kann die Darstellung in Ihrem Konsolenfenster natürlich wieder von der hier gezeigten abweichen).

```
Ursprungsanordnung:
00000005PlKa: Plenge, Katrin, 26201 Liebenburg
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim

Zugriff per Index:
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000005PlKa: Plenge, Katrin, 26201 Liebenburg

Zugriff per Index:
00000000AdKa: Ademmer, Katrin, 49440 Unterschleißheim
00000001LöLo: Löffelmann, Lothar, 21237 Lippetal
00000002HoBa: Hollmann, Barbara, 96807 Liebenburg
00000003AlMa: Albrecht, Margarete, 65716 Bad Waldliesborn
00000004PlKa: Plenge, Katrin, 93436 Liebenburg
00000005PlKa: Plenge, Katrin, 26201 Liebenburg
```

Sie erkennen, dass die Liste in der Tat nach dem Schlüssel umsortiert wurde. Dies gilt sowohl für den Zugriff über den Index als auch über den Enumerator mit `For/Each`.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 6

# Generics (Generika) und generische Auflistungen

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Einführung

Generics – oder »Generika« wie Microsoft die so schöne deutsche Übersetzung dafür fand – sind nicht neu in Visual Basic 2008. Das erste Argument, sie dennoch in diesem Buch zu behandeln, ist, dass Generics und vor allem generische Auflistungen entscheidend für das Verständnis des nächsten Kapitels sind, in dem es um die Grundlagen zu LINQ geht. Das zweite Argument: Nicht jeder VB-Entwickler hat bereits mit Generics und generischen Auflistungen gearbeitet – eine Veranschaulichung der Konzepte kann daher so oder so nicht schaden, denn auch wenn Sie nicht vorhaben, in Zukunft mit LINQ zu arbeiten: das Verstehen von Generics und das Verwenden generischer Auflistungen wird Ihnen in jedem Fall helfen, Ihre Anwendungen sicherer zu machen und sie damit auch professionellen Standards genügen zu lassen.

## Generics: Verwenden einer Codebasis für verschiedene Typen

Wenn Sie Methoden und Eigenschaften entwickeln, haben diese unter Umständen einen Nachteil: Sie verarbeiten, wenn sie typsicher sein sollen, nur einen bestimmten Datentyp.

Die Alternative dazu ist, dass Sie einen Typ schaffen, der die Aufnahme beliebiger Datentypen durch den auf `Object` basierenden Einsatz ermöglicht, doch ein solcher Typ ist dann nicht typsicher und kann zur Laufzeit Ausnahmen auslösen, mit denen Sie nicht rechnen.

**BEGLEITDATEIEN** Im Folgenden sehen Sie zunächst den Klassencode, der einen Adresseneintrag verwaltet – dieses Projekt finden Sie im Verzeichnis *.\Samples\Chapter06 - Generics\Generics01*.

Dieses Beispiel verwendet die aus dem vorherigen Kapitel bekannte Klasse `DynamicList` in leicht modifizierter Form. Das Modul verwendet eine Instanz von `DynamicList` und fügt ihr ein paar Elemente hinzu. Diese Elemente gibt es anschließend in einer `For/Each`-Schleife wieder im Konsolenfenster aus:

```
Module mdlMain

    Sub Main()
        Dim locListe As New DynamicList
        locListe.Add(123.32)
        locListe.Add(126.32)
        locListe.Add(124.52)
        locListe.Add(29.99)
        locListe.Add(13.54)
        'Der wird Probleme machen!
        locListe.Add(#12/31/2005 4:00:00 PM#)
        locListe.Add(43.32)
        For Each locItem As Double In locListe
            Console.WriteLine(locItem)
        Next

        Console.WriteLine()
        Console.WriteLine("Taste drücken zum Beenden!")
        Console.ReadKey()
    End Sub

End Module
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wenn Sie dieses Programm starten, sehen Sie anschließend Folgendes auf dem Bildschirm:

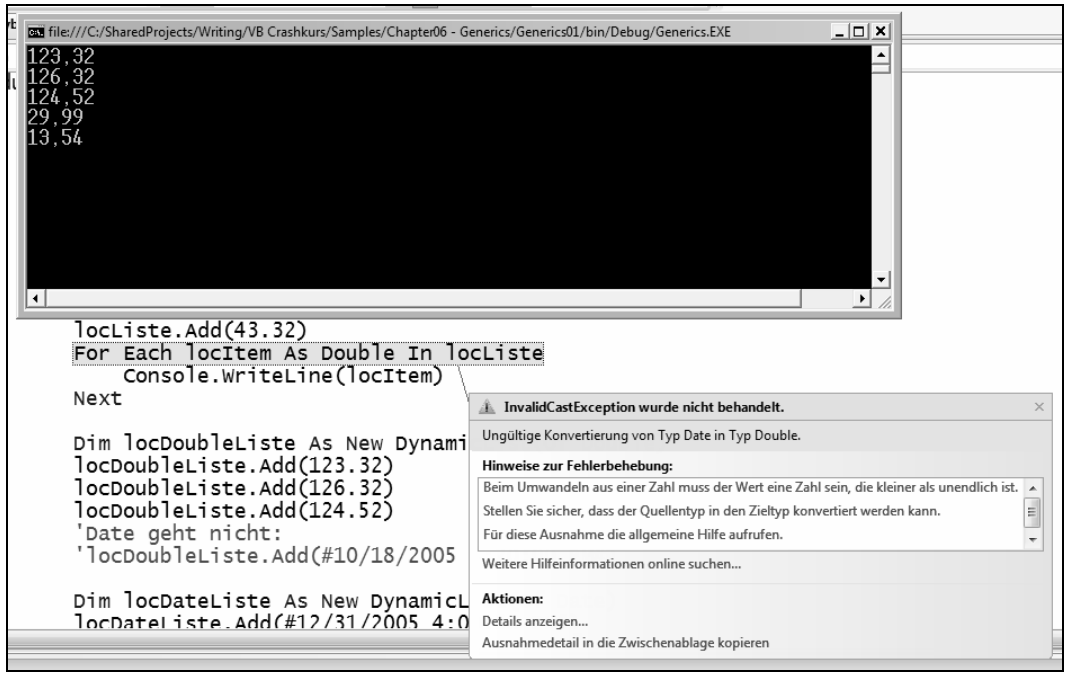



**Abbildung 6.1** Die Liste wird nur bis zum *Date* Element ausgegeben; das *Date*-Element ist nämlich nicht in *Double* konvertierbar

Es ist klar, wieso das passiert. Wir haben eine Liste mit reinen `Double`-Elementen aufbauen wollen, aber uns ist ein `Date`-Element »dazwischengerutscht«. Solche Fehler sind in einem so einfachen Programm, wie wir es hier als Beispiel verwenden, noch auf den ersten Blick zu erkennen – doch in größeren Projekten sollte man solche Fehler von vornherein zu vermeiden versuchen.

# Lösungsansätze

Und wie kann man das machen? Indem man eine Klasse wie die `DynamicList` typsicher macht. Indem man sie von vornherein erst gar keinen allgemein gültigen Datentyp wie `Object` akzeptieren lässt, sondern ausschließlich Werte vom Typ `Double`.

Und um das zu erreichen, nehmen wir uns den Quellcode der `DynamicList` vor, und führen die entsprechenden Änderungen durch. Konsequenterweise nennen wir diese Klasse dann auch `DynamicListDouble` (wie sie im angesprochenen Beispielprojekt übrigens bereits in einer eigenen Klassendatei vorhanden ist). Im folgenden Listing finden Sie all die Stellen in Fettschrift markiert, an denen der Datentyp `Object` in `Double` geändert wurde.

```
Class DynamicListDouble
    Implements IEnumerable

    Protected myStep As Integer = 4         ' Schrittweite, um die das Array erhöht wird.
    Protected myCurrentArraySize As Integer ' Aktuelle Array-Größe
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Protected myCurrentCounter As Integer   ' Zeiger auf aktuelles Element
    Protected myArray() As Double           ' Array mit den Elementen.

    Sub New()
        myCurrentArraySize = myStep
        ReDim myArray(myCurrentArraySize - 1)
    End Sub

    Sub Add(ByVal Item As Double)

        'Element im Array speichern
        myArray(myCurrentCounter) = Item

        'Zeiger auf nächstes Element erhöhen
        myCurrentCounter += 1

        'Prüfen, ob aktuelle Arraygrenze erreicht wurde
        If myCurrentCounter = myCurrentArraySize - 1 Then
            'Neues Array mit mehr Speicher anlegen,
            'und Elemente hinüberkopieren. Dazu:

            'Neues Array wird größer:
            myCurrentArraySize += myStep

            'temporäres Array erstellen
            Dim locTempArray(myCurrentArraySize - 1) As Double
            Array.Copy(myArray, locTempArray, myArray.Length)
            myArray = locTempArray
        End If
    End Sub
.
.
.
End Class
```

Wenn Sie die Klasse auf diese Weise abgeändert und das eigentliche Programm wie folgt modifiziert haben,

```
Module mdlMain

    Sub Main()
        Dim locListe As New DynamicListDouble
        locListe.Add(123.32)
        locListe.Add(126.32)
.
.
.
```

zeigt Ihnen Visual Basic schon zur Entwurfszeit eine Fehlermeldung, wie Sie sie in Abbildung 6.2 sehen können.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

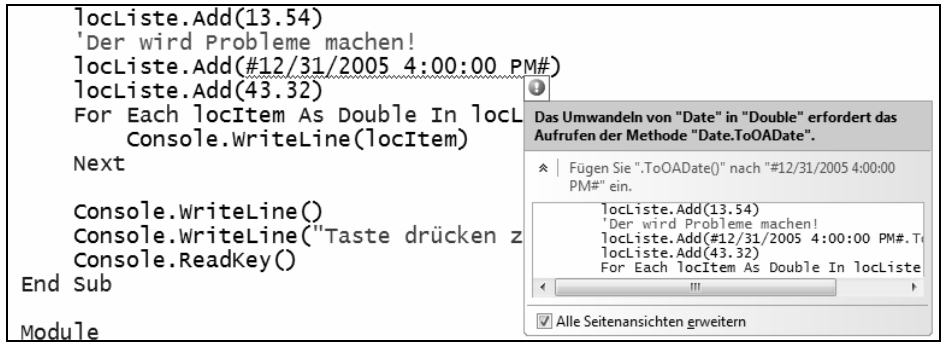



**Abbildung 6.2** Bei typsicheren Klassen sehen Sie Fehlermeldungen bei »falschen« Typen bereits im Editor zur Entwurfszeit – wenngleich die Fehlermeldungen ab und an über das Ziel hinausschießen

Gut – diese Fehlermeldung an dieser Stelle schießt ein wenig über das Ziel hinaus; hätte es der Editor dabei belassen, uns über den falschen Typ an dieser Stelle zu informieren, wäre das ausreichend gewesen. Aber immerhin: Sie sehen auf jeden Fall durch die Verwendung der typsicheren Klasse schon zur Entwurfszeit, dass Sie einen Typen verwendet haben, den Sie mit dieser Klasse nicht verwenden dürfen.

# Typengeneralisierung durch den Einsatz generischer Datentypen

Nun ist das Entwickeln einer Klasse wie `DynamicList` schon ein wenig aufwändiger. Und es wird ja schließlich Klassen und Strukturen geben, die noch viel, viel komplexer sind.

Doch gleichzeitig wird es gerade bei Klassen oder Strukturen, die große Datenmengen verwalten, immer wieder vorkommen, dass Sie sie für den Einsatz unterschiedlichster Typen verwenden wollen – für unser `DynamicList`-Beispiel trifft diese Aussage mehr als zu, denn:

Auch Zeichenketten ließen sich mit der Klasse wunderbar verwalten. Und auch `Integer`-Werte. Und auch `Decimals`. Und auch … – eigentlich ist dieser Typ für alle Datentypen und Objektinstanzen geeignet, die Sie in größeren Mengen in einer Liste speichern wollen.

Doch als Programmierer sollten Sie aus den genannten Gründen immer auch auf Typsicherheit bestehen, und so bliebe Ihnen bislang eigentlich nur die Möglichkeit, …

- … für jeden benötigten Datentyp eine neue Version der verarbeitenden Klasse zu erstellen. Dieser Aufwand ist allerdings enorm groß, und zieht ein mindestens ebenso großes Problem nach sich: Finden Sie einen Fehler in einer Klasse, müssen Sie diesen in allen Klassen ändern, die sich nur durch ihren verarbeitenden Typ unterscheiden (`DynamicListDouble`, `DynamicListString`, `DynamicListDate` und welche Klasse Sie auch immer sonst noch eingerichtet hätten).
- … eine Klasse auf Basis einer Schnittstelle zu erstellen, mit der die Typen durch Vererbung anpassbar sind. Damit hält sich der Pflegeaufwand in Grenzen, da eine Fehlerbehebung in der Basisklasse sich natürlich auch in den Klassenableitungen widerspiegelt. Doch solche Klassen zu implementieren erfordert extrem abstraktes Denken und sorgfältige Planung, und dafür steht nicht unbedingt immer die Zeit zur Verfügung.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Mit generischen Datentypen wird das anders. Bei generischen Datentypen (auf englisch »Generics« – für den Fall, dass Sie mal nach englischen Artikeln googlen müssen) legen Sie sich während der Entwicklung überhaupt noch nicht fest, welchen Typ Ihre Klasse oder Struktur später einmal verarbeiten soll. Sie arbeiten stattdessen mit so genannten Typparametern, durch die der Typ – dem JITter sei Dank – erst bei der ersten Verwendung zur Laufzeit ersetzt wird.

Mit anderen Worten: Das, was Sie mit dem Kopieren und Typanpassen Ihres Codes zur Entwicklungszeit manuell machen, dafür sorgen JITter und die Technik der Generics zur Laufzeit automatisch. Vereinfacht gesagt: So, wie Sie bei Word mit Suchen und Ersetzen arbeiten können, wird der IML-Code Ihrer generischen Klasse kopiert, und alle Typparameter werden durch den angegebenen Typ ersetzt.[1]

In der Praxis und für unser `DynamicList`-Beispiel sieht das wie folgt aus:

Anstelle sich von vornherein für einen Datentyp wie `Double`, `Integer` oder `String` zu entscheiden, platzieren Sie an den entscheidenden Stellen eine Art Platzhalter – einen Typparameter –, den Sie im Kopf der Klasse mit dem Zusatz `Of` benennen, etwa so:

```
Class DynamicList(Of flexiblerDatentyp)
```

Und anstatt anschließend innerhalb der Klasse einen fixen Datentyp zu verwenden, setzen Sie diesen Typparameter als Stellvertreter ein. Für unsere `DynamicList`-Klasse bedeutet das:

```
Class DynamicList(Of flexiblerDatentyp)
    Implements IEnumerable

    Protected myStep As Integer = 4          ' Schrittweite, um die das Array erhöht wird.
    Protected myCurrentArraySize As Integer  ' Aktuelle Array-Größe
    Protected myCurrentCounter As Integer    ' Zeiger auf aktuelles Element
    Protected myArray() As flexiblerDatentyp ' Array mit den Elementen.

    Sub New()
        myCurrentArraySize = myStep
        ReDim myArray(myCurrentArraySize - 1)
    End Sub

    Sub Add(ByVal Item As flexiblerDatentyp)

        myArray(myCurrentCounter) = Item
        myCurrentCounter += 1
        If myCurrentCounter = myCurrentArraySize - 1 Then
            myCurrentArraySize += myStep

            'temporäres Array erstellen
            Dim locTempArray(myCurrentArraySize - 1) As flexiblerDatentyp

            'Elemente kopieren;
            Array.Copy(myArray, locTempArray, myArray.Length)
            myArray = locTempArray
        End If
    End Sub
    'Liefert die Anzahl der vorhandenen Elemente zurück
    Public Overridable ReadOnly Property Count() As Integer
```

[1] Ganz so einfach geht es natürlich nicht. Es gibt für JITter bzw. Compiler durchaus die Möglichkeit, Codegemeinsamkeiten in generischen Klassen bestehen zu lassen, und nur dort neuen Code zu generieren, wo es nicht anders möglich ist.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Get
            Return myCurrentCounter
        End Get
    End Property

    'Erlaubt das Zuweisen
    Default Public Overridable Property Item(ByVal Index As Integer) As flexiblerDatentyp
        Get
            Return myArray(Index)
        End Get
        Set(ByVal Value As flexiblerDatentyp)
            myArray(Index) = Value
        End Set
    End Property

    Public Function GetEnumerator() As System.Collections.IEnumerator _
                        Implements System.Collections.IEnumerable.GetEnumerator
        Dim locTempArray(myCurrentArraySize) As flexiblerDatentyp
        Array.Copy(myArray, locTempArray, myArray.Length)
        Return myArray.GetEnumerator
    End Function
End Class
```

Und nun können Sie `DynamicList` für jeden Datentyp verwenden, den Sie möchten, und Sie müssen dabei nicht auf Typsicherheit verzichten, wie Abbildung 6.3 zeigt.

In der Grafik sehen Sie zweierlei. Zum einen, wie Sie einen generischen Datentyp anwenden. In der `Sub Main` des Moduls wird die generische Klasse einmal auf Basis des Datentyps `Double` definiert

```
Dim locDoubleListe As New DynamicList(Of Double)
```

und einmal auf Basis des Datentyps `Date`:

```
Dim locDateListe As New DynamicList(Of Date)
```

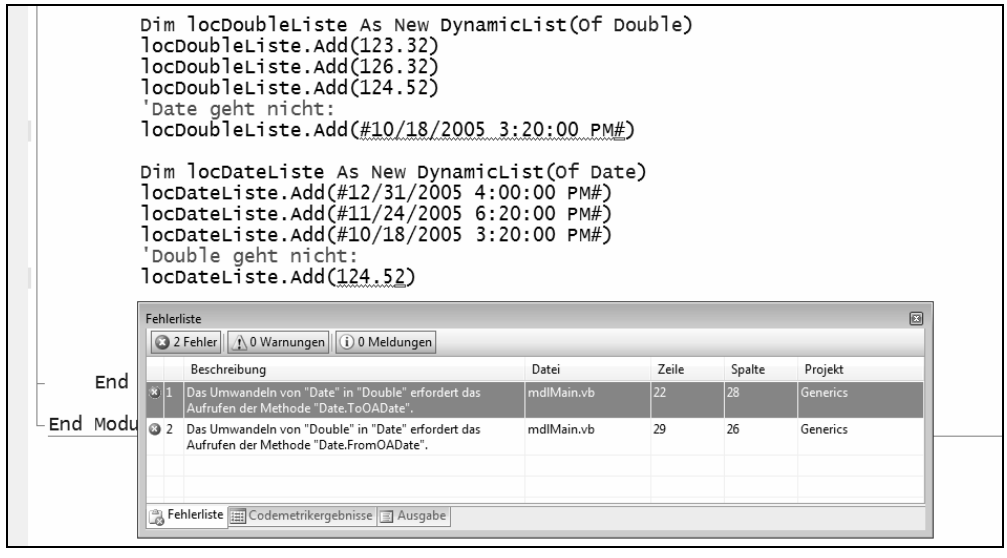



**Abbildung 6.3** Mit *Of* bestimmen Sie, für welchen Datentyp Ihre generische Klasse zur Anwendung kommen soll

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Und anhand der Fehlerliste, die Sie ebenfalls in der Grafik sehen können, erkennen Sie auch, dass beide »Typversionen« der Klasse auf ihre Weise typsicher sind. Sie können der einen nur Werte vom Typ `Double` und der anderen nur Werte vom Typ `Date` hinzufügen. Jeder Versuch, einer Liste einen jeweils anderen Typ unterzujubeln, wird schon zur Entwurfszeit mit einer entsprechenden Fehlermeldung bestraft.

**HINWEIS** Es gibt bei der Verwendung von Generics die Möglichkeit, den Compiler den einzusetzenden Typ auf Basis der tatsächlich übergebenden Parameter bestimmen zu lassen. Dieses Verfahren nennt man *Typrückschluss für Typparameter*, und viele Beispiele dafür finden sich bei der Verwendung der verschiedenen Überladungsversionen der Erweiterungsmethoden, die Sie in der `Enumerable`-Klasse finden, und auf denen die gesamte *Linq to Objects*-Infrastruktur basiert. Beispiele für *Typrückschluss für Typparameter* finden Sie im nächsten Kapitel.

# Beschränkungen (Constraints)

Im gezeigten Beispiel können Sie eine `DynamicList` so definieren, dass sie sich aus jedem beliebigen Typ bilden kann. Unter bestimmten Umständen kann das nicht erwünscht sein, und zwar genau dann, wenn Sie innerhalb einer generischen Klasse andere generische Typen verwenden, deren Typ Sie aber zur Entwicklungszeit noch nicht kennen.

**BEGLEITDATEIEN** Im Folgenden sehen Sie zunächst den Klassencode, der einen Adresseneintrag verwaltet – dieses Projekt finden Sie im Verzeichnis *.\Samples\Chapter06 - Generics\Generics02*.

## Beschränkungen für generische Typen auf eine bestimmte Basisklasse

Dazu ein Beispiel: Angenommen, Sie haben eine Anwendung geschaffen, die die verschiedensten Körper (Quader, Kugeln, Pyramiden) berechnen, verwalten und darstellen muss. Sie möchten jetzt eine generische Klasse schaffen, die nicht nur die verschiedenen Typen von Körpern in einer Liste wie der `DynamicList` speichert, sondern Sie möchten, dass diese Klasse auch deren Gesamtvolumen berechnen soll.

Nehmen wir weiter an, dass es in unserem Beispiel eine Basisklasse gibt, auf der alle Körperklassen basieren. Diese Basisklasse speichert dann die Position und die Farbe eines Körpers. Die einzelnen Körperklassen leiten anschließend von dieser Körperbasisklasse ab, damit sie die für alle gleich bleibenden Eigenschaften nicht ständig wieder implementieren müssen. Eine solche Klassenerbfolge sieht dann in etwa folgendermaßen aus:

```
Imports System.Drawing

'Stellt die Grundeigenschaften eines Körpers bereit
Public MustInherit Class KörperBasis

    Private myFarbe As Color
    Private myPosition As Point

    MustOverride ReadOnly Property Volumen() As Double

    Public Property Farbe() As Color
        Get
            Return myFarbe
        End Get
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Set(ByVal value As Color)
            myFarbe = value
        End Set
    End Property

    Public Property Position() As Point
        Get
            Return myPosition
        End Get
        Set(ByVal value As Point)
            myPosition = value
        End Set
    End Property
End Class

'Stellt die Grundeigenschaften eines Quaders bereit
Public Class Quader
    Inherits KörperBasis

    Private mySeitenLänge_a As Double
    Private mySeitenLänge_b As Double
    Private mySeitenLänge_c As Double

    Sub New(ByVal a As Double, ByVal b As Double, ByVal c As Double)
        mySeitenLänge_a = a
        mySeitenLänge_b = b
        mySeitenLänge_c = c
    End Sub

    Public Overrides ReadOnly Property Volumen() As Double
        Get
            Return mySeitenLänge_a * mySeitenLänge_b * mySeitenLänge_c
        End Get
    End Property
End Class

'Stellt die Grundeigenschaften einer Pyramide bereit
Public Class Pyramide
    Inherits KörperBasis

    Private myGrundfläche As Double
    Private myHöhe As Double

    Sub New(ByVal Grundfläche As Double, ByVal Höhe As Double)
        myGrundfläche = Grundfläche
        myHöhe = Höhe
    End Sub

    Public Overrides ReadOnly Property Volumen() As Double
        Get
            Return (myGrundfläche * myHöhe) / 3
        End Get
    End Property
End Class
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Rein theoretisch könnten wir natürlich nun die bereits vorhandene `DynamicList`-Klasse für die Speicherung der `Körper`-Objekte verwenden, wie der Code der folgenden Abbildung zeigt:

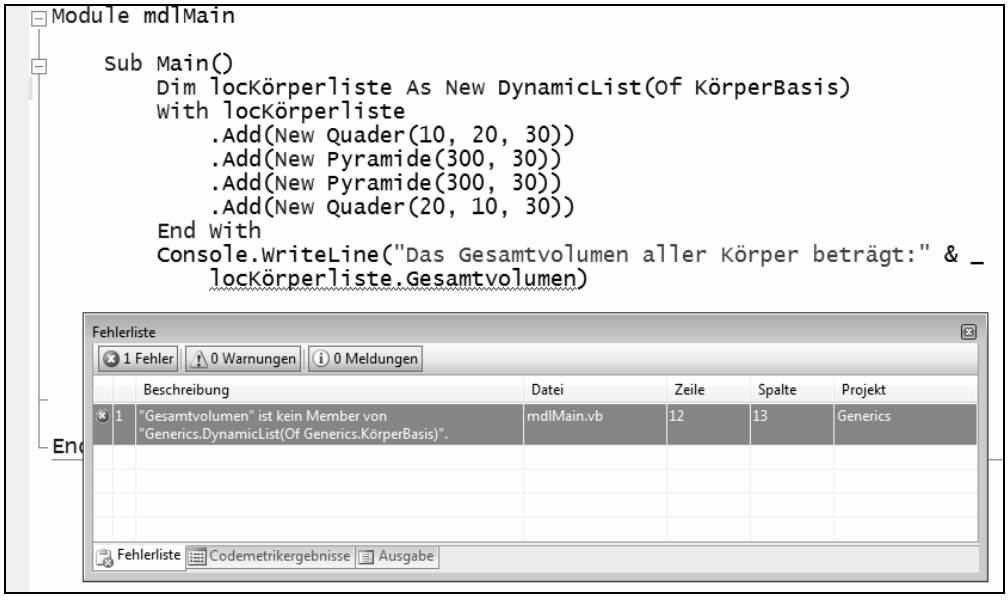



**Abbildung 6.4** Die *DynamicList* soll auch eine *Gesamtvolumen*-Eigenschaft zur Verfügung stellen, doch die ist zurzeit noch nicht implementiert

Doch dabei ergibt sich eine Kette von Problemen. Wie in Abbildung 6.4 zu sehen, möchten wir eine Eigenschaft der Liste verwenden, die es zu diesem Zeitpunkt noch nicht gibt. Und mit herkömmlichen Mitteln haben wir auch leider keine Chance, diese Eigenschaft zu implementieren, denn:

Innerhalb der generischen `DynamicList`-Klasse müssten wir eine Eigenschaft `Gesamtvolumen` erschaffen, die durch alle Elemente iteriert, die sie hält, und deren `Volumen`-Eigenschaft abfragt. Doch genau eine solche Prozedur können wir nicht implementieren, wie die folgende Grafik zeigt:

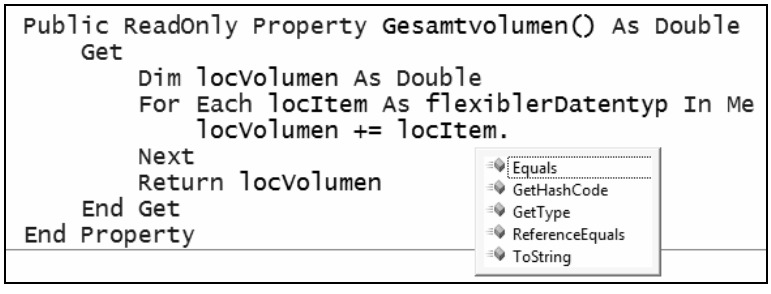



**Abbildung 6.5** Die Eigenschaft, die zur Berechnung des Gesamtvolumens benötigt wird, ist nicht erreichbar!

Wenn wir die Schleife zum Iterieren durch die Elemente der `DynamicList` erstellen, dann müssen wir natürlich darauf achten, dass ein einzelnes Element dieser Iteration vom gleichen Typ ist, wie für die gesamte generische Klasse – und damit muss es vom Typ `flexiblerDatentyp` sein (anderenfalls wäre die Klasse nicht mehr generisch, also allgemeingültig anwendbar).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

`flexiblerDatentyp` kann aber jeder beliebige Datentyp sein, und deswegen sind auch nur die Eigenschaften und Methoden erreichbar, die von jedem Datentyp *gleichermaßen* zur Verfügung gestellt werden. Das sind logischerweise genau die Eigenschaften und Methoden, die `Object` zur Verfügung stellt und die jede neue Klasse implizit erbt. Das wiederum sind die Elemente, die Ihnen IntelliSense, wie in Abbildung 6.5 zu sehen ist, anzeigt.

Das ist nun der richtige Zeitpunkt, Beschränkungen ins Spiel zu bringen. Wenn wir dem generischen Datentyp »sagen«, »pass auf, du darfst nur solche Datentypen als Typparameter akzeptieren, die auf `Körperbasis` basieren«, dann kann das .NET Framework ohne Probleme die Methoden und Eigenschaften über `locItem` anbieten, der vom Typ `flexiblerDatentyp` ist, die durch `Körperbasis` implementiert werden.

Diese Änderungen nehmen wir als Nächstes vor, allerdings nicht in der Klasse `DynamicList` selbst, denn: Damit wir nun nicht unsere `DynamicList` für alle Zeiten nur noch auf die Verwaltung von `Körper`-Objekten limitieren, implementieren wir diese Änderungen in einer Klasse, die identisch zur `DynamicList`-Klasse ist, jedoch nur die benötigten Änderungen noch zusätzlich innehat. Diese Klasse nennen wir `DynamicListKörper`, und die sieht folgendermaßen aus:

```
Class DynamicListKörper(Of flexiblerDatentyp As KörperBasis)
    Implements IEnumerable

    Protected myStep As Integer = 4          ' Schrittweite, um die das Array erhöht wird.
    Protected myCurrentArraySize As Integer  ' Aktuelle Array-Größe
    Protected myCurrentCounter As Integer    ' Zeiger auf aktuelles Element
    Protected myArray() As flexiblerDatentyp ' Array mit den Elementen.

    Sub New()
        myCurrentArraySize = myStep
        ReDim myArray(myCurrentArraySize - 1)
    End Sub

    Sub Add(ByVal Item As flexiblerDatentyp)
    .
    .
    .
    End Sub

    Public ReadOnly Property Gesamtvolumen() As Double
        Get
            Dim locVolumen As Double
            For Each locItem As flexiblerDatentyp In Me
                locVolumen += locItem.Volumen
            Next
            Return locVolumen
        End Get
    End Property
.
.
.
```

Aus Platzgründen sehen Sie in diesem Listing lediglich die Änderungen im Vergleich zur Klasse `DynamicList`.

Sie können in der ersten Zeile des Klassenlistings sehen, wie Beschränkungen implementiert werden: In den Klammern steht jetzt neben der Erweiterung `Of flexiblerDatentyp`, die den Typparameter für die weitere

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Verwendung des generischen Typs in der Klasse festlegt, obendrein der Zusatz `as KörperBasis`, der nun bestimmt, dass ausschließlich Klassen, die von `KörperBasis` abgeleitet wurden, als Basis für die Erstellung des Datentyps `DynamicListKörper` dienen dürfen.

Das befähigt uns jetzt auch, die Eigenschaft `Gesamtvolumen` zu implementieren. Da wir die Datentypen für die generische Klasse auf `KörperBasis` und deren Ableitungen beschränken, weiß das .NET Framework, dass es alle Methoden, die `KörperBasis` anbietet, sicher für alle Objekte zur Verfügung stellen kann, die auf `flexiblerDatentyp` basieren.

Nach der Implementierung dieser neuen Klasse, stellen wir das Testprogramm im Modul *mdlMain.vb* entsprechend um:

```
Module mdlMain

    Sub Main()
        Dim locKörperliste As New DynamicListKörper(Of KörperBasis)
        With locKörperliste
            .Add(New Quader(10, 20, 30))
            .Add(New Pyramide(300, 30))
            .Add(New Pyramide(300, 30))
            .Add(New Quader(20, 10, 30))
        End With
        Console.WriteLine("Das Gesamtvolumen aller Körper beträgt:" & _
            locKörperliste.Gesamtvolumen)

        Console.WriteLine()
        Console.WriteLine("Taste drücken zum Beenden!")
        Console.ReadKey()
    End Sub

End Module
```

Übrigens: Dass sich die Klasse wirklich nur noch verwenden lässt, wenn Sie sie tatsächlich auf einer Ableitung von `KörperBasis` basieren lassen, zeigt die folgende Abbildung.

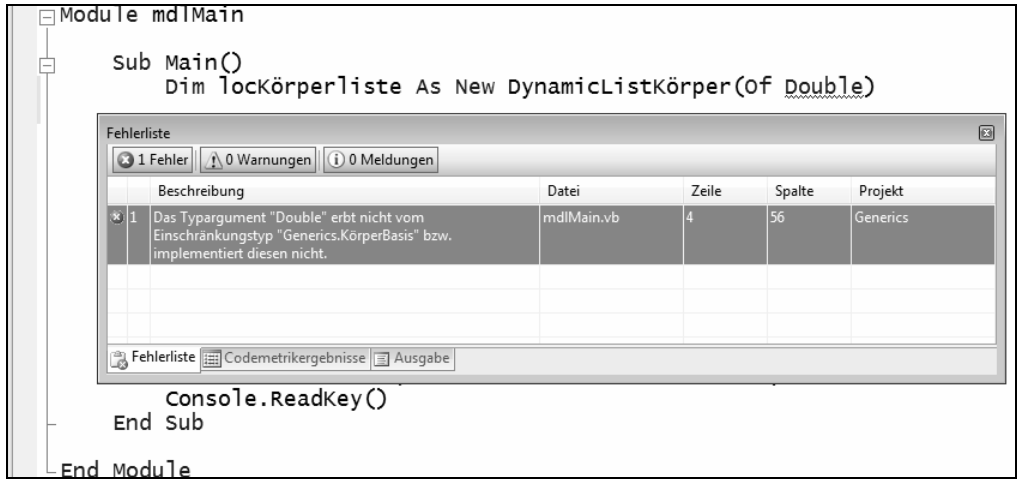



**Abbildung 6.6** Eine beschränkte generische Klasse können Sie tatsächlich nur noch auf Basis eines Datentyps instanziieren, der die Beschränkung erfüllt

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Das .NET Framework erlaubt es nicht, mehrere Basisklassen als Beschränkung für einen generischen Typ zu definieren. Das würde die Technik der Mehrfachvererbung implizieren, die das .NET Framework in der vorliegenden 2.0-Version nicht beherrscht (und wahrscheinlich auch nie beherrschen wird).

## Beschränkungen auf Klassen, die bestimmte Schnittstellen implementieren

Ungleich flexibler können Sie generische Klassen gestalten, wenn sich deren generische Typen nicht auf bestimmte Basisklassen sondern nur auf bestimmte Schnittstellen beschränken.

Darüber hinaus haben Schnittstellen den Vorteil, sich bereits in vielen »fertigen« Typen des .NET Frameworks »zu befinden«, sodass Sie generische Klassen erstellen können, die nicht nur auf Ihren eigenen Typen basieren (deren Einschränkungen Sie ja gut steuern können, da Sie über deren Quellcode verfügen); vielmehr können Sie auch die Typen des .NET Frameworks verwenden, die sich, da sie die unterschiedlichsten Schnittstellen implementieren, ebenfalls sehr gut selektieren lassen.

Ein Beispiel: Sie möchten die `DynamicList`-Klasse um eine Sortierfunktion erweitern. Zu diesem Zweck müssen Sie die Elemente, die die `DynamicList`-Klasse speichert, miteinander vergleichen können. Das .NET Framework stellt für Typen, deren einzelne Instanzen sich miteinander vergleichen lassen sollen, die `IComparable`-Schnittstelle bereit. Wenn ein Typ diese Schnittstelle einbindet, zwingt ihn diese Schnittstelle damit auch, eine Funktion namens `CompareTo` einzubinden. Und wenn Sie eine generische Klasse schaffen, dann können Sie die Typen, aus denen diese hervorgehen soll, auch auf die `IComparable`-Schnittstelle beschränken. Damit bleibt die Klasse generisch, und kann dennoch jeden beliebigen Datentyp typsicher speichern – jedenfalls solange er die `IComparable`-Schnittstelle selbst implementiert. *Wenn* er das allerdings macht, können Sie auch vom Vorhandensein einer `CompareTo`-Funktion sicher ausgehen und damit beispielsweise eine Sortierfunktion in der generischen Klasse implementieren. Die `IComparable`-Schnittstelle wird übrigens von allen primitiven Datentypen wie `String`, `Long`, `Decimal`, `Date` etc. eingebunden.

**BEGLEITDATEIEN** Ein Beispiel, das Sie im Verzeichnis *.\Samples\Chapter06 - Generics\Generics03* finden, soll das verdeutlichen.

Schauen wir uns die veränderte (und aus Platzgründen wieder gekürzte) Version der `DynamicList` an, die nun den Namen `DynamicListSortable` trägt, und die – neben der Einschränkung bei der Definition des Klassennamens – um eine `Sort`-Methode ergänzt wurde:

```
Class DynamicListSortable(Of flexiblerDatentyp As IComparable)
    Implements IEnumerable

    Protected myStep As Integer = 4          ' Schrittweite, um die das Array erhöht wird.
    Protected myCurrentArraySize As Integer  ' Aktuelle Array-Größe
    Protected myCurrentCounter As Integer    ' Zeiger auf aktuelles Element
    Protected myArray() As flexiblerDatentyp ' Array mit den Elementen.

    Sub New()
        myCurrentArraySize = myStep
        ReDim myArray(myCurrentArraySize - 1)
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Sub Add(ByVal Item As flexiblerDatentyp)
    .
    .
    .
    End Sub

    'Sortiert die Elemente, die die DynamicListSortable speichert
    Public Sub Sort()
        Dim locÄußererZähler, locInnererZähler As Integer
        Dim locDelta As Integer
        Dim locItemTemp As flexiblerDatentyp

        locDelta = 1

        'Größten Wert der Distanzfolge ermitteln
        Do
            locDelta = 3 * locDelta + 1
        Loop Until locDelta > myCurrentCounter

        Do
            locDelta \= 3

            'Shellsort's Kernalgorithmus
            For locÄußererZähler = locDelta To myCurrentCounter - 1
                locItemTemp = Me.Item(locÄußererZähler)
                locInnererZähler = locÄußererZähler
                Do While (Me.Item(locInnererZähler - locDelta).CompareTo(locItemTemp) > 0)
                    Me.Item(locInnererZähler) = Me.Item(locInnererZähler - locDelta)
                    locInnererZähler = locInnererZähler - locDelta
                    If (locInnererZähler <= locDelta) Then Exit Do
                Loop
                Me.Item(locInnererZähler) = locItemTemp
            Next
        Loop Until locDelta = 0
    End Sub
```

Möglich wird die Implementierung eines Sortieralgorithmus erst wegen der Beschränkung auf Typen, die die `IComparable`-Schnittstelle einbinden. Doch da die Typen die Schnittstelle einbinden müssen (denn anderenfalls wäre gar keine Instanzerstellung der `DynamicListSortable` möglich), kann die `Sort`-Methode auch gefahrlos auf die `CompareTo`-Methode jedes Elements zurückgreifen (siehe fett hervorgehobene Zeile im oben stehenden Listing).

Da, wie schon gesagt, beispielsweise alle primitiven Datentypen in .NET `IComparable` einbinden, können wir nun diese auch mit unserer Liste verwenden, wie der Modulcode im Folgenden zeigt:

```
Module mdlMain

    Sub Main()
        Dim locDoubleList As New DynamicListSortable(Of Double)
        locDoubleList.Add(124)
        locDoubleList.Add(1243)
        locDoubleList.Add(24)
        locDoubleList.Add(14)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        locDoubleList.Add(1)
        locDoubleList.Add(-32)
        locDoubleList.Add(231)
        locDoubleList.Add(143)

        locDoubleList.Sort()
        For Each locItem As Double In locDoubleList
            Console.WriteLine(locItem)
        Next
        Console.WriteLine()

        Dim locStringList As New DynamicListSortable(Of String)
        locStringList.Add("Klaus")
        locStringList.Add("Arnold")
        locStringList.Add("Sarah")
        locStringList.Add("Christiane")
        locStringList.Add("Jürgen")
        locStringList.Add("Uta")
        locStringList.Add("Helge")
        locStringList.Add("Uwe")

        locStringList.Sort()
        For Each locItem As String In locStringList
            Console.WriteLine(locItem)
        Next

        Console.WriteLine()
        Console.WriteLine("Taste drücken zum Beenden!")
        Console.ReadKey()
    End Sub

End Module
```

Auch hier sind die eigentlich interessanten Zeilen in Fettschrift gedruckt. Sie demonstrieren einerseits, dass die generische Liste auf unterschiedlichen Datentypen basieren kann und andererseits, dass dennoch solch komplexe Funktionen wie das Sortieren funktionieren – obwohl zum Zeitpunkt der Entwicklung der Liste noch gar nicht bekannt ist, mit welchen Typen es die Liste später zu tun haben wird.

Dass dieses Konzept auch funktioniert, zeigt die Ausführung des Programms, die folgendes Ergebnis ins Konsolenfenster zaubert:

```
-32
1
14
24
124
143
231
1243

Arnold
Christiane
Helge
Jürgen
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Klaus
Sarah
Uta
Uwe

Taste drücken zum Beenden!
```

**HINWEIS** Einen Sortieralgorithmus in eigenen Auflistungsklassen zu implementieren ist im Übrigen genau so überflüssig, wie eigene Auflistungsklassen von Grund auf neu zu kreieren. Das .NET Framework kennt nämlich eine Vielzahl von Auflistungsklassen für die unterschiedlichsten Zwecke. Doch es ist allemal interessant zu sehen, wie das Prinzip von Auflistungsklassen an sich funktioniert, und für das bessere Verständnis des vorherigen Kapitels, das die wichtigsten Auflistungen im .NET Framework vorstellt, bestimmt von Vorteil.

## Beschränkungen auf Klassen, die über einen Standardkonstruktor verfügen

In einigen Fällen ist es notwendig, dass eine generische Klasse in der Lage ist, den Typ, auf dem sie basieren soll, auch zu instanziieren. Das kann sie dann nicht, wenn es sich beim Typ, auf dem sie basiert, um eine abstrakte Klasse handelt, um eine Schnittstelle oder um eine Klasse, die ausschließlich über parametrisierte Konstruktoren verfügt.

Wenn Sie diese Fälle für den Typ ausschließen möchten, den eine generische Klasse einbindet, müssen Sie eine Beschränkung definieren, die vom Typ, auf dem sie basieren soll, einen Standardkonstruktor erfordert, und das geht folgendermaßen:

```
Public Class GenerischeKlasseMitInstanziierbaremTyp(Of flexiblerDatentyp As New)

    Public Sub TestMethode()
       'Das geht nur auf Grund der angegebenen Beschränkung:
        Dim locTest As New flexiblerDatentyp

        'Und hier ist locTest jetzt als Datentyp instanziiert!
        Console.WriteLine(locTest.ToString)
    End Sub
End Class
```

## Beschränkungen auf Wertetypen

Möchten Sie eine generische Klasse auf Wertetypen beschränken, verfahren Sie auf ähnliche Weise, wie im vorherigen Abschnitt beschrieben. Sie bestimmen durch `As Structure`, dass nur noch Wertetypen (in Visual Basic also Strukturen) innerhalb einer generischen Klasse als Typ zur Anwendung kommen dürfen, die auf `ValueType` basieren. Ein Beispiel:

```
Public Class GenerischeKlasseNurMitWertetypen(Of flexiblerDatentyp As Structure)

    Public Sub TestMethode()
        'Ist Wertetyp – keine Instanziierung durch New erforderlich!
        Dim locWerteTyp As flexiblerDatentyp
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Und hier ist locTest jetzt als Datentyp instanziiert.
        Console.WriteLine(locWerteTyp.ToString)
    End Sub
End Class
```

## Kombinieren von Beschränkungen und Bestimmen mehrerer Typparameter

In Visual Basic sind alle Beschränkungen für generische Datentypen untereinander kombinierbar. Im Gegensatz zu Beschränkungen bei Basisdatentypen können Sie darüber hinaus auch Beschränkungen für mehrere Schnittstellen bestimmen.

Wenn Sie verschiedene Beschränkungen oder mehrere Schnittstellen für einen generischen Datentyp einrichten, fassen Sie die verschiedenen Vorschriften in geschweiften Klammern zusammen.

Ein Beispiel soll auch diesen Sachverhalt verdeutlichen:

```
Public Class GenerischeBeschränkungskombi(Of flexiblerDatentyp As {Structure, IComparable, IDisposable})

    Public Sub TestMethode()
        Dim locWerteTyp As flexiblerDatentyp
        Dim locWerteTyp2 As flexiblerDatentyp

        'Direkt verwendbar, da Wertetyp durch Struktur
        'Vergleichbar, dank IComparable
        locWerteTyp.CompareTo(locWerteTyp2)

        'Disposable, dank IDisposable
        locWerteTyp.Dispose()
        locWerteTyp2.Dispose()
    End Sub
End Class
```

Zusätzlich können Sie eine generische Klasse auch für die Verwendung von mehreren Typparametern einrichten. Falls Sie beispielsweise eine Auflistung entwickeln möchten, die als ein Wörterbuch fungiert, dann benötigen Sie einen Typ zum Nachschlagen (den Schlüssel) und einen für den eigentlichen Wert. (Programmieren Sie das aber nicht selbst, denn auch das gibt es schon beispielsweise mit der generischen `KeyedCollection`-Klasse, die sich im `System.Collection.ObjectModel`-Namespace befindet.) Die Einschränkungen lassen sich dann für jeden Typparameter einzeln festlegen:

```
Public Class GenerischesWörterbuch(Of Schlüsseltyp As {Structure, IComparable}, _
                      Wertetyp As {New, IComparable, IDisposable})

    Public Sub TestMethode()
        Dim locWerteTyp As Schlüsseltyp
        Dim locWerteTyp2 As Wertetyp

        'Direkt verwendbar, da Wertetyp durch Struktur
        'Vergleichbar, dank IComparable
        locWerteTyp.CompareTo(locWerteTyp2)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Disposable, dank IDisposable
        locWerteTyp2.Dispose()
    End Sub
End Class
```

# Generische Auflistungen (Generic Collections)

Generische Auflistungen haben im Vergleich zu »herkömmlichen« Auflistungen, die Sie im vorherigen Kapitel kennen gelernt haben, einen entscheidenden Vorteil: Sie sind grundsätzlich typsicher. Anders als »normale« Auflistungen wie beispielsweise die `ArrayList`-Klasse nehmen sie, da sie nicht auf `Object` basieren, nicht jeden Datentyp als Element entgegen, sondern beschränken sich auf den Datentyp, der ihrer Definition zugrunde liegt.

Das bedeutet: Definieren Sie beispielsweise eine `Collection` auf Basis von `Integer`, etwa mit

```
Dim locGenColl As New Collection(Of Integer)
```

dann laufen Sie nicht Gefahr, später versehentlich der Auflistung ein Element hinzuzufügen, das nicht vom Typ `Integer` ist – oder mit anderen Worten: Die Zeile

```
locGenColl.Add("Ein Element")
```

würde bereits zur Entwurfszeit im Editor als fehlerhaft gekennzeichnet.

Ihre Programme werden durch den Einsatz von generischen Auflistungen somit robuster und auch die Entwicklungszeit reduziert sich, da Sie sich nicht mit Laufzeitfehlern herumärgern müssen, sondern bereits zur Entwurfszeit Fehler korrigieren können. Und weniger Programmtests bedeuten weniger Entwicklungszeit und -kosten.

Nun gibt es nicht wenige generische Auflistungen seit dem .NET Framework 2.0, und sie alle im Detail zu beschreiben ist nicht nur unnötig – schließlich funktionieren viele von ihnen wie ihre nicht generischen Verwandten – es würde auch den Rahmen des Buches sprengen.

Aus diesem Grund möchte ich mich auf die in meinen Augen wichtigen Besonderheiten beschränken, die Sie bei nicht-generischen Auflistungen nicht antreffen. Eine Tabelle, die Sie im Folgenden finden, gibt darüber hinaus Auskunft, welche generischen Auflistungen Ihnen zur Verfügung stehen, und zu welchem Zweck sie dienen.

| Namespace | Auflistung | Beschreibung |
|---|---|---|
| `System.Collection.ObjectModel` | `Collection(Of type)` | Stellt eine Standardauflistung für die einfache, unsortierte Verwaltung von Elementen eines bestimmten Typs zur Verfügung.<br>**Besonderheiten:** Im Gegensatz zu `List(Of type)` gibt es überschreibbare Methoden, mit denen man das Verhalten beim Einfügen, Löschen und Neuzuweisen von Elementen der Auflistung in abgeleiteten Klassen beeinflussen kann. ▶ |

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Namespace | Auflistung | Beschreibung |
|---|---|---|
| `System.Collection.ObjectModel` | `KeyedCollection(Of key,type)`. | Stellt eine Auflistung von Elementen zur Verfügung, die sowohl das Nachschlagen über einen Schlüssel (Key) eines bestimmten Typs *sowie* den Einsatz eines Indexes erlaubt.<br>**Besonderheiten:** Diese Auflistung können Sie nur in Ableitungen verwenden, da der Typ, den Sie speichern, selber einen Standard-Schlüssel erzeugen muss, und für diesen Umstand müssen Sie in der Ableitung sorgen.<br>**WICHTIG:** Vermeiden Sie den Einsatz von numerischen Schlüsseln (Integer, Long, etc.), da es hier beim Serialisieren der Auflistung zu Problemen kommt (Stand: *.NET Framework-Version 2.0.50727*). |
| `System.Collection.ObjectModel` | `ReadOnlyCollection(Of type)` | Stellt eine Auflistung dar, deren Elemente nur gelesen werden können.<br>**Besonderheiten:** Elemente, die in einer anderen generischen Auflistung gleichen Typs vorliegen, können nur bei der Instanziierung im Konstruktor dieser Auflistung übergeben werden. Ansonsten sind die Elemente nur lesbar und können nach der Instanziierung nicht mehr verändert werden. |
| `System.Collections.Generic` | `Dictionary(Of key, type)` | Stellt eine Auflistung von Schlüsseln und Werten dar.<br>**Besonderheiten:** Stellt eine Zuordnung von einem Satz von Schlüsseln zu einem Satz von Werten bereit. Jede Hinzufügung zum Wörterbuch besteht aus einem Wert und dem zugeordneten Schlüssel. Ein Wert kann anhand des zugehörigen Schlüssels sehr schnell abgerufen werden (beinahe ein O(1)-Vorgang), da die `Dictionary`-Klasse in Form einer Hashtable implementiert ist. |
| `System.Collections.Generic` | `LinkedList(Of type)` | Stellt eine doppelt verknüpfte Liste dar.<br>**Besonderheiten:** Ist eine verknüpfte Liste mit einzelnen Knoten vom Typ `LinkedListNode`; das Einfügen und Entfernen einzelner Elemente geht extrem schnell vonstatten. |
| `System.Collections.Generic` | `List(Of type)` | Stellt eine Standardauflistung für die einfache, unsortierte Verwaltung von Elementen eines bestimmten Typs zur Verfügung.<br>**Hinweis:** Diese Klasse eignet sich nicht für Ableitungen in eigenen Auflistungsklassen, bei denen Sie mit Code Einfluss auf die Bearbeitung der Liste nehmen müssen. Verwenden Sie stattdessen die `Collection(Of )`-Auflistung (siehe oben). |
| `System.Collections.Generic` | `Queue(Of type)` | Stellt eine FIFO-Auflistung (First-In-First-Out) von Objekten dar.<br>**Hinweis:** Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung finden Sie im vorherigen Kapitel. |
| `System.Collections.Generic` | `SortedDictionary(Of key,type)` | Stellt eine Auflistung von Schlüssel-Wert-Paaren dar, deren Reihenfolge anhand des Schlüssels bestimmt wird. ▶ |

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Namespace | Auflistung | Beschreibung |
|---|---|---|
| `System.Collections.Generic` | `SortedList(Of key, type)` | Verwaltet eine sortierte Liste, deren Elemente über Schlüssel abrufbar sind.<br>**Hinweis:** Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung finden Sie im vorherigen Kapitel.<br>Im Gegensatz zu `SortedDictionary` erfolgt die Sortierung über das Element und nicht über den Schlüssel! |
| `System.Collections.Generic` | `Stack(Of type)` | Stellt eine LIFO-Auflistung (Last-In-First-Out) von Objekten dar.<br>**Hinweis:** Prinzipiell funktioniert diese Auflistung wie ihr nicht generischer Verwandter. Mehr zur nicht generischen Version dieser Auflistung finden Sie im vorherigen Kapitel. |

**Tabelle 6.1:** Die wichtigsten generischen Auflistungstypen

# List(Of )-Auflistungen und Lambda-Ausdrücke

Die `List(Of )`-Klasse ist eine der am häufigsten (wenn nicht sogar die am häufigsten) eingesetzte generische Auflistungsklasse. Wenn Sie primitive Datentypen in einer Auflistung speichern, sollten Sie den Einsatz `List(Of type)` anderen Auflistungsklassen vorziehen – alleine schon aus Performance-Gründen. Das .NET Framework ist nämlich in der Lage, den eigentlich notwendigen Boxing-Vorgang bei `List(Of type)` zu umgehen, und damit ist `List(Of type)` die schnellere Alternative.

Und obwohl es sich bei dieser generischen Auflistung um solch eine populäre Klasse handelt, hält sie Funktionalitäten bereit, über die man – obwohl sie bereits im .NET Framework 2.0 vorhanden waren – jedenfalls im Rahmen von Visual Basic bislang wenig Brauchbares erfahren konnte; wohl auch, weil sie förmlich nach der Verwendung von Lambda-Funktionen schreien, die erst mit der 2008er Version Einzug in Visual Basic hielten. Das folgende Beispiel setzt das Verständnis von Lambda-Funktionen voraus – und falls Sie sich mit diesen noch nicht auseinander gesetzt haben sollten: Kapitel 3 klärt deren grundsätzliches Verständnis.

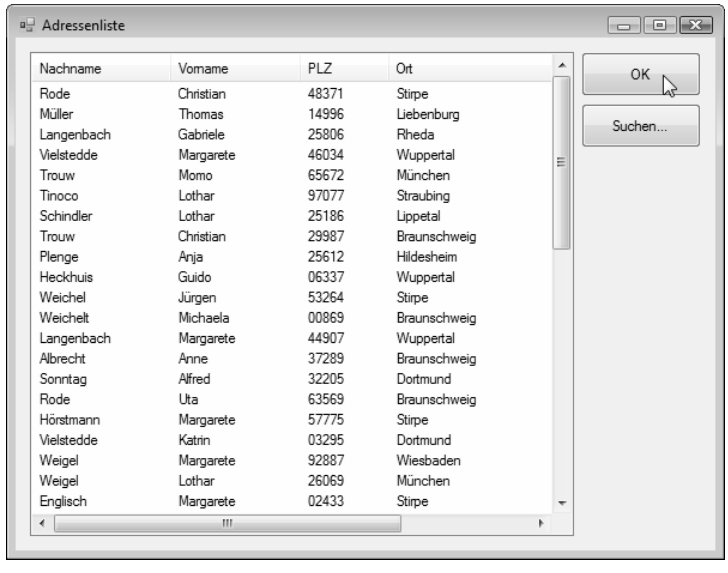



**Abbildung 6.7** Sortierung und Suche steuern Sie über die Spaltenköpfe in dieser Adressenliste

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**BEGLEITDATEIEN** Sie finden das folgende Beispielprojekt im Verzeichnis *.\Samples\Chapter06 - Generics\ListOfDemo.*

Wenn Sie das Programm starten, sehen Sie einen Dialog, etwa wie in Abbildung 6.7 zu sehen. Die im Formular vorhandene `ListView` wird mit 50 Zufallsadressen gefüllt. Sie können die Liste sortieren, indem Sie auf den entsprechenden Spaltenkopf klicken – so, wie Sie es vom Windows-Explorer in der *Details*-Ansicht gewohnt sind.

Eine Suchfunktion, die Sie über die entsprechende Schaltfläche erreichen, gestattet es Ihnen, nach dem Begriff innerhalb der Liste zu suchen. Gefunden wird dabei der Eintrag, dessen Text in der Spalte, nach der zuletzt sortiert wurde, dem eingegebenen Suchbegriff entspricht.

Soweit ist dieses Beispiel noch nichts Besonderes. Doch wenn Sie einen Blick in die Umsetzung riskieren, wird klar, wie alternativ der Lösungsansatz ist. Das geht schon beim Schreiben der Zufallsadressen in die Liste los:

## ForEach und die generische Action-Klasse

Wenn Sie den Inhalt eines Arrays oder einer Auflistung in einer Liste darstellen möchten, liegt die Vorgehensweise, um das zu erreichen, eigentlich auf der Hand: Sie iterieren mit einem For/Each-Konstrukt durch die Auflistung, verarbeiten damit jedes Element und bringen es mit geeigneten Methoden oder Eigenschaften in die sichtbare Liste eines Formulars.

Dieses Beispiel nutzt einen anderen Weg, wie im folgenden Listing-Ausschnitt zu sehen:

```
Sub ElementeDarstellen()

    'Unterdrückt Neuzeichnen-Ereignisse bis zum
    'nächsten EndUpdate; dadurch geht der Aufbau
    'der Elemente schneller und wackelt nicht.
    Me.lvwAdressen.BeginUpdate()

    'Alle Elemente der ListView löschen.
    Me.lvwAdressen.Items.Clear()

    'Für jedes Element der Liste wird ElementInListe aufgerufen.
    myAdressen.ForEach(New Action(Of Adresse)(AddressOf ElementInListe))

    'So werden die Spaltenbreiten optimal angepasst.
    For Each locCol As ColumnHeader In Me.lvwAdressen.Columns
        locCol.Width = -2
    Next

    'Aufbau der ListView ist beendet.
    Me.lvwAdressen.EndUpdate()
End Sub

'Der Action-Delegat: für jedes Element der Liste wird diese Aktion durchgeführt.
Sub ElementInListe(ByVal Element As Adresse)
    'Neues ListView-Element - Name kommt zuerst.
    Dim locLvwItem As New ListViewItem(Element.Name)

    'Die Untereinträge setzen
    With locLvwItem.SubItems
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            .Add(Element.Vorname)
            .Add(Element.PLZ)
            .Add(Element.Ort)
        End With

        'Zum Wiederfinden Referenz in Tag
        locLvwItem.Tag = Element

        'Zur Listview hinzufügen
        lvwAdressen.Items.Add(locLvwItem)
    End Sub
```

Sie können die `ForEach`-Methode verwenden, um durch eine Auflistung iterieren *zu lassen* und dabei für jedes Element einen Delegaten aufrufen, der eine bestimmte Aktion ausführt. Genau das Verfahren nutzen wir an dieser Stelle, um die Liste aufzubauen. Der Delegat wird im Beispiel nicht durch ein `Delegate`-Objekt (mehr zu Delegaten finden Sie in Kapitel 4) sondern durch die direkte Angabe einer Prozedur mithilfe des `Address Of`-Operators angegeben – der Compiler sorgt dann für den richtigen Ersatz einer Delegatvariablen. Aber natürlich könnten an dieser Stelle auch »manuelle« Delegatvariablen zum Einsatz kommen, die in Abhängigkeit von bestimmten Programmzuständen andere Prozeduren verwenden, und genau darin besteht der Reiz des Einsatzes dieser Verfahren. Und – auch das ist möglich – mit Visual Basic 2008 können Sie anstelle eines Delegaten auch einen Lambda-Ausdruck an dieser Stelle einsetzen.

Wichtig ist in jedem Fall, dass die Routinen, die Sie mithilfe der `Action`-Klasse aufrufen, den Signaturanspruch des Delegaten erfüllen, den Sie im Konstruktor von `Action` angeben. Im Fall von `ForEach` und der `Action`-Klasse muss es sich bei der Delegatenprozedur um eine Methode handeln, die kein Funktionsergebnis hat (also um eine Visual Basic-`Sub`) und als Parameter ein Element entgegen nimmt, dessen Typ auch der Basis der generischen `Action`-Klasse entspricht – `Adresse` in unserem Beispiel.

Im Ergebnis erreichen wir also, dass für jedes Element des `List(Of Adresse)`-Objektes `myAdressen` die Methode `ElementInListe` aufgerufen wird.

## Sort und die generische Comparison-Klasse

Prinzipiell funktioniert das nächste Pärchen ähnlich, das Sie verwenden, wenn Sie ein Array mit der Methode `Sort` ohne den Einsatz einer speziellen `Comparer`-Klasse (wie in Kapitel 5 gezeigt) sortieren möchten. In Abwandlung zur ersten Verwendung kommen hier jedoch Lambda-Ausdrücke zum Einsatz, sodass wir uns das buchstäbliche Delegieren an die richtige Sortierungsfunktion mit einer Delegatenvariable sparen können. Den relevanten Codeausschnitt des Beispiels finden Sie im Folgenden:

```
    'Wird aufgerufen, wenn eine der Spalten angeklickt wird.
    Private Sub lvwAdressen_ColumnClick(ByVal sender As Object, ByVal e As _
                        System.Windows.Forms.ColumnClickEventArgs) Handles lvwAdressen.ColumnClick

        'Spaltennummer, die in e.Column steht, in AdressenSortierenNach konvertieren
        mySortierenNach = CType(e.Column, AdressenSortierenNach)

        Select Case mySortierenNach
            Case AdressenSortierenNach.Name
                myAdressen.Sort(Function(adr1 As Adresse, adr2 As Adresse) _
                                String.Compare(adr1.Name, adr2.Name))
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            Case AdressenSortierenNach.Vorname
                myAdressen.Sort(Function(adr1 As Adresse, adr2 As Adresse) _
                                String.Compare(adr1.Vorname, adr2.Vorname))

            Case AdressenSortierenNach.PLZ
                myAdressen.Sort(Function(adr1 As Adresse, adr2 As Adresse) _
                                String.Compare(adr1.PLZ, adr2.PLZ))

            Case AdressenSortierenNach.Ort
                myAdressen.Sort(Function(adr1 As Adresse, adr2 As Adresse) _
                                String.Compare(adr1.Ort, adr2.Ort))

        End Select

        'Die Elemente neu sortiert darstellen
        ElementeDarstellen()
    End Sub
```

`Sort` arbeitet hier in diesem Beispiel mit Lambda-Ausdrücken, bei denen jeder der vier Lambda-Ausdrücke nach einer anderen Eigenschaft der Adresse sortiert – in Abhängigkeit davon, welche der vier Spalten in der `ListView` zum Sortieren eingestellt wurde.

## Find und die generische Predicate-Klasse

Das letzte generische »Dreamteam« schließlich kommt zum Einsatz, wenn es um das Finden eines bestimmten Objektes in einer generischen Liste geht: `Find` und der Delegat, der durch die generische `Predicate`-Klasse eingerichtet wird. Auch hier entspricht die grundsätzliche Vorgehensweise dem bereits Bekannten, wie der entsprechende Code im Beispiel zeigt:

```
    Private Sub btnSuchen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
                               Handles btnSuchen.Click

        'Suchbegriff abfragen
        Dim suchFormular As New frmSuchen
        Dim aktuellerSuchbegriff As String    ' Der zuletzt erfragte Suchbegriff

        'Den Suchbegriff merken, damit der Predicate-Delegat darauf
        'zugreifen kann.
        aktuellerSuchbegriff = suchFormular.Suchbegriff
        If String.IsNullOrEmpty(aktuellerSuchbegriff) Then
            Return
        End If

        'Hier kann die Suche beginnen!
        Dim locGefAdr As Adresse = Nothing

        Select Case mySortierenNach
            Case AdressenSortierenNach.Name
                locGefAdr = myAdressen.Find(Function(adr As Adresse) adr.Name = aktuellerSuchbegriff)
            Case AdressenSortierenNach.Vorname
                locGefAdr = myAdressen.Find(Function(adr As Adresse) adr.Vorname = aktuellerSuchbegriff)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            Case AdressenSortierenNach.PLZ
                locGefAdr = myAdressen.Find(Function(adr As Adresse) adr.PLZ = aktuellerSuchbegriff)
            Case AdressenSortierenNach.Ort
                locGefAdr = myAdressen.Find(Function(adr As Adresse) adr.Ort = aktuellerSuchbegriff)
        End Select

        'Wenn ein Element gefunden wurde, dieses markieren.
        If locGefAdr IsNot Nothing Then

            'Alle ListView-Elemente durchsuchen und überprüfen, ob ...
            For Each locLvwItem As ListViewItem In Me.lvwAdressen.Items

                '... die Tag-Referenz der Referenz des gesuchten Objekts entspricht.
                If locLvwItem.Tag Is locGefAdr Then

                    'Gefunden! ListView-Element markieren,
                    locLvwItem.Selected = True

                    'und dafür sorgen, dass es im sichtbaren Bereich liegt.
                    locLvwItem.EnsureVisible()
                    Return
                End If
            Next
        End If
    End Sub
```

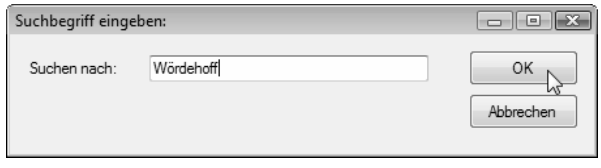



**Abbildung 6.8** Der Suchbegriff bezieht sich immer auf die Spalte, nach der zuletzt sortiert wurde

Auch hier machen Lambda-Ausdrücke, wie in den fetthervorgehobenen Codezeilen zu sehen, die Suche nach den richtigen Adressen vergleichsweise einfach. Je nach selektierter Sortierspalte wird einfach ein anderer Lambda-Ausdruck verwendet, der den Vergleich mit dem eingegebenen Suchbegriff übernimmt.

Nur eine kleine Herausforderung ist dann noch das Selektieren des gefundenen Begriffs in der Liste – und das ist im Übrigen auch der Grund, wieso wir beim Aufbauen der Liste eine Referenz jedes Elements in der `Tag`-Eigenschaft jedes `ListViewItem`-Elements speichern: Wenn der Begriff durch `Find` gefunden wurde, liegt uns das entsprechende `Adresse`-Objekt vor. Mit diesem können wir anschließend durch die `ListViewItem`-Auflistung iterieren und auf Objektübereinstimmung durch Abfrage der `Tag`-Eigenschaft testen. Dieser Aufwand ist nötig, da es keine andere Möglichkeit gibt, das richtige `ListViewItem`-Element zu finden, und nur dieses erlaubt es aber, die richtige Zeile in der Liste durch seine `Select`-Eigenschaft zu selektieren.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Teil B

# Language Integrated Query (LINQ)

**In diesem Teil:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 7

# Einführung in LINQ

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Falls Sie sich in Ihrer Entwicklerkarriere jemals schon mit Datenbankprogrammierung auseinandergesetzt haben, dann wissen Sie, dass alle Konzepte, die Microsoft in diesem Zusammenhang vorgestellt hat, zwar immer brauchbar, aber nie wirklich das Gelbe vom Ei waren.

Es geht um das Fehlen eines gescheiten Konzeptes, das man unter dem Schlagwort O/RM kennt. O/RM steht für Object Relational Mapping, und dabei handelt es sich um eine Programmiertechnik, um Daten zwischen den beiden verschiedenen, eigentlich nicht kompatiblen Typsystemen »Datenbanken« und denen der objektorientierten Programmierung mehr oder weniger automatisch zu konvertieren.

Wie Sie es in den vorangehenden Kapiteln schon erfahren konnten, verwendet man am besten Business-Objekte, um mit Daten in einer objektorientierten Programmiersprache umzugehen. Unter »Business-Objekt« versteht man im einfachsten Fall eine simple Klasse, die keine weitere Aufgabe hat, als eine Datenentität wie eine Kontaktadresse oder einen Artikel zu speichern. Mehrere dieser Adressen oder Artikel legt man dann in Auflistungen ab.

Datenbanken hingegen speichern ihre einzelnen Datensätze (also eine einzelne Adresse oder einen einzelnen Artikel) in Tabellen. In der Regel verwendet man dann hier die berühmten SQL-Abfragen, um Daten in verschiedenen Tabellen miteinander zu kombinieren, zu filtern und zu selektieren.

Die Aufgabe einer O/RM ist es nun, diese beiden Welten auf eine möglichst einfache Art und Weise miteinander zu kombinieren – denn bislang war das Aufgabe des Entwicklers, und die sah exemplarisch folgendermaßen aus: Wollte man beispielsweise die Adressen eines bestimmten Postleitzahlenbereichs aus einer SQL Server-Datenbank in einer Windows-Forms-Anwendung darstellen (wobei es hier im Folgenden nicht um die konkrete Programmierung sondern nur um das grundlegende Konzept geht), war in etwa Folgendes zu tun:

- Eine Windows Forms-Anwendung baut eine Verbindung zu einer Datenbank auf. Dabei muss sie sich an bestimmte Konventionen halten und vor allem provider-spezifische Klassen verwenden, um diese Verbindung herzustellen. Eine Verbindung zu einem SQL-Server erfordert so unter dem .NET Framework das `SqlConnection`-Objekt, eine Verbindung zu einer Access-Datenbank das `OleDBConnection`-Objekt und eine Verbindung zu einer Oracle-Datenbank eben ein `OracleConnection`-Objekt.
- Es überträgt eine Selektierungsabfrage in Form eines Command-Objektes. Auch dieses Command-Objekt ist wieder provider-spezifisch (`OleDbCommand`, `OracleCommand`, `SqlCommand`); zumal können sich die einzelnen Abfragetexte – auch wenn sie alle ANSI-SQL-basierend sind – schon ein wenig voneinander unterscheiden.[1] Um beispielsweise alle Adressen der Postleitzahlengebiete zwischen 50000 und 59999 abzufragen und die Ergebniszeilen nach Ort und Namen zu sortieren, lautet die entsprechende Abfrage:

```
SELECT [Name], [Vorname], [Strasse], [PLZ], [ORT] FROM [Adressen]
WHERE [PLZ]>='50000' AND [PLZ] <='59999'
ORDER BY [Ort], [Name]
```

- Anschließend können die Daten feldweise ausgelesen werden. Im einfachsten Fall funktioniert das zellenweise – also nacheinander kommen Name des ersten Datensatzes, dann Vorname, dann Straße, dann Postleitzahl und schließlich der Ort. Weiter geht es mit Name des zweiten Datensatzes, usw. Und diese Ergebnistexte, die nacheinander vom SQL Server zurückgeliefert werden, müssen jetzt wiederum

[1] Beispiel Datensätze löschen: In Access heißt es `DELETE * FROM Tabelle WHERE...`, in Oracle `DELETE Tabelle WHERE...` in SQL Server `DELETE FROM Tabelle WHERE...` (auch wenn SQL Server auch die Oracle-Syntax »kann«, da das ANSI-SQL ist).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

irgendwo gespeichert werden – beispielsweise in einer entsprechenden Auflistung, deren einzelne Elemente idealerweise den Schemainformationen der Datenbanktabelle entsprechen (die Eigenschaften eines Elements sollten sich also auf die einzelnen Spalten der Datenbanktabelle abbilden).

- Wenn das Programm den Anwender dann anschließend die Daten in den Business-Objekten hat verändern lassen – beispielsweise indem die Daten in einer Maske editierbar gemacht wurden – müssen sie aus den Masken zurück in die Business-Objekte und schließlich dann zurück in die Datenbank. Dazu muss das Programm alle Business-Objekte durchlaufen, schauen, ob sich die Daten dort geändert haben (dazu sollte es die Ursprungsdaten natürlich auch gespeichert haben, da es sonst ja gar nicht entscheiden kann, ob es Daten an den Business-Objekten gegeben hat), und dann für jedes Business-Objekt, das sich geändert hat, eine entsprechende SQL-INSERT bzw. UPDATE-Anweisung generieren, damit die Daten auch in den entsprechenden Tabellen der Datenbank aktualisiert werden.

Würde man das alles manuell machen, wäre das ein ziemlicher Coding-Aufwand. Und aus diesem Grund gab es auch bisher schon entsprechende Hilfsmittel – beispielsweise DataSets – mit deren Hilfe sich der ganze Kommunikationsvorgang extrem vereinfachen ließ. Abfragen an den SQL Server waren allerdings immer noch an den SQL Server zu stellen – und das Selektieren oder Sortieren von Business-Daten im Speicher funktionierte damit völlig anders, als die eleganten SQL-Abfragen, die an die verschiedenen SQL-Server gestellt wurden.

Das wird mit LINQ anders. Mit LINQ formulieren Sie SQL-Abfragen innerhalb des Quellcodes. Und wenn Sie vorher durch entsprechende Einstellungen gesagt haben, dass diese sich an eine SQL-Server-Datenbank richten sollen, dann sorgt LINQ dafür, dass diese im Quelltext hinterlegte Abfrage eben SQL Server-kompatibel dort hingelangt. LINQ erlaubt es aber mit der gleichen Abfragesyntax, auch XML-Dokumente abzufragen. Oder simple Auflistungen. Oder Inhalte von DataSets.

Diese verschiedenen Dinge, die Sie abfragen können, nennt man Entitäten. SQL Server ist eine Entität. Auflistungen eine andere. XML wieder eine andere. Und DataSets eine noch andere. So gibt es nicht nur LINQ, sondern immer auch etwas, auf das sich LINQ bezieht. Und im Moment[2] gibt es die folgenden Entitäten für LINQ:

- *LINQ to Objects*: Damit können Sie SQL-ähnliche Abfragen an Auflistungen stellen. Und zwar an alle Auflistungen, die die Schnittstelle `IEnumerable` implementieren – und das sind die meisten.
- *LINQ to SQL*: Damit können Sie SQL-ähnliche Abfragen direkt an den SQL Server stellen. Kapitel 9 zeigt Ihnen, wie das funktioniert.
- *LINQ to Xml*: Mit XML als Entität können Sie Abfragen direkt auf XML-Dokumente loslassen, um eine Auflistung entsprechender Elemente zu bekommen. Kapitel 10 zeigt Ihnen, wie das funktioniert.
- *LINQ to DataSets*: Funktioniert in etwa wie *LINQ to Objects*, nur dass eben DataSets abgefragt werden können.
- *LINQ to Entities*: Funktioniert prinzipiell wie *LINQ to SQL*, nur dass nicht der SQL Server der Abfrageempfänger ist, sondern eine beliebige Entität. *LINQ to Entities* erlaubt prinzipiell das LINQ-Mapping zu jeder Datenquelle, die als .NET-Datenprovider implementiert ist. Leider geht das noch nicht im Moment, sondern erst, wenn das Service Pack 1 für Visual Studio 2008 verfügbar wird.

[2] Stand: 20.01.2008

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Wie »geht« LINQ prinzipiell

LINQ erweitert die Basic-Sprache um Abfrageausdrücke. Was dabei genau abgefragt wird, ist Sache des jeweiligen LINQ-Providers – das haben wir im vergangenen Abschnitt schon erfahren. Das *Wie* ist dabei aber eher entscheidend, denn es ist für alle LINQ-fähigen Entitäten weitestgehend gleich.

**BEGLEITDATEIEN** Im Verzeichnis *\Samples\Chapter07 - Linq\ErsteLinqDemo* finden Sie eine Konsolen-Beispielanwendung, die mit Business-Objekten die Fähigkeiten von LINQ to Objects eindrucksvoll demonstriert. Das Programm konstruiert dabei folgendes Szenario: Es erstellt eine Kundenauflistung – die einzelnen Elemente dieser Auflistung werden dabei in entsprechenden Business-Objekten gespeichert, deren Schema genau dem der folgenden Tabelle entspricht.

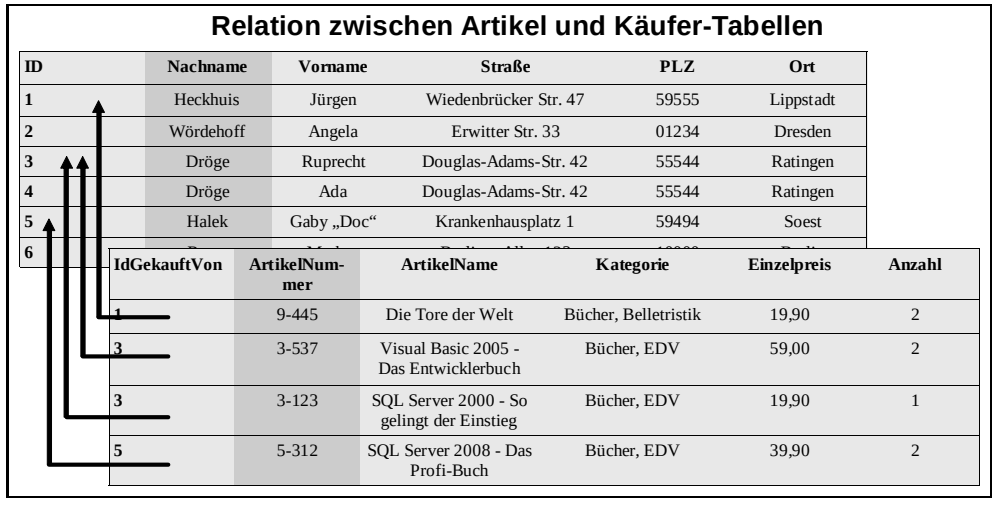



**Abbildung 7.1** Das LINQ-Beispielprogramm legt zwei Tabellen an, die nur logisch über die Spalten-ID miteinander verknüpft sind.

Es definiert ferner eine zweite Tabelle, die Daten mit gekauften Artikeln enthält. In dieser Tabelle ist jedoch der Name des Käufers nicht mehr enthalten; stattdessen gibt es nur eine ID in der Gekauften-Artikel-Tabelle, mit der man in der ersten nachschlagen und so die eigentlichen Käuferdaten ermitteln kann.

Und jetzt kommt das eigentliche Problem, bzw. die eigentliche Aufgabe: Das Programm soll die Daten so aufbereiten, dass die Kunden nicht nur nacheinander nach Nachnamen sortiert angezeigt werden; das Programm soll ebenfalls ermitteln, welcher Kunde wie viel Umsatz gemacht hat, und es soll die gekauften Artikel untereinander auflisten. Dabei soll es aber nur die Kunden berücksichtigen, die mehr als 1.000 Euro erzielt haben, und in einem definierten Postleitzahlgebiet wohnen, etwa wie folgt:

```
5: Lehnert, Theo - 11 Artikel zu insgesamt 1064,10 Euro
     Details:
     1-234: Das Leben des Brian(3 Stück für insgesamt 254,85 Euro)
     1-234: Das Leben des Brian(1 Stück für insgesamt 14,95 Euro)
     3-123: Das Vermächstnis der Tempelritter(1 Stück für insgesamt 29,95 Euro)
     3-537: Visual Basic 2005 - Das Entwicklerbuch(2 Stück für insgesamt 169,90 Euro)
     3-537: Visual Basic 2005 - Das Entwicklerbuch(1 Stück für insgesamt 89,95 Euro)
     5-506: Visual Basic 2008 - Das Entwicklerbuch(1 Stück für insgesamt 79,95 Euro)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
     5-518: Visual Basic 2008 - Neue Technologien - Crashkurs(1 Stück für insgesamt 4,95 Euro)
     7-321: Das Herz der Hölle(2 Stück für insgesamt 99,90 Euro)
     9-009: Die Wächter(2 Stück für insgesamt 39,90 Euro)
     9-009: Die Wächter(3 Stück für insgesamt 254,85 Euro)
     9-646: Was diese Frau so alles treibt(1 Stück für insgesamt 24,95 Euro)
10: Weigel, Momo - 14 Artikel zu insgesamt 1118,70 Euro
     Details:
      4-444: Harry Potter und die Heiligtümer des Todes(1 Stück für insgesamt 34,95 Euro)
     1-234: Das Leben des Brian(1 Stück für insgesamt 49,95 Euro)
     2-424: 24 - Season 6 [UK Import - Damn It!](2 Stück für insgesamt 89,90 Euro)
     2-424: 24 - Season 6 [UK Import - Damn It!](3 Stück für insgesamt 74,85 Euro)
     2-424: 24 - Season 6 [UK Import - Damn It!](1 Stück für insgesamt 4,95 Euro)
     3-534: Mitten ins Herz(3 Stück für insgesamt 59,85 Euro)
     3-537: Visual Basic 2005 - Das Entwicklerbuch(3 Stück für insgesamt 74,85 Euro)
     3-543: Microsoft Visual C# 2005 - Das Entwicklerbuch(1 Stück für insgesamt 44,95 Euro)
     3-543: Microsoft Visual C# 2005 - Das Entwicklerbuch(1 Stück für insgesamt 29,95 Euro)
     5-401: Programmieren mit dem .NET Compact Framework(1 Stück für insgesamt 29,95 Euro)
     7-321: Das Herz der Hölle(1 Stück für insgesamt 89,95 Euro)
     9-009: Die Wächter(3 Stück für insgesamt 179,85 Euro)
     9-423: Desperate Housewives - Staffel 2, Erster Teil(2 Stück für insgesamt 99,90 Euro)
     9-445: Die Tore der Welt(3 Stück für insgesamt 254,85 Euro)
```

LINQ ermöglicht es, aus zwei Auflistungen, die mit den Schemainformationen, wie Sie auch in Abbildung 7.1 zu sehen sind, *eine* neue Auflistung zu erstellen, die das hier gezeigte mit den genannten Konventionen einfach hintereinander ausgibt. Das entsprechende Listing dazu sieht folgendermaßen aus:

```
Module LinqDemo

    Sub Main()

        Dim adrListe = Kontakt.ZufallsKontakte(10)
        Dim artListe = Artikel.Zufallsartikel(adrListe)

        Dim adrListeGruppiert = From adrElement In adrListe _
                                Join artElement In artListe _
                                On adrElement.ID Equals artElement.IDGekauftVon _
                                Select adrElement.ID, adrElement.Nachname, _
                                       adrElement.Vorname, adrElement.PLZ, _
                                       artElement.ArtikelNummer, artElement.ArtikelName, _
                                       artElement.Anzahl, artElement.Einzelpreis, _
                                       Postenpreis = artElement.Anzahl * artElement.Einzelpreis _
                                Order By Nachname, ArtikelNummer _
                                Where (PLZ > "0" And PLZ < "50000") _
                                Group ArtikelNummer, ArtikelName, _
                                      Anzahl, Einzelpreis, Postenpreis _
                                By ID, Nachname, Vorname _
                                Into Artikelliste = Group, AnzahlArtikel = Count(), _
                                     Gesamtpreis = Sum(Postenpreis) _
                                Where Gesamtpreis > 1000

        For Each KundenItem In adrListeGruppiert
            With KundenItem
                Console.WriteLine(.ID & ": " & .Nachname & ", " & .Vorname & " - " & _
                          .AnzahlArtikel & " Artikel zu insgesamt " & .Gesamtpreis & " Euro")
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
                Console.WriteLine("     Details:")
                For Each ArtItem In KundenItem.Artikelliste
                    With ArtItem
                        Console.WriteLine("     " & .ArtikelNummer & ": " & .ArtikelName & _
                                          "(" & .Anzahl & " Stück für insgesamt " _
                                          & (.Einzelpreis * .Anzahl).ToString("#,##0.00") & " Euro)")
                    End With
                Next
            End With

        Next
        Console.ReadKey()

    End Sub

End Module
```

Wenn Sie sich dieses Listing anschauen, dann geht es Ihnen wahrscheinlich wie jedem, der sich das erste Mal mit SQL-Abfragen oder – wie hier im Beispiel von LINQ – mit an SQL angelehnten Abfragen beschäftigt, und es stellen sich Ihnen zwei Fragen:

- Wie formuliere ich eine Abfrage, um genau das Datenresultat zu bekommen, was ich mir vorstelle?
- Wie funktioniert LINQ, und wie gliedert der Compiler die LINQ-Abfrage in das Klassenkonzept und die Sprachelemente des .NET Frameworks ein?

Wir beginnen mit der Beantwortung der letzten Frage, denn die erste wird sich automatisch ergeben, wenn Sie verstanden haben, wie der Compiler diese Art von Abfragen verarbeitet und was im Grunde genommen dabei passiert.

Intern lebt LINQ und leben LINQ-Abfragen von Lambda-Ausdrücken. Und um zu verstehen, wie LINQ funktioniert, sollten Sie Lambda-Ausdrücke (Kapitel 4), Generics (Kapitel 6), Auflistungen und gerade auch generische Auflistungen (Kapitel 5 und Kapitel 6) verstanden haben.

LINQ basiert als allererstes auf einem Satz an Funktionen, die als Erweiterungsmethoden implementiert sind. Konkret wird die Schnittstelle `IEnumerable(Of type)` um Methoden erweitert, die sich ihrerseits wieder in der Klasse `Enumerable` befinden. Und bevor wir uns im Detail mit der LINQ-typischen Abfragesyntax beschäftigen, wollen wir als erstes diese Methoden ein wenig genauer unter die Lupe nehmen.

**BEGLEITDATEIEN** Sie finden die folgenden Beispiele in einem Projekt im Verzeichnis *\Samples\Chapter07 - Linq\LambdaLinqBasis*

Dabei sei Folgendes angemerkt: Die kommenden Abschnitte beschreiben die wichtigsten Erweiterungs-Methoden an verschiedenen Beispielen; sie erheben aber keinen Anspruch auf Vollständigkeit. Vielmehr geht es in den Abschnitten hauptsächlich darum, ein Verständnis für den Kontext zu vermitteln, in dem diese Methoden später bei der Verwendung der konkreten LINQ-Syntax durch den Visual Basic-Compiler zum Einsatz gelangen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Die Where-Methode

Das folgende Listing demonstriert die `Where`-Methode. Diese Methode ist eine Erweiterungsmethode für `IEnumerable(Of t)`, und steht damit auch an »normalen« Arrays zu Verfügung. Sie iteriert automatisch durch alle Elemente, und ruft für jedes Element eine *Predicat*-Bedingung (einen Delegaten, der für die Bedingungsprüfung jedes Elements zuständig ist – mehr zu Predicats, oder Prädikaten auf deutsch, finden Sie im vorherigen Kapitel) auf. Anstelle einer Delegatenvariablen kann hier natürlich auch ein Lambda-Ausdruck angegeben werden. Wird dieser Ausdruck als *wahr* ausgewertet – trifft die entsprechende Bedingung also zu –, wird das betroffene Element in eine neue Ergebnisliste überführt, die vom generischen Typ `IEnumerable` des entsprechenden Quelltyps ist: Im ersten Beispiel ist der Quelltyp vom Typ `Integer`, im zweiten vom Typ `String`.

```
Public Sub WhereBeispiel()
    ' Where iteriert durch alle Elemente der angegebenen
    ' Auflistung und führt für jedes Element die Prüfung in der Lambda-Funktion
    ' durch. Trifft sie zu, wird das entsprechende Element
    ' in die neue Auflistung überführt.
    ' Hier der erste Versuch mit Integer-Elementen ...
    Dim fnums = ziffern.Where(Function(einIntElement) einIntElement < 6)
    Console.WriteLine("Ziffern < 6")
    For Each intElement In fnums
        Console.WriteLine(intElement)
    Next
    Console.WriteLine("-----------------")

    ' ... und hier ein weiterer mit Strings.
    Dim fStrings = Namen.Where(Function(einName) einName.StartsWith("K"c))
    Console.WriteLine("Namen die mit 'K' beginnen")
    For Each strElement In fStrings
        Console.WriteLine(strElement)
    Next
End Sub
```

# Die Select-Methode

Weiter geht es mit der `Select`-Methode. Die `Select`-Methode ist in der Lage, ein Element der einen Auflistung gegen das entsprechende Elemente einer anderen Auflistung anderen Typs aufgrund seiner Ordinalnummer auszutauschen.

Auch hierzu ein Beispiel: Wir haben zwei Arrays, nämlich die folgenden beiden:

```
Private ziffern As Integer() = New Integer() {9, 1, 4, 2, 5, 8, 6, 7, 3, 0}
Private ziffernNamen As String() = New String() {"null", "eins", "zwei", "drei", _
                                                 "vier", "fünf", "sechs", "sieben", _
                                                 "acht", "neun"}
```

Mithilfe der `Select`-Anweisung können wir nun eine Auflistung aufbauen, die aufgrund der Inhalte der Elemente des ersten Arrays die Elemente des zweiten Arrays findet. Die folgende Methode gibt daher …

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Public Sub SelectBeispiel()
        ' Hier wird Select verwendet, um ein Element auf Grund seiner
        ' Wertigkeit durch ein zweites in einer anderen Auflistung zu ersetzen.
        Dim ziffernListe = ziffern.Select(Function(einIntElement) ziffernNamen(einIntElement))

        Console.WriteLine("Ziffern:")
        For Each s As String In ziffernListe
            Console.WriteLine(s)
        Next
    End Sub
```

… das folgende Ergebnis aus …

```
Ziffern:
neun
eins
vier
zwei
fünf
acht
sechs
sieben
drei
null
```

… das exakt der Reihenfolge der Inhalte des ersten Arrays entsprach.

## Anonymen Typen

Nun können wir dieses Spielchen mit der `Select`-Methode noch ein kleines bisschen weitertreiben, und lernen in diesem Zusammenhang das erste Mal den sinnvollen Einsatz anonymer Typen kennen. Anonyme Typen sind Klassen, die keinen Namen haben und direkt als Rückgabetypen – beispielsweise im Rahmen einer `Select`-Methode – definiert werden können.

Nun werden Sie denken: Wenn der Typ keinen Namen hat – wie soll ich dann eine Variable definieren von diesem nicht benannten Typen, oder gar eine Auflistung, die auf Basis eines anonymen Typs definiert wird? Möglich wird das auf Basis von lokalen Typrückschlüssen (siehe Kapitel 3) und Generics (Kapitel 6). Wir erinnern uns: Lokaler Typrückschluss bewirkt, dass Typen aufgrund ihrer initialen Zuweisung während der Deklarierung definiert werden – und das funktioniert natürlich auch für Typen, die keinen Namen haben, anonyme Typen eben:

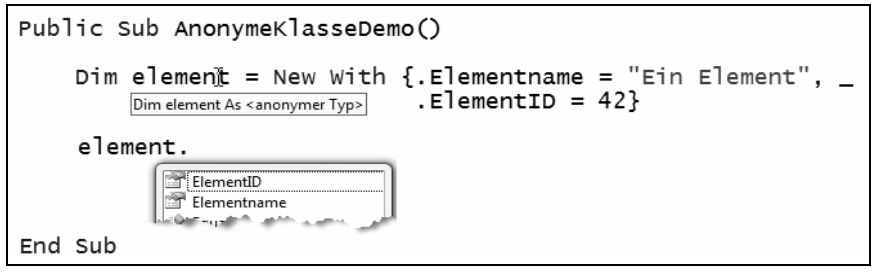



**Abbildung 7.2** Anonyme Klassen erlauben das Definieren von Typen, die keine Namen haben. Dank lokalem Typrückschluss lassen sich solche Typen aber dennoch an Objektvariablen zuweisen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## Typrückschluss für generische Typparameter

Für unser erweitertes `Select`-Beispiel brauchen wir eine weitere Technik, die sich Typrückschluss für generische Typparameter nennt. Dabei geht es darum, dass generische Funktionen ihre Typparameter über die ihnen tatsächlich übergebenen Parameter rückschließen, beispielsweise wie es in der folgenden Methode passiert:

```
Public Function ListeMitEinemElement(Of TSource)(ByVal dasEineElement As TSource) As _
                                                                                List(Of TSource)
    Dim lst As New List(Of TSource)
    lst.Add(dasEineElement)
    Return lst
End Function
```

Über den Sinn oder Unsinn dieser Routine kann man natürlich streiten, denn ihre Aufgabe ist folgende: Ihr wird eine Instanz beliebigen Typs übergeben, und die Methode macht daraus eine Auflistung, der sie exakt dieses Element hinzufügt. Es geht dabei auch weniger um die Aufgabe der Methode, sondern wie ihr Parameter übergeben werden können. Um diese Routine ohne Typrückschluss für generische Typparameter zu verwenden, müssten Sie sie folgendermaßen verwenden:

```
Dim dasEineElement As Integer = 10
Dim liste As List(Of Integer) = ListeMitEinemElement(Of Integer)(dasEineElement)
```

Nun wird `dasEineElement` als `Integer` definiert, und die Tatsache, dass Sie der Methode eine `Integer`-Variable übergeben, reicht dem Compiler, die Information zu ergänzen, um aus der generischen Methode eben eine vom Typ `Integer` zu machen. Ausreichend ist also:

```
Dim dasEineElement As Integer = 10
Dim liste As List(Of Integer) = ListeMitEinemElement(dasEineElement)
```

Dank Typrückschluss kann es jetzt noch einen Schritt weitergehen: Die im oben gezeigten Listing angegebenen Typdefinitionen sind eigentlich auch redundant: Die Konstante 10 definiert `dasEineElement` ausreichend als `Integer`. Und damit funktioniert der Typrückschluss für Generics natürlich auch bei der Zuweisung der zurückgegebenen Auflistung. Damit ergibt sich folgende, immer noch typmäßig ausreichend aufschlussreiche Sparversion des Beispiels:

```
Dim dasEineElement = 10
Dim liste = ListeMitEinemElement(dasEineElement)
```

Und da wir auf diese Weise sämtliche explizite Typangaben eliminieren konnten, funktioniert diese Sparversion natürlich auch für anonyme Typen, wie die folgende Abbildung zeigt:


**Abbildung 7.3** Ist der übergebene Parameter ein anonymer Typ, wird auch der generische Typparameter der generischen Methode anonym – genau wie die Auflistung, die die Methode in dessen Abhängigkeit zurückliefert

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Mit diesem Wissen ist das Verständnis der folgenden Routine ebenfalls kein Problem mehr. Wenn Sie das folgende Beispiel laufen lassen …

```
Public Sub SelectBeispielAnonym()
    ' Hier wird Select verwendet, um ein Element auf Grund seiner
    ' Wertigkeit durch ein zweites, anonymes in einer anderen Auflistung zu ersetzen.
    Dim anonymeListe = ziffern.Select(Function(einIntElement) _
                                      New With {.OriginalWert = einIntElement, _
                                                .Ziffernname = ziffernNamen(einIntElement)})
    Console.WriteLine("Inhalt der anonymen Klassenelemente:")
    For Each anonymerTyp In anonymeListe
        Console.WriteLine(anonymerTyp.OriginalWert & ", " & anonymerTyp.Ziffernname)
    Next
End Sub
```

erhalten Sie das folgende Ergebnis:

```
Inhalt der anonymen Klassenelemente:
9, neun
1, eins
4, vier
2, zwei
5, fünf
8, acht
6, sechs
7, sieben
3, drei
0, null
```

Und für diejenigen, die es genau interessiert, wie Methoden wie `Select` implementiert werden, sei die folgende Methode als Beispiel gezeigt, mit deren Hilfe sich Elemente einer Auflistung beliebigen Typs nummerieren lassen:

```
Public Function ItemNumberer(Of TSource, TResult)(ByVal source As IEnumerable(Of TSource), _
                    ByVal selector As Func(Of TSource, Integer, TResult)) As IEnumerable(Of TResult)

    Dim retColl As New List(Of TResult)
    Dim count = 1
    For Each item In source
        retColl.Add(selector(item, count))
        count += 1
    Next
    Return retColl.AsEnumerable
End Function
```

Diese Methode übernimmt eine generische `IEnumerable`-Auflistung eines beliebigen Typs und einen Lambda-Ausdruck, und liefert eine `IEnumerable`-Auflistung zurück, die durch den Rückgabewert des Lambda-Ausdrucks definiert wurde. Ziel ist es, durch alle Elemente der Quellauflistung hindurchzuiterieren, und der aufrufenden Methode über den Lambda-Ausdruck die Möglichkeit zu geben, eine aktuelle Elementnummer (`count`) in die Zielauflistung einzubauen. Mithilfe eines anonymen Typs könnte die Verwendung dieser Methode folgendermaßen aussehen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Public Sub ItemNumbererDemo()
        Dim numList = ItemNumberer(New String() {"eins", "zwei", "drei", "vier", "fünf", _
                                                 "sechs", "sieben", "acht", "neun", "zehn"}, _
                                   Function(element, count) New With {.ID = count, _
                                                                      .Element = element})

        For Each numListItem In numList
            Console.WriteLine(numListItem.ID & ": " & numListItem.Element)
        Next

    End Sub
```

Auch bei Aufruf dieser Methode wird wieder mit Rückschließen generischer Typparameter gearbeitet. Die Methode `ItemNumberer` wird nicht mit `(Of Irgendwas, Irgendwas)` genauer spezifiziert. Stattdessen leitet sich der Typ `TSource` durch das übergebene String-Array, der Rückgabewert durch den übergebenen, anonymen Typ ab.

Die anschließende Ausführung der Methode ergibt folgendes Ergebnis:

```
1: eins
2: zwei
3: drei
4: vier
5: fünf
6: sechs
7: sieben
8: acht
9: neun
10: zehn
```

# Die OrderBy-Methode

Mit der generischen `OrderBy`-Erweiterungsmethode können Sie Auflistungen quasi nach beliebigen Kriterien sortieren. Das zu sortierende Element wird als Argument dem Lambda-Ausdruck übergeben, und der Lambda-Ausdruck entscheidet dann quasi, nach welchem Kriterium die Auflistung sortiert wird.

Bei einer einfachen Auflistung aus String-Elementen, ist das Kriterium …

```
Public Sub OrderBy()
        'Für jedes Element liefert OrderBy das Sortierelement - bei einer Liste
        'von Strings ist das jedes Element selbst.
        Dim sortierteNamen = Namen.OrderBy(Function(einName) einName)

        For Each s In sortierteNamen
            Console.WriteLine(s)
        Next
End Sub
```

… gleich einem der Elemente, die sortiert werden sollen.

Anders wird das, wenn Sie Klasseninstanzen sortieren, wie das folgende Beispiel zeigt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Zunächst die Klasse, die die Elemente stellt:

```
Public Class Kontakt
    Private myID As Integer
    Private myVorname As String
    Private myNachname As String
    Private myOrt As String

    Public Property ID() As Integer
        Get
            Return myID
        End Get
        Set(ByVal value As Integer)
            myID = value
        End Set
    End Property

    Public Property Vorname() As String
        Get
            Return myVorname
        End Get
        Set(ByVal value As String)
            myVorname = value
        End Set
    End Property

    Public Property Nachname() As String
        Get
            Return myNachname
        End Get
        Set(ByVal value As String)
            myNachname = value
        End Set
    End Property

    Public Property Ort() As String
        Get
            Return MyOrt
        End Get
        Set(ByVal value As String)
            myOrt = value
        End Set
    End Property
End Class
```

Dann die Definition der Elemente:

```
Module Module1

    Private ziffern As Integer() = New Integer() {9, 1, 4, 2, 5, 8, 6, 7, 3, 0}
    Private ziffernNamen As String() = New String() {"null", "eins", "zwei", "drei", _
                                                     "vier", "fünf", "sechs", "sieben", _
                                                     "acht", "neun"}
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Private Namen As String() = New String() {"Jürgen", "Denise", "Angela", "Gaby", _
                                              "Andreas", "Katrin", "Klaus", "Arnold", _
                                              "Kerstin", "Anja", "Miriam", "Uta", "Momo", _
                                              "Leonie-Gundula"}

    Private Kontakte() As Kontakt = New Kontakt() { _
        New Kontakt With {.ID = 1, .Vorname = "Jürgen", .Nachname = "Heckhuis", .Ort = "Demohausen"}, _
        New Kontakt With {.ID = 2, .Vorname = "Ruprecht", .Nachname = "Dröge", .Ort = "Lippstadt"}, _
        New Kontakt With {.ID = 3, .Vorname = "Klaus", .Nachname = "Löffelmann", .Ort = "Demohausen"}, _
        New Kontakt With {.ID = 4, .Vorname = "Andreas", .Nachname = "Belke", .Ort = "Lippstadt"}, _
        New Kontakt With {.ID = 5, .Vorname = "Kerstin", .Nachname = "Lehnert", .Ort = "Ratingen"}, _
        New Kontakt With {.ID = 6, .Vorname = "Denise", .Nachname = "Wagner", .Ort = "Demohausen"}, _
        New Kontakt With {.ID = 7, .Vorname = "Anja", .Nachname = "Vielstedde", .Ort = "Lippstadt"}, _
        New Kontakt With {.ID = 8, .Vorname = "Angela", .Nachname = "Wördehoff", .Ort = "Demohausen"}, _
        New Kontakt With {.ID = 9, .Vorname = "Momo", .Nachname = "Weichel", .Ort = "Ratingen"}, _
        New Kontakt With {.ID = 10, .Vorname = "Leonie-Gundula", .Nachname = "Beckmann", _
                          .Ort = "Ratingen"}, _
        New Kontakt With {.ID = 11, .Vorname = "Miriam", .Nachname = "Sonntag", .Ort = "Lippstadt"} _
    }
.
.
.
```

Und schließlich die `OrderBy`-Routine, die die entsprechenden Elemente sortiert:

```
    Public Sub OrderBy()

        'Bei Klasseninstanzen liefert OrderBy die Instanz zum Sortieren,
        'und hier bestimmt die Eigenschaft, nach was sortiert wird.
        Dim sortierteKontakte = Kontakte.OrderBy(Function(einKontakt) einKontakt.Nachname)

        For Each kontaktElement In sortierteKontakte
            Console.WriteLine(kontaktElement.ID & ": " & _
                              kontaktElement.Nachname & ", " & _
                              kontaktElement.Vorname & ", " & _
                              kontaktElement.Ort)
        Next
    End Sub
```

## Sortieren nach Eigenschaften, die per Variable übergeben werden

In vielen Anwendungen stößt man auf Szenarien, bei denen es notwendig wird, dass beispielsweise eine Auflistung von Objekten nach wechselnden Kriterien sortiert wird, die sich zur Laufzeit festlegen lassen sollen.

In diesem Fall müssen Sie den eigentlichen Sortierstring – beispielsweise die Eigenschaft, die diesen für ein zu sortierendes Element ermittelt – dynamisch abrufen. Das funktioniert nur über .NET-Reflection. Reflection zu erklären, würde an dieser Stelle zu weit führen – Sie können sich aber, wie schon wiederholt vorgeschlagen, unter *www.activedevelop.de* das Buch *Visual Basic 2005 – Das Entwicklerbuch* kostenlos herunterladen, das Ihnen in einem eigenen Kapitel die Techniken von .NET Reflection näherbringt.

Die `OrderBy`-Methode per Reflection mit dem entsprechenden Sortierstring zu versorgen, ist jedenfalls nicht nur möglich, sondern auch kein großes Problem.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Dabei wird der Inhalt des Sortierkriteriums nicht direkt mit dem Eigenschaftennamen des Elements gelesen, sondern über einen Umweg:

```
Public Sub OrderBy(ByVal PropertyName As String)
    'Wenn die "Sortierspalte" frei wählbar sein soll, kann
    'das jeweilige Elemente nur über Reflection ermittelt werden:
    Dim sortierteNamen = Kontakte.OrderBy(Function(einKontakt) _
            einKontakt.GetType.GetProperty(PropertyName).GetValue(einKontakt, Nothing))

    'Funktioniert aber auch!
    For Each s In sortierteNamen
        Console.WriteLine(s.Nachname)
    Next
End Sub
```

1. Der Typ des Elements wird per `GetType` ermittelt.
2. `GetProperty` ermittelt anschließend ein so genanntes `PropertyInfo`-Objekt, das alle Informationen der Eigenschaft enthält und auch Zugriff auf den eigentlichen Wert gestattet. Da `GetProperty` der Eigenschaftenname als Argument übergeben wird, können die Eigenschafteninfos wie gewünscht dynamisch zur Laufzeit ermittelt werden. Man könnte diese Vorgehensweise quasi als »manuelles Late-Binding« bezeichnen.
3. `GetValue` ermittelt anschließend den eigentlichen Wert, und damit das Sortierkriterium.

Das Ergebnis zeigt, dass das Verfahren funktioniert, wenn die Methode folgendermaßen aufgerufen wird:

```
OrderBy("Nachname")
```

… führt zu folgender Ausgabe:

```
10: Beckmann, Leonie-Gundula, Ratingen
4: Belke, Andreas, Lippstadt
2: Dröge, Ruprecht, Lippstadt
1: Heckhuis, Jürgen, Demohausen
5: Lehnert, Kerstin, Ratingen
3: Löffelmann, Klaus, Demohausen
11: Sonntag, Miriam, Lippstadt
7: Vielstedde, Anja, Lippstadt
6: Wagner, Denise, Demohausen
9: Weichel, Momo, Ratingen
8: Wördehoff, Angela, Demohausen
```

# Die GroupBy-Methode

`GroupBy` ermöglicht es, Elemente einer Auflistung nach bestimmten Kriterien zu selektieren und in mehrere, verschiedene Auflistungsinstanzen zu überführen. Die einfachste Möglichkeit, die `GroupBy`-Erweiterungsmethode zu verwenden, ist im folgenden Beispiel durchgeführt:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Public Sub GroupBy()
        Dim gruppierteNamen = Kontakte.GroupBy(Function(einKontakt) einKontakt.Ort)

        For Each gruppenElement In gruppierteNamen
            Console.WriteLine(gruppenElement.Key)
            For Each kontaktElement In gruppenElement
                Console.WriteLine("---> " & kontaktElement.Nachname & ", " & kontaktElement.Vorname)
            Next
        Next
    End Sub
```

Wenn Sie dieses Beispiel ausführen, sehen Sie folgendes Ergebnis:

```
Demohausen
---> Heckhuis, Jürgen
---> Löffelmann, Klaus
---> Wagner, Denise
---> Wördehoff, Angela
Lippstadt
---> Dröge, Ruprecht
---> Belke, Andreas
---> Vielstedde, Anja
---> Sonntag, Miriam
Ratingen
---> Lehnert, Kerstin
---> Weichel, Momo
---> Beckmann, Leonie-Gundula
```

`GroupBy` hat also eine Auflistung zum Ergebnis, die ihrerseits wieder eine Auflistung kapselt. In der oberen Hierarchieebene gibt es eine `Key`-Eigenschaft, die dem Element entspricht, nachdem zuvor gruppiert wurde, und das den Zugriff auf die untergeordneten – also gruppierten – Elemente ermöglicht. Dementsprechend ist es am Geschicktesten, mit zwei ineinander verschachtelten `For/Each`-Schleifen durch alle Elemente zu iterieren, wie es das Beispiel auch zeigt.

In Kombination mit einer `Select`-Anweisung ist es darüber hinaus nicht nur möglich, sondern auch sinnvoll, eine neue Klasse anzulegen, die weiterführende Informationen über die Gruppen enthalten kann.

Das folgende Beispiel zeigt, wie eine solche Kombination realisiert werden kann:

```
    Public Sub GroupBy()
        Dim gruppierteKontakte = Kontakte.GroupBy(Function(einKontakt) _
                                                  einKontakt.Ort).Select(Function(eineGruppe) _
                                                    New With {.Ort = eineGruppe.Key, _
                                                    .Anzahl = eineGruppe.Count, _
                                                    .Elemente = eineGruppe.ToList})

        For Each gruppenElement In gruppierteKontakte
            Console.WriteLine(gruppenElement.Ort & " (Anzahl: " & gruppenElement.Anzahl & ")")
            For Each kontaktElement In gruppenElement.Elemente
                Console.WriteLine("---> " & kontaktElement.Nachname & ", " & kontaktElement.Vorname)
            Next
        Next
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In diesem Beispiel werden die vorhandenen Elemente in der Auflistung `Kontakte` nicht nur nach der Eigenschaft `Ort` gruppiert; die dabei entstehende Auflistung wird vielmehr in eine neue Auflistung übertragen, deren Schema durch einen anonymen Typ per `Select`-Methode definiert wird. Schön ist dabei auch zu erkennen, dass hier – am Beispiel der Eigenschaft `Anzahl` – Aggregat-Funktionen zum Einsatz kommen können, die Aufschluss über die eigentlichen `Kontakt`-Elemente geben, die den entsprechenden Gruppenelementen zugeordnet sind. Die Elemente selber werden ebenfalls »weitergereicht«, indem sie durch die Eigenschaft `ToList`, die eine generische `Kontakte`-Auflistung zurückgibt – in der Gruppenauflistung `gruppierteKontakte` durch die `Elemente`-Eigenschaft verfügbar gemacht wird.

# Kombinieren von LINQ-Erweiterungsmethoden

Nachdem Sie nunmehr ein paar der wichtigsten LINQ-Erweiterungsmethoden kennen gelernt haben, lassen Sie uns als nächstes einen Blick darauf werfen, wie sich diese Methoden miteinander vertragen.

Wie wir gelernt haben, geben alle diese Methoden Auflistungen zurück. Und diese Auflistungen haben alle eines gemeinsam: Sie implementieren, genau wie ihre Quellauflistungen – also die, aus denen sie entstanden sind – `IEnumerable(Of T)`. Das bedeutet aber weiterhin, dass die Ergebnisse der Erweiterungsmethoden sich wieder als Argument für eine weitere LINQ-Erweiterungsmethode nutzen lassen.

Ein Beispiel soll auch dies verdeutlichen:

```
Public Sub KombinierteErweiterungsmethoden()

    Dim ergebnisListe = Kontakte.Where(Function(einKontakt) einKontakt.Ort = "Lippstadt" _
                                      ).Select(Function(einKontakt) _
                                       New With {.ID = einKontakt.ID, _
                                                 .LastName = einKontakt.Nachname, _
                                                 .FirstName = einKontakt.Vorname, _
                                                 .City = einKontakt.Ort} _
                                                 ).OrderBy(Function(Contact) Contact.LastName)

    For Each contactItem In ergebnisListe
        Console.WriteLine(contactItem.ID & ": " & _
                    contactItem.LastName & ", " & _
                    contactItem.FirstName & " living in " & _
                    contactItem.City)

    Next
End Sub
```

Wenn Sie die Erweiterungsmethoden dieses Beispiels in dieser Reihenfolge nacheinander angewendet sehen, ahnen Sie bestimmt bereits, was bei LINQ-Abfragen, wie Sie sie am Anfang dieses Kapitels kennen gelernt haben, wirklich passiert.

Aber lassen Sie uns nicht zwei Schritte auf einmal machen, sondern die Methodenkombination, die Sie im obigen Listing fett hervorgehoben sehen, erklärungstechnisch Schritt für Schritt auseinander nehmen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

1. Als erstes verarbeitet das Programm die Ausgangsliste `Kontakte` mithilfe der `Where`-Erweiterungsmethode. In diesem konkreten Beispiel generiert die `Where`-Methode eine neue, `IEnumerable(Of t)`-implementierende Auflistung, die nur jene Elemente enthält, deren Ort der Zeichenkette »Lippstadt« entspricht.
2. Das Ergebnis dieser Auflistung ist Gegenstand der nächsten Erweiterungsmethode: `Select`. `Select` macht nichts anderes, als ein Element der ersten Auflistung durch ein anderes Element auszutauschen und dieses neue Element in einer abermals neu erstellten Auflistung einzugliedern. Die `Select`-Methode dieses konkreten Beispiels erstellt dazu eine anonyme Klasse, die prinzipiell den alten Elementen ähnlich ist, aber englische Eigenschaftenbezeichnungen enthält. Die alten Objekte, auf Basis der Klasse `Kontakte`, werden also als Basis für eine Reihe von neuen Objekten verwendet, die von der `Select`-Methode nacheinander angelegt werden. Wir erhalten so eine Auflistung mit anonymen Klasseninstanzen, die sich durch englische Eigenschaftenbezeichner auszeichnen.
3. Diese Auflistung wird abermals Gegenstand einer Methode – dieses Mal ist es die Erweiterungsmethode `OrderBy`. Sie nimmt nun die anonymen Klasseninstanzen mit den englischen Eigenschaftenbezeichnern und sortiert diese nach Nachnamen (`Contact.LastName`) in einer wiederum neuen Auflistung ein.
4. Diese zuletzt generierte Auflistung ist schließlich die Ergebnisauflistung, die der Objektvariablen `ergebnisListe` zugewiesen und in der anschließenden `For/Each`-Schleife ausgegeben wird:

```
4: Belke, Andreas living in Lippstadt
2: Dröge, Ruprecht living in Lippstadt
11: Sonntag, Miriam living in Lippstadt
7: Vielstedde, Anja living in Lippstadt
```

# Vereinfachte Anwendung von LINQ – Erweiterungsmethoden mit der LINQ-Abfragesyntax

OK – viel zu sagen gibt es jetzt nicht mehr, ich hoffe, dass Ihnen – um es mit einer Metapher zu sagen – mein vielleicht etwas von hinten aufgezäumtes Pferd didaktisch genehm war.

Ein Blick auf das folgende Listing reicht meiner Meinung nach jetzt aus, um zu verstehen, was der Visual Basic-Compiler aus einem LINQ-Ausdruck macht, und wie er ihn – jedenfalls im Fall von *LINQ to Objects* – in eine Kombination aus Erweiterungsmethoden umsetzt.

Die Abfrage des folgenden Listings entspricht nämlich exakt …

```
Public Sub KombinierteErweiterungsmethoden_á_la_LINQ()

    Dim ergebnisListe = From einKontakt In Kontakte _
                        Where einKontakt.Ort = "Lippstadt" _
                        Select ID = einKontakt.ID, _
                               LastName = einKontakt.Nachname, _
                               FirstName = einKontakt.Vorname, _
                               City = einKontakt.Ort _
                        Order By LastName
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        For Each contactItem In ergebnisListe
            Console.WriteLine(contactItem.ID & ": " & _
                        contactItem.LastName & ", " & _
                        contactItem.FirstName & " living in " & _
                        contactItem.City)

        Next

    End Sub
```

… der Methodenkombination des Listings im vorherigen Abschnitt. Und natürlich liefert es auch das exakt gleiche Ergebnis – probieren Sie es aus!

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 8

# LINQ to Objects

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Einführung in LINQ to Objects

Nach der Lektüre des vorherigen Kapitels wissen Sie ja im Grunde genommen schon, wie LINQ to Objects funktioniert, was gibt es also in Sachen Einführung zu LINQ to Objects noch zu sagen? – Was die generelle, interne Funktionsweise anbelangt, im Grunde gar nichts. Was die Auswirkungen von LINQ to Objects auf Ihre tägliche Routine anbelangt vielleicht eine ganze Menge!

Letzten Endes geht es nämlich beim Einsatz von LINQ to Objects nicht darum, möglichst eine Datenbank mit Business-Objekten quasi im Managed Heap Ihrer .NET-Anwendung nachzubilden – auch wenn die Beispiele dieses Kapitels das vielleicht vermuten lassen.

Nein? – Nein!

LINQ to Objects soll es Ihnen ermöglichen, die im letzten Kapitel vorgestellte, an die SQL-Abfragesprache angelehnte Syntax auch für ganz andere Zwecke, an die Sie im Moment vielleicht noch gar nicht denken, in ihre Programme einzubauen. Doch dazu müssen Sie wissen, mit welchen Komponenten einer Abfrage Sie welche Ergebnisse erzielen können – und dabei soll Ihnen dieses Kapitel behilflich sein. Aus diesem Grund wählen wir als Beispielträger auch einfachste Objektauflistungen (übrigens exakt die des letzten Kapitels), damit Sie sich bei den Beispielen auf LINQ konzentrieren können und nicht von Funktionalitäten anderer Auflistungsklassen des Systems abgelenkt werden. Dennoch können Sie LINQ nicht nur für ihre selbstgeschriebenen Business-Klassen und -Auflistungen einsetzen. Natürlich lassen sich auch Verzeichnis-, Grafik- oder Steuerelement-Auflistungen mit LINQ bearbeiten und organisieren.

**HINWEIS** Dieses Kapitel kann übrigens dabei keinen Anspruch auf Vollständigkeit erheben – dazu ist das Thema LINQ einfach zu mächtig und umfangreich. Es wird Ihnen aber auf jeden Fall ausreichend Informationen, Beispiele und Anregungen zum Thema liefern, so dass Sie Ihr Wissen durch eigenes Ausprobieren und aufmerksames Lesen der Online-Hilfe sicherlich perfektionieren können. Das Visual Basic 2008 Entwicklerbuch, das voraussichtlich gegen Ende des 2. Quartals erscheinen wird, hat zum Thema LINQ noch mehr Informationen und Beispiele parat.

Übrigens: Der Einsatz von LINQ kann immer dann sinnvoll sein, wenn Sie mit gleich welchen Auflistungen arbeiten müssen. Das gilt vor allen Dingen unter Beachtung des folgenden Aspektes, dem ich – wegen seiner Zukunftsträchtig- und -wichtigkeit –, einen eigenen Abschnitt widmen will; sicherlich ein wenig spekulativ, aber deswegen nicht minder wahrscheinlich – jedenfalls nach meinem bescheidenen Ermessen!

## Verlieren Sie die Skalierbarkeit von LINQ nicht aus den Augen!

Denken Sie daran: LINQ als *Engine* zur Verarbeitung von internen .NET-Auflistungen ist skalierbar. Worauf ich hinaus will, möchte ich an einem einfachen Denkmodell demonstrieren:

Stellen Sie sich vor, Sie benötigen in Ihrer Anwendung einen Algorithmus zur Sortierung einer Auflistung. Dann können Sie, was gleichermaßen unnötig und unklug wäre, einen eigenen Algorithmus dazu schreiben. Unnötig, weil es bereits ausreichend Sortieralgorithmen im .NET Framework für alle möglichen Auflistungstypen gibt, und unklug, weil Sie selbst dafür zuständig wären, Ihren Algorithmus zu optimieren, beispielsweise weil Sie bemerken, dass Sie – was passieren wird – es nur noch mit Multi-Core-Prozessoren zu tun haben, und Ihre Anwendung, weil sie ständig (aber ständig nur auf einem Core des Prozessors) sortiert, nur 50%, 25% oder gar 12,5% der gesamt zur Verfügung stehenden Prozessorleistung ausschöpft. Ihr Algorithmus arbeitet nämlich nicht multithreaded.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Es gibt bei LINQ bereits experimentelle Ansätze und Konzepte,[1] die durchzuführenden Operationen mit mehreren Prozessoren (oder eben mehreren Prozessor-Cores) zu parallelisieren – jedenfalls soweit das möglich ist. Eine LINQ-Abfrage würde dann eine Kombination von Erweiterungsmethoden »auslösen«, die ihre eigentliche Arbeit in mehrere Threads und damit mehrere Cores aufteilen – das Resultat wäre eindrucksvoll: Gleicher Programmaufwand, doppelte, vierfache oder sogar achtfache Leistung!

LINQ anstelle normaler, selbstgestrickter `For/Each`-Auflistungsverarbeitungsschleifen lohnt sich aus planungstechnischer Sicht also auf jeden Fall – mal ganz abgesehen davon, dass Ihr Coding-Aufwand auch geringer wird und damit die Fehleranfälligkeit Ihres Programms abnimmt.

# Der Aufbau einer LINQ-Abfrage

LINQ-Abfragen beginnen stets[2] mit der Syntax

```
From bereichsVariable In DatenAuflistung
```

und offen gesagt: Wenn ich auch die Syntax aus Programmiersicht nachvollziehen konnte, so verstand ich zunächst nicht, warum es unter rein sprachlichen Aspekten ausgerechnet `From` heißen musste – was wird denn da eigentlich von der Bereichsvariablen genommen, entwendet oder verwendet?

Also begann ich, ein wenig im Internet zu recherchieren, und ich stellte fest, dass nicht nur ich mir, sondern sich auch englische Muttersprachler diese Frage stellten, und so bemühte ich mich um ein wenig mehr Hintergrundinfos zur Entstehungsgeschichte der LINQ-Syntax bei jemanden, von dem ich glaubte, dass er es wissen müsse – und ich hatte Erfolg:

### Lisa Feigenbaum vom Visual Basic Team über die Syntax von LINQ-Abfragen

Je mehr ich darüber nachdachte, wieso ausgerechnet `From` (und vielleicht nicht `Through` oder `For`) eine LINQ-Abfrage einleitet, desto mehr ließ mich auch die Frage nicht mehr in Ruhe, wieso eine LINQ-Abfrage nicht mit `Select` eingeleitet würde – so, wie wir Entwickler es schließlich schon seit Jahren von SQL-Server-Abfragen gewohnt waren.

Ich stellte diese Frage Lisa Feigenbaum vom Visual Basic Team, und sie antwortete mir dazu Folgendes:

*»From« deutet auf Quelle oder Ursprungsort hin. Der* `From`*-Ausdruck als solcher ist die Stelle, an der man die Abfragequelle bestimmt (beispielsweise den Datenkontext, eine Auflistung im Speicher, XML, etc.). Aus diesem Grund fanden wir diesen Terminus als angemessen. Darüber hinaus handelt es sich bei »From« auch um ein SQL-Schlüsselwort. Wir haben uns bei LINQ um größtmöglichen SQL-Erhalt bemüht, sodass LINQ-Neueinsteiger, die bereits über SQL-Erfahrung verfügten, die Ähnlichkeit bereits »erfühlen« konnten. Ein interessanter Unterschied zwischen LINQ und SQL, den wir jedoch einbauen mussten, ist die Reihenfolge [der Elemente] innerhalb des Abfrageausdrucks. Bei SQL steht* `Select` *vor* `From`*. In LINQ ist es genau umgekehrt.* ▶

[1] *http://msdn.microsoft.com/msdnmag/issues/07/10/PLINQ/default.aspx?loc=de* stellt einen interessanten Artikel zu PLINQ (Parallel LINQ) zur Verfügung (Stand: 24.01.2007).

[2] Ausnahmen bilden reine Aggregat-Abfragen, die auch mit der Klausel `Aggregate` beginnen, wie im entsprechenden Abschnitt dieses Kapitels beschrieben.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

*Einer der großen Vorteile von LINQ ist, dass man mit verschiedenen Datenquellen in einem gemeinsamen Modell arbeiten kann. Innerhalb einer Abfrage arbeitet man nur mit Objekten, und wir können IntelliSense für diese Objekte zur Verfügung stellen. Der Grund dafür, dass wir* `From` *als erstes benötigen, ist, dass wir als erstes wissen müssen, von woher die Quelle stammt, über die wir die Abfrage durchführen, bevor wir die IntelliSense-Information zur Verfügung stellen können.*[3]

Sprachlich zu verstehen ist `From` also im Kontext des gesamten ersten Abfrageteilausdrucks – also quasi `From (bereichsVariable in Auflistung)` – und bezieht sich *nicht* nur auf die Bereichsvariable, die hinter `From` steht. Und nachdem dieser Ausdruck nun auch sprachlich geklärt ist, fassen wir zusammen:

`From` leitet eine LINQ-Abfrage ein, und er bedeutet ins »menschliche« übersetzt: »Hier, laufe mal bitte durch alle Elemente, die hinter dem Schlüsselwort `In` stehen, und verwende die Bereichsvariable hinter `From`, um mit deren Eigenschaften bestimmen zu können, wie die Daten der Auflistung selektiert, sortiert, gruppiert und dabei in eine neue Elementliste (mit – bei Bedarf – Instanzen anderer Elementklassen) überführt werden können.

**WICHTIG** Wichtig für die Profi-T-SQL-ler unter Ihnen ist übrigens: LINQ-Abfragen beginnen niemals mit `Select`, ja, sie müssen noch nicht einmal ein `Select` aufweisen. `DELETE`, `INSERT` und `UPDATE` gibt es im Übrigen auch nicht – auch nicht wie im nächsten Kapitel zu lesen, in *LINQ to SQL*.

Mit dem Wissen des vorherigen Kapitels schauen wir uns jetzt mal die folgenden beiden Abfragen an und überlegen uns, wie sie sich im Ergebnis unterscheiden.

**BEGLEITDATEIEN** Im Verzeichnis *\Samples\Chapter08 - Linq\LinqToObjects* finden Sie eine Konsolen-Beispielanwendung, die mit Business-Objekten weitere Möglichkeiten von *LINQ to Objects* erklären soll. Das Programm entspricht wieder dem aus dem letzten Kapitel: Es generiert per Zufallsgenerator zwei Auflistungen mit Business-Objekten (eine Kunden-, eine Artikelbestellauflistung) die logisch zu einander in Relation stehen. Das vorherige 7. Kapitel erklärt mehr zum Aufbau des Beispiels.

OK. Wir schauen uns zunächst das folgende Listing des Beispielprogramms an, und das schaut so aus:

```
Sub LINQAbfragenAufbau()

    Console.WriteLine("Ergebnisliste 1:")
    Console.WriteLine("---------------")

    Dim ergebnisListe = From adrElement In adrListe _
                        Order By adrElement.Nachname, adrElement.Vorname Descending _
                        Select KontaktId = adrElement.ID, _
                               adrElement.Nachname, _
                               adrElement.Vorname, _
                               PlzOrtKombi = adrElement.PLZ & " " & adrElement.Ort _
                        Where KontaktId > 50 And KontaktId < 100 And _
                               Nachname Like "L*"
```

[3] Quelle: Lisa Feigenbaum in einer E-Mail vom 01.02.2008. Lisa Feigenbaum ist VB IDE Program Manager im Visual Basic-Team bei Microsoft. Unter *http://blogs.msdn.com/vbteam/default.aspx,* der Team Blog-Seite, erreichen Sie Lisa Feigenbaum und das Visual Basic Team (Stand 03.02.2008).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        For Each ergebnisItem In ergebnisListe
            With ergebnisItem
                Console.WriteLine(.KontaktId & ": " & _
                                  .Nachname & ", " & .Vorname & _
                                  " - " & .PlzOrtKombi)
            End With
        Next

        Console.WriteLine()
        Console.WriteLine("Ergebnisliste 2:")
        Console.WriteLine("----------------")


        Dim ergebnisListe2 = From adrElement In adrListe _
                    Where adrElement.ID > 50 And adrElement.ID < 100 And _
                           adrElement.Nachname Like "L*" _
                    Select KontaktId = adrElement.ID, _
                           adrElement.Nachname, _
                           adrElement.Vorname, _
                           PlzOrtKombi = adrElement.PLZ & " " & adrElement.Ort _
                    Order By Nachname, Vorname

        For Each ergebnisItem In ergebnisListe2
            With ergebnisItem
                Console.WriteLine(.KontaktId & ": " & _
                                  .Nachname & ", " & .Vorname & _
                                  " - " & .PlzOrtKombi)
            End With
        Next
    End Sub
```

Sie sehen, dass sich die Abfragen syntaktisch und damit auf den ersten Blick schon einmal gar nicht gleichen. Und dennoch, wenn Sie das Beispiel starten, so sehen Sie ein Ergebnis etwa wie im folgenden Bildschirmauszug…

```
Ergebnisliste 1:
----------------
81: Langenbach, Barbara - 78657 Wiesbaden
76: Langenbach, Klaus - 61745 Dortmund
54: Lehnert, Katrin - 82730 Wiesbaden
96: Löffelmann, Barbara - 63122 Rheda
68: Löffelmann, Gabriele - 54172 Bad Waldliesborn

Ergebnisliste 2:
----------------
81: Langenbach, Barbara - 78657 Wiesbaden
76: Langenbach, Klaus - 61745 Dortmund
54: Lehnert, Katrin - 82730 Wiesbaden
96: Löffelmann, Barbara - 63122 Rheda
68: Löffelmann, Gabriele - 54172 Bad Waldliesborn
```

… und obwohl diese Liste – laut Codelisting – mal definitiv auf zwei verschiedenen Ereignisauflistungen beruht, die aus zwei verschiedenen Abfragen entstanden sind, ist das Ergebnis dennoch augenscheinlich dasselbe. Zufall? Keinesfalls – im Gegenteil, pure Absicht.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Denn dieses Beispiel demonstriert sehr schön, wie LINQ-Abfragen im Code funktionieren und wie sehr sie im Grunde genommen nichts mit den klassischen SQL-Datenbankabfragen zu tun haben. Gut, zugegeben, Ähnlichkeiten sind vorhanden, aber das war's auch schon.

Nun lassen Sie uns mal beide Abfragen auseinandernehmen und dabei schauen, wieso so unterschiedlich formuliertes dennoch zum gleichen Ergebnis führen kann. Los geht's:

- Die erste Abfrage startet, wie jede LINQ-Abfrage, mit der `From`-Klausel, und diese definiert `adrElement` als *Bereichsvariable* für alle kommenden Parameter bis – so vorhanden – zum ersten `Select`. Die Bereichsvariable ist also bis zum ersten `Select`-Befehl die Variable, mit der quasi intern durch die komplette Auflistung hindurchiteriert wird, und an der damit auch die Eigenschaften bzw. öffentlichen Felder der Auflistungselemente »hängen«.
- Es folgt hier im Beispiel (aber natürlich nicht notwendigerweise) die Sortieranweisung `Order By`, der die Felder bzw. Eigenschaften, nach denen die Elemente der Auflistung sortiert werden sollen, als Argumente über die Bereichsvariable übergeben werden: das sind in diesem Beispiel `adrElement.Nachname` und `adrElement.Vorname`. Die Bereichsvariable dient also dazu, auf die Eigenschaften zuzugreifen, über die durch die `Order By`-Klausel festgelegt werden kann, nach welchen Feldern sortiert wird.

**TIPP** Wenn nichts anderes gesagt wird, sortiert `Order By` *aufsteigend*. Durch die Angabe des Schlüsselwortes `Descending` können Sie den Sortierausdruck abändern, so dass die Elementauflistung *absteigend* sortiert wird. Wollte man – um beim Beispiel zu bleiben – die erste Liste nach Namen aufsteigend und nach Vornamen absteigend sortieren, hieße der LINQ-Abfrageausdruck folgendermaßen:

```
Dim ergebnisListe = From adrElement In adrListe _
                    Order By adrElement.Nachname, adrElement.Vorname Descending _
                    Select KontaktId = adrElement.ID, _
                           adrElement.Nachname, _
                           adrElement.Vorname, _
                           PlzOrtKombi = adrElement.PLZ & " " & adrElement.Ort _
                    Where KontaktId > 50 And KontaktId < 100 And _
                            Nachname Like "L*"
```

- Jetzt folgt ein `Select`, und Achtung: Wir haben ja im letzten Kapitel erfahren, dass `Select` dazu dient, ein Element einer Auflistung durch ein anderes zu ersetzen, das in einer andere Auflistung »landet«. Und dieses zu ersetzende Element wird eine Instanz einer neuen anonymen Klasse mit exakt den Feldern sein, die Sie hinter dem `Select` bestimmen.

**TIPP** Spätestens hier sehen Sie, dass ein `Select` für eine Abfrage, die sich auf nur eine Tabelle bezieht, und die keine Gruppierungen durchführt, eigentlich ein reiner Zeitkiller ist, denn Sie können natürlich mit *den* Ausgangselementen weiterarbeiten, die *ohnehin* schon da sind und in der späteren Ergebnisliste *schlimmstenfalls* ein paar redundante Informationen aufweisen. Anders als bei SQL-Abfragen benötigen Sie das `Select`-Schlüsselwort bei LINQ *nicht*, um Abfragen für Filterungen, Selektierungen oder Gruppierungen einzuleiten, sondern *nur*, um eine *neue Auflistung* mit Elementen zu erstellen, die aus einem neuen, anonymen Typ bestehen, der nur die Eigenschaften offenlegt, die Sie als Feldnamen angeben. Beherzigen Sie deswegen auch das, was der nächste Abschnitt über `Select` zu berichten weiß!

- Nachdem wir nun das `Select` hinter uns gelassen und der Abfrage-Engine damit mitgeteilt haben, dass wir nur noch mit `KontaktId`, `Nachname`, `Vorname` und einem `PlzOrtKombi`-Feld weitermachen wollen, können sich weitere LINQ-Abfrageelemente auch nur noch auf diese, durch `Select` neu eingeführte anonyme

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Klasse beziehen – dementsprechend würde hinter der `Where`-Klausel ein `adrElement.Id` als Argument nicht mehr funktionieren; `Select` hat `adrElement` der Auflistung schließlich durch Instanzen einer anonymen Klasse ersetzt, und die haben nur noch die Eigenschaft `KontaktID`. `Where` dient jetzt dazu, die neue Auflistung mit den Elementen der anonymen Klasse zu filtern: nur Elemente mit `KontaktID > 50` und `KontaktID < 100` sind mit von der Partie, so wie die Elemente, deren Nachname mit »L« beginnt.

Damit ist die erste Abfrage durch. Und jetzt schauen wir uns die zweite an.

- Hier geht es nach der obligatorischen Festlegung von Auflistung und Bereichsvariable, die als Elementiterator dienen soll, zunächst los mit der `Where`-Klausel, die die Elemente auf jene IDs einschränkt, die größer als 50 und kleiner als 100 sind. Die Einschränkung wird dann noch weiterführt, nämlich auf Nachnamen, die mit dem Buchstaben »L« beginnen. Hier kann `Where` direkt auf die Eigenschaften zurückgreifen, die über die Bereichsvariable erreichbar sind, denn sie entsprechen einem Element der Ausgangsauflistung.
- Das geht solange, bis es auch hier wieder ein `Select` gibt, was dem ein Ende setzt. Auch hier wird wieder ein Ersetzungsvorgang durchgeführt – unnötigerweise und nur zu Demonstrationszwecken –, der die Elemente der vorhandenen Auflistung durch neue einer anonymen Klasse ersetzt, die wiederum die angegebenen Eigenschaften haben.
- Das anschließende `Where` schränkt diese Auflistung wieder ein, und zwar prinzipiell in gleicher Weise, wie im ersten Beispiel – lediglich der Zeitpunkt, wann die Elemente oder welche Elemente eingeschränkt werden, ist ein anderer.

Eines ist in diesem Zusammenhang sehr wichtig zu erwähnen:

**WICHTIG** LINQ-Abfragen werden immer verzögert ausgeführt, und niemals direkt. Das bedeutet: Nicht die eigentliche Abfrage ist ausschlaggebend für das »Anstoßen« der Verarbeitung, sondern erst das erste Zugreifen auf die Ergebnisliste löst das Ausführen der eigentlichen Abfrage aus. Mehr dazu erfahren Sie im Abschnitt »Kombinieren und verzögertes Ausführen von LINQ-Abfragen«.

Die Ausführung von `Select` ergibt bei der Anwendung von nur einer Auflistung nur eingeschränkt Sinn. Das folgende Beispiel zeigt, wieso `Select` Rechenleistung kosten kann ohne einen nachvollziehbaren Nutzen dabei zu bringen.

Dieses Beispiel arbeitet mit einem Hochgeschwindigkeitszeitmesser, um die Laufdauer bestimmter Auswertungen zu messen. Dabei wird eine Auflistung mit 2.000.000 Kundenelementen einmal mit und einmal ohne `Select`-Abfrage ausgeführt.

**HINWEIS** Achten Sie beim Einrichten dieses Beispiels darauf, die ersten Zeilen des Beispielprojektes folgendermaßen abzuändern:

```
Module LinqDemo

    Private adrListe As List(Of Kontakt) = Kontakt.ZufallsKontakte(2000000)
    Private artListe As List(Of Artikel) '= Artikel.Zufallsartikel(adrListe)
```

Achten Sie darauf, diese Zeilen für die anderen Beispiele wieder zurückzubauen.

Die eigentliche Methode, die das Beispiel enthält, sieht wie folgt aus:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Sub SelectSpeedCompare()

        Dim hsp As New HighSpeedTimeGauge
        Console.WriteLine("Starte Test")
        hsp.Start()
        Dim ergebnisListe = From adrElement In adrListe _
                            Order By adrElement.Nachname, adrElement.Vorname _
                            Select KontaktId = adrElement.ID, _
                                   adrElement.Nachname, _
                                   adrElement.Vorname, _
                                   PlzOrtKombi = adrElement.PLZ & " " & adrElement.Ort

        Dim ersteAnzahl = ergebnisListe.Count
        hsp.Stop()
        Dim dauer1 = hsp.DurationInMilliSeconds

        hsp.Reset()
        hsp.Start()
        Dim ergebnisListe2 = From adrElement In adrListe
                             Order By adrElement.Nachname, adrElement.Vorname

        Dim zweiteAnzahl = ergebnisListe2.Count
        hsp.Stop()
        Dim dauer2 = hsp.DurationInMilliSeconds

        Console.WriteLine("Abfragedauer mit Select: " & dauer1 & " für " & ersteAnzahl & " Elemente.")
        Console.WriteLine("Abfragedauer ohne Select: " & dauer2 & " für " & zweiteAnzahl & " Elemente.")
    End Sub
```

Das Ergebnis dieses Tests offenbart den Unterschied. Bei 2.000.000 Elementen kommt es …

```
Starte Test
Abfragedauer mit Select: 16939 für 2000000 Elemente.
Abfragedauer ohne Select: 16061 für 2000000 Elemente.
```

… bis zum Unterschied von einer Sekunde. Natürlich ist das kein Wert, an dem man sich für die Praxis in irgendeiner Form orientieren sollte; das Beispiel ist – schon klar – zudem natürlich auch praxisfern, denn wer zwei Millionen Datensätze im Speicher hält, hat entweder eine Datenbankengine programmiert oder das Konzept von Datenbanken und Datenbankabfragen nicht verstanden. Aber es geht nur um Tendenzen und den Tipp, Prozessorleistung nicht an Stellen unnötig zu verheizen, an denen es nicht nötig ist.

Übrigens, und besser könnte diese Überleitung zum nächsten Abschnitt nicht sein, wenn Sie sich das vorherige Listing anschauen, sollte Ihnen etwas auffallen, das Sie erstaunen sollte. Sie sehen es nicht? OK – dann überlegen Sie sich mal, ob es Zufall ist, dass die `Count`-Eigenschaft noch innerhalb der Zeitmessung abgefragt wird. Denn: Das ist nicht nur bewusst so gemacht, anderenfalls würden Sie komplett andere Ergebnisse bekommen, denn wenn Sie das Listing folgendermaßen abändern …

```
    Sub SelectSpeedCompare()
.
.
.
        hsp.Stop()
        Dim dauer1 = hsp.DurationInMilliSeconds
        Dim ersteAnzahl = ergebnisListe.Count
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        hsp.Reset()
        hsp.Start()
        Dim ergebnisListe2 = From adrElement In adrListe _
                             Order By adrElement.Nachname, adrElement.Vorname

        hsp.Stop()
        Dim dauer2 = hsp.DurationInMilliSeconds
        Dim zweiteAnzahl = ergebnisListe2.Count
.
.
.
    End Sub
```

… erhalten Sie auf einmal das folgende Ergebnis!

```
Starte Test
Abfragedauer mit Select: 1 für 2000000 Elemente.
Abfragedauer ohne Select: 0 für 2000000 Elemente.
```

Ein Fehler? Nein – das ist genau das Ergebnis, was bei der Ausführung herauskommen muss!

# Kombinieren und verzögertes Ausführen von LINQ-Abfragen

Eines vorweg: LINQ-Abfragen werden immer verzögert ausgeführt, die Überschrift könnte also insofern in die Irre führen, als dass sie impliziert, der Entwickler, der sich LINQ-Abfragen bedient, hätte eine Wahl. Er hat sie nämlich nicht.

Wann immer Sie eine Abfrage definieren, und sei sie wie die folgende …

```
        Dim adrListeGruppiert = From adrElement In adrListe _
                                Join artElement In artListe _
                                On adrElement.ID Equals artElement.IDGekauftVon _
                                Select adrElement.ID, adrElement.Nachname, _
                                       adrElement.Vorname, adrElement.PLZ, _
                                       artElement.ArtikelNummer, artElement.ArtikelName, _
                                       artElement.Anzahl, artElement.Einzelpreis, _
                                       Postenpreis = artElement.Anzahl * artElement.Einzelpreis _
                                Order By Nachname, ArtikelNummer _
                                Where (PLZ > "0" And PLZ < "50000") _
                                Group ArtikelNummer, ArtikelName, _
                                      Anzahl, Einzelpreis, Postenpreis _
                                By ID, Nachname, Vorname _
                                Into Artikelliste = Group, AnzahlArtikel = Count(), _
                                     Gesamtpreis = Sum(Postenpreis)
```

… noch so lang; die Abfragen selbst werden nicht zum Zeitpunkt ihres »Darüberfahrens« ausgeführt. Im Gegenteil: In `adrListeGruppiert` wird – um bei diesem Beispiel zu bleiben – im Prinzip nur eine Art *Ausführungsplan* generiert, also eine Liste der Methoden, die nacheinander ausgeführt werden, sobald ein Element aus der (noch zu generierenden!) Ergebnisliste abgerufen wird, oder Eigenschaften bzw. Funktionen der Ergebnisliste abgerufen werden, die in unmittelbarem Zusammenhang mit der Ergebnisliste selbst stehen – wie beispielsweise die `Count`-Eigenschaft.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Und es wird noch besser: Verschiedene LINQ-Abfragen lassen sich so nacheinander »aufreihen«, wie das folgende Beispiel eindrucksvoll zeigt.

**HINWEIS** Achten Sie beim Einrichten dieses Beispiels darauf, die ersten Zeilen des Beispielprojektes folgendermaßen abzuändern:

```
Module LinqDemo

    Private adrListe As List(Of Kontakt) = Kontakt.ZufallsKontakte(500000)
    Private artListe As List(Of Artikel) '= Artikel.Zufallsartikel(adrListe)
```

Achten Sie darauf, diese Zeilen für die anderen Beispiele wieder zurückzubauen.

```
    Sub SerialLinqsCompare()

        Dim hsp As New HighSpeedTimeGauge
        Console.WriteLine("Starte Test")
        hsp.Start()
        Dim ergebnisListe = From adrElement In adrListe _
                            Order By adrElement.Nachname, adrElement.Vorname

        Dim ergebnisListe2 = From adrElement In ergebnisListe _
                             Where adrElement.Nachname Like "L*"

        ergebnisListe2 = From adrElement In ergebnisListe2 _
                         Where adrElement.ID > 50 And adrElement.ID < 200

        Dim ersteAnzahl = ergebnisListe2.Count
        hsp.Stop()
        Dim dauer1 = hsp.DurationInMilliSeconds

        hsp.Reset()
        hsp.Start()
        Dim ergebnisListe3 = From adrElement In adrListe _
                             Order By adrElement.Nachname, adrElement.Vorname _
                             Where adrElement.Nachname Like "L*" And _
                             adrElement.ID > 50 And adrElement.ID < 200
        Dim zweiteAnzahl = ergebnisListe3.Count
        hsp.Stop()
        Dim dauer2 = hsp.DurationInMilliSeconds

        Console.WriteLine("Abfragedauer serielle Abfrage: " & dauer1 & " für " & ersteAnzahl & "
               Ergebniselemente.")
        Console.WriteLine("Abfragedauer kombinierte Abfrage: " & dauer2 & " für " & zweiteAnzahl & "
               Ergebniselemente.")
    End Sub
```

Der zweite, in Fettschrift markierte Block entspricht im Grunde genommen dem ersten – nur das hier Abfragen hintereinandergeschaltet, aber eben nicht ausgeführt werden. Die eigentliche Ausführung findet statt, wenn auf die zu entstehende Elementauflistung zugegriffen wird – im Beispiel also die `Count`-Eigenschaft der »Auflistung« abgerufen wird, die die Anzahl der Elemente nur dann zurückgeben kann, wenn es eine Anzahl an Elementen gibt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Dass sich die beiden Ausführungspläne nicht nennenswert unterscheiden, zeigt das folgende Ergebnis …

```
Starte Test
Abfragedauer serielle Abfrage: 3531 für 17 Ergebniselemente.
Abfragedauer kombinierte Abfrage: 3561 für 17 Ergebniselemente.
```

… das mit 30ms Unterschied bei 500.000 Elementen wirklich nicht nennenswert ist.

**WICHTIG** Die Ausführungspläne, die Sie durch die Abfragen erstellen, werden jedes Mal ausgeführt, wenn Sie auf eine der Ergebnislisten zugreifen. Wiederholen Sie die letzte fettgeschriebene Zeile im obigen Listing …

```
.
.
        Dim zweiteAnzahl = ergebnisListe3.Count
        zweiteAnzahl = ergebnisListe3.Count
        hsp.Stop()
        Dim dauer2 = hsp.DurationInMilliSeconds
.
.
```

… ergibt sich das folgende Ergebnis:

```
Starte Test
Abfragedauer serielle Abfrage: 3675 für 12 Ergebniselemente.
Abfragedauer kombinierte Abfrage: 7104 für 12 Ergebniselemente.
```

## Faustregeln für das Erstellen von LINQ-Abfragen

Die Faustregeln für das Erstellen von Abfragen lauten:

- LINQ-Abfragen bestehen nur aus Ausführungsplänen – die eigentlichen Abfragen werden durch ihre Definition nicht ausgelöst!
- Wenn das Ergebnis einer Abfrage als Ergebnisliste Gegenstand einer weiteren Abfrage wird, kommt der erste Abfrageplan auch nicht zur Auslösung; beide Abfragepläne werden miteinander kombiniert.
- Erst der Zugriff auf ein Element der Ergebnisauflistung (über For/Each oder den Indexer) löst die Abfrage und damit das Erstellen der Ergebnisliste aus.
- Ein erneuter Zugriff auf eine elementeabhängige Eigenschaft oder auf die Elemente selbst löst – und das ist ganz wichtig – abermals das Erstellen der Ergebnisliste aus. Das gilt auch für Abfragen, die durch mehrere Abfragen kaskadiert wurden.

## Kaskadierte Abfragen

Das Beispiel, was ich Ihnen im letzten Listing vorgestellt habe, hat nämlich noch zwei weitere, auskommentierte Zeilen, die die Kaskadierungsfähigkeit (das Hintereinanderschalten) von Abfragen eindrucksvoll demonstrieren. Wenn Sie das Ergebnis dieser beiden zusätzlichen Zeilen des Listings …

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
adrListe.Add(New Kontakt(51, "Löffelmann", "Klaus", "Wiedenbrücker Straße 47", "59555", "Lippstadt"))
Console.WriteLine("Elemente der seriellen Abfrage: " & ergebnisListe2.Count)
```

… verstanden haben, dann haben Sie auch das Prinzip von LINQ verstanden!

```
Starte Test
Abfragedauer serielle Abfrage: 3515 für 7 Ergebniselemente.
Abfragedauer kombinierte Abfrage: 7101 für 7 Ergebniselemente.
Elemente der seriellen Abfrage: 8
```

Hier wird der Originalauflistung ein weiteres Element hinzugefügt, das exakt der Kriterienliste der kaskadierten Abfrage entspricht. Durch das Abfragen der `Count`-Eigenschaft von `ergebnisListe2` wird die *komplette* Abfragekaskade ausgelöst, denn von vorher 7 Elementen befinden sich anschließend 8 Elemente in der Ergebnismenge, was nicht der Fall wäre, würde LINQ nur die letzte Ergebnismenge, nämlich `ergebnisListe2` selbst auswerten, denn dieser haben wir das Element nicht hinzugefügt.

## Gezieltes Auslösen von Abfragen mit ToArray oder ToList

Nun haben wir gerade erfahren, dass Abfragen, bei jedem Zugriff auf die Ergebnisliste mit `For/Each` oder direkt über eine Elementeigenschaft bzw. den Index zu einer neuen Ausführung des Abfrageplans führen. Das kann, wie im Beispiel des vorherigen Abschnittes, durchaus wünschenswert sein – in vielen Fällen können sich daraus aber echte Performance-Probleme entwickeln.

Aus diesem Grund implementierten die Entwickler von LINQ spezielle Methoden, die eine auf `IEnumerable` basierende Ergebnisliste wieder in eine »echte« Auflistung bzw. in ein »echtes« Array umwandeln können.

Am häufigsten wird dabei sicherlich die Methode `ToList` zur Anwendung kommen, die das Ergebnis einer wie auch immer gearteten LINQ-Abfrage in eine generische `List(Of t)` umwandelt – und dabei ist es völlig gleich, ob es sich um eine *LINQ to Objects*, *LINQ to SQL*, *LINQ to XML* oder eine *LINQ to sonst zu was*-Abfrage handelt. Das Ausführen von `ToList` auf eine solche Ergebnisliste hat zwei Dinge zur Folge:

- Die LINQ-Abfrage wird ausgeführt.
- Eine `List(Of {Ausgangstyp|Select-Anonymer-Typ})` wird zurückgeliefert.

Die Elemente, die anschließend zurückkommen, sind völlig ungebunden – Sie können sie anschließend sooft indizieren, mit `For/Each` durchiterieren, ihre `Count`-Eigenschaft abfragen wie Sie wollen – mit der ursprünglichen LINQ-Abfrage haben sie nichts mehr zu tun.

`ToList` funktioniert dabei denkbar einfach:

```
Dim reineGenerischeListe = ergebnisListe.ToList
```

Und `ToList` ist auch nicht die einzige Methode, mit der Sie eine Ergebnisliste von ihrer Ursprungsabfrage trennen können. Das können Sie – in Form von Ergebnislisten unterschiedlichen Typs – auch mit den folgenden Methoden machen. `t` ist dabei immer der Typ eines Elements der Ausgangsauflistung oder ein anonymer Typ, der in der Abfrage beispielsweise durch die `Select`-Klausel entstanden ist.

- `ToList`: Überführt die Abfrageergebnisliste in eine generische `List(Of t)`.
- `ToArray`: Überführt die Abfrageergebnisliste in ein Array vom Typ `t`.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- `ToDictionary`: Überführt die Abfrageliste in eine generische Wörterbuchauflistung vom Typ `Dictionary(Of t.key, t)`. `t.key` muss dabei durch die Angabe eines Lambda-Ausdrucks festgelegt werden, also etwa

```
Dim reineGenerischeListe = ergebnisListe.ToDictionary(Function(einKontakt) einKontakt.ID)
```

  um beim Beispiel zu bleiben, und die ID zum Nachschlageschlüssel zu machen.

- `ToLookup`: Überführt die Abfrageliste in eine generische Lookup-Auflistung vom Typ `Lookup(Of t.key, t)`. `t.key` muss dabei durch die Angabe eines Lambda-Ausdrucks festgelegt werden, also etwa

```
Dim reineGenerischeListe = ergebnisListe.ToLookup(Function(einKontakt) einKontakt.ID)
```

**HINWEIS** Die `ToLookup`-Methode gibt ein generisches `Lookup`-Element zurück, ein 1:*n*-Wörterbuch, das Auflistungen von Werten Schlüssel zuordnet. Ein Lookup unterscheidet sich von Dictionary, das eine 1:*1*-Zuordnung von Schlüsseln zu einzelnen Werten ausführt.

# Verbinden mehrerer Auflistungen zu einer neuen

LINQ kennt mehrere Möglichkeiten, Auflistungen miteinander zu verbinden. Natürlich ist es nicht unbedingt sinnig, das willkürlich zu tun: Die Auflistungen müssen schon in irgendeiner Form miteinander in logischer Relation stehen – dann aber kann man sich durch geschicktes Formulieren eine Menge an Zeit sparen. Wie eine solche Relation ausschauen kann, in der zwei Auflistungen zueinander stehen, demonstriert Ihnen der Abschnitt »Wie ›geht‹ LINQ prinzipiell« des vorherigen Kapitels – und diese Relation soll auch noch mal Gegenstand der Beispiele dieses Abschnittes sein:

**Relation zwischen Artikel und Käufer-Tabellen**

| ID | Nachname | Vorname | Straße | PLZ | Ort |
|---|---|---|---|---|---|
| 1 | Heckhuis | Jürgen | Wiedenbrücker Str. 47 | 59555 | Lippstadt |
| 2 | Wördehoff | Angela | Erwitter Str. 33 | 01234 | Dresden |
| 3 | Dröge | Ruprecht | Douglas-Adams-Str. 42 | 55544 | Ratingen |
| 4 | Dröge | Ada | Douglas-Adams-Str. 42 | 55544 | Ratingen |
| 5 | Halek | Gaby „Doc“ | Krankenhausplatz 1 | 59494 | Soest |
| 6 | | | | | |

| IdGekauftVon | ArtikelNum-mer | ArtikelName | Kategorie | Einzelpreis | Anzahl |
|---|---|---|---|---|---|
| 1 | 9-445 | Die Tore der Welt | Bücher, Belletristik | 19,90 | 2 |
| 3 | 3-537 | Visual Basic 2005 - Das Entwicklerbuch | Bücher, EDV | 59,00 | 2 |
| 3 | 3-123 | SQL Server 2000 - So gelingt der Einstieg | Bücher, EDV | 19,90 | 1 |
| 5 | 5-312 | SQL Server 2008 - Das Profi-Buch | Bücher, EDV | 39,90 | 2 |

**Abbildung 8.1** Das LINQ-Beispielprogramm legt zwei Tabellen an, die nur logisch über die Spalten-ID miteinander verknüpft sind

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Implizite Verknüpfung von Auflistungen

Die einfachste Möglichkeit, zwei Auflistungen zu gruppieren, zeigt die folgende Abbildung.

```
Sub Auflistungsvereinigung()
    Dim ergebnisliste = From adrElement In adrListe, artElement In artListe

    For Each element In ergebnisliste
        element.
                adrElement
                artElement
                Allgemein   Alle
                Public ReadOnly Property adrElement() As LinqToObjectSamples.Kontakt
        With element
            Console.WriteLine(.adrElement.ID & ": " & .adrElement.Nachname & _
                              ", " & .adrElement.Vorname & ": " & _
                              .artElement.IDGekauftVon & ": " & .artElement.ArtikelName)
        End With
    Next
End Sub
```

**Abbildung 8.2** Schon mit der From-Klausel können Sie zwei Auflistungen per LINQ miteinander kombinieren

`From` kombiniert zwei Auflistungen miteinander. Im Ergebnis erhalten Sie eine neue Auflistung, bei der jedes Element der ersten mit jedem Element der zweiten Auflistung kombiniert wurde. Die Ausgabe dieser Auflistung, die über zwei Eigenschaften verfügt, die jeweils Zugriff auf die Originalelemente gestatten, zaubert dann folgendes Ergebnis auf den Bildschirm (aus Platzgründen gekürzt):

```
6: Clarke, Christian: 1: Visual Basic 2008 - Das Entwicklerbuch
6: Clarke, Christian: 1: Das Herz der Hölle
6: Clarke, Christian: 2: Visual Basic 2005 - Das Entwicklerbuch
6: Clarke, Christian: 2: Das Vermächstnis der Tempelritter
6: Clarke, Christian: 3: Das Leben des Brian
6: Clarke, Christian: 3: Die Tore der Welt
.
.
.
7: Ademmer, Lothar: 1: Visual Basic 2008 - Das Entwicklerbuch
7: Ademmer, Lothar: 1: Das Herz der Hölle
7: Ademmer, Lothar: 2: Visual Basic 2005 - Das Entwicklerbuch
7: Ademmer, Lothar: 2: Das Vermächstnis der Tempelritter
7: Ademmer, Lothar: 3: Das Leben des Brian
7: Ademmer, Lothar: 3: Die Tore der Welt
```

An der Ergebnisliste können Sie erkennen, wie redundant und nicht informativ diese Liste ist, denn jeder Artikel der Artikelliste wird einfach und stumpf mit jedem Kontakt kombiniert.

Wichtiger wäre es, Zuordnungen ausdrücklich bestimmen zu können, um zu sagen, dass ein Artikel mit einer bestimmten ID auch nur dem logisch dazugehörigen Kontakt zugeordnet werden soll. Und das funktioniert folgendermaßen:

```
    Sub Auflistungsvereinigung()
        Dim ergebnisliste = From adrElement In adrListe, artElement In artListe _
                            Where adrElement.ID = artElement.IDGekauftVon
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        For Each element In ergebnisliste
            With element
                Console.WriteLine(.adrElement.ID & ": " & .adrElement.Nachname & _
                                  ", " & .adrElement.Vorname & ": " & _
                                  .artElement.IDGekauftVon & ": " & .artElement.ArtikelName)
            End With
        Next
    End Sub
```

In dieser Version werden durch die `Where`-Klausel nur die Elemente in die Auflistung übernommen, die die gleiche `ID` haben wie das korrelierende Element der anderen Auflistung (`IDGekauftVon`). Das Ergebnis, was dann zu sehen ist, macht natürlich viel mehr Sinn, da es eine Aussagekraft hat (nämlich: welcher Kunde hat welche Artikel gekauft):

```
7: Sonntag, Uwe: 7: Visual Basic 2008 - Das Entwicklerbuch
7: Sonntag, Uwe: 7: The Da Vinci Code
7: Sonntag, Uwe: 7: O.C., California - Die komplette zweite Staffel (7 DVDs)
7: Sonntag, Uwe: 7: Desperate Housewives - Staffel 2, Erster Teil
7: Sonntag, Uwe: 7: Die Rache der Zwerge
8: Vüllers, Momo: 8: Programmieren mit dem .NET Compact Framework
9: Tinoco, Daja: 9: Das Herz der Hölle
9: Tinoco, Daja: 9: Mitten ins Herz
9: Tinoco, Daja: 9: The Da Vinci Code
9: Tinoco, Daja: 9: Der Schwarm
9: Tinoco, Daja: 9: Desperate Housewives - Staffel 2, Erster Teil
9: Tinoco, Daja: 9: Harry Potter und die Heiligtümer des Todes
9: Tinoco, Daja: 9: Der Teufel trägt Prada
9: Tinoco, Daja: 9: O.C., California - Die komplette zweite Staffel (7 DVDs)
10: Lehnert, Michaela: 10: Abgefahren - Mit Vollgas in die Liebe
10: Lehnert, Michaela: 10: Das Herz der Hölle
```

**HINWEIS** Die Verknüpfung zweier oder mehrerer Auflistungen mit `In` als Bestandteil der `From`-Klausel einer Abfrage nennt man implizite Verknüpfung von Auflistungen, da dem Compiler nicht ausdrücklich mitgeteilt wird, welche Auflistung auf Grund welchen Elementes mit einer anderen verknüpft wird. Eine ausdrückliche oder explizite Verknüpfung stellen Sie mit `Join` her, das im nächsten Abschnitt beschrieben wird. Explizit oder implizit hier im Beispiel ist aber letzten Endes einerlei – das Ergebnis ist in beiden Fällen dasselbe.

## Explizite Auflistungsverknüpfung mit Join

Im Gegensatz zu impliziten Auflistungsverknüpfungen, die Sie, wie im letzten Abschnitt beschrieben, mit `In` als Teil der `From`-Klausel bestimmen, erlaubt Ihnen die `Join`-Klausel sogenannte explizite Auflistungsverknüpfungen festzulegen. Die generelle Syntax der `Join`-Klausel lautet:

```
Dim ergebnisliste = From bereichsVariable In ersterAuflistung _
                    Join verknüpfungsElement In zweiterAuflistung _
                    On bereichsVariable.JoinKey Equals verknüpfungsElement.ZweiterJoinKey
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

`Join` verknüpft die erste mit der zweiten Tabelle über einen bestimmten Schlüssel (`JoinKey`, `ZweiterJoinKey`), der beide Tabellen zueinander in Relation stellt. In unserem Beispiel wird für jede Bestellung in der Artikeltabelle ein Schlüssel (ein *Key*, eine *ID*) definiert, der der Nummer der *ID* in der Kontakttabelle entspricht.

Im Vergleich zur impliziten Verknüpfung von Tabellen ändert sich im Ergebnis nichts; die Umsetzung des vorherigen Beispiels mit `Join` gestaltet sich folgendermaßen:

```
Sub JoinDemo()
    Dim ergebnisliste = From adrElement In adrListe _
                        Join artElement In artListe _
                        On adrElement.ID Equals artElement.IDGekauftVon

    For Each element In ergebnisliste
        With element
            Console.WriteLine(.adrElement.ID & ": " & .adrElement.Nachname & _
                              ", " & .adrElement.Vorname & ": " & _
                              .artElement.IDGekauftVon & ": " & .artElement.ArtikelName)
        End With
    Next
End Sub
```

Es gibt die Möglichkeit, mit der Klausel `GroupJoin` Verknüpfungen mehrerer Tabellen auf bestimmte Weise in Gruppen zusammenzufassen. Wie das funktioniert erfahren Sie im Abschnitt »Group Join« ab Seite 171.

# Gruppieren von Auflistungen

Die Klausel `GroupBy` erlaubt es, Dubletten von Elementen einer oder mehrerer Auflistungen in Gruppen zusammenzufassen. Sie möchten also beispielsweise eine Auflistung von Kontakten nach Nachnamen gruppieren, und dann in einer geschachtelten Schleife durch die Namen und innen durch alle den Namen zugeordneten Kontakte iterieren. Mit der `GroupBy`-Klausel können Sie genau das erreichen, wie das folgende Beispiel zeigt:

```
Sub GroupByDemo()
    Dim ergebnisliste = From adrElement In adrListe _
                        Group By adrElement.Nachname Into Kontaktliste = Group _
                        Order By Nachname

    For Each element In ergebnisliste
        With element
            Console.WriteLine(element.Nachname)
            For Each Kontakt In element.Kontaktliste
                With Kontakt
                    Console.WriteLine(.ID & ": " & .Nachname & ", " & .Vorname)
                End With
            Next
            Console.WriteLine()
        End With
    Next
End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wörtlich ausformuliert hieße die `Group By`-Klausel dieses Beispiels: »Erstelle eine Liste aller Nachnamen (`Group By adrElement.Nachname`) und mache diese unter der Nachname-Eigenschaft in der Liste zugänglich.[4] Fasse alle Elemente der Ausgangsliste in jeweiligen Auflistungen zusammen, die dem Nachnamen zugehörig sind (`Into … = Group`), und mache diese Auflistung über die Eigenschaft `Kontaktliste` verfügbar.«

**HINWEIS** Falls Sie keine Aliasbenennung der Gruppe vornehmen (der Ausdruck würde dann einfach

```
Dim ergebnisliste = From adrElement In adrListe _
                    Group By adrElement.Nachname Into Group _
                    Order By Nachname
```

heißen, würde die Eigenschaft, mit der Sie die zugeordneten Elemente erreichen können, einfach `Group` heißen.

Wenn Sie das Beispiel starten, sehen Sie das gewünschte Ergebnis, etwa wie im folgenden Bildschirmauszug (aus Platzgründen gekürzt):

```
Weichelt
19: Weichelt, Anne
39: Weichelt, Gabriele
47: Weichelt, Uta
69: Weichelt, Franz
97: Weichelt, Hans

Weigel
37: Weigel, Lothar
40: Weigel, Rainer
43: Weigel, Lothar
58: Weigel, Hans
60: Weigel, Anja

Westermann
11: Westermann, Margarete
21: Westermann, Alfred
28: Westermann, Alfred
41: Westermann, Alfred
77: Westermann, Guido
98: Westermann, José
100: Westermann, Michaela

Wördehoff
14: Wördehoff, Bernhard
```

[4] Wenn nichts anderes gesagt wird, heißt das Feld bzw. die Eigenschaft, nach der Sie gruppieren, in der späteren Auflistung so wie das Ausgangsfeld. Wenn Sie das nicht wünschen, können Sie das durch das Davorsetzen eines neuen Namens etwa mit `Group By Lastname = adrElement.Nachname Into Kontaktliste = Group` – an Ihre Wünsche anpassen. Anders als im Beispiel wäre hier `Lastname` die Eigenschaft, mit der Sie später die Nachnamen abfragen könnten.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## Gruppieren von Listen aus mehreren Auflistungen

`Group By` eignet sich auch sehr gut dazu, mit `Join` kombinierte Listen zu gruppieren und auszuwerten. Stellen Sie sich vor, Sie möchten eine Liste mit Kontakten erstellen, mit der Sie über jeden Kontakt wieder auf eine Liste mit Artikeln zugreifen zu können, um auf diese Weise herauszufinden, welche Kunden welche Artikel gekauft hätten. Die entsprechende Abfrage und die anschließende Iteration durch die Ergebniselemente sähen dann folgendermaßen aus:

```
Sub GroupByJoinedCombined()
    Dim ergebnisliste = From adrElement In adrListe _
                        Join artElement In artListe _
                        On adrElement.ID Equals artElement.IDGekauftVon _
                        Group artElement.IDGekauftVon, artElement.ArtikelNummer, _
                            artElement.ArtikelName _
                        By artElement.IDGekauftVon, adrElement.Nachname, adrElement.Vorname _
                        Into Artikelliste = Group, AnzahlArtikel = Count() Order By Nachname

    For Each kontaktElement In ergebnisliste
        With kontaktElement

            Console.WriteLine(.IDGekauftVon & ": " & .Nachname & .Vorname)
            For Each Artikel In .Artikelliste
                With Artikel
                    Console.WriteLine("   " & .ArtikelNummer & ": " & .ArtikelName)
                End With
            Next
            Console.WriteLine()
        End With
    Next
End Sub
```

Hier sehen Sie eine Zusammenfassung dessen, was wir in den letzten Abschnitten kennen gelernt haben. Die Abfrage beginnt mit einem `Join` und vereint Artikel und Kundenliste zu einer flachen Auflistung, die sowohl die Namen als auch die Artikel für jeden Namen beinhaltet. Und dann wird gruppiert: Anders als beim ersten Gruppierungsbeispiel, in dem alle Elemente in der untergeordneten Liste einbezogen werden, geben wir hier zwischen `Group` und `By` die Felder an, die in der inneren Auflistung als Eigenschaften erscheinen sollen, wir ändern also, ähnlich dem `Select`-Befehl, das Schema der inneren Auflistung. Die Elemente, die wir anschließend hinter dem `By` angeben, sind die, nach denen gruppiert wird, und die damit auch in der äußeren Auflistung verfügbar sind. Das Ergebnis entspricht dann unseren Erwartungen:

```
21: Wördehoff, Theo
   4-444: The Da Vinci Code
   2-424: 24 - Season 6 [UK Import - Damn It!]
   2-134: Abgefahren - Mit Vollgas in die Liebe
   3-534: Mitten ins Herz
   3-333: Der Schwarm
   3-537: Visual Basic 2005 - Das Entwicklerbuch
   4-444: Harry Potter und die Heiligtümer des Todes
   5-554: O.C., California - Die komplette zweite Staffel (7 DVDs)
   2-134: Abgefahren - Mit Vollgas in die Liebe
   7-321: Das Herz der Hölle
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
75: Wördehoff, Katrin
   2-134: Abgefahren - Mit Vollgas in die Liebe
   2-134: Abgefahren - Mit Vollgas in die Liebe
   2-134: Abgefahren - Mit Vollgas in die Liebe
   3-123: Das Vermächstnis der Tempelritter
   2-134: Abgefahren - Mit Vollgas in die Liebe
   1-234: Das Leben des Brian
   3-543: Microsoft Visual C# 2005 - Das Entwicklerbuch
   3-543: Microsoft Visual C# 2005 - Das Entwicklerbuch
   9-423: Desperate Housewives - Staffel 2, Erster Teil
   7-321: Das Herz der Hölle
   5-506: Visual Basic 2008 - Das Entwicklerbuch
   5-513: Microsoft SQL Server 2008 - Einführung in Konfiguration, Administration, Programmierung
   9-646: Was diese Frau so alles treibt
   5-506: Visual Basic 2008 - Das Entwicklerbuch
   2-321: Die Rache der Zwerge
   9-445: Die Tore der Welt
   4-444: Harry Potter und die Heiligtümer des Todes

77: Wördehoff, Theo
   7-321: Das Herz der Hölle
   1-234: Das Leben des Brian
   9-009: Die Wächter
   3-123: Das Vermächtnis der Tempelritter
   5-518: Visual Basic 2008 - Neue Technologien - Crashkurs
```

## Group Join

Exakt das gleiche Ergebnis bekommen Sie, allerdings mit sehr viel weniger Aufwand, wenn Sie sich der `Group Join`-Klausel bedienen, die `Join` und `Group By` miteinander kombiniert – das folgende Beispiel zeigt, wie's geht:

```
    Sub GroupJoin()
       Dim ergebnisliste = From adrElement In adrListe _
                           Group Join artElement In artListe _
                           On adrElement.ID Equals artElement.IDGekauftVon Into Artikelliste = Group

       For Each kontaktElement In ergebnisliste
           With kontaktElement

               Console.WriteLine(.adrElement.ID & ": " & _
                                 .adrElement.Nachname & ", " _
                                 & .adrElement.Vorname)
               For Each Artikel In .Artikelliste
                   With Artikel
                       Console.WriteLine("   " & .ArtikelNummer & ": " & .ArtikelName)
                   End With
               Next
               Console.WriteLine()
           End With
       Next
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Aggregatfunktionen

Aggregatfunktionen sind Funktionen, die ein bestimmtes Feld einer Auflistung aggregieren – zum Beispiel aufsummieren, den Durchschnitt berechnen, das Maximum oder das Minimum ermitteln oder einfach nur zählen, wie viele Elemente in einer Auflistung vorhanden sind. Anders als bei »normalen« Abfragen werden beim Aggregieren also die Elemente der Auflistung einer Aggregatfunktion alle nacheinander übergeben, und diese Funktion arbeitet bzw. rechnet dann mit jedem einzelnen dieser Elemente, betrachtet sie aber als Menge.

**HINWEIS** LINQ-Abfragen fangen grundsätzlich mit der FROM-Klausel an – reines Aggregieren von Auflistungen sind allerdings die einzige Ausnahme. Wenn Sie eine reine Aggregierung auf eine Auflistung ausführen wollen, leiten Sie die LINQ-Abfrage nicht mit `From` sondern mit der Klausel `Aggregate` ein – die folgenden Beispiele werden es gleich demonstrieren.

Lassen Sie uns ganz einfach beginnen. Die erste Demo …

```
    Sub AggregateDemo()

        Dim scheinbarerUmsatz = Aggregate artElement In artListe _
                            Into Summe = Sum(artElement.Einzelpreis)
        Console.WriteLine("Scheinbar beträgt der Umsatz {0:#,##0.00} Euro", scheinbarerUmsatz)
```

… aggregiert unsere Artikelliste in eine Gesamtsumme und ermittelt so scheinbar, wie viel Gesamtumsatz mit den Artikeln erzielt wurde, etwa wie hier zu sehen:

```
Scheinbar beträgt der Umsatz 47.117,60 Euro
```

Doch diese Aussage stimmt wirklich nur scheinbar. Denn bei der Abfrage wurden nur die Artikelpreise in einer `Decimal`-Variable (`scheinbarerUmsatz`) summiert – die Anzahl eines Artikel blieb unberücksichtigt.

Dieses Manko behebt das direkt anschließende Beispiel …

```
        Dim gesamtUmsatz = Aggregate artElement In artListe _
                           Let Artikelumsatz = artElement.Einzelpreis * artElement.Anzahl _
                           Into Summe = Sum(Artikelumsatz)
        Console.WriteLine("Der Umsatz beträgt {0:#,##0.00} Euro", gesamtUmsatz)
```

… das mit einem sehr, sehr alten Bekannten – nämlich der `Let`-Klausel – eine neue Variable einführt, die das *Produkt* von Anzahl und Preis aufnimmt und anschließend als Ergebnis aufsummiert:

```
Der Umsatz beträgt 93.306,00 Euro
```

## Zurückliefern mehrerer verschiedener Aggregierungen

Wenn eine Aggregat-Abfrage nur eine einzige Aggregat-Funktion verwendet, dann ist das Abfrageergebnis eine einzige Variable, die dem Rückgabetyp der Aggregatfunktion entspricht. Liefert beispielsweise die `Sum`-Aggregatfunktion ein Ergebnis vom Typ `Decimal`, dann ist das Abfrageergebnis ebenfalls vom Typ `Decimal`.

Sie können aber auch mehrere Aggregate innerhalb einer einzigen Abfrage verwenden, wie das dritte Beispiel zeigt:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Dim mehrereInfos = Aggregate artElement In artListe _
                   Let Artikelumsatz = artElement.Einzelpreis * artElement.Anzahl _
                   Into Summe = Sum(Artikelumsatz), _
                        Minimalpreis = Min(artElement.Einzelpreis), _
                        MaximalPreis = Max(artElement.Einzelpreis)
Console.WriteLine("Der Umsatz beträgt {0:#,##0.00} Euro" & vbNewLine & _
                  "Minimal- und Maximalpreis jeweils " & _
                  "{1:#,##0.00} und {2:#,##0.00} Euro", _
                  mehrereInfos.Summe, mehrereInfos.Minimalpreis, _
                  mehrereInfos.MaximalPreis)
```

In diesem Beispiel werden »in einem Rutsch« Artikelpostensumme, Maximalpreis und Minimalpreis ermittelt; die Rückgabevariable ist in dem Fall vom Typ anonyme Klasse mit den drei Eigenschaften, die über die entsprechenden Variablen innerhalb der Abfrage (Summe, Minimalpreis und Maximalpreis) definiert wurden. Hätten Sie diese Variablendefinitionen weggelassen und stattdessen nur die Aggregatfunktionen angegeben, hätte der Compiler für die Eigenschaften des anonymen Rückgabetypen Namen gewählt, die den Namen der Aggregatfunktionen entsprächen.

**WICHTIG** Anders als bei Standardabfragen werden reine Aggregatabfragen, wie Sie sie hier in den letzten drei Beispielen kennen gelernt haben, *nicht* verzögert sondern sofort ausgeführt.

## Kombinieren von gruppierten Abfragen und Aggregatfunktionen

Das Gruppieren von Abfragen eignet sich naturgemäß gut für den Einsatz von Aggregatfunktionen, denn beim Gruppieren von Datensätzen werden verschiedene Elemente zusammengefasst, auf die sich dann die unterschiedlichsten Aggregatfunktionen anwenden lassen.

Ein Beispiel soll auch das verdeutlichen: Angenommen, Sie möchten herausfinden, welcher Kunde durch seine Käufe wie viel Umsatz produziert hat. In diesem Fall würden Sie die Abfrage nach Artikeln gruppieren, und zwar nach dem Feld `IDGekauftVon` – denn dieses Feld repräsentiert »*wer* hat gekauft«. Damit in der Ergebnisliste nicht nur eine nichtssagende Nummer sondern ein greifbarer Vor- sowie ein Nachname zu finden sind, verknüpfen Sie Artikeltabelle und Kundentabelle mit einer `Join`-Abfrage oder – noch besser – fassen Gruppierung und Zusammenfassung der Tabellen mit einem `Group Join` zusammen. In die `Group Join`-Abfrage bauen Sie gleichzeitig noch die schon bekannte `Order By`-Klausel ein, mit der Sie die Ergebnisliste nach dem errechneten Umsatz sortieren – durch die Angabe von `Descending` als Schlüsselwort sogar absteigend, um eine Top-Ranking-Umsatzliste zu bekommen. Die eigentliche Aggregierung der Umsätze erfolgt dann mit der hinter `Into` stehenden Klausel, in der ebenfalls bestimmt wird, mit welcher Eigenschaft später die innere Liste (`JoinedList`) im Bedarfsfall durchiteriert werden kann (und dabei dann einzelne `artListe`-Elemente vom Typ `Artikel` zurückliefert). Die Aggregatfunktion `Sum`, die dafür notwendig ist, summiert im Beispiel das Produkt von Menge und Artikeleinzelpreis auf, was – für jeden Kontakt gruppiert – exakt dem Umsatz jedes Kunden entspricht.

**TIPP** Wie ebenfalls im folgenden Beispiel zu sehen, verwendet die Abfrage die `Take`-Klausel, um die Ergebnisliste auf 10 Elemente zu beschränken. Es gibt weitere Klauseln, die Sie für solche Einschränkungen verwenden können, wie beispielsweise `Take While` oder `Skip` – deren Anwendungsweise ist ähnlich und über dieses Beispiel leicht ableitbar. Die Online-Hilfe verrät Ihnen zu diesen Klauseln mehr.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Sub JoinAggregateDemo()
    Dim ergebnisListe = From adrElement In adrListe _
                        Group Join artElement In artListe _
                        On adrElement.ID Equals artElement.IDGekauftVon _
                        Into _
                        KundenUmsatz = Sum(artElement.Einzelpreis * artElement.Anzahl) _
                        Order By KundenUmsatz Descending _
                        Take 10

    For Each item In ergebnisListe
        With item.adrElement
            Console.WriteLine(.ID & ": " & .Nachname & ", " _
                              & .Vorname & " --- Umsatz:" _
                              & item.KundenUmsatz.ToString("#,##0.00") & _
                              " Euro")
        End With
    Next
End Sub
```

Wenn Sie dieses Beispiel ausführen, produziert es in etwa die folgenden Zeilen:

```
3: Braun, Franz --- Umsatz:2.358,00 Euro
34: Plenge, Franz --- Umsatz:2.287,80 Euro
91: Jungemann, Katrin --- Umsatz:2.183,05 Euro
94: Tinoco, Anne --- Umsatz:2.128,00 Euro
100: Hollmann, Gareth --- Umsatz:2.108,15 Euro
18: Westermann, Anja --- Umsatz:2.098,05 Euro
66: Hörstmann, Katrin --- Umsatz:1.923,00 Euro
54: Albrecht, Uta --- Umsatz:1.843,40 Euro
97: Hollmann, Daja --- Umsatz:1.798,50 Euro
27: Westermann, Alfred --- Umsatz:1.788,15 Euro
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 9

# LINQ to SQL

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Einleitung

Als ich meine ersten Gehversuche mit LINQ machte, war ich begeistert. Nur nach und nach stellte ich fest, welche enormen Möglichkeiten sich mir mit LINQ auf einmal boten – und ich hatte zu diesem Zeitpunkt doch noch gar nichts gesehen! Während meiner anfänglichen Gehversuche mit einer der frühen Betas von Visual Studio 2008 wusste ich nämlich zum damaligen Zeitpunkt, irgendwann im April 2007, noch gar nicht, wie sehr ich erst an der Spitze des Eisberges 'rumdokterte.

Doch nach und nach begriff ich, wohin »uns« diese ganze LINQ-Geschichte führen sollte, und irgendwann war es dann auch so weit, dass es die ersten Visual Studio Betas gab, die uns die ersten Versionen von *LINQ to SQL* vorführten. Wir waren alle total begeistert.

Das Prinzip ist eigentlich ganz einfach. Sie »sagen«: »Ich programmiere im Folgenden genau so, wie ich es mit LINQ to Objects gelernt habe, die Quelle meiner Daten ist jedoch keine Auflistung, sondern eine SQL Server-Datenbank. Ansonsten bleibt alles genau so«.

Und so einfach soll das sein? Keine neuen Objekte, Klassen und Verfahren, die man lernen muss? Nichts Spezielles, was man zu beachten hat? Gibt es keine Falltüren, auf die man aufpassen sollte?

Doch, na klar, ein paar gibt es, denn Sie wollen Daten ja nicht nur aus dem SQL Server »abholen«, Sie wollen sie ja schließlich nach ihrer Verarbeitung auch wieder zurück in den Server bekommen, und dazu muss die vorhandene Infrastruktur natürlich ein wenig »aufgebohrt« werden, damit sie das gestattet. Und dann muss es ja immerhin auch noch möglich sein, mit ein paar SQL Server-Schmankerln zu arbeiten, beispielsweise mit gespeicherten Prozeduren! Und zu guter Letzt soll dieses Kapitel auch nicht nur diese, sondern ein paar Seiten mehr haben, und das hat es schließlich auch…

# Voraussetzungen für die Beispiele dieses Kapitels

Dieses Kapitel nennt sich LINQ to SQL.[1] Und mit SQL im Part dieses Namens ist nicht etwa irgendein SQL gemeint, sondern SQL in Form von Microsoft SQL Server. Mit der Visual Studio Version, mit der dieses Buch entstand, war es lediglich möglich, mit dem SQL Server der Versionen 2000 und 2005 von Microsoft zu arbeiten. Erst mit dem Erscheinen des SQL Server 2008 und dem Service Pack 1 für Visual Studio 2008 wird es möglich sein, auch LINQ to SQL mit dieser Serverversion zu betreiben. Für andere SQL-Server, wie Oracle oder MySQL gibt es zur Drucklegung noch keine entsprechenden Treiber.

Um die Beispiele dieses Buches nachvollziehen zu können, benötigen Sie also eine funktionsfähige SQL Server-Instanz – am besten in der Version 2005. SQL Server 2005 in der Express Edition ist dazu ausreichend, und Sie können sie, falls Sie sie nicht im Rahmen von Visual Studio 2008 ohnehin haben mitinstallieren lassen, auch ohne Probleme von den Microsoft-Seiten herunterladen.

[1] Übrigens: Tatsächlich sollte es »Link tuh Ess Kju Äll« ausgesprochen werden, aber genau wie bei Microsofts SQL Server, den (fast) jeder »*ßiekwäl* Sörwä« ausspricht, nennen es die meisten »Link to *ßiekwäl*«.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Darüber hinaus brauchen Sie für die Beispiele dieses Kapitels auch die Beispieldatenbank *AdventureWorks* die Sie unter *http://codeplex.com/SqlServerSamples* herunterladen und installieren können.

Wählen Sie das MSI-Installerpaket *AdventureWorksLT.msi* zum Download und zur Installation aus.[2]

## Beta-Warnung

Dieses Buch entstand, wie bereits angedeutet, nicht mit der RTM-Version von Visual Studio 2008, sondern mit dem Release Candidate. Auch aus diesem Grund könnte es zu folgenden Abweichungen kommen:

Je nach eingesetzter Visual Studio Version werden ggf. durch den Klassendesigner unterschiedliche Klassennamen generiert. Beim Ausprobieren mit der RC-Version und der finalen Visual Basic 2008 Express Edition konnten wir folgende Unterschiede feststellen:

| Version | Tabelle | Klasse, die einen Kunden abbildet | Table(of Customer) im DataContext |
|---|---|---|---|
| Visual Studio 2008 RC1 | Customer | Customer | Customer***s*** |
| Visual Studio Express 2008 | Customer | Customer | Customer |

**Tabelle 9.1:** Unterschiedliche Namensgebung der verschiedenen Visual Studio-Versionen

Desweiteren verwendet Visual Basic 2008 Express einen anderen Datenbankauswahldialog, da Visual Basic 2008 in der Express Edition nur den Modus des Anhängens von Datenbankdateien in SQL Express unterstützt.

# Wie es bisher war – Visual Studio 2005 vs. Visual Studio 2008

Auch in Visual Studio 2005 war es natürlich schon möglich, Client/Server-Anwendungen auf Basis von SQL Server 2000/2005 zu erstellen. Man bediente sich dabei regelmäßig eines interaktiven DataSet-Designers, der es ermöglichte, so genannte typisierte DataSets zu erstellen. Diese übernahmen die Infrastruktur für den Austausch von Daten zwischen Client und SQL-Server.

[2] Eine recht ausführliche Anleitung zur Installation von SQL Express finden Sie unter *www.activedevelop.de* in der SQL Server-Section.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

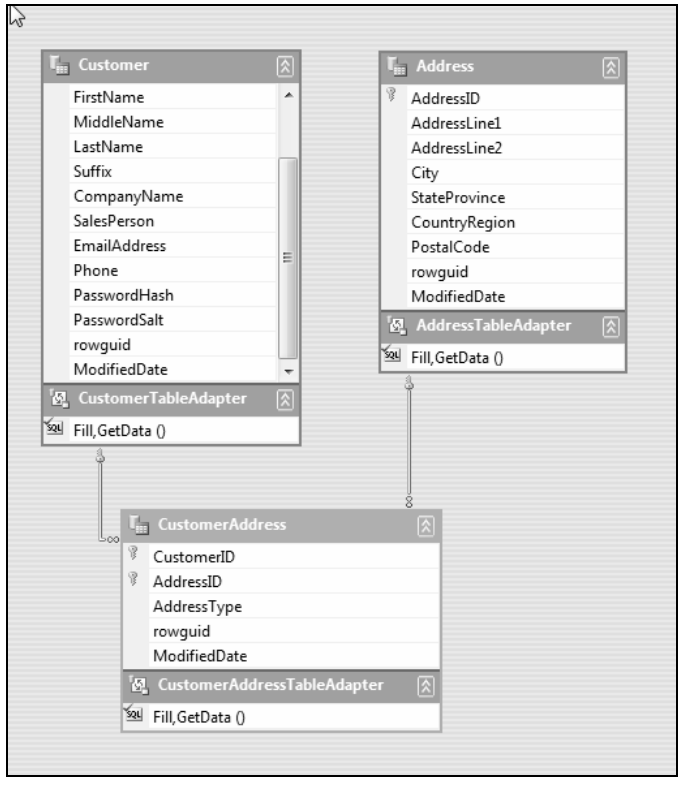



**Abbildung 9.1** Der DataSet-Designer von VS 2005

Pro Tabelle wurde eine `DataTable`-Klasse erstellt und zudem noch jeweils eine Klasse mit dem `TableAdapter`. Letzterer enthielt die Methoden, um den Austausch der Daten (Selektieren, Aktualisieren, Einfügen und Löschen) mit dem SQL Server zu gewährleisten.

**HINWEIS** Visual Studio 2008 bietet auch die Möglichkeit zu *LINQ to DataSets*. Dabei wird die Kommunikation zwischen SQL Server und der Client-Anwendung wie in bisherigen Versionen über DataSets abgewickelt, und auch die Aktualisierungslogik läuft ganz normal, wie gewohnt, über die DataSets ab. Die Abfrage, Selektierung und Sortierung der Daten innerhalb der DataSets kann dann aber über LINQ-Abfragen erfolgen. Durch die sehr große Ähnlichkeit zu *LINQ to Objects* wollen wir dieses Thema im Rahmen dieses Buches nicht näher vertiefen.

Auch für LINQ to SQL gibt es einen Designer, der dem seines Vorgängers vergleichsweise ähnlich sieht:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

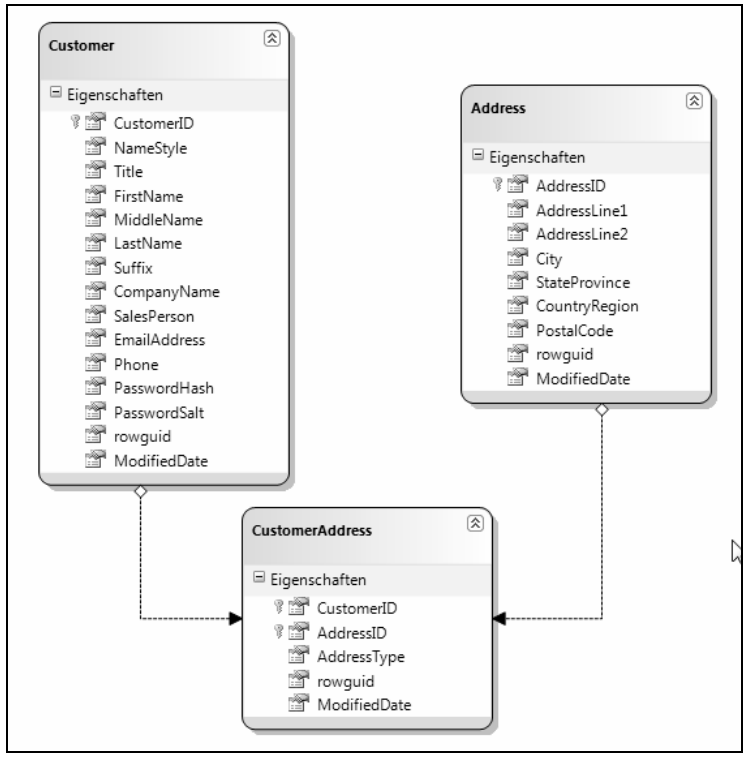



**Abbildung 9.2** Der O/R-Designer von VS 2008

Wie Sie sehen, fehlen die TableAdapter, und Verhaltensweisen, die man zuvor über Tabellenattribute einstellen konnte, werden nun über Eigenschaften gesteuert.

Und damit enden die Gemeinsamkeiten auch schon – lassen Sie sich deswegen nicht über die oberflächlichen Ähnlichkeiten hinwegtäuschen: LINQ to SQL verfolgt einen gänzlich anderen Ansatz als die typisierten DataSets (mit denen Sie, falls Sie es wünschten, natürlich auch in dieser Version von Visual Basic noch weiterarbeiten könnten).

LINQ to Objects haben Sie bereits kennen gelernt, und falls nicht, sollten Sie sich auf alle Fälle das vorherige Kapitel zu Gemüte führen. Die gleiche Syntax und ein sehr, sehr ähnlicher Ansatz wird nämlich auch von LINQ to SQL verwendet.

# Die ersten Schritte

Learning by Doing ist immer noch der beste Weg, neue Technologien kennen zu lernen, deswegen lassen Sie uns im Folgenden ein kleines LINQ to SQL-Beispiel erstellen.

**BEGLEITDATEIEN** Unter *.\Samples\Chapter09 - LinqToSql\LinqToSqlDemo* finden Sie die Beispieldateien für dieses Projekt, falls Sie die Beispiele nicht »händisch« nachvollziehen wollen. Denken Sie daran, dass Sie für das Nachvollziehen der Beispiele Zugriff auf eine Instanz von SQL Server 2005 haben müssen und dort die *AdventureWorks*-Beispiele eingerichtet sind.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Erstellen Sie ein neues Visual Basic-Projekt (als Konsolenanwendung).
- Fügen Sie mithilfe des Projektmappen-Explorers ein neues Element in die Projektmappe ein – den entsprechenden Dialog erreichen Sie über das Kontext-Menü des Projektnamens –, und wählen Sie für das neue Element, wie in der Abbildung zu sehen, die Vorlage *LINQ to SQL-Klasse* aus.
- Nennen Sie sie `AdventureWorks`.

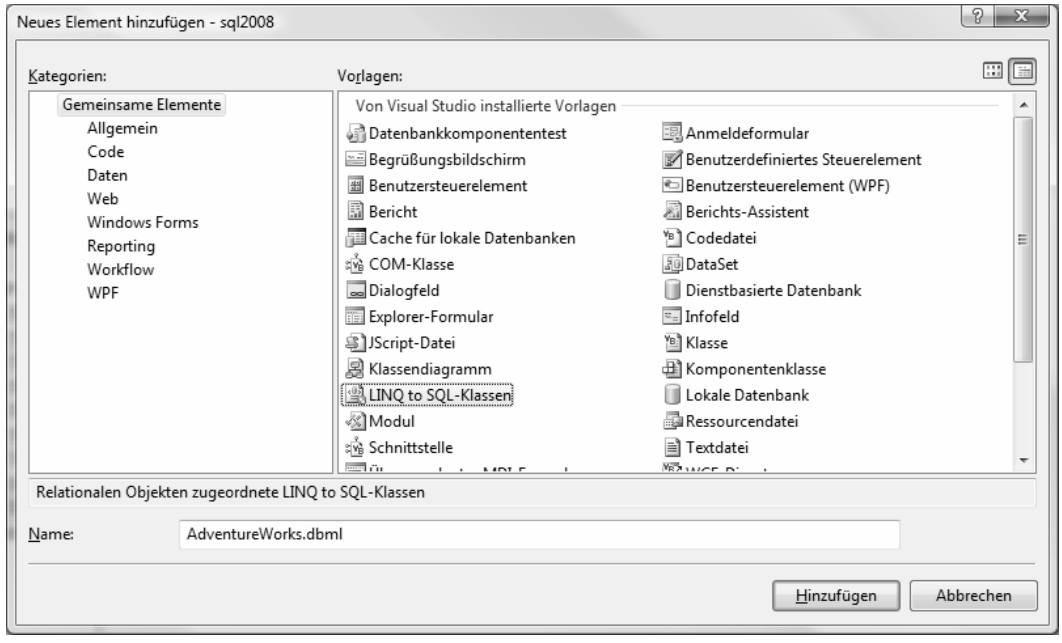



**Abbildung 9.3** Hinzufügen einer LINQ to SQL-Klasse

- Schließen Sie den Dialog mit Mausklick auf *Hinzufügen* ab.
- Im nächsten Schritt sehen Sie den so genannten Object Relational Designer (O/R Designer), der es Ihnen gestattet, neue Business-Objektklassen auf Basis von Datenbankobjekten zu entwerfen. Öffnen Sie den Server-Explorer (falls Sie ihn nicht sehen, lassen Sie ihn über das *Ansicht*-Menü anzeigen), öffnen Sie das Kontextmenü von *Datenverbindungen*, und fügen Sie eine neue Verbindung hinzu.

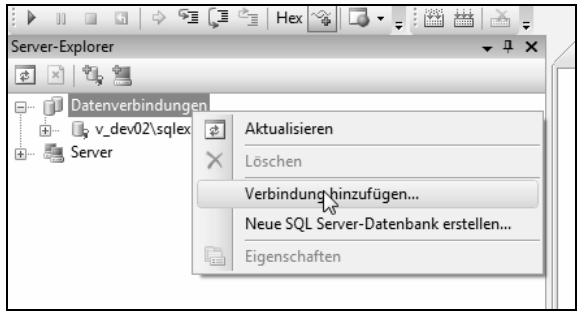



**Abbildung 9.4** Erstellen einer neuen Datenverbindung

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Wählen Sie in dem nun folgenden Dialog die SQL Server-Instanz sowie die Datenbank AdventureWorks aus, die wir im folgenden Beispiel verwenden wollen.

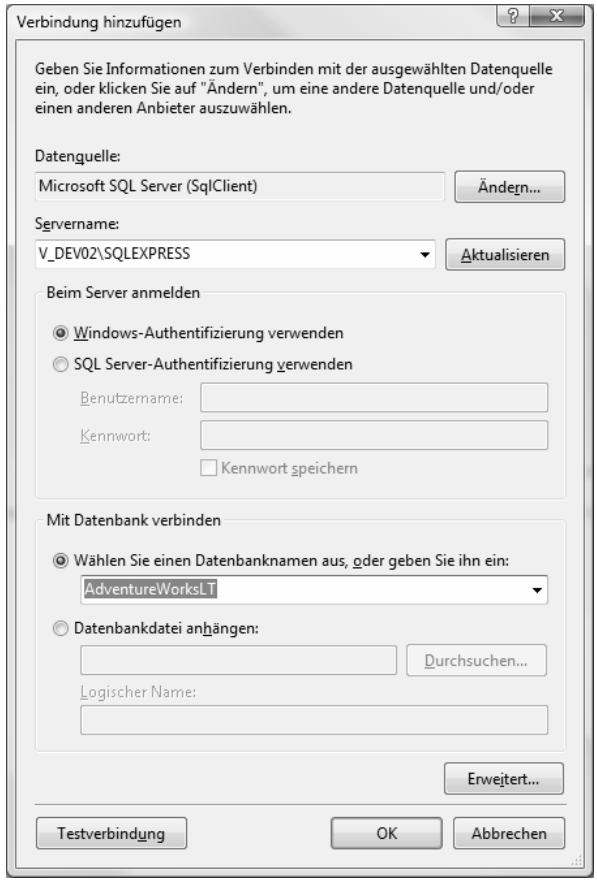



**Abbildung 9.5** Auswählen von SQL Server und Datenbank

- Sobald Sie die Verbindung getestet und den Dialog mit *OK* beendet haben, können Sie sich im *Server-Explorer* durch die Datenbankstrukturen bewegen. Ziehen Sie die Tabellen *Customer*, *CustomerAddress* und *Address* per Drag & Drop in den O/R-Designer. Sie erhalten eine ähnliche Ansicht wie in Abbildung 9.2.
- Dabei ist es zunächst übrigens völlig gleich, ob Sie Tabellen der Datenbank oder Sichten in den O/R-Designer ziehen – das hat bestenfalls Einfluss auf die entsprechenden generierten Klassen. Auch gespeicherte Prozeduren lassen sich auf diese Weise mit dem O/R-Designer verarbeiten, jedoch landen diese, anders als Sichten oder Tabellen, nicht im Hauptbereich sondern im rechten Bereich des O/R-Designers, wie in Abbildung 9.6 gekennzeichnet.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

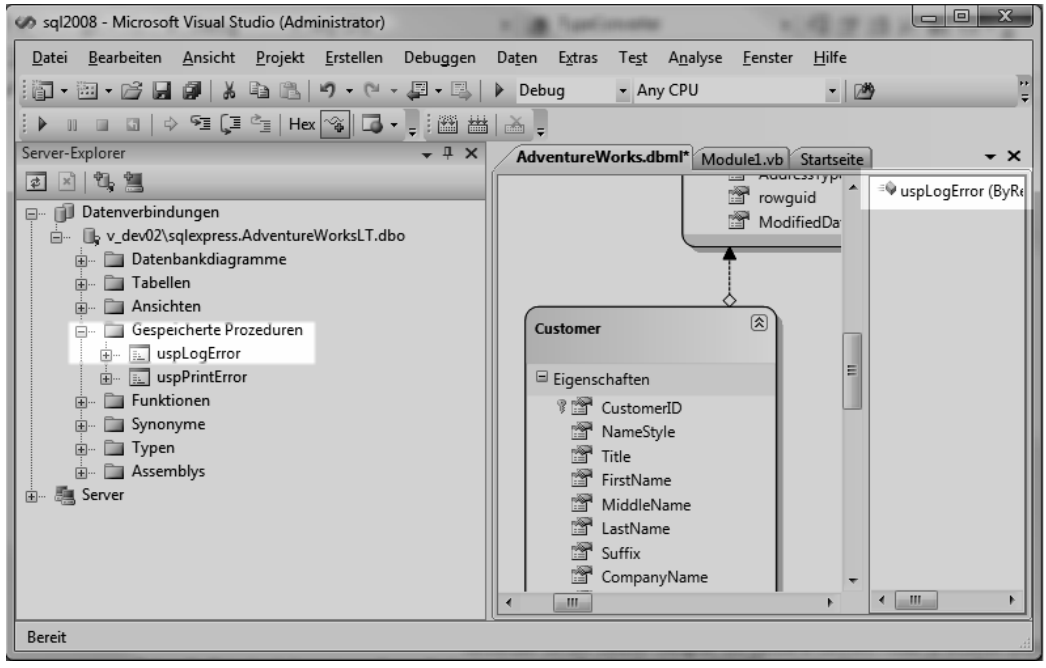



**Abbildung 9.6** Hinzufügen der gespeicherten Prozedur uspLogError

- Speichern Sie Ihre Design-Änderungen ab. Wenn Sie mit dem entsprechenden Symbol des Projektmappen-Explorers *Alle Dateien* im Projektmappen-Explorer eingeblendet haben, finden Sie dort eine Datei namens *AdventureWorks.designer.vb*.

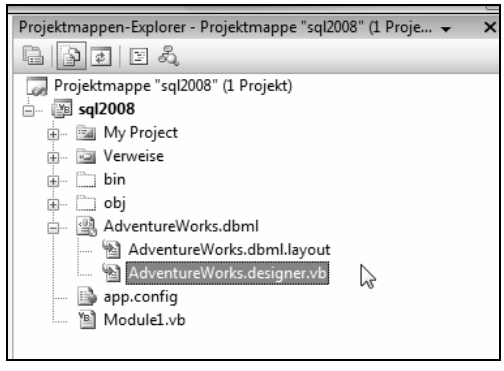



**Abbildung 9.7** Der O/R-Designer erstellt eine VB-Codedatei

Diese Datei beinhaltet unter anderem den Quellcode für einen so genannten *DataContext*. Der DataContext ist die Verwaltungsinstanz bei LINQ to SQL, etwas, was wir bislang bei LINQ to Objects nicht kennen gelernt haben, weil es nicht benötigt wurde: Der DataContext verwaltet beispielsweise die Verbindungszeichenfolge für den Verbindungsaufbau zur Datenbank und übernimmt die Transaktionssteuerung. Sämtliche Tabellenstrukturen werden im DataContext definiert. Der DataContext übernimmt zur Laufzeit in etwa die Rolle, die der TableAdapter bei typisierten DataSets spielt – er kümmert sich ebenfalls um die Aktualisierungslogik.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Wechseln Sie nun zu *Module1.vb*, und ändern Sie die von `Sub Main` definierte Methode folgendermaßen ab:

```
Sub Main()
    Dim datacontext As New AdventureWorksDataContext
    Dim customeraddress = From adr In datacontext.Addresses _
                          Join custadrassoc In datacontext.CustomerAddresses _
                          On adr.AddressID Equals custadrassoc.AddressID _
                          Join cust In datacontext.Customers _
                          On cust.CustomerID Equals custadrassoc.CustomerID _
                          Where cust.LastName = "Geist" _
                          Select New With {.Title = cust.Title, _
                                           .Firstname = cust.FirstName, _
                                           .LastName = cust.LastName, _
                                           .City = adr.City, _
                                           .CountryRegion = adr.CountryRegion, _
                                           .Phone = cust.Phone _
                          }
    For Each ds In customeraddress
        Console.WriteLine("{0} {1} {2} wohnt in {3}/{4} " & _
                          "und ist unter der Telefonnummer {5} " & _
                          "erreichbar.", ds.Title, _
                          ds.Firstname, ds.LastName, _
                          ds.City, ds.CountryRegion, _
                          ds.Phone)
    Next
    Console.WriteLine("Bitte mit Return bestätigen")
    Console.ReadLine()
End Sub
```

Die Syntax von LINQ haben Sie bereits in Kapitel 8 »LINQ to Objects« kennen gelernt – und jetzt lernen Sie die Stärken von LINQ kennen: Kennen Sie eine, kennen Sie alle! (Einmal von den Feinheiten abgesehen, die Sie in Ergänzung kennen müssen.)

In dieser Abfrage werden die drei Tabellen *Address*, *CustomerAddress* und *Customer* durch eine `Join`-Klausel miteinander verknüpft. Zudem wird die Abfrage auf den Nachnamen *Geist* beschränkt. Für alle gefundenen Datensätze (hier nur einer) werden anschließend Informationen zu dem Kunden und der Adresse ausgegeben – für die Ergebnismenge wird dazu mit `Select` eine Auflistung mit Instanzen einer entsprechenden anonymen Klasse erstellt.

Die Ausgabe dieses Programms liefert:

```
Mr. Jim Geist wohnt in Puyallup/United States und ist unter der Telefonnummer 724-555-0161 erreichbar.
Bitte mit Return bestätigen
```

Und jetzt kommt das Entscheidende: Im Gegensatz zu LINQ to Objects wird das Ergebnis dieser Abfrage nicht auf dem Rechner ermittelt, auf dem das Programm läuft, sondern aus der LINQ-Abfrage wird ein SQL-Kommando erstellt, zur Datenbank versendet und das Ergebnis wird anschließend von der Datenbank quasi »abgeholt«.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Sie können das einfach überprüfen, indem Sie vor der For/Each-Schleife der `Log`-Eigenschaft des Datenkontexts eine `TextWriter`-Instanz hinzufügen. Wir nutzen hierfür `Console.Out`:

```
        datacontext.Log = Console.Out
```

Die Ausgabe des Programms liefert nun etwas mehr Informationen, und verrät das Geheimnis, wie oder vielmehr mit welchen SQL-Statements die Kommunikation zwischen dem Client und dem SQL Server über den Datenkontext vonstatten geht:

```
SELECT [t2].[Title], [t2].[FirstName] AS [Firstname], [t2].[LastName],
       [t0].[City], [t0].[CountryRegion], [t2].[Phone]
FROM [SalesLT].[Address] AS [t0]
INNER JOIN [SalesLT].[CustomerAddress] AS [t1]
      ON [t0].[AddressID] = [t1].[AddressID]
INNER JOIN [SalesLT].[Customer] AS [t2]
      ON [t1].[CustomerID] = [t2].[CustomerID]
WHERE [t2].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Mr. Jim Geist wohnt in Puyallup/United States und ist unter der Telefonnummer 72
4-555-0161 erreichbar.
Bitte mit Return bestätigen
```

Wie Sie sehen, werden nur die Daten selektiert, die wir auch in dem `Select`-Ausdruck festgelegt haben. Zudem wird unsere `Where`-Klausel ebenso an den SQL Server übermittelt. (Der Parameter *@p0* wird, wie in der nächsten Zeile aufgeführt, durch *Geist* ersetzt.)

Diese Umwandlung von LINQ to SQL in eine `Select`-Anweisung wird durch einen datenbankspezifischen LINQ-Provider durchgeführt, und ich kann Ihnen sehr empfehlen, ein wenig mit verschiedenen Abfragen gegen die AdventureWorks-Datenbank zu experimentieren, um einerseits ein Gefühl für LINQ to SQL an sich, andererseits aber auch eines für die Umsetzung von lokalem LINQ auf »echtes« T-SQL zu bekommen.

# Kaskadierte Abfragen

Aufgrund der verzögerten Ausführung von LINQ-Abfragen, die für LINQ to SQL genau so gilt wie für LINQ to Objects (Kapitel 8 hält mehr darüber bereit), können sehr einfache und übersichtliche Sub-Select-Abfragen mit LINQ to SQL durchgeführt werden.

Im folgenden Beispiel wiederholen wir die Adressabfrage für unseren Herrn Geist. Diesmal interessieren uns jedoch nicht die Kundendaten, sondern lediglich seine Adresse:

```
    Dim datacontext As New AdventureWorksDataContext
        Dim cust = From row In datacontext.Customers _
                   Where row.LastName = "Geist"
        Dim custadrassoc = From row In datacontext.CustomerAddresses _
                           Where cust.Any(Function(c) c.CustomerID = row.CustomerID)
        Dim adr = From row In datacontext.Addresses _
                  Where custadrassoc.Any(Function(a) a.AddressID = row.AddressID)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        datacontext.Log = Console.Out
        For Each ds In adr
            Console.WriteLine("Herr Geist wohnt in {0}/{1}", ds.City, ds.CountryRegion)
        Next
        Console.WriteLine("Bitte mit Return bestätigen")
        Console.ReadLine()
    End Sub
```

Als erstes selektieren wir die Kundendaten des Herrn Geist. Diese Abfrage wird in `cust` gespeichert. Als nächstes selektieren wir aus der Tabelle `CustomerAddresses` den Datensatz, der zur Kundennummer von Herrn Geist passt. Diese Abfrage wird in `custadrassoc` gespeichert.

Nun können wir in der Tabelle `Addresses` suchen, und wir benutzen dazu die `Any`-Klausel, um den entsprechenden Datensatz zu finden, der den Kriterien entspricht, da die Ausgangsliste natürlich ebenfalls komplett durchsucht werden muss – und das ist genau das, was `Any` leistet. Diese Abfrage wird in `adr` gespeichert.

**HINWEIS** Noch einmal zur Erinnerung: Das Prinzip der verzögerten Ausführung bestimmt, dass bislang noch keine Abfrage ausgeführt wurde. Es wurden lediglich die Abfragen definiert. Erst durch die `For/Each`-Schleife werden die drei Abfragen zu einer zusammengesetzt und ausgeführt, weil – wie wir es bei LINQ to Objects bereits kennen gelernt haben – das erste Mal der Versuch gestartet wird, ein Element der Ergebnismenge abzurufen.

Als Ergebnis erhalten wir die Anschrift von Herrn Geist (und, da das Logging für die SQL-Abfragegenerierung noch aktiv ist, auch die eigentliche SQL-Abfrage, die an den SQL-Server gesendet wird).

```
SELECT [t0].[AddressID], [t0].[AddressLine1], [t0].[AddressLine2], [t0].[City],
       [t0].[StateProvince], [t0].[CountryRegion], [t0].[PostalCode], [t0].[rowguid],
       [t0].[ModifiedDate]
FROM [SalesLT].[Address] AS [t0]
WHERE EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [SalesLT].[CustomerAddress] AS [t1]
    WHERE ([t1].[AddressID] = [t0].[AddressID]) AND (EXISTS(
        SELECT NULL AS [EMPTY]
        FROM [SalesLT].[Customer] AS [t2]
        WHERE ([t2].[CustomerID] = [t1].[CustomerID]) AND ([t2].[LastName] = @p0)
        ))
    )
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Herr Geist wohnt in Puyallup/United States.
Bitte mit Return bestätigen
```

Übrigens: Wenn Sie schon ein wenig mit den verschiedenen Abfragemöglichkeiten von LINQ to Objects bzw. LINQ to SQL herumexperimentiert haben, haben Sie sich möglicherweise gefragt, warum wir hier im Beispiel die `Any`-Methode von den Abfragen `cust` und `custadrassoc` verwendet haben, obwohl `All` in diesem Zusammenhang sinnvoller erscheinen könnte. Um Ihnen diesen Unterschied vor Augen zu führen, ändern wir den Code ab und verwenden nun die `All`-Methode für das weitere Selektieren der Daten:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    Dim datacontext As New AdventureWorksDataContext
        Dim cust = From row In datacontext.Customers _
                   Where row.LastName = "Geist"
        Dim custadrassoc = From row In datacontext.CustomerAddresses _
                           Where cust.All(Function(c) c.CustomerID = row.CustomerID)
        Dim adr = From row In datacontext.Addresses _
                  Where custadrassoc.All(Function(a) a.AddressID = row.AddressID)

        datacontext.Log = Console.Out
        For Each ds In adr
            Console.WriteLine("Herr Geist wohnt in {0}/{1}", ds.City, ds.CountryRegion)
        Next
        Console.WriteLine("Bitte mit Return bestätigen")
        Console.ReadLine()
```

Wenn das Programm nun gestartet wird erhalten wir folgende Ausgabe:

```
SELECT [t0].[AddressID], [t0].[AddressLine1], [t0].[AddressLine2], [t0].[City],
       [t0].[StateProvince], [t0].[CountryRegion], [t0].[PostalCode], [t0].[rowguid],
       [t0].[ModifiedDate]
FROM [SalesLT].[Address] AS [t0]
WHERE NOT (EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [SalesLT].[CustomerAddress] AS [t1]
    WHERE ((
        (CASE
            WHEN [t1].[AddressID] = [t0].[AddressID] THEN 1
            ELSE 0
         END)) = 0) AND (NOT (EXISTS(
        SELECT NULL AS [EMPTY]
        FROM [SalesLT].[Customer] AS [t2]
        WHERE ((
            (CASE
                WHEN [t2].[CustomerID] = [t1].[CustomerID] THEN 1
                ELSE 0
             END)) = 0) AND ([t2].[LastName] = @p0)
        )))
    ))
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Herr Geist wohnt in Puyallup/United States.
Bitte mit Return bestätigen
```

Diese Abfrage funktioniert ebenfalls, aber wie Sie sehen, ist das erstellte `Select` deutlich komplexer.

Die `All`-Methode betrachtet jeden Datensatz, während `Any` nur die Datensätze betrachtet, die den Kriterien entsprechen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Daten verändern und speichern

Wenn Sie mit den aus den Abfragen erhaltenen Daten weiterarbeiten möchten, ohne sie jedes Mal neu zu selektieren, wandeln Sie das Ergebnis der Abfrage in eine Liste oder ein Array um. Dieses erfolgt beispielsweise mit den Methoden `ToList` oder `ToArray` – wir haben das im LINQ to Objects-Kapitel bereits kennengelernt.

**HINWEIS** Das gilt für LINQ to SQL umso mehr, als dass Sie es nur in den wenigsten Fällen wirklich wünschen, dass die eigentliche SQL-Abfrage immer und immer wieder zur Ausführung an den SQL-Server übermittelt wird. In den meisten Fällen wird es reichen, die Abfrage ein Mal auszuführen, und die Ergebnisliste schließlich mit `ToArray` bzw. `ToList` von der eigentlichen Abfrage zu trennen.

```
Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
           Where row.LastName = "Geist"
datacontext.Log = Console.Out
Dim custlist = cust.ToList
```

Sie können jetzt mit der Liste arbeiten, ohne dass die Daten bei jedem Zugriff neu geladen werden.

Konstruieren wir unser Beispiel weiter, und nehmen wir an, dass unser Herr Geist die Telefongesellschaft gewechselt und nun eine neue Telefonnummer bekommen hat:

```
Dim herrGeist = custlist(0)
herrGeist.Phone = "555/1234567"
```

LINQ to SQL überwacht die geladenen Daten und kann so auch Veränderungen an diesen erkennen. Anhand dieser Veränderungen können `Insert`-, `Update`- und `Delete`-SQL-Kommandos für die Datenbank erstellt und die Änderungen automatisiert zurückgeschrieben werden.

Dieses erfolgt durch Aufruf der Methode `SubmitChanges` am verwendeten DataContext.

```
datacontext.SubmitChanges()
```

Wir verwenden folgendes Beispiel, um uns die Ausgaben des DataContext anzusehen:

```
Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
           Where row.LastName = "Geist"

datacontext.Log = Console.Out
Dim custlist = cust.ToList
Dim herrGeist = custlist(0)

Console.WriteLine("Ändern der Telefonnummer")
herrGeist.Phone = "555/1234567"

Console.WriteLine("Änderungen Speichern")
datacontext.SubmitChanges()
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Es werden folgende Ausgaben erzeugt:

```
SELECT [t0].[CustomerID], [t0].[NameStyle], [t0].[Title], [t0].[FirstName], [t0]
.....(Ausgabe gekürzt) .....
FROM [SalesLT].[Customer] AS [t0]
WHERE [t0].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
Ändern der Telefonnummer
Änderungen speichern
UPDATE [SalesLT].[Customer] SET [Phone] = @p12
WHERE ([CustomerID] = @p0) AND (NOT ([NameStyle] = 1)) AND ([Title] = @p1) AND (
[FirstName] = @p2) AND ([MiddleName] IS NULL) AND ([LastName] = @p3) AND ([Suffi
x] IS NULL) AND ([CompanyName] = @p4) AND ([SalesPerson] = @p5) AND ([EmailAddre
ss] = @p6) AND ([Phone] = @p7) AND ([PasswordHash] = @p8) AND ([PasswordSalt] =
@p9) AND ([rowguid] = @p10) AND ([ModifiedDate] = @p11)
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
.....(Ausgabe gekürzt) .....
-- @p10: Input UniqueIdentifier (Size = 0; Prec = 0; Scale = 0) [c6ebb29a-cc67-4
59c-90e3-339e0f912906]
-- @p11: Input DateTime (Size = 0; Prec = 0; Scale = 0) [13.10.2004 11:15:07]
-- @p12: Input NVarChar (Size = 11; Prec = 0; Scale = 0) [555/1234567]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
```

Der Parameter `@p12` beinhaltet nun unsere geänderte Telefonnummer. Die Änderungen wurden also zurück in die Datenbank gespeichert.

### Einfügen und löschen von Datensätzen

`SubmitChanges` repliziert Änderungen an der Auflistung im vorherigen Beispiel in die Datenbank. Beim Einfügen *neuer* Datensätze gibt es die Möglichkeit, direkt mit der `InsertOnSubmit`-Methode zu arbeiten, die als Argument eine Instanz der Klasse übernimmt, die einen Datensatz abbildet. Das Löschen eines Datensatzes funktioniert wiederum mit `DeleteOnSubmit`. Auch hier werden Änderungen nicht sofort, sondern erst *nach* einem abschließenden `SubmitChanges` an die Datenbank übermittelt. Das folgende Beispiel verdeutlicht auch diese Funktionsweise.

```
Dim datacontext As New AdventureWorksDataContext, customerObjektInstanz As Customer
'Im Objektinitialisierer müssen die "Muss"-Felder (die DBNull nicht enthalten dürfen) angegeben werden
datacontext.Customers.InsertOnSubmit(New Customer() with { .FirstName="Andreas", .LastName="Belke", ...}
'So würden Sie eine vorhandene (aber instanzierte!) customer-Instanz löschen
datacontext.Customers.DeleteOnSubmit(customerObjektInstanz)
'Erst hier werden eingefügte/gelöschte Datensätze in die entsprechende SQL Server-Tabelle übernommen
datacontext.SubmitChanges()
```

## Concurrency-Check (Schreibkonfliktprüfung)

Was aber, wenn die Daten bereits von einem anderen Anwender geändert wurden? In diesem Fall wird eine `ChangeConflictException` ausgelöst. LINQ überprüft beim Aktualisieren, ob der zu ändernde Datensatz noch in der Version vorliegt, in der er ursprünglich geladen wurde. Ist das nicht der Fall, wird die Ausnahme ausgelöst. Standardmäßig werden alle Felder einer Tabelle überprüft. Dieses kann jedoch im Designer für jedes Feld einzeln eingestellt werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

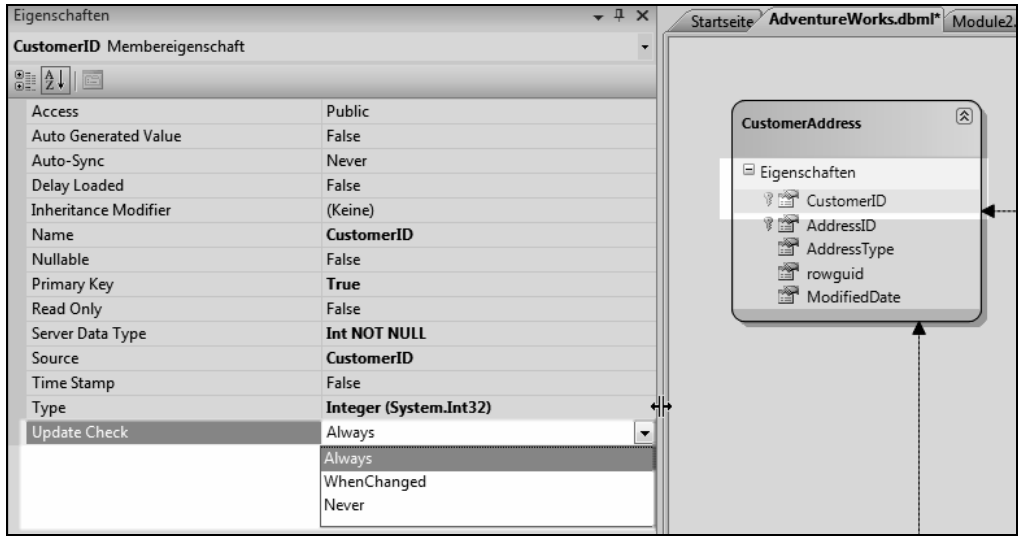



**Abbildung 9.8** Ändern der Kollisionsüberprüfung eines Attributes

Wenn eine Tabelle einen Änderungszeitstempel besitzt oder eine Versionsnummer des Datensatzes gepflegt wird, können die Daten auch anhand dieser Angaben auf Eindeutigkeit geprüft werden. Hierzu muss die `Time Stamp` Eigenschaft bei dem jeweiligen Attribut auf `true` gesetzt werden.

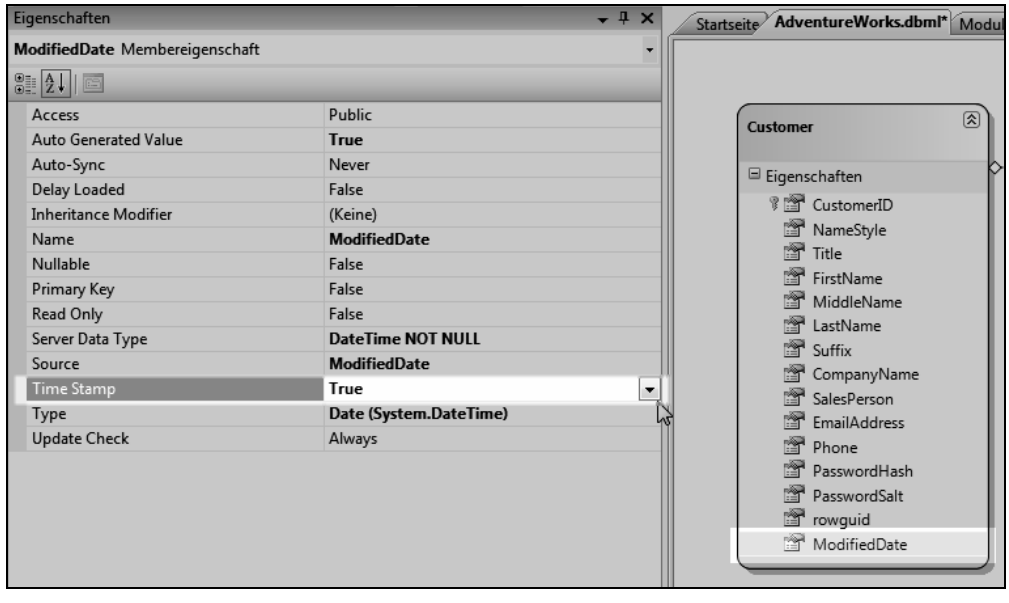



**Abbildung 9.9** Verwenden eines Zeitstempels zur Kollisionsüberprüfung

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Häufig wird anstelle von `Time Stamp` auch von `IsVersion` gesprochen. Tatsächlich wird die Einstellung von `Time Stamp` in dem `ColumnAttribut` unter `IsVersion` gespeichert. Es handelt sich also letzten Endes um die gleiche Eigenschaft, die im Designer nur »über Umwege« angesprochen wird.

**WICHTIG** Wird bei einem Feld eine Tabelle `Time Stamp` auf `true` gesetzt, werden für alle anderen Felder die `Update Check`-Angaben ignoriert.

Ändern Sie für unser Beispiel im O/R Designer das `ModifiedDate` der Tabelle `Customer`. Setzen Sie die Eigenschaft `Time Stamp` auf `true`. Ändern Sie den Code anschließend folgendermaßen ab:

```
        Console.WriteLine("Ändern der Telefonnummer")
        herrGeist.Phone = "555/123456"
```

**WICHTIG** Ändern Sie auch die Telefonnummer, die Sie `herrGeist.Phone` zuweisen. Da das Programm bereits einmal gelaufen ist, wurde die Telefonnummer bereits in der Datenbank geändert. Demzufolge wird diese Telefonnummer auch wieder geladen und das Zuweisen derselben Telefonnummer wäre keine Änderung und es würden auch keine Update-Kommandos erstellt.

```
SELECT [t0].[CustomerID], [t0].[NameStyle], [t0].[Title], [t0].[FirstName],
[t0].[MiddleName], [t0].[LastName], [t0].[Suffix], [t0].[CompanyName],
[t0].[SalesPerson], [t0].[EmailAddress], [t0].[Phone], [t0].[PasswordHash],
[t0].[PasswordSalt], [t0].[rowguid], [t0].[ModifiedDate]
FROM [SalesLT].[Customer] AS [t0]
WHERE [t0].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Ändern der Telefonnummer
Änderungen speichern
UPDATE [SalesLT].[Customer]
SET [Phone] = @p2
WHERE ([CustomerID] = @p0) AND ([ModifiedDate] = @p1)

SELECT [t1].[ModifiedDate]
FROM [SalesLT].[Customer] AS [t1]
WHERE ((@@ROWCOUNT) > 0) AND ([t1].[CustomerID] = @p3)
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
-- @p1: Input DateTime (Size = 0; Prec = 0; Scale = 0) [13.10.2004 11:15:07]
-- @p2: Input NVarChar (Size = 11; Prec = 0; Scale = 0) [555/1234567]
-- @p3: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Bitte mit Return bestätigen
```

**HINWEIS** Falls im obigen Bildschirmauszug der Teil mit dem Update-Kommando fehlen sollte, müssen Sie die Telefonnummer in der Datenbank ändern! Wie Sie sehen, wurde bei dem Update in der `Where`-Klausel nur auf das `ModifiedDate` geprüft.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Eine weitere Möglichkeit, die Kollisionsüberprüfung zu steuern, liegt in der Methode `SubmitChanges` selbst. Es existiert eine überladene Version, bei der eine `ConflictMode`-Einstellung angegeben werden kann.

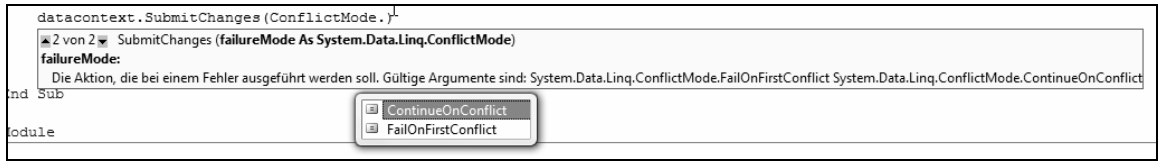



**Abbildung 9.10** Mögliche ConflictModes-Einstellungen

Mögliche Werte sind:

- `ContinueOnConflict`: Es bedeutet *nicht*, dass Fehler durch parallele Zugriffe ignoriert werden, sondern es ist vielmehr so, dass auftretende Kollisionen quasi »gesammelt« und erst zum Schluss durch eine entsprechende Ausnahme »bekannt gegeben« werden.
- `FailOnFirstConflict`: Schon beim Auftreten des *ersten* Konfliktes wird eine Ausnahme ausgelöst.

# Transaktionen

Standardmäßig werden Änderungen an die Datenbank durch den DataContext automatisch als Transaktion übermittelt, falls keine explizite Transaktion im Gültigkeitsbereich vorgefunden wird. Um eine übergreifende Transaktion zu nutzen, stehen zwei Möglichkeiten zur Verfügung.

## TransactionScope (Transaktionsgültigkeitsbereich)

Um Transaktionen mit `TransactionScope` nutzen zu können, muss die `System.Transactions.dll` Assembly in das Projekt aufgenommen werden.

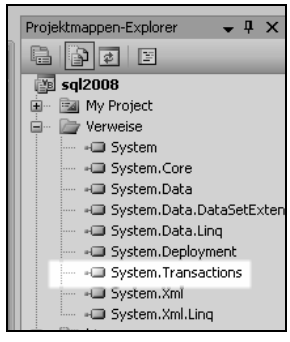



**Abbildung 9.11** Einbinden der `System.Transaction.dll` Assembly

```
Dim ts As New Transactions.TransactionScope
Using ts
    datacontext.ExecuteCommand("exec protokolliere_aenderungen")
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            datacontext.SubmitChanges(ConflictMode.ContinueOnConflict)
            ts.Complete()
        End Using
```

Exemplarisch für weitere Änderungen an dem Datenhaushalt wurde hier die `ExecuteCommand`-Methode des DataContext verwendet, um eine datenverändernde gespeicherte Prozedur aufzurufen.

TransactionScope sorgt automatisch für das Durchführen eines Rollbacks, falls Fehler in der Transaktion auftreten. Sie müssen lediglich dafür sorgen, dass zum richtigen Zeitpunkt die `Complete`-Methode des `TransactionScope`-Objekts aufgerufen wird, um die geänderten Daten in der Datenbank mit einem *Commit* festzuschreiben.

## Verwenden der Transaktionssteuerung des DataContext

Gerade für Anwendungen die noch mehr ADO.Net-orientiert sind, existiert eine weitere Möglichkeit mit Transaktionen zu arbeiten.

DataContext besitzt eine `Transaction`-Eigenschaft. Es ist ebenso möglich, über diese Eigenschaft Transaktionen zu steuern. Jedoch ist hier ein deutlicher Mehraufwand notwendig – der folgende Codeauszug soll das exemplarisch demonstrieren:

```
        datacontext.Transaction = datacontext.Connection.BeginTransaction()
        Try
            datacontext.ExecuteCommand("exec protokolliere_aenderungen")
            datacontext.SubmitChanges()
            datacontext.Transaction.Commit()
        Catch ex As Exception
            datacontext.Transaction.Rollback()
            Throw ex
        Finally
            datacontext.Transaction = Nothing
        End Try
```

Als erstes muss eine neue Transaktion eingeleitet werden. Dazu wird auf dem `Connection`-Objekt des DataContexts die `BeginTransaction`-Methode aufgerufen. Sie erstellt ein neues Transaktionsobjekt. Änderungen am Datenbankhaushalt müssen nun in einem Try-Catch-Block durchgeführt werden. Im Fehlerfall wird eine Ausnahme ausgelöst. Sie muss abgefangen werden, um das Rollback durchzuführen.

Zudem sollte im `Finally`-Block das Transaktionsobjekt wieder auf `Nothing` gesetzt werden. So startet der DataContext automatisch eine neue Transaktion bei einem `SubmitChanges`, sofern nicht explizit über `BeginTransaction` eine neue Transaktion begonnen wird.

# Was, wenn LINQ einmal nicht reicht

Unter Umständen kommen Sie einmal in eine Situation, in der Sie eine `Select`-Abfrage mit LINQ nicht abbilden können. In diesem Fall können Sie selbst das T-SQL-Kommando für SQL Server erstellen, um die Datenbank abzufragen. Das Schöne daran ist, dass Sie, nachdem die Daten geladen wurden, nicht auf die LINQ-Features zu verzichten brauchen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der DataContext stellt eine Methode namens `ExecuteQuery` bereit, mit dem beliebige T-SQL-Abfragen direkt an den SQL Server geschickt werden können:

```
Dim allrows = From ds In datacontext.Customers
Dim cmd As Common.DbCommand = datacontext.GetCommand(allrows)
Dim sel = cmd.CommandText & " where lastname = {0}"
Dim res = datacontext.ExecuteQuery(Of Customer)(sel, New Object() {"Geist"})
For Each ds In res
    Console.WriteLine("Herr Geist hat die KundenNr {0}", ds.CustomerID)
Next
```

Wir definieren hier im Beispielcode eine Abfrage auf der Tabelle `Customers` und speichern sie in `allrows`. Der DataContext stellt die Methode `GetCommand` zur Verfügung, mit der man das generierte Select-Kommando erfragen kann.

Dieses wird als `cmd` gespeichert. Der `CommandText` enthält das eigentliche `Select`-Kommando im Klartext.

Wir erweitern es um

```
where lastname = {0}
```

Bei `{0}` handelt es sich um einen Parameter, und zwar genauer gesagt, um den ersten Parameter, welcher der Abfrage übergeben wurde. Den zweiten erreichen Sie mit `{1}`, usw. Beim Aufrufen der Methode `ExecuteQuery` muss die Klasse angegeben werden, die einen Datensatz der Tabelle aufnehmen kann. In diesem Fall die Klasse `Customer` (demgegenüber steht `Customers`, welches eine *Tabelle* von `Customer`-Instanzen ist).[3] Als Parameter wird das `Select`-Kommando übergeben, sowie ein Objekt-Array, das die im *Select* festgelegten Parameter in der entsprechenden Reihenfolge enthält (die Parameter können ebenso kommasepariert – also ohne Verwendung eines Arrays – übergeben werden). In diesem Fall wird nur ein Parameter vom Typ `String` übergeben, nämlich »Geist«. Als Ergebnis erhalten wir:

```
Herr Geist hat die KundenNr 37
```

Die Methode `ExecuteQuery` ist für Datenbankabfragen gedacht. Für Datenbankänderungen existiert die Methode `ExecuteCommand` des DataContext.

Als Beispiel:

```
datacontext.ExecuteCommand("update saleslt.customer set phone={0} where lastname={1}", "555-987654321",
"Geist")
```

[3] Die Namen werden durch den O/R-Designer von LINQ to SQL vergeben – leider haben wir hier Abweichungen in der deutschen Visual Basic 2008 Express Edition und der Visual Studio 2008 Professional Edition (deutsche Vorabversion auf Basis des RC) bei der automatischen Namensvergabe für die jeweiligen Objektnamen feststellen müssen. Es ist davon auszugehen, dass diese Inkonsistenten in der finalen Version behoben sein werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 10

# LINQ to XML

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Einführung in LINQ to XML

XML steht für *Extensible Markup Language,* was auf Deutsch soviel wie *erweiterbare Auszeichnungssprache* bedeutet. XML ist eine Untermenge von SGML, welches bereits 1986 vom World Wide Web Consortium (W3C) standardisiert wurde.

Bevor wir uns mit dem eigentlichen Kapitelthema *LINQ to XML* beschäftigen, lassen Sie uns kurz XML als solches rekapitulieren: XML besteht, vereinfacht gesagt, aus Elementen, Attributen und Werten.

**BEGLEITDATEIEN** Unter *.\Samples\Chapter10 - LinqToXml\LinqToXmlDemo* finden Sie die Beispieldateien für dieses Projekt, falls Sie die Beispiele nicht händisch nachvollziehen wollen.

Beispiel:

```
<software Installierbar = "Ja" >Visual Studio 2008 </software>
```

Das Element `Software` besitzt ein Attribut `Installierbar` und den Wert `Visual Studio 2008`. Elemente in XML beginnen mit einem Starttag `<name>` und enden mit dem Endtag `</name>`, wobei *name* durch ein beliebiges Wort getauscht werden kann (Achtung: keine Leerzeichen!).

Zwischen `<name>` und `</name>` steht der Wert. Ist kein Wert vorhanden, kann anstelle von `<name></name>` auch `<name/>` geschrieben werden.

Attribute werden in dem Element notiert, etwa:

```
<person name="Müller" vorname="Hans"/>
```

Dieses Beispiel definiert ein Element ohne Wert, dass zudem die Attribute `name` und `vorname` beinhaltet.

Eine detaillierte Beschreibung von XML würde den Umfang dieses Buches sprengen, daher sei hier auf die gute deutsche Einführung von *SelfHtml* [1] bzw. auf die Seiten von *W3C* [2] verwiesen.

# Vergleich Visual Basic 8 und Visual Basic 9

Um unter Visual Basic 8 eine XML-Struktur zu erstellen oder zu ändern, musste einiger Aufwand betrieben werden. Als Beispiel soll hier folgende XML-Struktur um das Element `baujahr` erweitert werden:

```
 <fuhrpark>
  <kennzeichen>MS - VB 2008</kennzeichen>
  <ladung menge="10 Tonnen">Salz</ladung>
  <hersteller>MAN</hersteller>
  <baujahr>1998</baujahr>
</fuhrpark>
```

[1] *http://de.selfhtml.org*

[2] *http://www.w3c.org*

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der Code lautete hierzu für VB8:

```
Dim xml As New XmlDocument
xml.Load(Application.StartupPath + "\fuhrpark1.xml")
Dim nodeList As XmlNodeList = xml.GetElementsByTagName("fuhrpark")
For Each node As XmlNode In nodeList
    Dim xmlElement As XmlElement = xml.CreateElement("baujahr")
    xmlElement.InnerText = "1998"
    node.AppendChild(xmlElement)
Next
```

In Visual Basic 9 gibt es viele Vereinfachungen bezüglich der XML-Verarbeitung. Es sind neue Klassen hinzugekommen. Die wichtigsten sind `XDocument`, `XElement` und `XAttribute`.

Um nun die obere Struktur zu erhalten, reicht es aus, folgenden Code zu erstellen:

```
Dim xml As New XDocument
xml.Add(New XElement("fuhrpark", _
            New XElement("kennzeichen", "MS - VB 2008"), _
            New XElement("ladung", _
                        New XAttribute("menge", "10 Tonnen"), _
                        "Salz" _
                        ), _
            New XElement("hersteller", "MAN") _
))
```

Unter dem Element `fuhrpark` werden die Elemente `kennzeichen`, `ladung` und `hersteller` gespeichert. Zudem erhält das Element `ladung` ein weiteres Attribut namens `menge`. Das Hinzufügen des `baujahr`-Elements kann nun durch folgende Zeile erreicht werden.

```
xml.Element("fuhrpark").Add(New XElement("baujahr", 1998))
```

Der Wert eines `XElements` (z. B. `1998`) kann über die `Value` Property erfragt werden.

# XML-Literale – XML direkt im Code ablegen

Mit Visual Basic 9 kann man `XDocument`-und `XElement`-Objekte direkt im Code erstellen – die entsprechenden Elemente dafür nennen sich *XML-Literale*. Mithilfe von XML-Literalen ist es möglich, XML inline direkt im Quellcode zu erstellen, wie das folgende Beispiel zeigt:

```
Dim fuhrpark As XElement = <fuhrpark>
                               <kennzeichen>MS - VB 2008</kennzeichen>
                               <ladung menge="10 Tonnen">Salz</ladung>
                               <hersteller>MAN</hersteller>
                           </fuhrpark>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Beim `fuhrpark`-Objekt handelt es sich um ein gewöhnliches `XElement` und `baujahr` kann nun wie gewohnt hinzugefügt werden:

```
fuhrpark.Add(New XElement("baujahr", 1998))
```

Mit XML-Literalen geht das sogar noch einfacher:

```
fuhrpark.Add(<baujahr>1998</baujahr>)
```

Darüber hinaus ist es auch möglich, Ausdrücke mit `<%=  %>` als XML in den Code einzubetten, wie die folgenden beiden Zeilen zeigen:

```
        Dim myBaujahr As Integer = 1998
        fuhrpark.Add(<baujahr><%= myBaujahr %></baujahr>)
```

Beim Aufrufen der `Add`-Methode wird der Wert des `baujahr`-Elements durch die Variable `myBaujahr` ersetzt.

# Erstellen von XML mithilfe von LINQ

Mithilfe von LINQ können ebenso XML-Dokumente bzw. -Elemente erstellt werden.

Aus einer bestehenden Liste, die durch die folgenden Codezeilen erstellt wird …

```
        Dim fahrzeugliste As New List(Of Fahrzeug)
        fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-VB 2008", .Hersteller = "MAN", _
                                             .Ladung = New Fahrzeug.LadungsBeschreibung With
                                              {.Menge = 10000, .Gut = "Salz"}})
        fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-C# 2008", .
                                              Hersteller = "Mercedes-Benz",
                                             .Ladung = New Fahrzeug.LadungsBeschreibung With
                                              {.Menge = 20000, .Gut = "Erdnüsse"}})
        fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-J# 2008", .Hersteller = "DAF",
                                             .Ladung = New Fahrzeug.LadungsBeschreibung With
                                              {.Menge = 5000, .Gut = "Wackeldackel"}})

    Private Class Fahrzeug
        Public Class LadungsBeschreibung
            Private _menge As Double
            Private _gut As String

            Public Property Menge() As Double
                        …
            End Property

            Public Property Gut() As String
                        …
            End Property
        End Class
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private _kennzeichen As String
        Private _ladung As LadungsBeschreibung
        Private _hersteller As String

        Public Property Kennzeichen() As String
                …
        End Property

        Public Property Ladung() As LadungsBeschreibung
                …
        End Property

        Public Property Hersteller() As String
                …
        End Property
    End Class
```

… soll folgendes XML-Element erstellt werden:

```
<fuhrpark>
  <fahrzeug>
    <kennzeichen>MS-VB 2008</kennzeichen>
    <hersteller>MAN</hersteller>
    <ladung menge="10000">Salz</ladung>
  </fahrzeug>
  <fahrzeug>
    <kennzeichen>MS-C# 2008</kennzeichen>
    <hersteller>Mercedes-Benz</hersteller>
    <ladung menge="20000">Erdnüsse</ladung>
  </fahrzeug>
  <fahrzeug>
    <kennzeichen>MS-J# 2008</kennzeichen>
    <hersteller>DAF</hersteller>
    <ladung menge="5000">Wackeldackel</ladung>
  </fahrzeug>
</fuhrpark>
```

Das Anlegen der XML-Strukturen erfolgt dabei mit `XElement` und `XAttribute`:

```
        Dim fuhrpark = New XElement("fuhrpark", _
                From fzg In fahrzeugliste _
                Select New XElement("fahrzeug", _
                                New XElement("kennzeichen", fzg.Kennzeichen), _
                                New XElement("hersteller", fzg.Hersteller), _
                                New XElement("ladung", fzg.Ladung.Gut,
                                     New XAttribute("menge", fzg.Ladung.Menge)) _
                                )
                      )
```

Dem `XElement` mit den Namen `Fuhrpark` wird eine Liste von XElementen (`fahrzeug`) hinzugefügt. Diese Liste (genaugenommen `IEnumerable(of XElement)`) wird mit *LINQ to Objects* erstellt. Das `fahrzeug`-Element wird zudem um die entsprechenden Unterelemente erweitert.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Auch hier ist wieder die Verwendung von Literalen möglich:

```
fuhrpark = <fuhrpark>
               <%= From fzg In fahrzeugliste _
                   Select <fahrzeug>
                              <kennzeichen><%= fzg.Kennzeichen %></kennzeichen>
                              <hersteller><%= fzg.Hersteller %></hersteller>
                              <ladung menge=<%= fzg.Ladung.Menge %>>
                                  <%= fzg.Ladung.Gut %>
                              </ladung>
                          </fahrzeug> _
               %>
           </fuhrpark>
```

# Abfragen von XML-Dokumenten mit LINQ to XML

LINQ bietet mit *LINQ to XML* auch eine Abfrageunterstützung für XML-Strukturen an.

Ein Beispiel:

```
Dim fuhrpark = <?xml version="1.0"?>
               <fuhrpark>
                   <fahrzeug>
                       <kennzeichen>MS-VB 2008</kennzeichen>
                       <hersteller>MAN</hersteller>
                       <ladung menge="10000">Salz</ladung>
                   </fahrzeug>
                   <fahrzeug>
                       <kennzeichen>MS-C# 2008</kennzeichen>
                       <hersteller>Mercedes-Benz</hersteller>
                       <ladung menge="20000">Erdnüsse</ladung>
                   </fahrzeug>
                   <fahrzeug>
                       <kennzeichen>MS-J# 2008</kennzeichen>
                       <hersteller>DAF</hersteller>
                       <ladung menge="5000">Wackeldackel</ladung>
                   </fahrzeug>
               </fuhrpark>
Dim manFahrzeuge = From fahrzeug In fuhrpark...<fahrzeug> _
Where fahrzeug.<hersteller>.Value = "MAN"
For Each fahrzeug In manFahrzeuge
    Console.WriteLine(fahrzeug)
Next
```

Bei der Variablen `fuhrpark` handelt es sich um ein `XDocument`. In der `From`-Klausel wird auf alle XElemente mit dem Namen `Fahrzeug` zugegriffen. Die `Where`-Klausel prüft den `Hersteller` des `Fahrzeugs` auf den Eintrag »MAN«. Der Wert eines `XElements` muss über die `Value` Property erfragt werden.

Es werden zwei Zugriffmöglichkeiten für Elemente bereitgestellt:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der Zugriff auf direkte Unterelemente erfolgt über `.<Element-name>`. Zudem kann auf darunterliegende Elemente mit `…<Element-name>` zugegriffen werden, wobei sich das Element durchaus tief in den XML-Strukturen befinden darf.

Auf den Inhalt eines Attributes kann über `.@<Attribut-name>` zugegriffen werden.

Beispiel: Es sollen alle schwer beladenen Fahrzeuge ermittelt werden.

```
        Dim schwerBeladen = From fahrzeug In fuhrpark...<fahrzeug> _
                            Where fahrzeug.<ladung>.@menge > 15000
        For Each fahrzeug In schwerBeladen
            Console.WriteLine(fahrzeug)
        Next
```

ergibt folgende Ausgabe:

```
<fahrzeug>
  <kennzeichen>MS-C# 2008</kennzeichen>
  <hersteller>Mercedes-Benz</hersteller>
  <ladung menge="20000">Erdnüsse</ladung>
</fahrzeug>
```

# IntelliSense-Unterstützung für LINQ to XML-Abfragen

Bei der Arbeit mit XML-Daten ist jede Unterstützung der IDE sehr hilfreich. Visual Basic 2008 bietet genau diese an, wie in der folgenden Abbildung zu sehen:

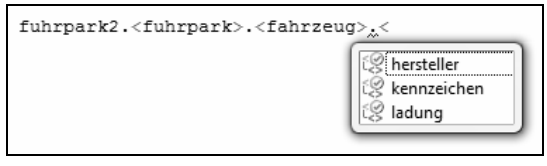



**Abbildung 10.1** XML IntelliSense-Unterstützung

Diese Unterstützung erhält man, sobald eine XML Schema Definition (XSD)-Datei mit `Imports` in die Codedatei importiert wird.

Aber fangen wir mal vorne an. Eine XSD-Datei kann Visual Studio selbst erstellen. Hierzu erstellt man zunächst die komplette XML-Datei mit allen möglichen Ausprägungen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

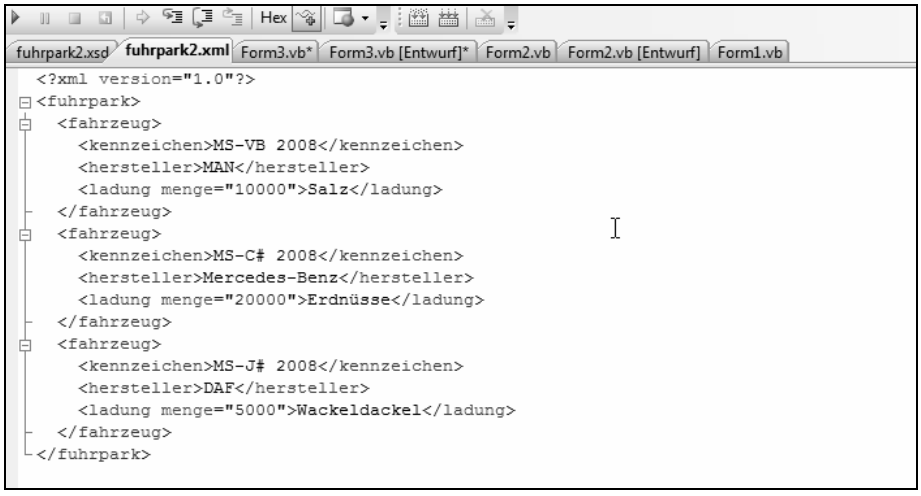



**Abbildung 10.2** Eine XML-Datei in Visual Studio

Danach kann man mithilfe des gleichlautenden Befehls im Menü *XML* ein Schema erstellen:

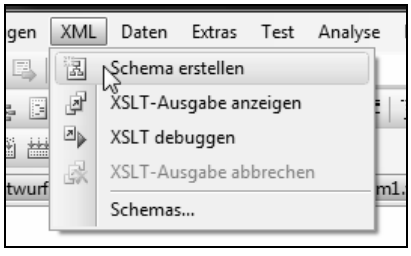



**Abbildung 10.3** Erstellen einer XML Schema Definition (XSD)

Das erstellte Schema muss nun im Projektverzeichnis gespeichert und in das Projekt mitaufgenommen werden.

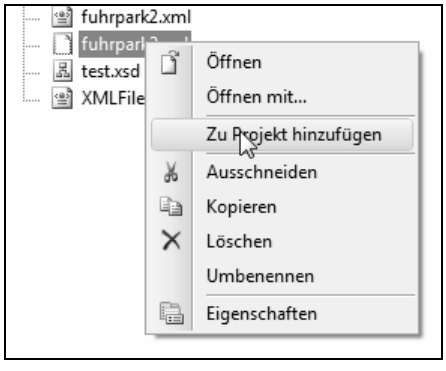



**Abbildung 10.4** Hinzufügen einer Datei im Projektmappen-Explorer

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In der soeben erstellten XSD-Datei muss nun noch ein Ziel-Namensbereich angegeben werden. Dieser Namensbereich muss nicht existieren, sondern dient lediglich der Klassifikation. In diesem Fall wurde `http://mysample.local.de/fuhrpark2` verwendet.

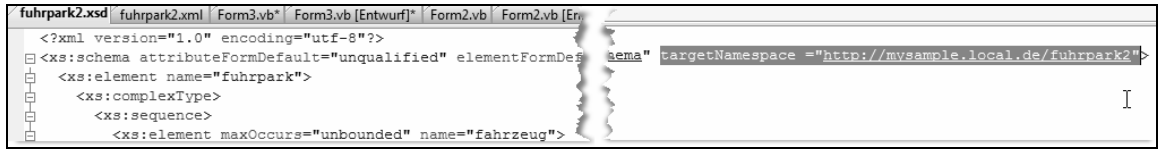



**Abbildung 10.5** Erweitern einer XSD-Datei um den Ziel-Namensbereich

In der VB Datei, die eine XML-Unterstützung erhalten soll, kann jetzt der soeben festgelegte Ziel-Namensbereich importiert werden.

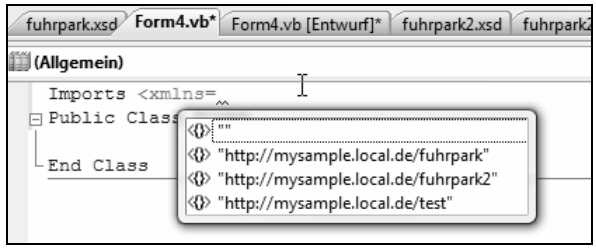



**Abbildung 10.6** Hinzufügen einer Imports Anweisung für XML Schema Dateien

Werden mehrere Namensräume verwendet, bietet es sich an, den Namensraum mit einem Präfix zu versehen. Das Präfix wird nach `xmlns` notiert und mit einem Doppelpunkt von `xmlns` getrennt. In diesem Fall wird als Präfix `fuhrpark` verwendet. Bei nur einem Namensraum, der in die VB-Codedatei importiert wird, ist das nicht erforderlich.

```
Imports <xmlns:fuhrpark="http://mysample.local.de/fuhrpark2">
Imports <xmlns:artikel="http://mysample.local.de/artikel">
```

Die XML-Strukturen müssen in diesem Fall auch um die Präfixe erweitert werden.

```
        Dim fuhrpark = <fuhrpark:fuhrpark>
                           <fuhrpark:fahrzeug>
                               <fuhrpark:kennzeichen>MS-VB 2008</fuhrpark:kennzeichen>
                               <fuhrpark:hersteller>MAN</fuhrpark:hersteller>
                               <fuhrpark:ladung menge="10000">Salz</fuhrpark:ladung>
                           </fuhrpark:fahrzeug>
                           <fuhrpark:fahrzeug>
                               <fuhrpark:kennzeichen>MS-C# 2008</fuhrpark:kennzeichen>
                               <fuhrpark:hersteller>Mercedes-Benz</fuhrpark:hersteller>
                               <fuhrpark:ladung menge="20000">Erdnüsse</fuhrpark:ladung>
                           </fuhrpark:fahrzeug>
                           <fuhrpark:fahrzeug>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
                    <fuhrpark:kennzeichen>MS-J# 2008</fuhrpark:kennzeichen>
                    <fuhrpark:hersteller>DAF</fuhrpark:hersteller>
                    <fuhrpark:ladung menge="5000">Wackeldackel</fuhrpark:ladung>
                </fuhrpark:fahrzeug>
            </fuhrpark:fuhrpark>
```

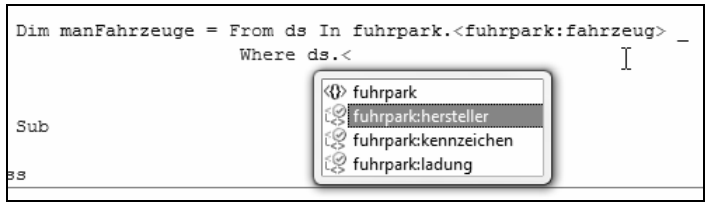



**Abbildung 10.7** XML-Unterstützung der IDE

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Teil C

# Windows Presentation Foundation (WPF)

**In diesem Teil:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 11

# WPF – Einführung und Grundlagen

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Einführung

Die Bibliotheken, die in Microsoft Windows für die Grafikausgaben und die Benutzerschnittstelle verantwortlich sind (GDI32.DLL und USER32.DLL), gibt es schon seit den Anfangszeiten von Windows. Damals waren es noch 16-Bit-Bibliotheken. Diese DLLs sind im Laufe der letzten 20 Jahre immer wieder migriert, angepasst und natürlich stark erweitert worden.

Nun stehen wieder große Erweiterungen für Grafik und Benutzerschnittstelle an. Windows Presentation Foundation (kurz: WPF, immer noch unter ihrem Codenamen *Avalon* bekannt) bietet neue Möglichkeiten, die bisher unter Windows entweder gar nicht, oder nur mit Zusatzbibliotheken zur Verfügung standen. Aus diesem Grund wurde das alte Grafik-Subsystem nicht erweitert, stattdessen wurde ein neues Subsystem implementiert, das der Anwender sowohl mit Windows XP,[1] Windows Server 2003/2008 und Windows Vista benutzen kann. In Windows XP und Windows Server 2003 müssen die neuen Bibliotheken extra installiert werden. In Windows Vista sind diese bereits enthalten und können nach der Installation des Betriebssystems sofort genutzt werden.

In diesem Kapitel wollen wir die Grundlagen von Windows Presentation Foundation besprechen. Und glauben Sie mir, es gibt viel zu berichten. Um Ihnen nur einen kleinen Vorgeschmack auf das Kommende zu geben, hier einige Stichworte: Vektorgrafiken, Fließkomma-Koordinaten, deklarative Programmierung, weitergeleitete Kommandos, weitergeleitete Eigenschaften, Abhängigkeitseigenschaften, XAML und vieles mehr.

**HINWEIS** Bevor es mit WPF losgeht, noch ein Hinweis in eigener Sache. Basis des kompletten WPF-Teils ist das Buch *WPF Crashkurs* von Bernd Marquardt.[2] Bernd hat alle Texte dieses Teils geschrieben, und von ihm stammen auch die C#-Originalbeispiele, auf denen die Visual Basic-Beispiele dieses Buchteils basieren. Auch wenn ich meinen Dank schon im Vorwort angebracht habe – ich möchte es nicht versäumen, mich noch einmal an der dafür am besten geeigneten Stelle für das Überlassen seiner Texte und seiner C#-Beispiele zu bedanken! Dank gilt an dieser Stelle auch nochmals meinem Mitarbeiter Jürgen Heckhuis, der die Beispiele von C# nach Visual Basic 2008 »übersetzt« hat.

**BEGLEITDATEIEN** Unter .*Samples**Chapter11*\ finden Sie die Beispieldateien für dieses Kapitel.

# Aller Anfang ist schwer …

Wo soll man eigentlich anfangen, wenn so vieles neu oder anders ist? Beginnen wir mit einer kurzen Beschreibung von WPF. Windows Presentation Foundation ist ein neues grafisches Subsystem für Microsoft Windows. Es bietet eine Unterstützung und eine einheitliche Sichtweise auf:

[1] Mindestens Service Pack 2 ist für Windows XP erforderlich.

[2] *Windows Presentation Foundation Crashkurs*, ISBN: 978-3-86645-504-7, erschienen bei Microsoft Press. Sie erreichen den Autor über seine Website *www.gosky.de*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

- Benutzerschnittstellen
- 2D-Grafiken
- 3D-Grafiken
- Dokumente
- Medien (Bilder, Filme,…)

Aus diesem Grund besteht WPF aus einer Reihe von Diensten, die der Softwareentwickler in Anspruch nehmen kann, um seine Softwarelösung zu bauen. Diese Dienste können folgendermaßen unterteilt werden:

- Basis-Dienste
  - XAML, Eingaben, Eigenschaften-System, Ereignisse, Zugänglichkeit
- Benutzerschnittstellen-Dienste
  - Anwendungs-Dienste, Verteilung, Steuerelemente, Layout, Datenbindung
- Medien-Dienste
  - 2D-Grafik, 3D-Grafik, Audio, Video, Texte, Fotos, Animationen, Effekte
- Dokument-Dienste
  - XPS-Dokumente, Open Packaging

Alle diese Dienste können Sie gleichzeitig und ohne Einschränkungen in einer WPF-Applikation benutzen. WPF bietet Ihnen für alle Dienste ein einheitliches Programmiermodell, eine einheitliche Schnittstelle und eine einfache Integration der Dienste in ihre Applikation.

# Architektur

In Abbildung 11.1 sehen Sie den Aufbau von WPF. Der untere Teil der Grafik zeigt den Teil des Gesamtsystems, der noch als *unmanaged* Code, also Maschinencode vorliegt. Dazu zählen in erster Linie das Betriebssystem, die Treiber, DirectX und der Desktop Window Manager (DWM). Auf das Betriebssystem setzt das .NET Framework 2.0 auf. Hierbei handelt es sich um *managed* Code. Wie Sie weiter aus der Grafik sehen können, setzt das Presentation Framework mit seinen Neuerungen auf das Betriebssystem mit DirectX und das .NET Framework 2.0 auf. Das bedeutet, dass ein Einsatz von Windows Presentation Foundation das Vorhandensein des .NET Frameworks 2.0 voraussetzt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

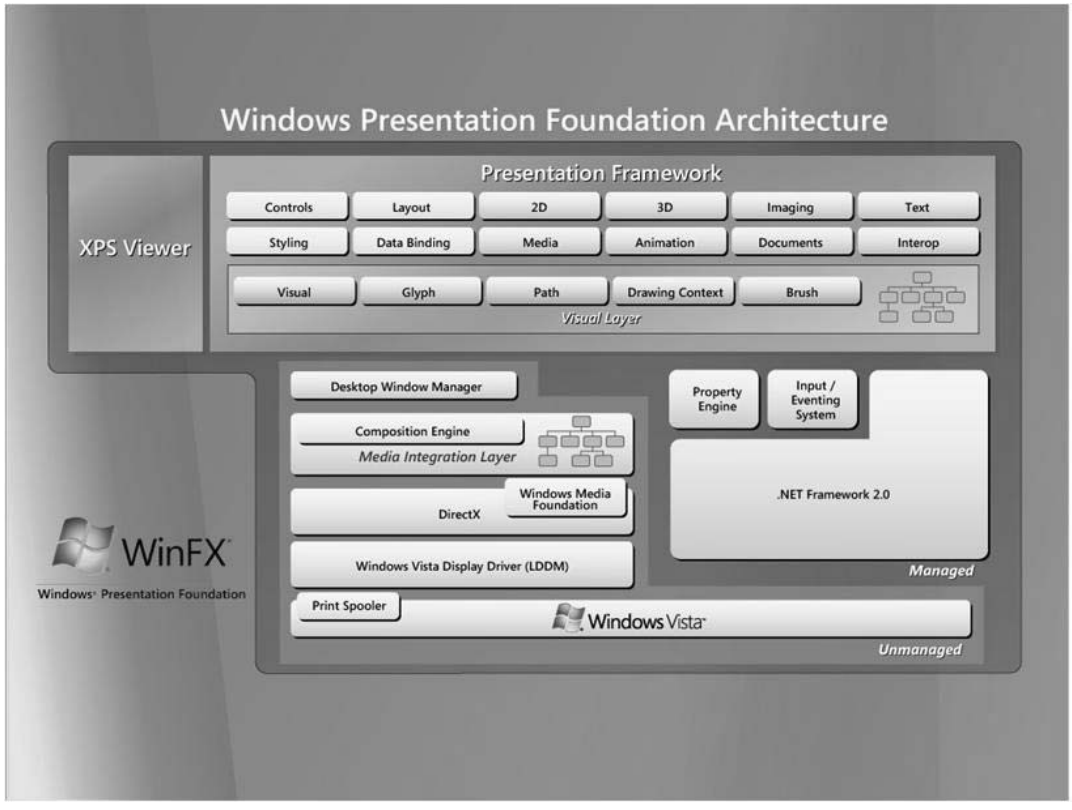



**Abbildung 11.1** Die Architektur von Windows Presentation Foundation

Alle Klassen, die im .NET Framework 2.0 vorhanden sind, können mit WPF genutzt werden. Hier müssen Sie nichts Neues lernen.

Anders dagegen im Presentation Framework selbst. Hier gibt es tausende von neuen Klassen, Strukturen, Aufzählungen und Ereignissen. In Abbildung 11.1 können Sie die einzelnen WPF-Dienste, die oben erwähnt wurden, wieder erkennen.

# Das .NET Framework ab Version 3.0

Im Zusammenhang mit Windows Presentation Foundation werden Sie oft den Namen ».NET Framework 3.0« hören. Hierbei handelt es sich um vier große Komponenten, die von Microsoft zu einem Produkt, nämlich dem .NET Framework 3.0, zusammengefasst wurden. Dazu gehören:

- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- Windows Workflow Foundation (WF)
- Windows Card Services (WCS)

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Dieses Buch behandelt als einzigen .NET 3.0-Schwerpunkt das Thema WPF – zum einen, weil es mit der vorherigen Version von Visual Studio schlicht nicht möglich war, WPF-Projekte zu entwickeln – erst mit Visual Studio 2008 steht ein WPF-Designer erstmals in einem Visual Studio-Produkt serienmäßig zur Verfügung. Zum anderen ist über das Thema WPF und Visual Basic im deutsprachigen Buchmarkt bislang noch nicht viel zu bekommen.

## WPF, .NET Framework 3.5 und Version Targeting

Wenn Sie WPF-Anwendungen entwickeln, bedeutet das nicht notwendigerweise, dass Sie in Ihren Anwendungen auf LINQ, das ja das .NET Framework 3.5 voraussetzt, verzichten müssen. Sie können natürlich auch eine neue WPF-Anwendung erstellen und diese auf dem .NET Framework 3.5 oder höheren Versionen[3] basieren lassen.

Die Version 3.0 ist allerdings die kleinste Version, auf der Ihre WPF-Anwendungen basieren müssen. Windows XP SP2 ist damit das älteste Betriebssystem, für das Sie WPF-Anwendungen entwickeln können.

# Vektorgrafik

Im Vergleich zu unseren gewohnten Windows-Forms-Anwendungen gibt es einige grundlegende konzeptionelle Neuerungen, die mit Windows Presentation Foundation eingeführt wurden. Zunächst einmal werden alle WPF-Elemente (Ausnahme: Pixel-Bilder, Bitmaps,…) als Vektorgrafik dargestellt und verarbeitet. Dies bedeutet, dass eine Schaltfläche keine Bitmap mehr ist, sondern dass diese Schaltfläche als Rechteck mit abgerundeten Ecken, einer Füllfarbe und einem Text in der Mitte an die Grafikkarte übermittelt und dort entsprechend dargestellt wird. Die Objekte »Rechteck mit Füllfarbe« und »Text« können normalerweise schneller an die Grafik-Hardware übertragen werden, als mehr oder weniger große Bitmaps. Die weitere Darstellung der Elemente übernimmt die Grafik-Hardware (GPU, Graphical Processing Unit), die mittlerweile in modernen PCs extrem leistungsfähig ist.

In einem Szenario mit vektororientierter Grafik ist es sinnvoll, alle Koordinaten als Fließkommazahlen anzugeben, da diese Grafiken nun beliebig skalierbar, also in der Größe änderbar, sind. Die Koordinatenwerte können Sie in WPF sowohl als Integerzahlen, als auch als Fließkommazahlen in einfacher und in doppelter Genauigkeit angeben.

# Trennung von Design und Logik

Wenn Sie bisher eine Benutzerschnittstelle entworfen und programmiert haben, dann wurden normalerweise Design und Logik schnell vermischt. In einer Windows Forms-Anwendung wird in einer Klasse, die von `System.Windows.Forms.Form` abgeleitet wird, in der Methode `InitializeComponent` das Aussehen eines Fensters in Form von Visual Basic .NET-Code festgelegt. Nun hat ein Grafik-Designer mit Programmcode sehr

[3] Sobald diese verfügbar werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

wenig im Sinn. Der Grafiker benutzt diverse Werkzeuge, um schöne, bunte Grafiken zu erstellen. Er wird aber wohl kaum VB-Code schreiben wollen, um das Aussehen eines Fensters oder Steuerelements zu verändern. Andererseits ist es unmöglich, den logischen Teil der Applikation, also z. B. die Datenzugriffe oder das Berechnen von Zahlen mit einem Grafikwerkzeug durchzuführen.

Auf unserer Welt gibt es leider nur wenige Programmierer, die gleichzeitig auch hervorragende Designer sind. Das kann man natürlich auch anders herum betrachten: Es gibt kaum Top-Designer, die auch noch perfekt programmieren können.

Hier stoßen also zwei Welten aneinander, für die es gilt, eine für beide Seiten akzeptable Brücke zu errichten. Auf der einen Seite befindet sich der Grafiker mit seinen Werkzeugen, die irgendetwas abliefern, das der Softwareentwickler auf der anderen Seite mit der Applikationslogik »unterfüttern« kann. Dabei sollte sich der Grafiker so wenig wie möglich mit Programmierung auseinandersetzen müssen, und der Entwickler benötigt kaum eine Ahnung über die eingesetzten Designerwerkzeuge.

Ziel ist letztendlich, dass ein Top-Designer zusammen mit einem Top-Softwareentwickler eine Top-Applikation erstellt!

1. Beginnen wir im alten Stil und schreiben eine WPF-Applikation nur mit Visual Basic .NET-Code. Wir führen zunächst einmal keine Trennung von Design und Code ein. Es handelt sich natürlich um eine weitere Version des bekannten *HalloWelt*-Programms. Die Vorgehensweise ist folgende:
2. Starten Sie Visual Studio 2008 und wählen Sie *Datei* > *Neu* > *Projekt* aus.
3. Im Dialogfeld *Neues Projekt* wählen Sie als Programmiersprache Visual Basic aus. Darunter wählen Sie ein leeres »Windows«-Projekt aus.
4. Als Referenzen fügen Sie Verweise auf folgende Bibliotheken ein: `System`, `WindowBase`, `PresentationFramework` und `PresentationCore`. Benutzen Sie hierzu im Projektmappen-Explorer durch Anklicken mit der rechten Maustaste den Befehl *Verweis hinzufügen*. Wählen Sie die angegebenen Referenzen im Dialogfeld aus.
5. Fügen Sie nun im Projektmappen-Explorer eine neue VB-Code-Datei mit dem Namen »Hallo.vb« hinzu. Benutzen Sie wieder die rechte Maustaste mit dem Befehl *Hinzufügen* > *Neues Element*.
6. Tippen Sie nun den Code aus Listing 11.1 ein.
7. In den Projekt-Eigenschaften stellen Sie auf der Seite *Anwendung* den Ausgabetyp *Windows-Application* ein.
8. Führen Sie das Programm aus, indem Sie aus dem Menü *Debuggen* den Befehl *Starten ohne Debugging* aufrufen.

```
Imports System
Imports System.Windows

Namespace Hallo

    Public Class Hallo
        Inherits System.Windows.Window
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        <STAThread()> _
        Shared Sub main()
            Dim w As New Window With {.Title = "Hallo, Welt!", .Width = "200", .Height = "200"}
            w.Show()

            Dim app As New Application
            app.Run()
        End Sub

    End Class
End Namespace
```

**Listing 11.1** Das erste WPF-Programm nur in Visual Basic

Wenn wir das Programm aus Listing 11.1 laufen lassen, erscheint ein ziemlich eintöniges Fenster (Listing 11.2) mit dem Titel »Hallo, Welt!«.

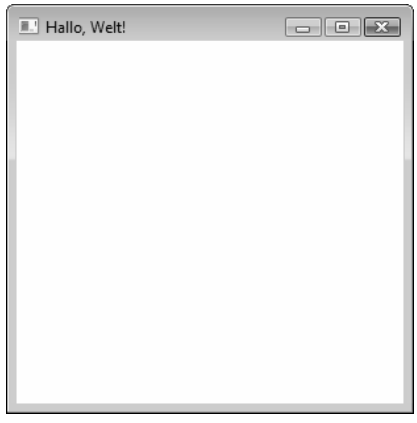



**Abbildung 11.2** Das erste WPF-Fenster

Schauen wir uns den Code des Beispiels nun etwas genauer an. Nach dem Einfügen der benötigten Namensräume aus den Referenzen definieren wir einen eigenen Namensraum mit der Klasse »Hallo«. In WPF muss vor der statischen `Main`-Methode das Attribut `[STAThread]` angewendet werden, um das Threading-Modell für die Applikation auf »Single-Threaded Appartment« zu setzen. Dies ist ein Relikt aus alten COM-Zeiten, aber es sorgt ggf. für eine Kompatibilität in die Welt des *Component Object Model*.

In der `Main`-Methode erzeugen wir zunächst ein `Window`-Objekt, welches über die Eigenschaft `Title` den Text für die Titelzeile erhält. Die Methode `Show` aus der `Window`-Klasse sorgt für die Darstellung des Fensters auf dem Bildschirm. Wenn wir die beiden letzten Code-Zeilen der `Main`-Methode nicht eingeben und unser Programm starten, dann werden wir das Fenster nur sehr, sehr kurz sehen. Die Applikation wird nämlich sofort wieder beendet. Hier kommen nun diese beiden letzten Zeilen ins Spiel. Im erzeugten `Application`-Objekt wird die Methode `Run` aufgerufen, um für dieses Hauptfenster eine Meldungsschleife (Message Loop) zu erzeugen. Nun kann unser kleines Programm Meldungen empfangen. Das Fenster bleibt solange sichtbar, bis diese Meldungsschleife beendet wird, z. B. durch Anklicken der Schließ-Schaltfläche oben rechts in der Titelzeile des Fensters.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Nun haben wir in diesem Beispiel allerdings den Designer arbeitslos gemacht, denn wir haben als Softwareentwickler das Design im VB-Code implementiert. Eigentlich besteht das Design dieser Applikation nur aus einer Zeile:

```
Dim w As New Window With {.Title = "Hallo, Welt!", .Width = "200", .Height = "200"}
```

Alles andere möchte ich in diesem einfachen Beispiel einmal als Applikationslogik bezeichnen. Da ein Designer jedoch nicht mit VB-Code oder anderen Programmiersprachen arbeiten will, muss eine andere »Sprache« her, mit der man Benutzeroberflächen und Grafiken »deklarieren« kann.

In Windows Presentation Foundation wird hierzu die Extensible Application Markup Languge (XAML, sprich: »gsämmel«) benutzt.

## XAML: Extensible Application Markup Language

XAML ist, wie XML, eine hierarchisch orientierte Beschreibungssprache. Wir können mit XAML Benutzerschnittstellen oder Grafiken deklarieren. Kommen wir wieder zu unserem ersten Beispiel (Abbildung 11.2) zurück. Als nächstes deklarieren wir nun den Designer-Teil mit XAML. Vorab jedoch noch ein kleiner Hinweis: Ein Designer wird natürlich den XAML-Code nicht »von Hand« eintippen, so wie wir es nun im nächsten Beispiel machen. Er wird stattdessen Werkzeuge verwenden, welche das erstellte Design schließlich als XAML-Code zur Verfügung stellen. In Listing 11.2 können Sie nun den erforderlichen XAML-Code für unser erstes Beispielprogramm sehen.

```
<Window x:Class="Hallo2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hallo, Welt!"
    Height="250"
    Width="250"
       >
  </Window>
```

**Listing 11.2** Unser Fenster in XAML

Was wir in Listing 11.2 sehen, ist nicht gerade überwältigend! Diese wenigen Zeilen XAML-Code machen im Grunde genommen nichts anderes, als ein `Window`-Objekt zu deklarieren, den Titel des Fensters auf »Hallo, Welt!« zu setzen und die Größe des Fensters auf 250 mal 250 Einheiten zu definieren. Hierzu werden die Eigenschaften `Title`, `Width` und `Height` auf die gewünschten Werte gesetzt.

Alle Eigenschaften in XAML werden mit Texten gefüllt. Darum müssen die Werte in Anführungszeichen gesetzt werden. Um XAML-Code zu nutzen, sollten Sie zwei Namensräume deklarieren:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Visual Studio 2008 fügt diese Namensräume automatisch in Ihre .NET Framework 3.0-Projekte in die XAML-Dateien ein.

Dieses Projekt wurde mit Visual Studio angelegt, und darum übernimmt Visual Studio auch den Rest der Arbeit. Im Hintergrund wird nämlich eine weitere Datei erzeugt, die VB-Code enthält und unter anderem ein `Application`-Objekt erstellt und die dazugehörige `Run`-Methode aufruft. Dieser, von Visual Studio erzeugte Code, soll uns aber gar nicht weiter interessieren. Wenn wir das Programm starten, werden wir wieder das gleiche Fenster wie im ersten Beispiel sehen.

Für das zweite Beispiel ist die Vorgehensweise mit Visual Studio 2008 folgende:

1. Starten Sie Visual Studio 2008 und wählen Sie *Datei* > *Neu* > *Projekt* im Datei-Menü aus.
2. Im Dialogfeld *Neues Projekt* wählen Sie als Programmiersprache Visual Basic aus. Darunter wählen Sie ein *WPF-Anwendung*-Projekt aus.
3. Öffnen Sie nun im Projektmappen-Explorer die XAML-Datei mit dem Namen *Window1.xaml*.
4. Tippen Sie nun den Code aus Listing 11.2 ein. Ändern Sie den Code, welchen Visual Studio erzeugt hat, einfach ab.
5. Führen Sie das Programm aus, indem Sie aus dem Menü *Debuggen* den Befehl *Starten ohne Debugging* aufrufen.

Nun haben wir die Deklaration der Benutzerschnittstelle in XAML hinterlegt. Dieser XAML-Code kann durch Grafikwerkzeuge erzeugt werden.

Jetzt aber wieder zurück zum Programmieren. Wir kommen nun auf die Applikationslogik zurück. Im nächsten Beispiel wollen wir eine minimale Logik implementieren. Für die Benutzerschnittstelle verwenden wir wiederum XAML, für die Logik wird Visual Basic eingesetzt.

Das Beispiel soll zeigen, wie eine Benutzerschnittstelle und die Applikationslogik in einer WPF-Applikation zusammenarbeiten. Dazu deklarieren wir mitten im Fenster eine Schaltfläche in einer bestimmten Größe (Listing 11.3). Das ist unsere Benutzerschnittstelle, die von einem Designer erstellt wurde.

```
<Window x:Class="Hallo3.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Hallo 3" Height="200" Width="200">
   <Grid>
       <Button Click="OnClick" Width="120" Height="40" Margin="28,58,21,64"
              >Bitte anklicken!</Button>
   </Grid>
</Window>>
```

**Listing 11.3** Die Benutzerschnittstelle in XAML

Innerhalb der Deklaration für das Hauptfenster wird ein weiteres WPF-Element definiert: die Schaltfläche. In XAML heißt dieses Element `Button`. Eine Schaltfläche hat natürlich Eigenschaften, die Sie setzen können. Hier wird die Breite (`Width`) der Schaltfläche mit 120 Einheiten angegeben und die Höhe (`Height`) mit 40 Einheiten. Der Text auf der Schaltfläche lautet: »Bitte anklicken!«. Wenn wir den XAML-Code eingeben und das Programm starten, passiert natürlich beim Klicken auf die Schaltfläche zunächst einmal gar nichts.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wenn der Anwender des Programms auf die Schaltfläche klickt, dann soll jedoch ein Text ausgegeben werden. Sie ahnen, welcher Text? Natürlich »Hallo, Welt!«. Das ist die Applikationslogik, welche ein Softwareentwickler erstellt (Listing 11.4). Die Verbindung zwischen den `Button`-Element und dem VB-Code wird über ein Ereignis hergestellt. In XAML (Listing 11.3) wird im `Button`-Element das `Click`-Ereignis benutzt, welches auf die Ereignismethode `OnClick` zeigt. Diese Methode wird nun in Visual Basic implementiert (Listing 11.4).

Dieses Projekt wurde ebenfalls mit Visual Studio 2008 erzeugt. Zunächst einmal werden nur die Namensräume `System` und `System.Windows` benötigt. Visual Studio erzeugt fast den gesamten Code aus Listing 11.4, wir müssen nur die Methode `OnClick` (im unteren Bereich) eingeben.

```
Imports System
Imports System.Windows

Namespace Hallo3
    Partial Public Class Window1
        Inherits System.Windows.Window

        ''' <summary>
        ''' Logik für dieses Beispiel
        ''' </summary>

        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnClick(ByVal sender As System.Object, ByVal e As _
                            System.Windows.RoutedEventArgs)
            MessageBox.Show("Hallo, Welt!")
        End Sub
    End Class
End Namespace
```

**Listing 11.4** Die Applikationslogik in Visual Basic

Im Konstruktor der Fensterklasse wird die Methode `InitializeComponent` aufgerufen. Auch der Code dieser Methode wird in einer separaten VB-Codedatei von Visual Studio generiert. Der VB-Befehl, der hier zum Einsatz kommt, um die verschiedenen Dateien für den Compiler korrekt zusammenzuführen, heißt `partial`. Der gesamte Code der Klasse kann auf mehrere VB-Dateien verteilt werden: Ein Teil der Klasse ist der Code aus Listing 11.4 und der andere, unsichtbare Teil ist der von Visual Studio erzeugte Zusatzcode, der die Methode `InitializeComponent` und eine Art Abbildung des XAML-Codes aus Listing 11.3 in Visual Basic enthält. Beim Übersetzen werden beide Dateien zusammengefügt. Aus diesem Grund erhalten Sie auch eine Fehlermeldung vom Compiler, wenn Sie das Beispiel übersetzen, ohne vorher die Methode `OnClick` einzugeben.

Die Benutzerschnittstelle, die in XAML definiert wurde, wird übrigens nicht von einem XML-Interpreter abgearbeitet, also interpretiert. Das wäre sicherlich zu langsam. Auch XAML wird kompiliert. Letztendlich entsteht daraus ganz normaler MSIL-Code, wie wir ihn aus jedem .NET-Programm gewohnt sind. Dieser »Zwischen-Code«, der in der EXE-Datei steht, wird dann vom JIT-Compiler (Just-In-Time) zur Laufzeit des Programms in die Maschinensprache der jeweiligen Zielplattform übersetzt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In die Ereignismethode (Listing 11.4) werden zwei Parameter übergeben: Von Typ `object` bekommen wir die Variable `sender`, die uns angibt, welches Objekt das Ereignis ausgelöst hat. Der Parameter `e` vom Typ `EventArgs` gibt uns zusätzliche Daten an, die zu dem jeweiligen Ereignis gehören. Diese Art der Ereignisprogrammierung kennen wir schon aus Windows Forms. In der Methode selbst wird dann einfach die Methode `MessageBox.Show` mit dem gewünschten Text aufgerufen.

Das Ergebnis unserer Bemühungen zeigt Abbildung 11.3.

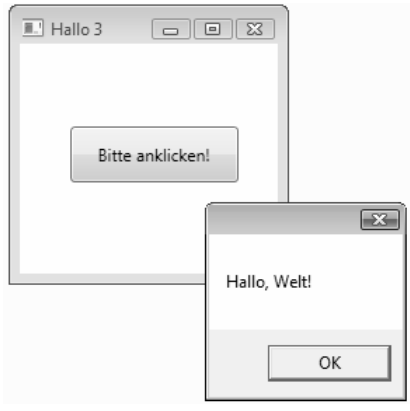



**Abbildung 11.3** XAML und Logik werden ausgeführt

Wir sind aber noch nicht ganz am Ziel unserer Wünsche! Das ganze Programm nützt uns nichts, wenn wir die Objekte, die wir in XAML deklariert haben, nicht aus unserem Applikationscode manipulieren können. Die Schaltfläche »Bitte anklicken!« ist ein Element vom Typ `Button` und enthält diverse Eigenschaften und Methoden. Wie kann man diese nun aus dem VB-Code aufrufen?

Hierzu wollen wir das letzte Beispiel wieder ein bisschen ändern. Zunächst muss die Schaltfläche einen Namen bekommen. Dies erledigen wir durch Setzen der `Name`-Eigenschaft (Listing 11.5). Es handelt sich hierbei um die einzige Änderung am XAML-Code.

```
<Window x:Class="Hallo4.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hallo 4" Height="200" Width="200">
        <Button Name="btn" Click="OnClick" Width="120" Height="40"
                Margin="34,61,24,61">Bitte anklicken!</Button>
</Window>
```

**Listing 11.5** Die Schaltfläche bekommt einen Namen

Da die Schaltfläche nun einen Namen (`btn`) hat, können wir sie ganz normal aus dem VB-Code heraus ansprechen und die Eigenschaften neu setzen oder Methoden des Objektes `btn` aufrufen. Im Beispiel (Listing 11.6) ändern wir mit der Eigenschaft `Content` zunächst den Text, der auf der Schaltfläche ausgegeben wird. Schließlich werden die Fontgröße (`FontSize`) und die Vordergrundfarbe (`Foreground`) neu gesetzt. In allen Fällen wird vor der jeweiligen Eigenschaft der Name der Schaltfläche, der in XAML definiert wurde, angegeben.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Imports System
Imports System.Windows

Namespace Hallo4
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnClick(ByVal sender As System.Object, _
                            ByVal e As System.Windows.RoutedEventArgs)
            MessageBox.Show("Hallo Welt!")

            ' Nach der Messagebox: Schaltfläche ändern
            With btn
                .Content = "Guten Tag!"
                .FontSize = 20
                .Foreground = Brushes.Red
            End With

        End Sub
    End Class
End Namespace
```

**Listing 11.6** Die erweiterte Applikationslogik

Abbildung 11.4 zeigt das Fenster nach dem Ausführen der Ereignismethode. Die Eigenschaften der Schaltfläche wurden entsprechend geändert, nachdem im `MessageBox`-Element die *OK*-Schaltfläche angeklickt wurde.


**Abbildung 11.4** Darstellung des Fensters nach dem MessageBox.Show-Aufruf

Nun können wir »beide Richtungen« zwischen XAML und Code benutzen. Um aus XAML die Applikationslogik »aufzurufen«, verwenden wir Ereignisse. Diese stehen uns in den vielen WPF-Elementen in großer Anzahl zur Verfügung. Wir implementieren Ereignismethoden, welche die Applikationslogik enthalten. Um umgekehrt die in XAML deklarierten WPF-Elemente zur Laufzeit zu verändern, benutzen wir das ganz normale Objektmodell dieser Elemente und rufen die Eigenschaften und Methoden aus dem Programmcode auf. Die einzelnen Objekte werden durch ihre Namen identifiziert. Die in XAML deklarierten Eigenschaftswerte sind also die Initialeinstellungen bei der Darstellung der WPF-Elemente.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wie eben bereits erwähnt wurde, stellt die XAML-Deklaration der Benutzerschnittstelle den Startzustand der Anwendung dar. Sie können allerdings, wenn erforderlich, die Startwerte sofort aus dem Programmcode ändern, indem Sie eine Methode für das Ereignis `Loaded` implementieren. Dies wird in einem Beispiel in Listing 11.7 (XAML-Teil) und Listing 11.8 (VB-Teil) gezeigt.

**HINWEIS** Aus diesem Beispiel wurde der nicht benötigte VB-Code, der von den Visual Studio-Erweiterungen erzeugt wurde, entfernt, um das Listing etwas kürzer zu halten.

Für das Hauptfenster mit dem Klassennamen `Window1` wird das Ereignis `Loaded` an die Methode `OnLoaded` gebunden. Der Code, der in der entsprechenden VB-Methode implementiert ist, wird nun direkt nach dem Konstruktoraufruf von `Window1` ausgeführt. Sobald das Fenster auf dem Bildschirm erscheint, enthält es bereits die vergrößerte Schaltfläche, da die beiden Eigenschaften `Width` und `Height` in der Methode `OnLoaded` entsprechend gesetzt wurden. Wird die Schaltfläche danach angeklickt, so wird die Ereignismethode `OnClick` aufgerufen und ausgeführt.

```
<Window x:Class="Hallo5.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hallo 5" Height="300" Width="300"
    Loaded="OnLoaded">
       <Button Name="btn" Click="OnClick" Width="120" Height="40"
               Margin="34,61,24,61">Bitte anklicken!</Button>
</Window>
```

**Listing 11.7** Fenster mit einem Loaded-Ereignis

```
Imports System
Imports System.Windows

Namespace Hallo5
    Partial Public Class Window1
        Inherits System.Windows.Window

        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnClick(ByVal sender As System.Object, _
                            ByVal e As System.Windows.RoutedEventArgs)
            MessageBox.Show("Hallo Welt!")

            ' Nach der Messagebox: Schaltfläche ändern
            With btn
                .Content = "Guten Tag!"
                .FontSize = 20
                .Foreground = Brushes.Red
            End With
        End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private Sub OnLoaded(ByVal sender As System.Object, _
                             ByVal e As System.Windows.RoutedEventArgs)
            btn.Height = 80
            btn.Width = 150
        End Sub
    End Class
End Namespace
```

**Listing 11.8** Der Code in OnLoaded wird zuerst ausgeführt


**Abbildung 11.5** Nach der Ausführung der OnLoaded-Methode

**WICHTIG** Grundsätzlich können wir sagen, dass alles, was in XAML angegeben und deklariert werden kann, auch in VB.NET mithilfe von Programmcode erzeugt werden kann. Umgekehrt gilt das allerdings nicht!

Wir können nun das Design der letzten Beispielanwendung ändern, ohne den Applikationscode zu modifizieren. Dazu wollen wir etwas ziemlich Verrücktes machen: Die Schaltfläche soll schräg im Fenster stehen und wir benötigen dazu eine Transformation, die wir in Kapitel 14 noch ausführlich behandeln werden. Benutzen Sie den entsprechenden XAML-Code mit der Transformation im Moment einfach so, wie in Listing 11.9 aufgeführt.

```
<Window x:Class="Hallo5a.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hallo 5a" Height="300" Width="300"
    Loaded="Window_Loaded">
        <Button Name="btn" Click="OnClick" Width="120" Height="40" Margin="34,61,24,61">
            Bitte anklicken!
            <Button.RenderTransform>
                <SkewTransform AngleX="10" />
            </Button.RenderTransform>
        </Button>
</Window>
```

**Listing 11.9** Eine schräge Schaltfläche

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Logik, die in Visual Basic implementiert wurde (Ereignismethoden `OnLoaded`, `OnClick`), bleibt unverändert und wird für dieses Beispiel gar nicht mehr aufgelistet. Wir benutzen weiterhin den Code aus Listing 11.8. Wenn Sie das Programm starten, sehen Sie zunächst die schräge Schaltfläche (Abbildung 11.6 links); die Funktionalität hinter dieser Schaltfläche ist jedoch unverändert geblieben. Ein Anklicken löst die `MessageBox.Show`-Methode aus und danach werden die Eigenschaften der nun schräg liegenden Schaltfläche wie bisher geändert (Abbildung 11.6 rechts).


**Abbildung 11.6**
Eine Applikation mit geändertem Design

Wir haben nun also eine Trennung zwischen dem Design der Applikation und der Logik erreicht. Der Softwareentwickler kann die Logik der Applikation ändern, ohne das Design (versehentlich) zu modifizieren. Umgekehrt kann ein Designer das Aussehen der Anwendung ändern, ohne dass der Programmcode ihm dabei ständig »in die Quere kommt«.

**WICHTIG** Wir können also sagen: XAML + Code = Anwendung.

# Der WPF-Designer

Natürlich müssten Sie eine Anwendung nicht unbedingt komplett codieren – bislang haben wir ja sowohl Programmlogik als auch den XAML-Designpart in guter alter Entwicklermanier mit wenn auch verschiedenen text-basierten Editoren erstellt.

Zumindest beim Design-Part war das in Windows Forms anders. Wann immer es galt, Windows Forms-Anwendungen zu gestalten, war der WinForms-Designer von der Partie – sollte das nicht in WPF genau so sein? Sollte man meinen. Tatsache ist, dass der WPF-Designer einen WPF-Entwickler sicherlich in die Lage versetzt, rudimentärste WPF-Formulare zu erstellen. Wenn es aber darum geht, komplexere Objektverschachtelungen aus designtechnischer Sicht zu bauen, sind die Grenzen schnell erreicht. Den Komfort eines WinForms-Designers – beispielsweise beim interaktiven Erstellen von Werkzeugleisten oder Dropdown-Menüs – werden Sie beim derzeitigen Stand des WPF-Designers vergeblich suchen. Und von einer Unterstützung der viel weiter reichenden Fähigkeiten von WPF (Animationen, 3D) kann aus designertechnischer Sicht schon gar keine Rede sein.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Aus diesem Grund und der Tatsache, dass es aus didaktischer Sicht sicherlich besser ist, trittsicher in XAML zu werden, lassen wir den WPF-Designer in den folgenden Abschnitten und Kapiteln weitestgehend außen vor und bedienen uns nur seiner Split-View-Fähigkeit.

**HINWEIS** In einigen Fällen ist es notwendig, dass der Designer aufgrund von Codeänderungen seinen Inhalt neu laden muss – eine entsprechende Warnmeldung finden Sie dann am oberen Rand des Design-Fensters. Ein einfacher Mausklick genügt dann, um den Inhalt des Designfensters an die aktualisierten Gegebenheiten anzupassen.

## Ereignisbehandlungsroutinen in WPF und Visual Basic

Im Zusammenhang mit dem Designer möchte ich auf das »Verdrahten« von Ereignisbehandlungsroutinen in WPF-Anwendungen ein kleines bisschen näher eingehen.

In Windows-Forms-Anwendungen sind Visual Basic-Entwickler es gewohnt, Ereignisprozeduren per Doppelklick auf das entsprechende Steuerelement zu verdrahten. Das geht in WPF-Anwendungen mit dem Designer prinzipiell ebenfalls. In beiden Fällen fügt Visual Basic dabei den Rumpf der Ereignisroutine mit einem geeigneten Namen ein, die automatisch die korrekte Signatur für das Behandeln des Ereignisses erhält. Das `Handles`-Schlüsselwort zeigt dem Visual Basic-Compiler an, dass hier die Infrastruktur für das Hinzufügen der Ereignisdelegatenliste in die `Form`- (bzw. – für WPF – `Window`-) Klasse eingefügt werden muss.

Bei WPF-Anwendungen gibt es in Ergänzung dazu auch eine Möglichkeit, die Sie auch bei der Entwicklung in jedem Fall vorziehen sollten, nämlich im XAML-Code selbst zu bestimmen, welche Ereignisprozedur für ein Ereignis eines bestimmten Objekts aufgerufen werden soll. Das funktioniert ähnlich einfach, wie das Doppelklicken auf ein Steuerelement, wie die nachstehende Abbildung zeigt:

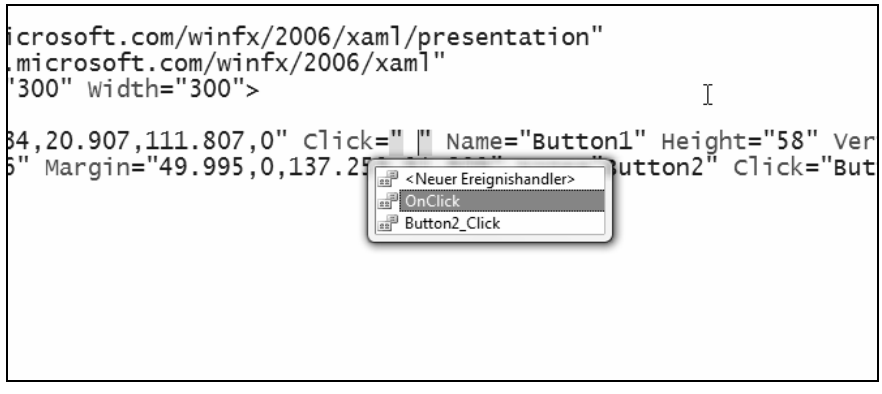



**Abbildung 11.7** Ereignisprozeduren lassen sich in WPF auch in Visual Basic-Programmen, anders als Sie es von Windows-Forms-Anwendungen gewohnt sind, auch per XAML-Code »verdrahten«

Sie können den Namen des Ereignisses direkt in die XAML-Elementdefinition hineinschreiben, und IntelliSense hilft Ihnen anschließend, das Ereignis entweder mit einer bereits vorhandenen Methode (die natürlich über die entsprechende Signatur des Ereignisses verfügen muss) zu verknüpfen, oder den Rumpf einer neuen Ereignisbehandlungsroutine einzufügen. In letztem Fall würden Sie in der IntelliSense-Objekteliste einfach den ersten Eintrag *<Neuer Ereignishandler>* auswählen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Logischer und visueller Baum

In Windows Presentation Foundation gibt es zwei Hierarchien für die WPF-Elemente, die von großer Wichtigkeit sind. Beginnen wir mit dem logischen Baum (*Logical Tree*). Der logische Baum bildet den Zusammenhang zwischen den Objekten ab. Diese Hierarchie ist entscheidend für die Vererbung von Eigenschaften zwischen den Elementen. Innerhalb dieser Hierarchie gibt es verschiedene Methoden, um durch den Baum zu navigieren:

- `GetParent`
- `GetChildren`
- `FindLogicalNode`

Die drei genannten statischen Methoden finden Sie in der Klasse `LogicalTreeHelper`.

```
<Window x:Class="LogicalTree.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="LogicalTree" Height="300" Width="300"
    >
    <Grid>
      <Button Name="btn" Width="200" Height="30" Click="OnClick">Hallo</Button>
    </Grid>
</Window>
```

**Listing 11.10** Eine Beispiel-Hierarchie

Die logische Hierarchie für den XAML-Code in Listing 11.10 ist einfach:

- `Window1`-Element
- `Grid`-Element
- `Button`-Element

Diese Hierarchie können wir mit den Methoden `GetParent` und `GetChildren` durchlaufen. Die Methoden geben Objekte vom Typ `DependencyObject` zurück, wie das folgende Listing 11.11 zeigt:

```
Imports System
Imports System.Windows
Imports System.Windows.Controls

Namespace LogicalTree
    Partial Public Class Window1
        Inherits System.Windows.Window
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub onClick(ByVal sender As System.Object, _
                            ByVal e As System.Windows.RoutedEventArgs)

            ' Richtung Wurzel-Element das Eltern-Element
            Dim Grid As DependencyObject
            Grid = LogicalTreeHelper.GetParent(CType(sender, FrameworkElement))
            MessageBox.Show(Grid.GetType().ToString)

            ' Richtung Wurzel-Element das nächste Eltern-Element
            Dim Window As DependencyObject
            Window = LogicalTreeHelper.GetParent(Grid)
            MessageBox.Show(Window.GetType().ToString)

            ' Alle Kindelelemente des Grids
            Dim obj As DependencyObject
            For Each obj In LogicalTreeHelper.GetChildren(Grid)
                MessageBox.Show(obj.GetType().ToString)
            Next

            ' Ein bestimmtes Element suchen
            Dim depobj As DependencyObject
            depobj = LogicalTreeHelper.FindLogicalNode(Window, "btn")

            Dim btn As New Button
            btn = CType(depobj, Button)
            btn.Content = "WPF"
        End Sub
    End Class
End Namespace
```

**Listing 11.11** Abfragen der Hierarchie aus Listing 11.10

In Listing 11.11 können Sie sehen, wie die Hierarchie abgearbeitet werden kann. In diesem einfachen Fall geben wir die Typen aus der Hierarchie nur in Meldungsfenstern aus. Im zweiten Teil des VB-Codes wird ermittelt, welche Kindelemente es im `Grid`-Element gibt. Hier wird nur ein Element, nämlich die Schaltfläche, gefunden. Im dritten Teil des Beispiels wird ein bestimmtes Element mit der Methode `LogicalTreeHelper.FindLogicalNode` aufgesucht. Das gefundene Objekt wird in ein `Button`-Element konvertiert. Dann können wir das Objekt benutzen und verschiedene Eigenschaften ändern.

Die zweite wichtige Hierarchie, die visuelle Hierarchie (Visual Tree) entscheidet darüber, wie die einzelnen Elemente »gerendert«, also dargestellt werden. Eine Schaltfläche ist nicht einfach nur eine Bitmap, die in der Grafikkarte dargestellt wird, sondern besteht aus einem Rechteck mit abgerundeten Ecken, einem Hintergrund und einem Inhalt (der wiederum beliebig kompliziert gestaltet sein kann). Die visuelle Hierarchie entscheidet also über die Darstellung der einzelnen Teile eines gesamten Elements. Hier wird auch die Reihenfolge in der Z-Richtung berücksichtigt, d.h., wie die Elemente übereinander liegen. Weiterhin spielt die visuelle Hierarchie für das Hit-Testing und die Transformationen der WPF-Elemente eine große Rolle.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Die XAML-Syntax im Überblick

Dieser Abschnitt gibt Ihnen einen kurzen Überblick über die XAML-Syntax. Der XAML-Code soll auch mit normalem VB-Code verglichen werden. Viele Leser werden sicherlich schon Erfahrung mit XML gesammelt haben. Trotzdem soll hier mit einigen Beispielen die Syntax von XAML erläutert werden.

Eine XAML-Hierarchie beginnt immer mit einem Wurzelelement, einem `Window`- oder `Page`-Element. Dort werden die benötigten XML-Namensräume definiert.

Nun wird die Hierarchie der Benutzerschnittstelle deklariert. Alle Elemente, die in XAML benutzt werden können, existieren als normale Klassen im .NET Framework 3.0. Als Programmierer wissen wir, dass Klassen unter anderem Eigenschaften, Methoden und Ereignisse enthalten. Wir wollen nun betrachten, wie sich das in XAML verhält.

```
<Button Name="btnClear" Width="80" Height="25" Click="OnClear">Löschen</Button>
```

In der obigen XAML-Zeile wird ein Element vom Typ `Button` mit dem Namen `btnClear` deklariert. Zur Laufzeit wird also ein Objekt vom Typ `Button` erzeugt. Die Eigenschaften `Width` und `Height` werden gesetzt und außerdem wird das `Click`-Ereignis mit der Ereignismethode `OnClear` verbunden. Der Text auf der Schaltfläche wird über die Default-Eigenschaft des `Button`-Elements gesetzt. Das Gleiche könnten Sie mit folgendem VB-Code erreichen:

```
Dim btnClear As New Button
btnClear.Width = 80
btnClear.Height = 25
btnClear.Content = "Löschen"
AddHandler btnClear.Click, AddressOf OnClear
Me.Content = btnClear
```

Auf Grund des Inhaltsmodells von WPF müssen Sie in der letzten VB-Zeile die Schaltfläche als Inhalt in das Elternelement einbringen. In XAML wird das einfach durch die Deklarationshierarchie erledigt:

```
<Window ...
  Width="300" Height="300">
  <!-- Die folgende Schaltfläche ist der Inhalt des Fensters -->
  <Button Click="OnClear">Test</Button>
</Window>
```

Die Ereignismethode `OnClear` wird im VB-Code implementiert und hat normalerweise folgende Kopfzeile:

```
        Public Sub OnClear(ByVal sender As System.Object, _
                           ByVal e As System.Windows.RoutedEventArgs)
```

Angehängte Eigenschaften (*attached properties*) werden Sie dann benutzen, wenn Sie auf Eigenschaften des Elternelements zugreifen müssen:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
<Grid>
...
  <Button Grid.Row="0" Grid.Column="0">Button 1</Button>
  <Button Grid.Row="1" Grid.Column="0">Button 2</Button>
</Grid>
```

In den obigen Zeilen wird von den `Button`-Deklarationen aus auf die Eigenschaften `Row` und `Column` des äußeren `Grid`-Elements zugegriffen.

Oftmals werden Sie in XAML Elemente deklarieren, ohne deren Standorteigenschaft zu benutzen. In diesem Fall können Sie eine gekürzte Schreibweise benutzen. Statt

```
<TextBox Name="text" Width="100"></TextBox>
```

können Sie auch schreiben

```
<TextBox Name"text" Width"100" />
```

Häufig müssen Sie einer Eigenschaft nicht einfach nur eine Zahl oder einen Text, sondern ein komplexes Element, welches wiederum eigene Eigenschaften besitzt, zuweisen. Als Beispiel wollen wird der Eigenschaft `RenderTransform` einer Schaltfläche (`Button`) ein Element vom Typ `RotateTransform` zuweisen, für welches die Eigenschaften `Angle`, `CenterX` und `CenterY` gesetzt werden sollen:

```
<Button Name="btnClear" Width="80" Height="25" Click="OnClear">
  Löschen
  <Button.RenderTransform>
    <RotateTransform Angle="25" CenterX="40" CenterY="12.5" />
  </Button.RenderTransform>
</Button>
```

Zunächst werden für die Schaltfläche selbst einige Eigenschaften »direkt« gesetzt (`Name`, `Width`,…). In der `Button`-Klasse gibt es die Eigenschaft `RenderTransform`, der nun ein `RotateTransform`-Objekt zugewiesen werden soll. Darum wird innerhalb der Hierarchie das `RotateTransform`-Element deklariert und die Eigenschaften `Angle`, `CenterX` und `CenterY` werden gesetzt. Der entsprechende VB-Code sieht folgendermaßen aus:

```
' Button-Element erzeugen
Dim btnClear As New Button
btnClear.Width = 80
btnClear.Height = 25
btnClear.Content = "Löschen"
AddHandler btnClear.Click, AddressOf OnClear

' Rotation erzeugen
Dim rot As New RotateTransform
rot.Angle = 25
rot.CenterX = 40
rot.CenterY = 12.5
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
' Rotation der Schaltfläche zuweisen
btnClear.RenderTransform = rot
Me.Content = btnClear
```

Diese Hierarchien können beliebig tief geschachtelt werden. Wie Sie am letzten Beispiel sehen können, ist XAML bei der Definition von Hierarchien meistens wesentlich einfacher und übersichtlicher als eine normale Programmiersprache.

Oft gibt es in den WPF-Objekten Eigenschaften, denen Sie mehrere Elemente eines Typs zuweisen können. In eine `ListBox` können Sie mehrere `ListBoxItem`-Elemente einfügen:

```
<ListBox>
  <ListBox.Items>
    <ListBoxItem>Test1</ListBoxItem>
    <ListBoxItem>Test2</ListBoxItem>
    <ListBoxItem>Test3</ListBoxItem>
  </ListBox.Items>
</ListBox>
```

Die drei `ListBoxItem`-Elemente werden in einem Collection-Objekt angelegt und der `ListBox`-Eigenschaft `Items` zugewiesen. In dem gezeigten Fall gibt es auch noch eine einfachere Schreibweise:

```
<ListBox>
  <ListBoxItem>Test1</ListBoxItem>
  <ListBoxItem>Test2</ListBoxItem>
  <ListBoxItem>Test3</ListBoxItem>
</ListBox>
```

Zum Vergleich auch hier der passende VB-Code zur Erzeugung der `ListBox`:

```
Dim lb As New ListBox
Dim item As New ListBoxItem
item.Content = "Test1"
lb.Items.Add(item)
item = New ListBoxItem
item.Content = "Test2"
lb.Items.Add(item)
item = New ListBoxItem
item.Content = "Test3"
lb.Items.Add(item)
Me.Content = lb
```

Mit XAML können Sie oft sehr einfach ganze Listen von Objekten deklarieren und zuweisen. Im folgenden Beispiel wird ein `MeshGeometry3D`-Element mit Daten initialisiert:

```
<MeshGeometry3D Positions="0,0,0 5,0,0 0,0,5"
                TriangleIndices="0 2 1"
                Normals="0,1,0 0,1,0 0,1,0" />
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In diesem Beispiel wird die Eigenschaft `Positions` mit drei Elementen vom Typ `Point3D` initialisiert. Jedes `Point3D`-Element wird wiederum mit drei `double`-Zahlen initialisiert, die jeweils durch Kommata getrennt in der Liste angegeben werden. Ganz ähnlich wird die Eigenschaft `Normals` gesetzt. Für die Eigenschaft `TriangleIndices` umfasst die Liste nur drei Zahlen, die einfach hinzugefügt werden. Der VB-Code zu diesem XAML-Beispiel sieht folgendermaßen aus:

```
Dim mesh As New MeshGeometry3D
mesh.Positions.Add(New Point3D(0.0, 0.0, 0.0))
mesh.Positions.Add(New Point3D(5.0, 0.0, 0.0))
mesh.Positions.Add(New Point3D(0.0, 0.0, 5.0))
mesh.TriangleIndices.Add(0)
mesh.TriangleIndices.Add(2)
 mesh.TriangleIndices.Add(1)
 mesh.Normals.Add(New Vector3D(0.0, 1.0, 0.0))
 mesh.Normals.Add(New Vector3D(0.0, 1.0, 0.0))
 mesh.Normals.Add(New Vector3D(0.0, 1.0, 0.0))
```

Wie Sie in diesem Beispiel leicht erkennen können, gibt es für die Eigenschaften `Positions`, `TriangleIndices` und `Normals` der `MeshGeometry3D`-Klasse immer eine `Add`-Methode, welche die Daten aus den XAML-Listenangaben korrekt verarbeitet und hinzufügt.

Mit diesen wenigen Beispielen haben wir die wichtigsten Syntaxelemente von XAML kennen gelernt und sollten in der Lage sein, die XAML-Hierarchien in diesem Buch zu lesen und zu verstehen. Mit etwas Übung werden Sie dann auch eigenen XAML-Code erstellen können.

**HINWEIS** Denken Sie daran, dass in Zukunft die XAML-Hierarchien nicht »von Hand« eingegeben, sondern von grafisch orientierten Werkzeugen erzeugt werden (wenn auch nicht gerade vom eingebauten WPF-Designer – Microsoft stellt aber weitere Werkzeuge zur Verfügung, über die Sie sich unter *http://www.microsoft.com/expression/* informieren können).

# Ein eigenes XAMLPad

Im Windows-Software Development Kit (SDK) wird ein kleines Werkzeug mitgeliefert, welches eine XAML-Hierarchie in einem Fenster darstellen kann. Das Werkzeug heißt XAMLPad (Abbildung 11.8). Wir wollen zum Schluss dieses Kapitels versuchen, unser eigenes einfaches *MeinXAMLPad* zu entwickeln.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

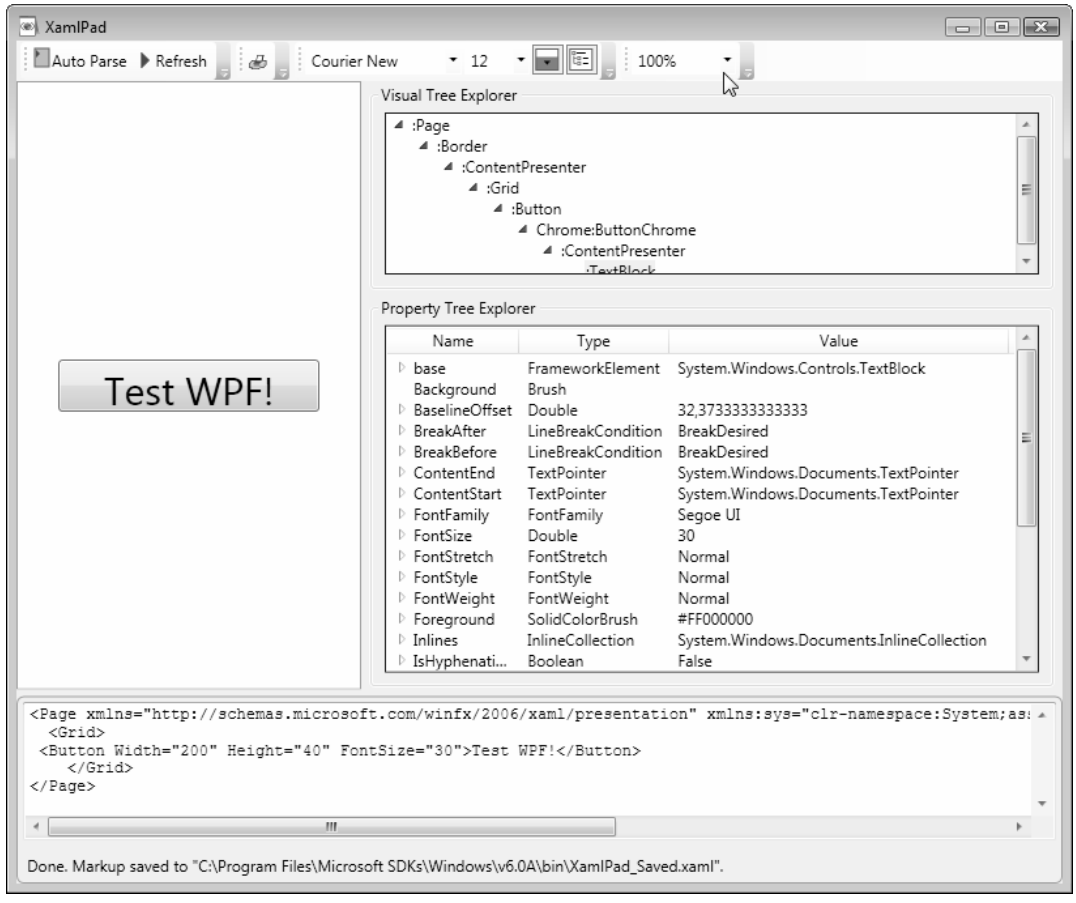



**Abbildung 11.8** Das XAMLPad aus dem Windows Software Development Kit

In Abbildung 11.8 wird im unteren Bereich des Fensters eine XAML-Hierarchie eingegeben, die im oberen Bereich visualisiert wird. Das Programm eignet sich gut, um ein bisschen mit XAML zu experimentieren und die Hierarchien für ein WPF-Element zu erforschen. Es eignet sich allerdings nicht dazu, WPF-Applikationen zu entwickeln und zu testen. Sie können mit diesem Werkzeug keinen VB-Code definieren, der z. B. als Ereignismethode für das `Click`-Ereignis der Schaltfläche aufgerufen werden kann.

Zunächst wollen wir eine einfache Benutzerschnittstelle wie in Listing 11.12 definieren.

```
<Window x:Class="MeinXAMLPad.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MeinXAMLPad" Height="500" Width="600"
    >
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
      <RowDefinition Height="40"/>
    </Grid.RowDefinitions>

    <Grid Name="gridCode" Grid.Column="0" Grid.Row="0" />

    <TextBox Grid.Column="0" Grid.Row="1" Name="textCode"
             VerticalScrollBarVisibility="Visible" HorizontalScrollBarVisibility="Visible"
             FontFamily="Lucida Console" FontSize="14" AcceptsReturn="True" />

    <StackPanel Grid.Column="0" Grid.Row="2" Orientation="Horizontal">
      <Button Name="btnClear" Width="80" Margin="5" Click="OnClear">Löschen</Button>
      <Button Name="btnCompile" Width="80" Margin="5" Click="OnCompile">Darstellen</Button>
      <Button Name="btnExit" Width="80" Margin="5" Click="OnExit">Beenden</Button>
    </StackPanel>
  </Grid>
</Window>
```

**Listing 11.12** Die Benutzerschnittstelle für MeinXAMLPad

Die hier verwendeten WPF-Steuerelemente (`Grid`, `StackPanel`, `Button` und `TextBox`) werden wir an späterer Stelle noch genauer kennen lernen. Das verwendete `Grid`-Element enthält in unserem Beispiel drei Zellen untereinander. Jede Zelle kann weitere WPF-Elemente enthalten. Die oberste Zelle enthält wiederum ein `Grid`-Element mit dem Namen `gridCode`. Dieses `Grid` wird später die WPF-Elemente darstellen, welche wir im darunter liegenden Textfeld eingeben.

Die Texteingabe wird über ein mehrzeiliges `TextBox`-Element realisiert. Um aus dem VB-Code auf das Element zugreifen zu können, vergeben wir auch hier einen Namen: `textCode`. Außerdem setzen wir einige weitere Eigenschaften, um die Eingabe etwas zu vereinfachen. Wichtig ist, die Eigenschaft `AcceptsReturn` auf `True` zu setzen, damit wir problemlos mehrzeilige Texte eingeben können. Weiterhin können Sie die Benutzbarkeit der Bildlaufleiste (*Scrollbar*) mit den Eigenschaften `VerticalScrollBarVisibility` und `HorizontalScrollBarVisibility` einschalten. Eine andere Schriftart (`FontFamily`) mit einer etwas größeren `FontSize` rundet das Bild ab.

Im unteren Teil des Fensters legen wir nun ein `StackPanel`-Element mit horizontaler Anordnung an. Hier befinden sich mehrere Schaltflächen, die *MeinXAMLPad* steuern. Alle Schaltflächen sind mit Ereignismethoden verbunden, deren Funktion sich aus dem Namen ergibt. Die interessanteste Schaltfläche mit dem Namen `btnCompile` sorgt für die Übersetzung und Darstellung des eingegebenen XAML-Codes. Der dazugehörige VB-Code ist in Listing 11.13 zu sehen.

```
Imports System
Imports System.Windows
Imports System.Windows.Markup
Imports System.Xml
Imports System.IO

Namespace MeinXAMLPad
    Partial Public Class Window1
        Inherits Windows.Window
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        'Das folgende Element enthält den Code für unser WPF-Element
        Dim uie As UIElement
        Dim code As UIElement
        Public Sub New()
            uie = Nothing
            InitializeComponent()
            InitTextBox()
        End Sub

        Private Sub InitTextBox()
            'Starttext in die Textbox schreiben
            textCode.Text = _
            "<Grid xmlns=""http://schemas.microsoft.com/winfx/2006/xaml/presentation""" _
            & vbNewLine
            textCode.AppendText _
            ("  xmlns:x=""http://schemas.microsoft.com/winfx/2006/xaml""" _
             & vbNewLine)
            textCode.AppendText("  >" & vbNewLine & vbNewLine & "</Grid>" & vbNewLine)
            textCode.Focus()
        End Sub

        Private Sub OnClear()
            If Not uie Is Nothing Then
                'Existierende WPF-Elemente entfernen
                gridCode.Children.Remove(uie)
                uie = Nothing
            End If
            InitTextBox()
        End Sub

        Private Sub OnCompile()
            'WPF-Element in Grid darstellen
            If Not uie Is Nothing Then
                gridCode.Children.Remove(uie)
            End If

            'WPF-Code aus TextBox übersetzen
            uie = GetTheCode(textCode.Text)

            If Not uie Is Nothing Then
                'WPF im Grid darstellen
                gridCode.Children.Add(uie)
            End If
        End Sub

        Private Function GetTheCode(ByVal strText As String) As UIElement
            uie = Nothing
            Try
                ' Stream aus Eingabetext erzeugen
                Dim sr As StringReader = New StringReader(strText)
                ' Text aus der Box als XmlReader darstellen
                Dim xr As XmlReader = XmlReader.Create(sr)
                ' Laden und übersetzen des XAML-Codes, Zuordnen zum UIElement
                code = CType(XamlReader.Load(xr), UIElement)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            Catch ex As Exception
                'Einfachste Fehlerbehandlung
                MessageBox.Show("Fehler im XAML-Code:" _
                                & vbNewLine & vbNewLine & ex.Message)
            End Try
            Return code
        End Function

        Private Sub OnExit()
            Me.Close()
        End Sub
    End Class
End Namespace
```

**Listing 11.13** Die Implementierung der Logik für MeinXAMLPad

Bei der Initialisierung der Applikation wird aus dem Konstruktor der Fensterklasse die Methode `InitTextBox` aufgerufen. Hier wird nur ein XAML-Grundgerüst mit den erforderlichen Namensräumen in das `TextBox`-Element geschrieben, damit wir beim Testen nicht so viel tippen müssen. Der Code für die Schaltfläche *Beenden* ist eigentlich klar. Hier schließen wir einfach das Hauptfenster der Applikation.

Kommen wir nun zur Ereignismethode `OnCompile`, die natürlich die Hauptarbeit macht. Der übersetzte XAML-Code wird in einem `UIElement` gespeichert, welches in der Fensterklasse deklariert wird. Das Objekt `UIElement` liegt in der WPF-Klassenhierarchie ziemlich weit oben:

- `System.Object`
  - `System.Windows.Threading.DispatcherObject`
    - `System.Windows.DependencyObject`
      - `System.Windows.Media.Visual`
        - **`System.Windows.UIElement`**
          - `System.Windows.FrameworkElement`
            - `System.Windows.Controls.Control`
              - `System.Windows.Controls.ContentControl`
                - `System.Windows.Controls.Primitives.ButtonBase`
                  - `System.Windows.Controls.Button`

Sie sehen in der obigen Liste, dass ein normales WPF-Steuerelement (hier: `Button`-Element) von `UIElement` abgeleitet wird, sodass wir in unserem Beispiel ein Objekt von diesem Typ für die Speicherung des eingegebenen XAML-Segments gut benutzen können. Wenn bereits ein `UIElement` dargestellt wurde, dann wird dieses Element in `OnCompile` zunächst gelöscht, indem für das Kindelement in `gridCode` die `Remove`-Methode aufgerufen wird. Nun kann der eingegebene XAML-Code über die Methode `GetTheCode` verarbeitet werden. In dieser Methode wird der Text aus dem Eingabefeld `textCode` als `StringReader`-Element geöffnet. Der erzeugte Datenstrom wird nun als `XmlReader` mit der `Create`-Methode geöffnet. Diese Schritte sind erforderlich, da das nun verwendete `XamlReader`-Element XML als Eingabe für die `Load`-Methode benötigt. Dieser Methodenaufruf im `XamlReader` erzeugt nun das `UIElement`, welches schließlich als Kindelement im Grid `gridCode` dargestellt wird.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wenn Sie die Schaltfläche *Löschen* anklicken, passieren in der Ereignismethode `OnClear` zwei Dinge. Sollten bereits WPF-Elemente im `Grid`-Element `gridCode` dargestellt werden, dann müssen diese mit der `Remove`-Methode gelöscht werden. Nun sehen Sie wieder ein leeres Ausgabefenster; das Element `gridCode` enthällt kein `UIElement` mehr. Außerdem wird das `TextBox`-Element wieder initialisiert.

*MeinXAMLPad* ist nun fertig. Der Aufruf der Applikation zeigt uns Abbildung 11.9. Im Eingabefeld unserer Applikation haben wir ein `StackPanel`-Element mit einer Schaltfläche, einem Eingabefeld und einem Kontrollkästchen eingegeben. Natürlich ist die Deklaration von Benutzerschnittstellen mit unserem Werkzeug nicht so angenehm. Zum einen vermissen wir natürlich die IntelliSense-Möglichkeit aus Visual Studio 2008, außerdem können wir keine Ereignismethoden definieren und der XAML-Code wird in nacktem Schwarz dargestellt. Wenn Sie Spaß daran haben, können Sie dieses Beispiel natürlich weiter entwickeln.


**Abbildung 11.9** MeinXAML-Pad in der Anwendung

# Zusammenfassung

In diesem ersten Kapitel haben Sie die Grundlagen von Windows Presentation Foundation kennen gelernt. Sie wissen nun, dass Code und Design getrennt voneinander vorliegen. Der Code wird in Visual Basic implementiert, während das Design der Benutzeroberfläche in XAML deklariert wird.

Alle WPF-Elemente, egal ob Steuerelemente oder grafische Figuren, werden als Vektorgrafik ausgegeben. Dadurch wird immer eine hervorragende Darstellungsqualität erzielt, auch wenn Elemente der Benutzerschnittstelle extrem vergrößert werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Verbindung von Design und Code wird über den Aufruf von Ereignismethoden aus XAML bzw. über die mit Namen versehenen Elemente aus dem Programmcode vollzogen.

Wenn Sie mit WPF arbeiten, wird niemals XAML-Code zur Laufzeit »interpretiert«. Letztendlich liegt auch die mit XAML deklarierte Benutzeroberfläche als MSIL-Code in der auszuführenden Datei vor. Dieser wird dann zur Laufzeit vom JIT-Compiler der Common Language Runtime in Maschinencode übersetzt. Somit sollten WPF-Benutzerschnittstellen ähnlich schnell wie bereits bekannte .NET Frameworks arbeiten.

Die Trennung von Code und Logik mit WPF bringt uns folgenden großen Vorteil: Sie bringen einen Top-Designer mit einem Top-Softwareentwickler zusammen und die beiden entwickeln für Sie eine Top-Software!

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 12

# Steuerelemente

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Mit Windows Presentation Foundation wollen wir die Benutzerschnittstellen unserer Applikationen deklarieren. Das funktioniert natürlich nicht ohne Steuerelemente wie Schaltflächen, Eingabefelder, Bildlaufleisten, usw. Diese Elemente stehen Ihnen auch in WPF zur Verfügung. Jedes WPF-Element enthält im .NET Framework 3.0 eine entsprechende Klasse, die das Element mit allen Eigenschaften, Methoden und Ereignissen abbildet.

In diesem Buch beschäftigen sich zwei Kapitel mit den Steuerelementen von WPF. In diesem Kapitel lernen Sie die Standardsteuerelemente, wie z. B. `TextBox`- oder `Button`-Elemente, kennen. In einem zweiten Kapitel werden die Elemente behandelt, die sich mit dem Anordnen von Steuerelementen (Layout) beschäftigen.

**BEGLEITDATEIEN** Unter .*Samples**Chapter12*\ finden Sie die Beispieldateien für dieses Kapitel.

# Einführung

Steuerelemente sind kleine Einheiten der Benutzerschnittstelle einer Applikation. Normalerweise bestehen sie selbst wieder aus einer meist einfachen Benutzerschnittstelle (Aussehen des Steuerelements) und einer implementierten Logik (Verhalten). Häufig bilden die Steuerelemente eine Abstraktionsschicht in einem Framework für Benutzerschnittstellen. Dies ist in WPF allerdings nicht so, da die Steuerelemente hier nicht ihr Aussehen »implementieren«. Das Aussehen der WPF-Steuerelemente wird über Templates (Vorlagen) realisiert. Diese Templates oder Vorlagen können jederzeit ausgetauscht werden. Dadurch wird zwar das Aussehen eines Steuerelements modifiziert, sein Verhalten bleibt jedoch unverändert. Dieses Verhalten wird in Abbildung 12.1 dargestellt.

Das Steuerelement besteht weiterhin aus der Logik, die sein Verhalten festlegt. Hierzu gehören die Methoden, die Eigenschaften und Ereignisse, die ausgelöst werden können. Ein Anwender interagiert mit dem Steuerelement über diese Logik. Die Darstellung des Steuerelements ist in Abbildung 12.1 allerdings losgelöst von der Logik als Template (Vorlage) implementiert. Über diese Vorlage wird dem Anwender das Steuerelement mit seinem Inhalt dargestellt. Auch bei den WPF-Steuerelementen ist also Logik und Design getrennt. Eine solche Trennung ist bei Steuerelementen meistens nicht vollständig zu erreichen. Trotzdem macht diese Aufteilung Sinn, wie wir in diesem und im Kapitel über Vorlagen sehen werden.

Der Vorteil dieser Vorgehensweise in WPF liegt eigentlich auf der Hand. Wenn Sie aus einer normalen, rechteckigen Schaltfläche mit grauem Hintergrund eine elliptische Schaltfläche mit einem Bild und einem Farbgradienten machen wollen, so ist das möglich, ohne ein neues Steuerelement zu programmieren. Das Verhalten (Logik) der Schaltfläche soll erhalten bleiben. Was geändert werden muss, ist nur das Aussehen, welches über eine Vorlage abgebildet wird und separat geändert werden kann.

Alle WPF-Steuerelemente können Ereignisse auslösen, auf die ein Softwareentwickler in Ereignismethoden reagieren kann. In dieser Hinsicht unterscheidet sich WPF nicht von anderen Frameworks für Benutzerschnittstellen. Steuerelemente können natürlich auch Methoden enthalten, die ganz bestimmte Arbeiten erledigen. So können Sie z. B. mit einer `SelectAll`-Methode alle Elemente in einem `ListBox`-Steuerelement auswählen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Ereignisse (*Events*), Befehle (*Commands*) und Eigenschaften (*Properties*) von WPF-Steuerelementen wollen wir in den folgenden Abschnitten noch etwas genauer betrachten.

**HINWEIS** Beachten Sie bitte auch bitte das im vorherigen Kapitel zu Ereignissen Gesagte, das sich auf die spezielle Eigenart von Visual Basic-Anwendungen bezieht, Ereignprozeduren mit dem `Handels`-Schlüsselwort mit Ereignissen bestimmter Objekte zu verknüpfen.

# Weitergeleitete Ereignisse (Routed Events)

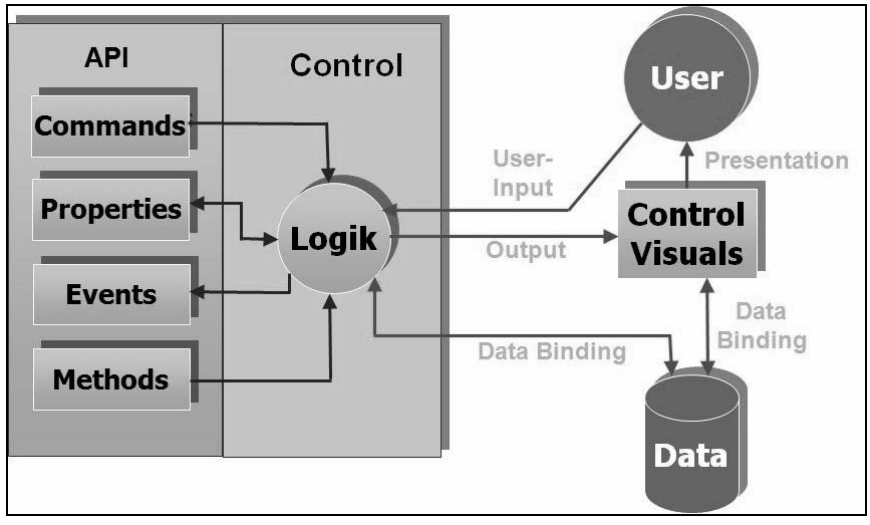



**Abbildung 12.1** Aufbau der Steuerelemente in WPF

Ein sehr wichtiges Konzept für Steuerelemente unter Windows Presentation Foundation sind die *Routed Events*. Routed Events sind Ereignisse, die an verschiedene WPF-Elemente in der Hierarchie weiter geleitet werden können. Hierzu wollen wir uns einmal vorstellen, dass z.B. eine WPF-Schaltfläche aus mehreren Elementen bestehen kann: Auf der Schaltfläche kann sich ein `Grid`-Element mit zwei Zellen befinden. Eine Zelle enthält ein `Canvas`-Element mit mehreren Grafik-Objekten. Die zweite Zelle kann ein `Border`- und ein `TextBlock`-Element beinhalten. Alle Objekte in diesem Szenario (auch die grafischen Elemente) können nun z.B. ein `MouseDown`-Ereignis auslösen. Abbildung 12.2 zeigt ein entsprechendes Beispiel.

In WPF gibt es drei Ereignistypen:

- Direkte Ereignisse (*Direct Events* etwa: *direkte Ereignisse*)
- In der Hierarchie nach oben laufende Ereignisse (*Bubbling Events* etwa: *aufsprudelnde Ereignisse*)
- In der Hierarchie nach unten laufende Ereignisse (*Tunneling Events* etwa: *tunnelnde Ereignisse*)

Die drei Ereignistypen verhalten sich unterschiedlich und müssen jeweils an der richtigen Stelle angewendet werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Direct Events sind mit den normalen CLR-Ereignissen identisch. Ein solcher Event wird nur in dem Element verarbeitet, in dem er erzeugt wurde. Denken Sie bei diesen Ereignissen z. B. an ein `MouseLeave`- oder `MouseEnter`-Ereignis. Diese Ereignisse sind eigentlich nur für das jeweilige Element interessant, welches vom Mauszeiger betreten oder verlassen wird. Andere WPF-Elemente in der logischen Hierarchie werden sich nicht für diese Ereignisse interessieren.

Bubbling Events und Tunneling Events laufen nun im Gegensatz zu den Direct Events durch die logische Hierarchie und werden somit an andere WPF-Elemente weiter gegeben.

Ein Bubbling Event beginnt seine Reise durch die logische Hierarchie beim auslösenden Element und wird dann nach oben bis zum Wurzelelement weiter geleitet. In welchem Element Sie das Ereignis nun bearbeiten, liegt ganz bei Ihnen. Sie können das Ereignis auch in mehreren WPF-Elementen verarbeiten. In diesem Fall implementieren Sie die Ereignismethode mehrfach unter verschiedenen Namen. Ein typisches Bubbling Event ist z. B. das `MouseDown`-Ereignis.

Ein Tunneling Event startet in der logischen Hierarchie oben beim Wurzelelement und fällt dann nach unten bis zum auslösenden Element durch. Auch hier können Sie das Ereignis an beliebiger Stelle in der Hierarchie bearbeiten.

Wenn ein WPF-Element ein Tunneling Event und einen Bubbling Event auslöst, so wird zunächst immer das von oben beginnende Ereignis ausgelöst. Wir sprechen hier von einem *Vorschau*-Ereignis (Preview-Event). Ein ausgelöstes Vorschau-Ereignis können Sie sehr gut benutzen, um das Ereignis selbst zu blockieren, sodass es nicht durch die logische Hierarchie hindurch bis zum auslösenden Element wandert. Das Vorschau-Ereignis zum `MouseDown`-Ereignis heißt dann `PreviewMouseDown`. Vorschau-Ereignisse haben also immer die Vorsilbe `Preview`.

Die Elementhierarchie im hier verwendeten Beispiel (Abbildung 12.2) sieht vereinfacht folgendermaßen aus:

- `Window`
- `Grid`
- `Canvas`
- `Ellipse`
- `Rectangle`
- `Border`
- `TextBlock`


**Abbildung 12.2** Ein Fenster mit einer Elementhierarchie

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In dieser Hierarchie können Sie sehen, dass die beiden grafischen Elemente `Rectangle` und `Ellipse` auf einer Ebene liegen. Trotzdem liegen sie in der Darstellung übereinander, da das Rechteck im XAML-Code erst nach der Ellipse deklariert wurde. Wir wollen nun sehen, welche Ereignisse in dieser Hierarchie ausgelöst werden. Dazu implementieren wir im Beispiel die diversen Ereignismethoden für das Bubbling und das Tunneling `MouseDown`-Ereignis.

```
<Window x:Class="Hierarchie.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hierarchie"
    Height="170" Width="250"
    MouseDown="OnWindow" PreviewMouseDown="OnWindowPrev"
    >
    <Grid ShowGridLines="True" MouseDown="OnGrid" PreviewMouseDown="OnGridPrev">
      <!-- Spalten für Grid-Element definieren -->
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>

      <!-- Element für Grafik in der linken Grid-Zelle -->
      <Canvas Grid.Column="0" Width="100" Height="100"
              MouseDown="OnCanvas" PreviewMouseDown="OnCanvasPrev">
        <Ellipse Canvas.Left="5" Canvas.Top="5" Stroke="Black" Fill="Red" Width="70" Height="50"
                 MouseDown="OnEllipse" PreviewMouseDown="OnEllipsePrev" />
        <Rectangle Canvas.Left="10" Canvas.Top="30" Stroke="Black" Fill="Yellow" Width="50" Height="50"
                   MouseDown="OnRectangle" PreviewMouseDown="OnRectanglePrev"/>
      </Canvas>

      <!-- Rand- mit Text-Element in der rechten Grid-Zelle -->
      <Border Grid.Column="1" Width="100" Height="100" BorderThickness="2"
              BorderBrush="Black" MouseDown="OnBorder" PreviewMouseDown="OnBorderPrev">
        <TextBlock FontSize="15" TextWrapping="Wrap" TextAlignment="Center"
                   MouseDown="OnTextBlock" PreviewMouseDown="OnTextBlockPrev"
                   HorizontalAlignment="Center" VerticalAlignment="Center">
          Ein Border-Element mit Text...
        </TextBlock>
      </Border>
    </Grid>
</Window>
```

**Listing 12.1** Die logische Hierarchie für das Beispiel in Abbildung 12.2

Da wir noch nicht so viele XAML-Hierarchien erzeugt haben, wollen wir hier noch einmal etwas genauer auf das Listing 12.1 eingehen. Das Wurzelelement ist hier ein `Window`-Element mit dem Klassennamen `Window1`. Für dieses Element wurden sowohl das `MouseDown`-Ereignis als auch das `PreviewMouseDown`-Ereignis implementiert. Das Bubbling Event wird in der Methode `OnWindow` und das dazugehörige Tunneling Event in der Methode `OnWindowPrev` bearbeitet. Den VB-Code zur XAML-Deklaration finden Sie in Listing 12.2.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Dort werden auch alle Ereignismethoden, welche dieses Beispiel benötigt, implementiert. In den einzelnen Ereignismethoden wird immer ein Informationstext an eine `String`-Variable angehängt. Hier werden keine `MessageBox`-Ausgaben benutzt, da die Erzeugung eines neuen Fensters den Durchlauf des `MouseDown`-Ereignisses beeinflusst. Um den Ereignisdurchlauf ohne irgendwelche Wechselwirkungen mit anderen WPF-Elementen zu gewährleisten, werden die in der `String`-Variablen gesammelten Informationen im Finalizer der Hauptfensterklasse mithilfe eines `StreamWriter`-Objekts in eine Datei geschrieben, die wir dann nach dem Schließen der Applikation mit dem Notepad von Windows anschauen können.

**HINWEIS** Wenn Sie für die Ausgabe der Informationen in den Ereignismethoden eine `MessageBox` benutzen, werden normalerweise die Ereignismethoden der Bubbling Events nicht aufgerufen.

Zurück zum XAML-Code (Listing 12.1). Innerhalb des Hauptfensters wird zunächst ein `Grid`-Element deklariert, für das auch das `MouseDown`- und das `PreviewMouseDown`-Ereignis implementiert werden. In diesem Fall werden die Methoden `OnGrid` bzw. `OnGridPrev` angesprungen. Über die `Grid.ColumnDefinition`-Eigenschaft werden dann zwei Spalten für das `Grid` definiert. Nun geht es weiter in der logischen Hierarchie mit den WPF-Elementen in den beiden `Grid`-Zellen. In der linken Zelle (`Grid.Column="0"`) wird ein `Canvas`-Element deklariert, welches die beiden Grafikelemente `Rectangle` und `Ellipse` enthält. Für alle Elemente werden die beiden Ereignisse `MouseDown` und `PreviewMouseDown` implementiert. Schließlich wird die rechte `Grid`-Zelle zunächst mit einem Rand (`Border`) versehen, welcher dann ein `TextBlock`-Element mit den jeweiligen Ereignis-Aufrufen enthält.

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Input
Imports System.IO

Namespace Hierarchie
    Partial Public Class Window1
        Inherits System.Windows.Window
        Dim strTest As String = ""

        Public Sub New()
            InitializeComponent()
        End Sub

        Protected Overrides Sub Finalize()
            MyBase.Finalize()
            ' Im Finalizer der Windows1-Klasse werden die gesammelten
            ' Daten in eine Datei gespeichert.
            Dim sw As StreamWriter = New StreamWriter("C:\test.txt")
            sw.Write(strTest)
            sw.Flush()
            sw.Close()
        End Sub

        ' Nun kommen alle Ereignismethoden
        Private Sub OnGrid(ByVal sender As Object, _
                           ByVal e As RoutedEventArgs)
            strTest = strTest + "Grid" & vbNewLine
        End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private Sub OnGridPrev(ByVal sender As Object, _
                               ByVal e As RoutedEventArgs)
            strTest = strTest + "Grid - Preview" & vbNewLine
        End Sub

        Private Sub OnWindow(ByVal sender As Object, _
                             ByVal e As RoutedEventArgs)
            strTest = strTest + "Window" & vbNewLine
        End Sub

        Private Sub OnWindowPrev(ByVal sender As Object, _
                                 ByVal e As RoutedEventArgs)
            strTest = strTest + "Window - Preview" & vbNewLine
        End Sub

        Private Sub OnCanvas(ByVal sender As Object, _
                             ByVal e As RoutedEventArgs)
            strTest = strTest + "Canvas" & vbNewLine
        End Sub

        Private Sub OnCanvasPrev(ByVal sender As Object, _
                                 ByVal e As RoutedEventArgs)
            strTest = strTest + "Canvas - Preview" & vbNewLine
        End Sub

        Private Sub OnTextBlock(ByVal sender As Object, _
                                ByVal e As RoutedEventArgs)
            strTest = strTest + "Textblock" & vbNewLine
        End Sub

        Private Sub OnTextBlockPrev(ByVal sender As Object, _
                                    ByVal e As RoutedEventArgs)
            strTest = strTest + "Textblock - Preview" & vbNewLine
        End Sub

        Private Sub OnBorder(ByVal sender As Object, _
                             ByVal e As RoutedEventArgs)
            strTest = strTest + "Border" & vbNewLine
        End Sub

        Private Sub OnBorderPrev(ByVal sender As Object, _
                                 ByVal e As RoutedEventArgs)
            strTest = strTest + "Border - Preview" & vbNewLine
        End Sub

        Private Sub OnEllipse(ByVal sender As Object, _
                              ByVal e As RoutedEventArgs)
            strTest = strTest + "Ellipse" & vbNewLine
        End Sub

        Private Sub OnEllipsePrev(ByVal sender As Object, _
                                  ByVal e As RoutedEventArgs)
            strTest = strTest + "Ellipse - Preview" & vbNewLine
        End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private Sub OnRectangle(ByVal sender As Object, _
                                ByVal e As RoutedEventArgs)
            strTest = strTest + "Rectangle" & vbNewLine
        End Sub

        Private Sub OnRectanglePrev(ByVal sender As Object, _
                                    ByVal e As RoutedEventArgs)
            strTest = strTest + "Rectangle - Preview" & vbNewLine
        End Sub
    End Class
End Namespace
```

**Listing 12.2** Der Code zum Beispiel in Abbildung 12.2

Die gesamte logische Hierarchie des Beispiels mit einem `MouseDown`- und einem `PreviewMouseDown`-Ereignis können Sie in Abbildung 12.3 verfolgen.

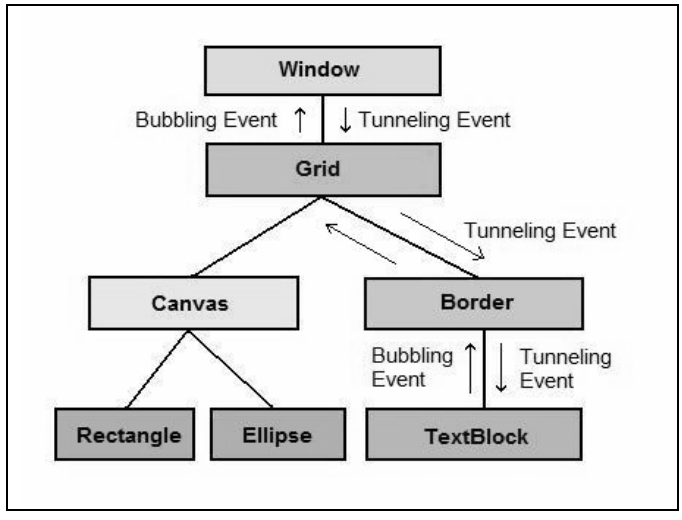



**Abbildung 12.3** Ereignisse in der logischen Hierarchie

Nun können wir das Beispielprogramm endlich aufrufen (Abbildung 12.2). Wenn wir mit der Maus auf den Text in der rechten `Grid`-Zelle klicken und danach die Applikation beenden, finden wir in der Datei »C:\Test.txt« folgende Informationen:

```
Window – Preview
Grid – Preview
Border – Preview
TextBlock – Preview
TextBlock
Border
Grid
Window
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Sie können sehen, dass zuerst das »Tunneling Event« ausgelöst wird. Der Durchlauf beginnt im Wurzelelement der Hierarchie, also im Hauptfenster der Applikation. Dann läuft das `Preview`-Ereignis durch den logischen Baum bis zum `TextBlock`-Element, welches das Ereignis ausgelöst hat. Nun läuft das »Bubbling Event« wieder den Weg zurück bis hin zum Wurzelelement. In jeder Ereignismethode könnten Sie das Ereignis bearbeiten. Sie müssen aber die Weitergabe des Ereignisses nicht programmieren, denn dies erledigt WPF für Sie. Das Ereignis läuft natürlich auch dann durch die logische Hierarchie, wenn einzelne Ereignismethoden nicht implementiert werden.

Im zweiten Beispiel wollen wir nun auf das gelbe Rechteck in der linken Grid-Zelle klicken und zwar so, dass wir in dem Bereich klicken, in dem auch darunter die rote Ellipse liegt (Abbildung 12.4). Das Ergebnis in der Datei »C:\Test.txt« lautet nun:

```
Window – Preview
Grid – Preview
Canvas – Preview
Rectangle – Preview
Rectangle
Canvas
Grid
Window
```


**Abbildung 12.4** Der zweite Versuch mit den Ereignissen

Hier wird das »Tunneling Event« zuerst vom Hauptfenster aus bis zum grafischen Element `Rectangle` durchlaufen. Wie Sie sehen, wird die Ereignismethode `OnEllipsePrev` des `Ellipse`-Elements nicht aufgerufen, da diese Ellipse in der logischen Hierarchie auf der gleichen Ebene wie das Rechteck liegt. Beide grafischen Objekte liegen im gleichen `Canvas`-Element. Auch hier wird dann das »Bubbling Event« durchlaufen, welches schließlich im Hauptfenster ankommt.

Welche Ereignismethoden in der Hierarchie aufgerufen werden, kann letztendlich von Ihnen gesteuert werden. Jedes weitergeleitete Ereignis erhält beim Aufruf der Ereignismethode ein Objekt vom Typ `RoutedEventArgs` oder ein Objekt, welches von dieser Klasse abgeleitet ist:

```
Private Sub OnCanvasPrev(ByVal sender As Object, _
                         ByVal e As RoutedEventArgs)
    strTest = strTest + "Canvas - Preview" & vbNewLine
End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Sie können die Eigenschaft Handled im Parameter e benutzen, um den weiteren Durchlauf eines Ereignisses durch die logische Hierarchie zu unterbrechen:

```
Private Sub OnCanvasPrev(ByVal sender As Object, _
                         ByVal e As RoutedEventArgs)
    strTest = strTest + "Canvas - Preview" & vbNewLine
    e.Handled = True
End Sub
```

Als Standard wird in einer Ereignismethode für die Eigenschaft Handled der Wert False zurückgegeben. In diesem Fall wird das Ereignis weitergeleitet, wie wir es in den vorangegangenen Beispielen gesehen haben. Setzen Sie nun den Wert von Handled auf True, ist das Ereignis sozusagen »erledigt« und die Ereignisweiterleitung wird an dieser Stelle unterbrochen. Als Beispiel wollen wir die Eigenschaft Handled im Ereignis PreviewMouseDown des Canvas-Elements auf True setzen. Nun werden die in der Hierarchie darunter liegenden Preview-Ereignismethoden und alle Methoden des Bubbling Events nicht aufgerufen, wie Sie an der ausgegebenen Test-Datei sehen können:

```
Window – Preview
Grid – Preview
Canvas – Preview
```

Direkte Ereignisse (Direct Events) werden nur für das Element verarbeitet, in dem sie auch ausgelöst wurden. In die Kategorie der direkten Ereignisse zählen z. B. das MouseEnter- oder das MouseLeave-Ereignis. Diese Ereignisse sind im Normalfall nur für das auslösende Element von Interesse.

Wenn Sie sich die Eigenschaften der Klasse RoutedEventArgs anschauen, werden Sie dort jedoch einige wichtige Informationen vermissen. Da wir ein Mausereignis verarbeitet haben, wollen wir meistens auch die Klickposition auswerten. Die Lösung des Problems ist sehr einfach. Wir müssen als zweiten Parameter in der Ereignismethode ein Element vom Typ MouseEventArgs annehmen. Listing 12.3 zeigt ein einfaches Fenster, welches wieder die beiden Ereignisse MouseDown und PreviewMouseDown implementiert (Listing 12.4). Auch hier benutzen wir im Code zunächst einen String in den Ereignismethoden, um die anfallenden Informationen zu sammeln. Da wir nun wissen, in welcher Reihenfolge die Ereignisse auftreten, können wir im »Bubbling Event« eine MessageBox mit dem Ergebnis anzeigen. Zusätzlich benötigen wir im Beispiel ein einfaches Rechteck, welches in einem Canvas-Element platziert wird.

Im MouseEventArgs-Objekt e gibt es allerdings immer noch keine X- und Y-Eigenschaften für die Position der Maus. Hier gibt es stattdessen eine interessante Vereinfachung für die Positionsabfrage. Wir benutzen die Methode GetPosition aus dem MouseEventArgs-Objekt. Beim Aufruf von GetPosition müssen Sie als einzigen Parameter ein Objekt in der Benutzerschnittstelle angeben, zu dem die Mauskoordinaten dann relativ umgerechnet werden sollen. Die Ausgabe der Koordinaten bezieht sich nun auf die obere linke Ecke des Hauptfensters. Im »Bubbling Event« beziehen wir uns in der Methode GetPosition auf das Rechteck mit dem Objektnamen rect. Der Unterschied der beiden Aufrufe ist im Ausgabetext in der MessageBox leicht erkennbar.

```
<Window x:Class="EventArgs.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="EventArgs" Height="200" Width="250"
    MouseDown="OnWindow"
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    PreviewMouseDown="OnWindowPrev"
    >
    <Grid>
      <Canvas>
        <Rectangle Canvas.Left="30" Canvas.Top="30" Width="150" Height="100"
                   Fill="Red" Stroke="Black" Name="rect"/>
      </Canvas>
    </Grid>
</Window>
```

**Listing 12.3** Benutzung der MouseEventArgs-Klasse

```
Imports System
Imports System.Windows
Imports System.Windows.Input

Namespace EventArgs
    Partial Public Class Window1
        Inherits System.Windows.Window
        Private strTest As String = String.Empty

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub OnWindow(ByVal sender As Object, _
                            ByVal e As MouseEventArgs)
            Dim pt As New Point
            pt = e.GetPosition(Me)
            strTest = strTest + "Bubble X: " _
            & pt.X.ToString & _
            "Y: " & pt.Y.ToString & vbNewLine
            MessageBox.Show(strTest)
        End Sub

        Public Sub OnWindowPrev(ByVal sender As Object, _
                                ByVal e As MouseEventArgs)
            Dim pt As New Point
            pt = e.GetPosition(rect)
            strTest = strTest + "Tunnel X: " _
            & pt.X.ToString & _
            "Y: " & pt.Y.ToString & vbNewLine
        End Sub
    End Class
End Namespace
```

**Listing 12.4** Benutzung der Klasse MouseEventArgs

Abschließend sei noch gesagt, dass nicht alle weitergeleiteten Ereignisse bis zum obersten Wurzel-Element der Anwendung laufen. So ist für das `MouseDown`-Ereignis z.B. ein `Button`-Element eine Schranke, die das Ereignis nicht ohne weiteres überwinden kann. Dadurch laufen bestimmte Ereignisse nicht sinnloserweise durch die gesamte Objekthierarchie.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Weitergeleitete Befehle (Routed Commands)

Ein anderes wichtiges Konzept in WPF sind die weitergeleiteten Befehle (Routed Commands). Hier geht es um das Problem, dass eine Methode mit mehreren Elementen aus der Benutzerschnittstelle verbunden werden soll. Denken Sie z. B. an den Fall, dass Sie einen Menüpunkt und eine Schaltfläche in der Symbolleiste der Anwendung mit einer einzigen Ereignismethode verbinden wollen. Dies ist im Grunde genommen einfach zu realisieren, indem wir das Klickereignis der beiden Elemente mit der gleichen Methode im Code verbinden.

Nun wollen wir noch einen Schritt weiter gehen. Es ist häufig erforderlich, die beiden Elemente der Benutzerschnittstelle (Menüpunkt und Schaltfläche in der Symbolleiste) in bestimmten Situationen ein- oder auszuschalten. Normalerweise erledigen wir die Einstellungen für das Menü erst dann, wenn das Popup-Menü heruntergeklappt werden soll. Dafür gibt es auch die entsprechenden Ereignisse, die wir benutzen können. Nun gibt es aber ein Problem, denn die Schaltfläche in der Symbolleiste muss sofort bearbeitet werden, wenn sich der Status für die Benutzerschnittstelle ändert. Bauen wir also die Aktivierung und Deaktivierung der Schaltfläche in die gleiche Methode ein, welche die Logik für den Menüpunkt enthält, so wird die Schaltfläche zu spät bearbeitet. Wir müssen also eine eigene Behandlungsmethode für die Elemente in der Symbolleiste programmieren. Das ist natürlich keine gute Lösung.

Hier kommen nun die Routed Commands ins Spiel. Sie können mit diesem WPF-Konzept verschiedene Elemente der Benutzerschnittstelle mit Ereignismethoden verbinden, die dann auch entsprechend zeitnah aufgerufen werden.

Wir wollen das in einem Beispiel vertiefen. Im Hauptfenster der Applikation gibt es ein Menü in einem `DockPanel`-Element mit zwei Menüpunkten *Neu* und *Beenden*. Zusätzlich haben wir mitten im Fenster eine Schaltfläche *Neu*, die an die gleiche Ereignismethode gebunden werden soll wie der Menüpunkt. Der dazugehörige XAML-Code ist in Listing 12.5 zu sehen.

```
<Window x:Class="Commands.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Commands" Height="200" Width="300"
    >
  <DockPanel>
    <Menu DockPanel.Dock="Top">
      <MenuItem Header="Datei">
        <MenuItem Command="New" Header="Neu..."/>
        <Separator />
        <MenuItem Click="OnBeenden" Header="Beenden" />
      </MenuItem>
    </Menu>

    <Button Command="New" Width="100" Height="25">Neu...</Button>
  </DockPanel>
</Window>
```

**Listing 12.5** Benutzerschnittstelle für das Beispiel mit Routed Commands

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Was hier auffällt, ist die Eigenschaft `Command`, die im ersten `MenuItem`- und im `Button`-Element verwendet wird. Für den zweiten Menüpunkt *Beenden* wird dagegen eine ganz normale Ereignismethode für das Klickereignis implementiert.

Hinter der Eigenschaft `Command` steckt nun ein so genannter »Routed Command«. Wir wollen darum zunächst den VB-Code in Listing 12.6 anschauen.

```
Imports System
Imports System.Windows
Imports System.Windows.Input

Namespace Commands
    Partial Public Class Window1
        Inherits System.Windows.Window
        Private strDokument As String = "Das ist ein Test-Text!"
        Public Sub New()
            InitializeComponent()

            ' CommandBinding-Objekt anlegen und Ereignismethoden zuweisen
            Dim commandNew As New CommandBinding(ApplicationCommands.[New])
            AddHandler commandNew.Executed, AddressOf NeuHandler
            AddHandler commandNew.CanExecute, AddressOf NeuAusfuehrenHandler

            'Neues CommandBinding hinzufügen
            Me.CommandBindings.Add(commandNew)
        End Sub

        Private Sub NeuHandler(ByVal sender As Object, _
                               ByVal e As ExecutedRoutedEventArgs)
            Dim strHelp As String = "Wollen Sie wirklich alles löschen?"
            Dim res As New MessageBoxResult
            res = MessageBox.Show(strHelp, "Test", MessageBoxButton.YesNo)
            If res = MessageBoxResult.Yes Then
                strDokument = String.Empty
            End If
        End Sub

        Private Sub NeuAusfuehrenHandler(ByVal sender As Object, _
                                         ByVal e As CanExecuteRoutedEventArgs)
            ' Darf der Befehl ausgeführt werden?
            e.CanExecute = strDokument.Length > 0
        End Sub

        Private Sub OnBeenden(ByVal sender As Object, _
                              ByVal e As RoutedEventArgs)
            Me.Close()
        End Sub
    End Class
End Namespace
```

**Listing 12.6** CommandBinding im Einsatz

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Im Fensterobjekt wird eine `String`-Variable deklariert, in welcher sich ein kurzer Text befindet. Dieser Text soll hier im Beispiel unser Dokument darstellen. Das heißt, dass der *Neu*-Befehl als Menüpunkt und als Schaltfläche nur dann aktiviert sein soll, wenn sich in der `String`-Variablen auch tatsächlich ein Text befindet. Ist die `String`-Variable leer, darf der *Neu*-Befehl nicht ausgeführt werden. Hierzu benutzen wir ein Objekt der `CommandBinding`-Klasse, welches wir im Konstruktor des Hauptfensters anlegen. Bei der Instanziierung des Objekts wird im Konstruktor als Parameter der Wert `ApplicationCommands.[New]` übergeben. In WPF gibt es verschiedene, vordefinierte `CommandBinding`-Klassen, die Sie in Ihren Applikationen sofort benutzen können.

**WICHTIG** Es gibt folgende Grundklassen für Routed Commands in WPF:

- `ApplicationCommands`
- New, Open, Save, SaveAs, Print, PrintPreview, Copy, Cut, Paste, Replace, SelectAll,...
- `ComponentCommands`
- MoveDown, MoveUp, MoveLeft, MoveRight, ScrollPageDown, ScrollPageUp,...
- `EditingCommands`
- AlignCenter, AlignJustify, Backspace, Delete, EnterLineBreak, MoveDownByLine,...
- `MediaCommands`
- BoostBass, ChannelUp, ChannelDown, FastForward, MiteVolume, NextTrack, Play, Stop,...

Die obige Liste enthält nicht alle Befehle der jeweiligen *Commands*-Klassen. Die Klasse `ApplicationCommands` enthält als statische Eigenschaften für die diversen Befehle, die normalerweise im Menü einer Applikation zu finden sind. In der Klasse `ComponentCommands` finden Sie Befehle, die mit Komponenten zu tun haben. `EditingCommands` enthält die Befehle, die mit dem Ändern von Dokumenten zu tun haben. Schließlich gibt es noch die Klasse `MediaCommands`, die Befehle enthält, die mit Mediendateien arbeiten.

Die vier *Commands*-Klassen enthalten nicht den Code, um die verschiedenen Befehle auszuführen (z. B. einen Text zu kopieren oder eine Datei zu speichern), sondern nur statische Eigenschaften, die es Ihnen ermöglichen, mehrere Elemente der Benutzerschnittstelle unter die Kontrolle einer Ereignismethode zu stellen.

Wenn Ihnen die vorgegebenen Befehle in den vier *Commands*-Klassen nicht ausreichen, können Sie eigene Befehlsklassen erzeugen.

Nach der Erzeugung des `CommandBinding`-Objekts in unserem Beispiel können Sie zwei Ereignismethoden definieren, die einerseits den Code aufrufen, der ausgeführt werden soll, wenn der Menüpunkt oder die Schaltfläche angeklickt werden (Ereignis: `Executed`) und andererseits die Steuerung für die Aktivierung und Deaktivierung der Elemente der Benutzerschnittstelle übernehmen (Ereignis: `CanExecute`). Diese Ereignisse, die hier ausgelöst werden, sind übrigens ganz normale Routed Events, die die logische Hierarchie der Benutzerschnittstelle durchlaufen.

Schließlich fügen wird das erzeugte `CommandBindings`-Objekt mit der Methode `Add` in die Liste der `CommandBindings` des Elternelements ein. Nun »weiß« unser Hauptfenster von den Bindungen der Schaltfläche und des Menüpunktes und kann diese bei Bedarf durch Aufruf der Ereignismethoden ausführen.

Die Ereignismethoden selbst erhalten ganz normalen Code. In der Methode `NeuHandler` wird der Anwender gefragt, ob er das Dokument (der `string strDokument`) löschen will. Wenn ja, wird die `String`-Variable geleert.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In der Methode `NeuAusfuehrenHandler` muss die Eigenschaft `CanExecute` des `CanExecuteRoutedEventArgs`-Objekt, der als Parameter übergeben wird, auf `True` gesetzt werden, wenn der Befehl ausgeführt werden darf. Ansonsten wird hier `False` zurückgegeben. Im Beispiel prüfen wir hierzu, ob die Länge des Textes in `strDokument` größer als Null ist.

Wenn wir die Applikation nun starten (Abbildung 12.5), ist der *Neu*-Befehl sowohl als Menüpunkt, sowie als Schaltfläche aktiviert, da in der String-Variable ein Text steht.


**Abbildung 12.5** Nach dem Start des CommandBindings-Beispiels

Klicken Sie nun auf die Schaltfläche *Neu*. Die Abfrage, ob Sie löschen wollen, beantworten Sie mit »Ja«. Sobald das `MessageBox`-Fenster geschlossen wird, ändert sich auch der Status in der Benutzerschnittstelle. Die Schaltfläche und der Menüpunkt werden deaktiviert und der Befehl *Neu* kann nicht mehr ausgeführt werden.

Die Ereignismethode `NeuHandler` wird natürlich nur dann aufgerufen, wenn der Befehl selbst über das Menü oder die Schaltfläche ausgelöst wird. Es stellt sich jedoch die Frage, wie häufig die Methode `NeuAusfuehrenHandler` aufgerufen wird. Um dies zu prüfen, können wir in diese Ereignismethode zusätzlichen Code einbauen, der jedes Mal einen kleinen Text im Ausgabe-Fenster von Visual Studio ausgibt. Wir benutzen in diesem Fall kein `MessageBox`-Element für die Datenausgabe, um den Fluss der Ereignisse nicht zu ändern. Zuerst werden wir feststellen, dass die Ereignismethode nicht einfach regelmäßig aufgerufen wird (Polling). Die Ereignismethoden für die Aktivierung und Deaktivierung der Elemente werden immer dann aufgerufen, wenn »irgendwie etwas Wichtiges« in Ihrer Benutzerstelle passiert. So wird die Methode `NeuAusfuehrenHandler` z. B. aufgerufen, wenn Sie in das Hauptfenster klicken, wenn Sie Popup-Menüs herunterklappen, wenn Sie auf die Schaltfläche *Neu* klicken oder wenn die Applikation gestartet wird. Die Ereignismethode wird nicht aufgerufen, wenn Sie z. B. den Mauszeiger durch das Fenster oder über die Schaltfläche bewegen, ohne mit der Maus zu klicken.

Um alle Aktivierungsmethoden von Hand zu starten, müssen Sie die statische Methode `InvalidateRequerySuggested` aus der Klasse `CommandManager` aufrufen:

```
CommandManager.InvalidateRequerySuggested()
```

Ein solcher »außerordentlicher« Aufruf kann dann wichtig sein, wenn ein separater Thread bestimmte Rechenarbeiten erledigt hat, welche den Status der Benutzerschnittstelle beeinflussen. Beim Beendigen eines Threads wird nicht automatisch die Methode `NeuAusfuehrenHandler` aufgerufen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Das bedeutet natürlich, dass der Code in den Ereignismethoden für die Aktivierung und Deaktivierung der Steuerelemente möglichst kurz sein sollte, damit er sehr schnell ausgeführt werden kann. Sie müssen bedenken, dass es in der Applikation mehrere `ComandBindings` im Hauptfenster geben kann. In den oben skizzierten Fällen werden dann natürlich alle Ereignismethoden aufgerufen, die am entsprechenden Elternelement gebunden sind.

Weitergeleitete Ereignisse unterstützen die Trennung von Logik und Design. Der Softwareentwickler implementiert Befehle mit ganz bestimmten Namen. Der Designer kann diese Befehle mit den Steuerelementen in der Benutzerschnittstelle verbinden, ohne dass er einen Namen für eine spezielle Ereignismethode vergeben muss. Er benutzt einfach den Namen des gewünschten Befehls:

```
<MenuItem Command="New" Header="Neu..."/>
<Button Command="Copy">Kopieren</Button>
```

# Eigenschaften der Abhängigkeiten

Kommen wir nun zu den Eigenschaften der Abhängigkeiten (Dependency Properties) bei den Steuerelementen von Windows Presentation Foundation.

Eigenschaften werden in der logischen Hierarchie von WPF weiter vererbt. Wird z. B. in einem `StackPanel`-Element eine bestimmte Schriftartgröße (Eigenschaft `FontSize`) gesetzt, so wird diese an alle Elemente weitergegeben, die in diesem Layout-Element enthalten sind. Dabei wird auch das Layout der Elemente neu berechnet, denn durch eine größere oder kleinere Schriftart für die Kindelemente im `StackPanel` können sich natürlich auch die Abmessungen und Positionen ändern (sofern es das Elternelement erlaubt). Das folgende Beispiel soll die Möglichkeiten, die sich durch die Abhängigkeitseigenschaften ergeben, demonstrieren (Listing 12.7 und Listing 12.8).

```
<Window x:Class="Depend.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Depend" Height="200" Width="650"
    >
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Button Margin="5" Tag="8" Grid.Row="0" Grid.Column="0" Click="WindowOnClick">Window: 8pt</Button>
    <Button Margin="5" Tag="16" Grid.Row="0" Grid.Column="1" Click="WindowOnClick">Window: 16pt</Button>
    <Button Margin="5" Tag="32" Grid.Row="0" Grid.Column="2" Click="WindowOnClick">Window: 32pt</Button>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Button Margin="5" Tag="8" Grid.Row="1" Grid.Column="0" Click="ButtonOnClick">Button: 8pt</Button>
    <Button Margin="5" Tag="16" Grid.Row="1" Grid.Column="1" Click="ButtonOnClick">Button: 16pt</Button>
    <Button Margin="5" Tag="32" Grid.Row="1" Grid.Column="2" Click="ButtonOnClick">Button: 32pt</Button>
  </Grid>
</Window>
```

**Listing 12.7** Test der Eigenschaften von Abhängigkeiten

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Input

Namespace Depend
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub WindowOnClick(ByVal sender As Object, _
                                 ByVal e As RoutedEventArgs)
            'FontSize-Eigenschaften des Fensters ändern
            Dim btn As New Button
            btn = CType(e.Source, Button)
            Me.FontSize = Convert.ToDouble(btn.Tag)
        End Sub

        Private Sub ButtonOnClick(ByVal sender As Object, _
                                  ByVal e As RoutedEventArgs)
            'FontSize-Eigenschaft einer Schaltfläche ändern
            Dim btn As New Button
            btn = CType(e.Source, Button)
            btn.FontSize = Convert.ToDouble(btn.Tag)
        End Sub
    End Class
End Namespace
```

**Listing 12.8** Die Schriftartgröße wird als Eigenschaft einer Abhängigkeit geändert

In diesem Beispiel werden sechs Schaltflächen dargestellt, welche die Schriftgrößen im Fenster oder auf den Schaltflächen selbst ändern können. Der Code für die Änderung der Schriftgrößen wurde in den Klick-Ereignismethoden `WindowOnClick` und `ButtonOnClick` untergebracht. Dort wird zunächst mit Hilfe des Methodenparameters `sender` die Schaltfläche ermittelt, auf die geklickt wurde. In der Eigenschaft `Tag` der einzelnen Schaltflächen ist die einzustellende Schriftgröße hinterlegt, die dann konvertiert und entweder für das gesamte Fenster oder nur für die jeweilige Schaltfläche gesetzt wird. Nach dem Start des Programms haben alle Schaltflächen die gleiche Schriftgröße (Abbildung 12.6).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

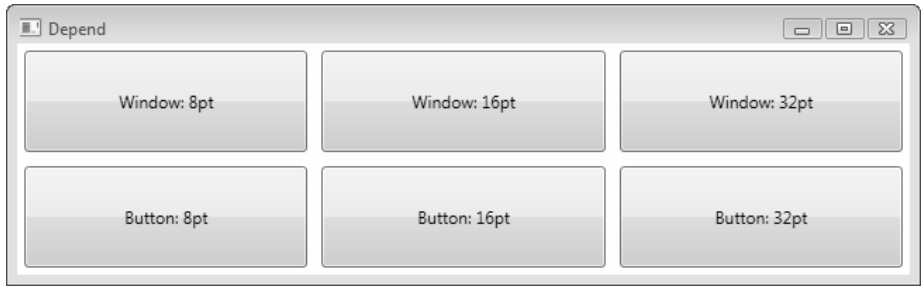



**Abbildung 12.6** Start des Testprogramms

Die Schaltflächen erben in diesem Beispiel die Schriftgröße des Elternelements, also des umgebenden Fensters. Klicken Sie nun auf die Schaltfläche *Button: 8pt*. Es wird nur die Schriftgröße dieser einen Schaltfläche geändert (Abbildung 12.7).

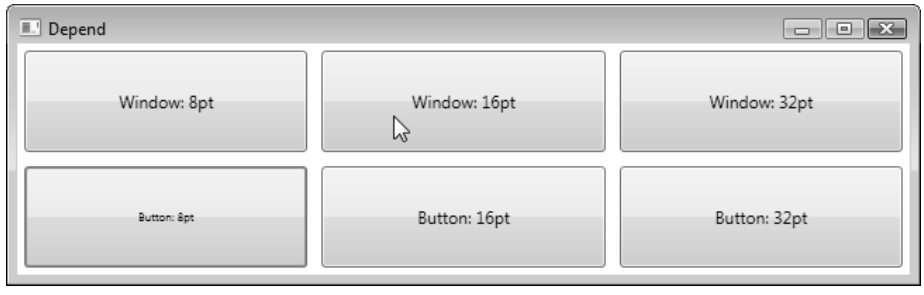



**Abbildung 12.7** Schaltfläche mit kleiner Schriftgröße

Im nächsten Test klicken wir nun auf die Schaltfläche *Window: 16pt*. Nun sollte für das gesamte Fenster die Schriftgröße 16pt verwendet werden, außer für die Schaltfläche, die wir eben im ersten Test auf die kleine Schriftgröße (8pt) gesetzt haben (Abbildung 12.8):

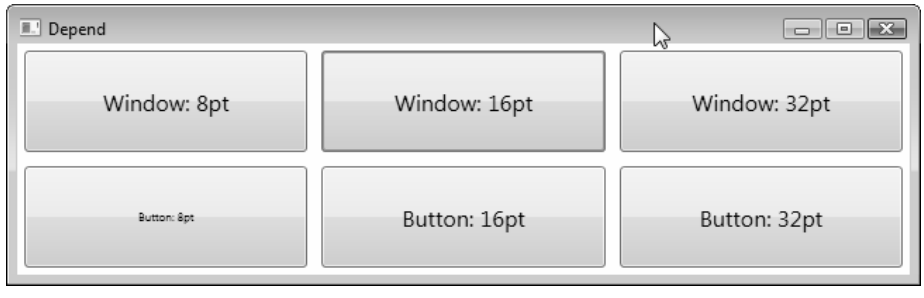



**Abbildung 12.8** Das Fenster mit 16pt-Schriftgröße

Die eingestellte Schriftgröße wurde bei diesem Versuch durch die gesamte logische Hierarchie der Benutzerschnittstelle weiter vererbt. Da für die Schaltfläche *Button: 8pt* bereits explizit eine andere Schriftgröße gesetzt wurde, ist an dieser Stelle die Vererbung unterbrochen worden. Klicken Sie nun noch auf die Schaltfläche *Button: 32pt* und Sie werden sehen, dass nur die Schriftart dieser Schaltfläche vergrößert wird (Abbildung 12.9).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

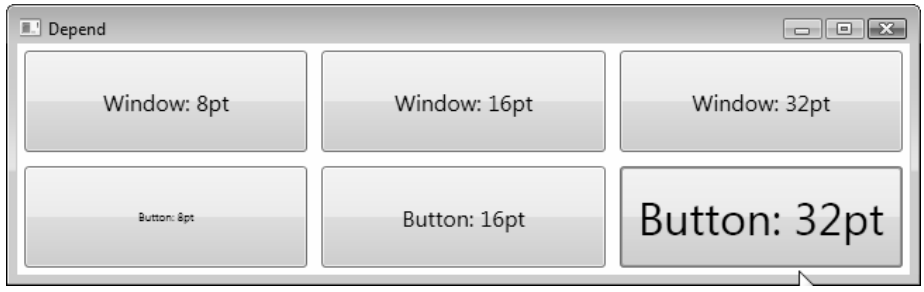



**Abbildung 12.9** Der letzte Test

Ein weiterer Klick auf die Schaltfläche *Window: 8pt* würde nun durch die Vererbung der Eigenschaften alle Schaltflächen in der oberen Reihe und die Schaltfläche *Button: 16pt* in der unteren Reihe des Fensters ändern.

Da im letzten Beispiel ein `Grid`-Element benutzt wurde, um die Schaltfläche zu positionieren, hat sich die Größe der Kindelemente bei der Änderung der Schriftgröße nicht geändert. Sie können natürlich andere Szenarien definieren, bei denen die Größe der Kindelemente durch eine Layout-Berechnung von WPF neu ermittelt wird.

Wenn Sie die Vererbung einer Eigenschaft durch das explizite Setzen dieser Eigenschaft »unterbrochen« haben, wird der vorgegebene Wert für die Eigenschaft dieses Steuerelements benutzt und natürlich von dort an andere Steuerelemente, die sich in der Hierarchie darunter befinden, weitergegeben. Sie können das explizite Setzen der Eigenschaft wieder aufheben, indem Sie die Methode `ClearValue` anwenden:

```
Me.ClearValue(Window.FontSizeProperty)
```

Mit der obigen Codezeile können Sie die Eigenschaft `FontSize` wieder durch die gesamte Hierarchie, einschließlich des Steuerelements `Me`, vererben.

# Eingaben

Eingaben können mit der Maus, über die Tastatur und mit dem Stift gemacht werden. Die Mauseingaben werden immer zunächst zu dem Element der Benutzerschnittstelle geleitet, das sich genau unter dem Mauszeiger befindet. Handelt es sich um ein Routed Event, läuft das Ereignis nun durch die logische Hierarchie und wird ggf. in den verschiedenen Ereignismethoden abgearbeitet. Ein direktes Ereignis kann nur Element-spezifisch verarbeitet werden.

| Maus-Ereignis | Weiterleitung | Aktion |
|---|---|---|
| `GotMouseCapture` | Bubbling | Das Element hält den Mausfokus fest |
| `LostMouseCapture` | Bubbling | Das Element hat den Mausfokus verloren |
| `MouseEnter` | Direkt | Der Mauszeiger wird in das Element hinein bewegt |
| `MouseLeave` | Direkt | Der Mauszeiger wird aus dem Element hinaus bewegt |
| `MouseDown, PreviewMouseDown` | Bubbling, Tunneling | Eine Maustaste wurde gedrückt |
| `MouseUp, PreviewMouseUp` | Bubbling, Tunneling | Eine Maustaste wurde losgelassen ▶ |

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

| Maus-Ereignis | Weiterleitung | Aktion |
|---|---|---|
| MouseMove, PreviewMouseMove | Bubbling, Tunneling | Die Maus wurde bewegt |
| MouseWheel, PreviewMouseWheel | Bubbling, Tunneling | Das Mausrad wurde gedreht |
| MouseLeftButtonDown, PreviewMouseLeftButtonDown | Bubbling, Tunneling | Die linke Maustaste wurde gedrückt |
| MouseLeftButtonUp, PreviewMouseLeftButtonUp | Bubbling, Tunneling | Die linke Maustaste wurde losgelassen |
| MouseRightButtonDown, PreviewMouseRightButtonDown | Bubbling, Tunneling | Die rechte Maustaste wurde gedrückt |
| MouseRightButtonUp, PreviewMouseRightButtonUp | Bubbling, Tunneling | Die rechte Maustaste wurde losgelassen |
| QueryCursor | Bubbling | Ein Element fragt nach dem momentanen Mauscursor |

**Tabelle 12.1** Maus-Ereignisse

Beim Programmieren mit der Maus gibt es noch eine interessante Eigenschaft, die aus der Klasse `UIElement` kommt, von der jedes WPF-Steuerelement abgeleitet ist. Die Eigenschaft heißt `IsMouseOver` und liefert einen boolschen Wert zurück. Die Eigenschaft `IsMouseOver` gibt immer dann der Wert `True` zurück, wenn sich der Mauszeiger über dem Steuerelement oder einem seiner sichtbaren Kindelemente befindet.

Bitte beachten Sie, dass das schon oft verwendete `Click`-Ereignis in Tabelle 12.1 der Maus-Ereignisse nicht enthalten ist. Ein `Click`-Ereignis kann nicht nur durch die Maus ausgelöst werden, sondern auch durch eine Tastatur. Außerdem können sich hinter einem `Click`-Ereignis mehrere Maus-Ereignisse verbergen (`MouseUp`, `MouseDown`). Darum finden Sie in der Klasse `Control` die beiden zusätzlichen Ereignisse `MouseDoubleClick` und `PreviewMouseDoubleClick`. Die Klasse `ButtonBase` implementiert dann das `Click`-Ereignis. `ButtonBase` ist die Basisklasse der Klassen `Button`, `RadioButton` und `CheckBox`.

Auch für die Tastatur stehen diverse Ereignisse zur Verfügung (Tabelle 12.2). Hier spielt das Konzept des Eingabefokus eine entscheidende Rolle. Nur das Steuerelement enthält Eingaben von der Tastatur, welches den Eingabefokus hat. Der Fokus kann durch Anklicken mit der Maus, durch die ⇆-Taste oder durch Navigieren mit den Cursor-Tasten zu einem bestimmten Steuerelement gebracht werden.

| Tastatur-Ereignis | Weiterleitung | Aktion |
|---|---|---|
| GotFocus, PreviewGotFocus | Bubbling, Tunneling | Ein Element erhält den Eingabefokus |
| LostFocus, PreviewLostFocus | Bubbling, Tunneling | Ein Element verliert den Eingabefokus |
| KeyDown, PreviewKeyDown | Bubbling, Tunneling | Eine Taste wird gedrückt |
| KeyUp, PreviewKeyUp | Bubbling, Tunneling | Eine Taste wird losgelassen |
| TextInput, PreviewTextInput | Bubbling, Tunneling | Ein Element empfängt eine Texteingabe |

**Tabelle 12.2** Tastatur-Ereignisse

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In Tabelle 12.3 können Sie schließlich die Ereignisse für den Stift des Tablett-PCs sehen.

| Stift-Ereignis | Weiterleitung | Aktion |
|---|---|---|
| `GotStylusCapture` | Bubbling | Das Element hält den Stiftfokus fest |
| `LostStylusCapture` | Bubbling | Das Element verliert den Stiftfokus |
| `StylusDown, PreviewStylusDown` | Bubbling, Tunneling | Der Stift berührt den Bildschirm über einem Element |
| `StylusUp, PreviewStylusUp` | Bubbling, Tunneling | Der Stift wird vom Bildschirm über einem Element hoch gehoben |
| `StylusEnter` | Direkt | Der Stift wird in das Element hinein bewegt |
| `StylusLeave` | Direkt | Der Stift wird aus dem Element hinaus bewegt |
| `StylusInRange, PreviewStylusInRange` | Bubbling, Tunneling | Der Stift befindet sich dicht über dem Bildschirm |
| `StylusOutOfRange, PreviewStylusOutOfRange` | Bubbling, Tunneling | Der Stift befindet sich außerhalb des Bildschirmerfassungs-abstands |
| `StylusMove, PreviewStylusMove` | Bubbling, Tunneling | Der Stift wird über das Element bewegt |
| `StylusInAirMove, PreviewStylusInAirMove` | Bubbling, Tunneling | Der Stift wird über das Element bewegt, berührt dabei aber nicht den Bildschirm |
| `StylusSystemGesture, PreviewStylusSystemGesture` | Bubbling, Tunneling | Es wird eine Stift-Geste erkannt |
| `TextInput, PreviewTextInput` | Bubbling, Tunneling | Das Element empfängt eine Texteingabe |

**Tabelle 12.3** Stift-Ereignisse

# Schaltflächen

Schaltflächen sind einfache Steuerelemente, die angeklickt werden können. Hierbei unterscheiden wir verschiedene Typen: Neben der Standard-Schaltfläche (`Button`) gibt es in WPF auch noch die Steuerelemente Kontrollkästchen (`CheckBox`) und Auswahlkästchen (`RadioButton`). Die drei Klassen sind von `ButtonBase` abgeleitet und stellen das `Click`-Ereignis zur Verfügung. Die Anwendung dieser Elemente ist sehr einfach:

```
<Button Click="OnButtonClicked" Name="btn">Button</Button>
<CheckBox Click="OnCheckboxClicked" Name="check">CheckBox</CheckBox>
<RadioButton Click="OnRadiobuttonClicked" Name="radio">RadioButton</RadioButton>
```

Die Methode für das `Click`-Ereignis der Schaltfläche kann folgendermaßen implementiert werden:

```
Private Sub OnButtonClick(ByVal sender As Object, ByVal e As RoutedEventArgs)
        MessageBox.Show("Schaltfläche geklickt!")
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Entsprechende Methoden können auch für die beiden anderen Steuerelemente erstellt werden. Wenn Sie aus dem Code auf die Steuerelemente zugreifen wollen, müssen diese mit einem Namen versehen werden.

Die drei Schaltflächenarten können nicht nur einen einfachen Text enthalten, sondern sie können beliebige andere Elemente, auch Grafik, darstellen. Im folgenden Listing 12.9 werden die drei Elemente mit einem erweiterten Inhalt dargestellt.

```
<Window x:Class="ButtonTest.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Test mit Schaltflächen" Height="200" Width="300"
    >
    <Grid>
      <StackPanel Width="150">

        <Button>
          <TextBlock>
            <Ellipse Margin="0,0,5,0" Width="20" Height="10" Fill="Red" Stroke="Black" />
            Button
            <Ellipse Margin="5,0,0,0" Width="20" Height="10" Fill="Green" Stroke="Black" />
          </TextBlock>
        </Button>

        <CheckBox>
          <TextBlock Foreground="White" FontSize="18">
            <TextBlock.Background>
              <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                <GradientStop Color="Blue" Offset="0" />
                <GradientStop Color="Red" Offset="1" />
              </LinearGradientBrush>
            </TextBlock.Background>
            CheckBox
          </TextBlock>
        </CheckBox>

        <RadioButton>
          <TextBlock FontSize="20" FontFamily="Algerian">
            RadioButton
          </TextBlock>
        </RadioButton>

      </StackPanel>
    </Grid>
</Window>
```

**Listing 12.9** Steuerelemente mit erweitertem Inhalt

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 12.10** Steuerelemente mit erweitertem Inhalt

In Abbildung 12.10 sehen Sie das Ergebnis unserer Programmierung. Einige der oben benutzten WPF-Elemente werden Sie in den folgenden Kapiteln noch näher kennen lernen. Das Beispiel soll Ihnen hier nur zeigen, dass es mit WPF sehr einfach ist, auch komplexe grafische Effekte mit den vorhandenen Steuerelementen zu nutzen.

Für alle Schaltflächenarten wird in Listing 12.9 ein `TextBlock`-Element verwendet, um eine weitere Gestaltung und Formatierung durchzuführen. Die normale Schaltfläche wird mit grafischen Grundelementen (Ellipsen) erweitert. Für das `CheckBox`-Element wird ein neuer Hintergrund mit einem Farbverlauf (`LinearGradientBrush`) definiert und für das `RadioButton`-Element wurde eine andere Schriftart ausgewählt.

# Bildlaufleisten und Schieberegler

In WPF finden Sie Steuerelemente, mit denen Sie einen Wert aus einem vorgegebenen Bereich auswählen können. Zu diesen Steuerelementen gehören die Klassen `Slider` und `ScrollBar`. Die Benutzung der `Slider`-Elemente ist in Listing 12.10 dargestellt.

```
<Window x:Class="Schieber.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Schieberegler" Height="280" Width="300"
    >
    <Grid>
      <Canvas>
        <Slider Name="sldVert" Orientation="Vertical" Margin="15,15,15,15"
                Minimum="0" Maximum="100" Height="200" Value="0" ValueChanged="OnValueVert" />
        <Slider Name="sldHorz" Orientation="Horizontal" Margin="60,15,15,15"
                Minimum="0" Maximum="100" Width="200" Value="0" ValueChanged="OnValueHorz" />

        <Label Name="lblHorz" FontSize="15" Margin="120,30,15,15" />
        <Label Name="lblVert" FontSize="15" Margin="30,100,15,15" />
      </Canvas>
    </Grid>
</Window>
```

**Listing 12.10** Slider-Elemente

```
Imports System
Imports System.Windows

Namespace Schieber
    Partial Public Class Window1
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Inherits System.Windows.Window

        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnValueHorz(ByVal sender As Object, ByVal e As RoutedEventArgs)
            lblHorz.Content = sldHorz.Value.ToString
        End Sub

        Private Sub OnValueVert(ByVal sender As Object, ByVal e As RoutedEventArgs)
            lblVert.Content = sldVert.Value.ToString
        End Sub
    End Class
End Namespace
```

**Listing 12.11** Ereignismethoden für die Schieberegler

Im Beispiel werden zwei Schieberegler deklariert, die in horizontaler und vertikaler Richtung verlaufen (Listing 12.10). Dies wird über die Eigenschaft `Orientation` eingestellt. Das hier verwendete `Canvas`-Element dient nur zur Positionierung aller Elemente und muss für unser Beispiel nicht weiter betrachtet werden. Für die Schieberegler werden jeweils ein `Minimum`- und ein `Maximum`-Wert festgelegt. Die Ausgaben der `Value`-Eigenschaft bewegen sich genau in diesem vorgegebenen Bereich. In unserem Beispiel wurden zusätzlich zwei `Label`-Elemente definiert, welche die Eigenschaft `Value` der beiden Schieberegler ausgeben.

Nun wollen wir zwei Ereignismethoden im Code implementieren (Listing 12.11). Das Ereignis `ValueChanged` der beiden Schieberegler wird auf die Methoden `OnValueHorz` und `OnValueVert` geleitet. Dort wird die Ausgabe der `Value`-Eigenschaft der Schieberegler in einen `String` konvertiert und im dazugehörigen `Label`-Element dargestellt. Die `Value`-Eigenschaft des `Slider`-Elements liefert übrigens einen Wert vom Typ `Double` zurück, wie Sie es beim Ausführen des Beispielprogramms sofort erkennen können (Abbildung 12.11).

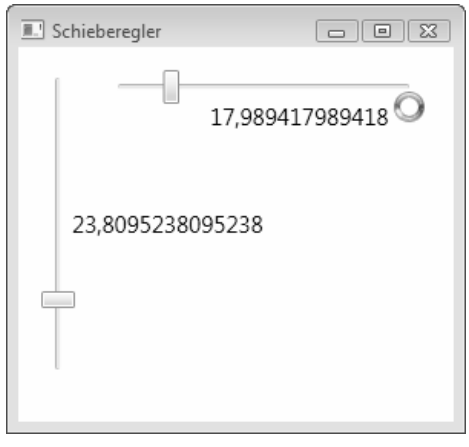



**Abbildung 12.11** Zwei Schieberegler im Einsatz

Mit den Bildlaufleisten (`ScrollBar`) können Sie ganz ähnlich wie mit den Schiebereglern (`Slider`) arbeiten. Bildlaufleiten bieten zusätzlich die Eigenschaften `LargeChange` und `SmallChange` an, mit denen Sie das Blättern durch ein größeres Dokument steuern können.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Ein sehr interessantes Steuerelement von WPF ist das `ScrollViewer`-Element, mit dem es sehr leicht möglich ist, einen Inhalt mit Bildlaufleisten darzustellen, wenn dieser nicht in das Elternelement passt (Listing 12.12). Der darzustellende Inhalt wird als Kindelement im `ScrollViewer`-Element deklariert. Ist das Kindelement größer als der verfügbare Platz, werden die benötigten Bildlaufleisten dargestellt (Abbildung 12.12) und können benutzt werden, ohne dass eine weitere Programmierung erforderlich ist.

```
<Window x:Class="Scroller.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ScrollViewer" Height="300" Width="300"
    >
    <Grid>
      <ScrollViewer HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">
        <Ellipse Width="1000" Height="1000" Fill="Red" Stroke="Black" StrokeThickness="5" />
      </ScrollViewer>
    </Grid>
</Window>
```

**Listing 12.12** Viel Inhalt mit den ScrollViewer-Element

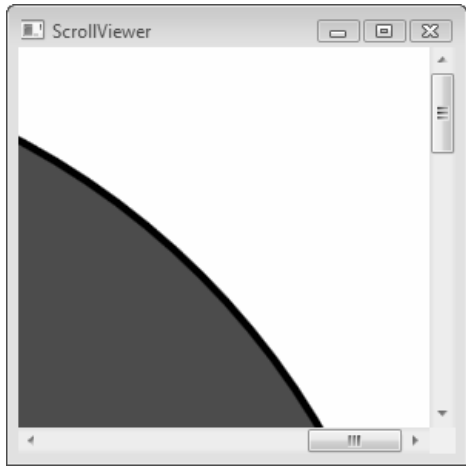



**Abbildung 12.12** Benutzung des ScrollViewer-Elements

Das `ScrollViewer`-Element kann in beliebigen Layout-Elementen eingesetzt werden. So können Sie z.B. die Zelle eines `Grid`-Elements sehr einfach mit Bildlaufleisen versehen. Auch die Ausgabe von Texten kann von einem `ScrollViewer`-Element unterstützt werden. Hierbei ist es meistens sinnvoll, die horizontale Bildlaufleiste auszuschalten und nur die senkrechte Leiste zu benutzen. Listing 12.13 zeigt ein Beispiel mit einem `Grid`-Element, bestehend aus zwei Zeilen und zwei Spalten, in dem eine Ellipse, ein Bild und ein Text in den einzelnen Zellen ausgegeben werden. Für die Ausgabe des Textes in der unteren `Grid`-Zeile wird die horizontale Bildlaufleiste ausgeschaltet und der Zeilenumbruch wird für das `TextBlock`-Element aktiviert.

Versuchen Sie nach dem Start des Beispielprogramms das Fenster zu vergrößern. Sobald der Inhalt vollständig in einer `Grid`-Zelle dargestellt werden kann, verschwinden die Bildlaufleisten (Abbildung 12.13).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
<Window x:Class="Scroller2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ScrollViewer" Height="300" Width="300"
    >
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
      </Grid.RowDefinitions>

      <ScrollViewer Grid.Column="0" Grid.Row="0" HorizontalScrollBarVisibility="Auto"
                    VerticalScrollBarVisibility="Auto">
        <Ellipse Width="200" Height="200" Fill="Red" Stroke="Black" StrokeThickness="3" />
      </ScrollViewer>

      <ScrollViewer Grid.Column="1" Grid.Row="0" HorizontalScrollBarVisibility="Auto"
                    VerticalScrollBarVisibility="Auto">
        <Image Source="IC5146.Jpg" />
      </ScrollViewer>

      <ScrollViewer Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2" VerticalScrollBarVisibility="Auto"
                    HorizontalScrollBarVisibility="Disabled" >
        <TextBlock TextWrapping="Wrap" FontSize="20">
          Hier werden mehrere ScrollViewer-Elemente von Windows Presenation Foundation eingesetzt.
          Es ist nun sehr einfach möglich, große Elemente mit Bildlaufleisten darzustellen.
        </TextBlock>
      </ScrollViewer>
    </Grid>
</Window>
```

**Listing 12.13** ScrollViewer-Elemente in einem Grid


**Abbildung 12.13** Mehrere ScrollViewer-Elemente

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Steuerelemente für die Texteingabe

Das einfachste Steuerelement für die Texteingabe ist das `TextBox`-Element, mit dem Sie normalerweise eine Zeile Text eingeben können. Durch Setzen der Eigenschaft `AcceptsReturn` auf `True` ist es jedoch möglich, mehrere Textzeilen einzugeben. Den eingegebenen Text können Sie über die Eigenschaft `Text` abfragen oder vorgeben.

Eine weitere Variante bei der Texteingabe stellt das `PasswordBox`-Element dar. Die Texte in diesem Steuerelement werden nicht lesbar dargestellt. Außerdem wird dieses Element über die Eigenschaft `Password` abgefragt.

**HINWEIS** Denken Sie daran, dass Kennworte, die in einem normalen `String`-Objekt gespeichert werden, solange im Speicher stehen, bis der Garbage Collector den Speicher löscht bzw. überschreibt. Kennworte sollten darum in `SecureString`-Objekten abgelegt werden. Wenn diese nicht mehr benötigt werden, kann der Speicher mit dem Kennwort sofort gelöscht werden. Außerdem wird das Kennwort im Speicher verschlüsselt abgelegt, sodass es nicht mit einem Debugger oder aus der Swap-Datei ausgelesen werden kann.

Die Anwendung des `TextBox`-Elements wird in Listing 12.14 gezeigt. Wenn Sie die Anwendung starten, können Sie die vorgegebenen Texte modifizieren (Abbildung 12.14). Bei der Eingabe eines Kennworts (ganz unten) wird mit der Eigenschaft `PasswordChar` das Zeichen »#« gesetzt. Dies ist das Zeichen, welches als Platzhalter für die Kennwortzeichen im `PasswordBox`-Element ausgegeben wird.

```
<Window x:Class="TextBoxen.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Textboxen" Height="300" Width="300"
    >
    <Grid>
      <StackPanel Orientation="Vertical">
        // Einfache TextBox
        <Label Margin="5">Einfache TextBox:</Label>
        <TextBox Name="text1" Margin="5">Einfache TextBox</TextBox>

        // Mehrzeilige TextBox
        <Label Margin="5">TextBox mit mehreren Zeilen:</Label>
        <TextBox Name="text2" AcceptsReturn="True" Margin="5" Height="70">Mehrere Zeilen</TextBox>

        // TextBox für ein Kennwort
        <Label Margin="5">Kennwort-Eingabe:</Label>
        <PasswordBox Name="text3" PasswordChar="#" Margin="5" />
      </StackPanel>
    </Grid>
</Window>
```

**Listing 12.14** Verschiedene Texteingaben

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 12.14** TextBox-Elemente und PasswordBox-Element

Ein `TextBox`-Element erlaubt nur die einfache Eingabe von Buchstaben. Wesentlich flexibler ist dagegen das `RichBoxText`-Element. Hier können Texte und grafische Elemente gemischt werden (Listing 12.15).

```
<Window x:Class="RichText.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="RichTextBox" Height="140" Width="500"
    >
    <Grid>
      <RichTextBox>
        <FlowDocument>
          <Paragraph>
            <Run xml:space="preserve" FontSize="20">Ein RichTextBox-Element </Run>
            <Ellipse Width="50" Height="20" Fill="Red" Stroke="Black" />
            <Run xml:space="preserve" FontSize="20"> kann neben einem Text auch </Run>
            <Run FontWeight="Bold" FontStyle="Italic" FontSize="20">grafische Elemente</Run>
            <Run xml:space="preserve" FontSize="20"> enthalten.</Run>
            <Polyline Stroke="Green" StrokeThickness="3" Points="0,0 20,20 40,0 60,20 80,0 100,20" />
          </Paragraph>
        </FlowDocument>
      </RichTextBox>
    </Grid>
</Window>
```

**Listing 12.15** Text und Grafik in einem RichTextBox-Element

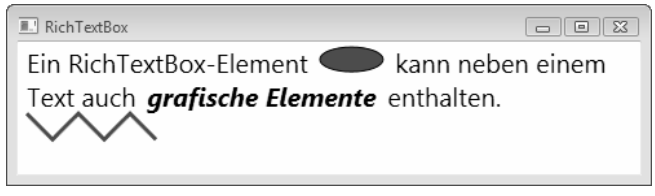



**Abbildung 12.15** Das RichTextBox-Element mit Text und Grafik

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Innerhalb des RichTextBox-Elementes haben wir zunächst ein FlowDocument-Element deklariert, welches dann ggf. mehrere Paragraph-Elemente enthalten kann. Jedes Paragraph-Element stellt einen kompletten Abschnitt des Dokuments dar und kann beliebige andere Elemente enthalten. Dazu gehören Texte, grafische Elemente und auch normale Steuerelemente. Texte werden in Run-Elementen vorgegeben, die sehr vielfältige Formatierungsanweisungen enthalten können. Um Zeichnungen innerhalb der Texte darzustellen, werden an den entsprechenden Stellen einfach die normalen grafischen WPF-Elemente benutzt (Listing 12.15).

Sie können die einzelnen Elemente, die in einem RichTextBox-Element enthalten sind, auch zur Laufzeit modifizieren. Hierzu müssen Sie die Elemente, die verändert werden sollen, mit einem Namen versehen. Danach können Sie aus dem Code ganz normal auf die Elemente zugreifen. Listing 12.16 zeigt den XAML-Teil eines Beispiels.

```
<Window x:Class="RichText2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Änderbares Rechteck" Height="190" Width="400"
    >
    <Grid>
      <StackPanel Orientation="Vertical">
        // Eingabe von Breite und Höhe des Rechtecks
        <StackPanel Orientation="Horizontal">
          <Label FontSize="20">Breite:</Label>
          <TextBox Name="textBreite" FontSize="20" Width="100" TextChanged="OnTextChanged">50</TextBox>
          <Label FontSize="20">Höhe:</Label>
          <TextBox Name="textHoehe" FontSize="20" Width="100" TextChanged="OnTextChanged">20</TextBox>
        </StackPanel>

        // Text mit einem Rechteck
        <RichTextBox Margin="10" Height="100">
          <FlowDocument>
            <Paragraph>
              <Run xml:space="preserve" FontSize="20">Hier ist ein </Run>
              <Rectangle Name="rect" Width="50" Height="20" Fill="Red" Stroke="Black"
                  StrokeThickness="2" />
              <Run FontSize="20">, welches in Breite und Höhe geändert werden kann.</Run>
            </Paragraph>
          </FlowDocument>
        </RichTextBox>
      </StackPanel>
    </Grid>
</Window>
```

**Listing 12.16** Ein Text mit einem variablen Rechteck

Zunächst erzeugen wir zwei TextBox-Elemente mit den Namen »textBreite« und »textHoehe« im oberen Bereich eines StackPanel-Elements. Das TextChanged-Ereignis dieser Eingabeelemente wird an die Methode OnTextChanged gebunden (Listing 12.17). In unteren Teil deklarieren wir ein RichTextBox-Element, welches einen Text und das zu steuernde Rechteck mit dem Namen »rect« enthält.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Imports System
Imports System.Windows

Namespace RichText2
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnTextChanged(ByVal sender As Object, _
                                  ByVal e As RoutedEventArgs)

            ' Prüfen, ob alle Elemente vorhanden sind
            If textBreite Is Nothing Or _
            textHoehe Is Nothing Or _
            rect Is Nothing Then Return

            Dim dBreite As Double
            Dim DHoehe As Double
            Dim bTest As Boolean

            ' Fehlerhafte Eingaben werden mit TryParse erkannt
            bTest = Double.TryParse(textBreite.Text, dBreite)
            bTest = Double.TryParse(textHoehe.Text, DHoehe)

            ' Alle Werte sind OK: Rechteck ändern
            If bTest And dBreite > 0.0 And DHoehe > 0.0 Then
                rect.Width = dBreite
                rect.Height = DHoehe
            End If

        End Sub
    End Class
End Namespace
```

**Listing 12.17** Steuerung des Rechtecks im RichTextBox-Element

Wenn wir die Ereignismethode programmieren, müssen wir allerdings ein bisschen aufpassen. Zuerst wird in der logischen Hierarchie die `TextBox` für die Breite deklariert und dementsprechend wird sie zur Laufzeit des Programms auch als erste erzeugt und dargestellt. Nun wird bei der Initialisierung dieser `TextBox` im XAML-Code die Zahl »50« zugewiesen und dadurch das `TextChanged`-Ereignis ausgelöst. In der Ereignismethode müssen wir also berücksichtigen, dass beim ersten Aufruf die zweite `TextBox` (Höhe) und das Rechteck im `RichTextBox`-Element noch gar nicht existieren. Darum werden in der Ereignismethode die einzelnen Objekte auf `nothing` getestet.

Außerdem müssen wir damit rechnen, dass ein Anwender unseres Programms fehlerhafte Eingaben tätigt. Dazu gehören negative Zahlen oder auch Buchstaben. Aus diesem Grund wird die Methode `TryParse` aus dem Typ `Double` benutzt, um die Eingaben zu prüfen, bevor das Rechteck manipuliert wird (Abbildung 12.16).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

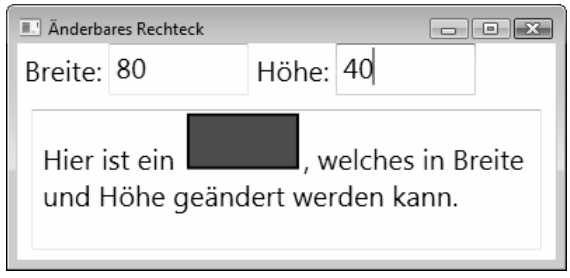



**Abbildung 12.16** Das Rechteck wurde geändert

# Das Label-Element

Das `Label`-Element haben Sie in den vorangegangenen Beispielen schon kennen gelernt. Es dient in erster Linie dazu, vor anderen Steuerelementen einen Text auszugeben, wie in Listing 12.16 bereits gezeigt wurde. Im ersten Moment sieht es so aus, als ob das `Label`-Element überflüssig wäre und durch eine einfache `TextBox` simuliert werden könnte. Bei einem `Label` kommt jedoch als neue Möglichkeit die Verarbeitung des Fokus hinzu.

Sie sollten ihre Benutzerschnittstellen immer so aufbauen, dass der Anwender auch mit der Tastatur die verschiedenen Steuerelemente in einem Dialogfenster ansteuern kann. Hierbei spielt das `Label`-Element eine große Rolle. Sie können für jedes `Label` eine Kurztaste (Access Key) definieren, die der Anwender in Kombination mit der Alt-Taste zur Ansteuerung des `Label`-Elements verwenden kann. Da nun das `Label` selbst jedoch nicht den Eingabefokus erhalten kann, wird dieser an das im `Label` gebundene Element weitergegeben.

In Listing 12.18 wird statt dem normalen Text im `Label` ein `AccessText`-Element benutzt. Der Buchstabe hinter dem Unterstreichungszeichen ist die Kurztaste für das `Label`-Element. Sie können also den Eingabefokus der `TextBox` für die Breite zuordnen, indem Sie die Tastenkombination Alt B eingeben. Das Zielelement, dem der Eingabefokus zugewiesen werden soll, wird in der `Target`-Eigenschaft des `Label`-Elements angegeben. Hier müssen Sie folgende Syntax benutzen:

```
Target="{Binding ElementName=textBreite}"
```

Als `ElementName` wird das Steuerelement angegeben, welches den Eingabefokus enthalten soll.

**HINWEIS** Die Kurztasten werden in der Benutzerschnittstelle erst dann durch die Unterstreichungszeichen gekennzeichnet, wenn Sie die Alt-Taste drücken (Abbildung 12.17).

Wie Sie in Listing 12.18 sehen können, ist das Element `AccessText` nicht unbedingt erforderlich, um die Kurztaste zu definieren. Das `AccessText`-Element ist dann sehr nützlich, wenn Sie die Textausgabe formatieren oder die vielfältigen Möglichkeiten mit Animationen benutzen wollen.

```
<Window x:Class="Labels.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Label" Height="100" Width="450"
    >
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Grid>
      <StackPanel Orientation="Horizontal" Height="40">
        <Label FontSize="20" Target="{Binding ElementName=textBreite}">
          <AccessText>_Breite:</AccessText>
        </Label>
        <TextBox Name="textBreite" FontSize="20" Width="100" ToopTip="Breitenangabe">50</TextBox>

        <Label FontSize="20" Target="{Binding ElementName=textHoehe}">
          _Höhe:
        </Label>
        <TextBox Name="textHoehe" FontSize="20" Width="100" ToopTip="Höhenangabe">20</TextBox>

        <Button FontSize="20" Margin="5,0,0,0" Click="OnAusgabe" ToopTip="Ausgabe der Daten">
          <AccessText>_Ausgabe...</AccessText>
        </Button>
      </StackPanel>
    </Grid>
</Window>
```

**Listing 12.18** Label-Elemente und Schaltflächen mit Kurztasten


**Abbildung 12.17** Die dargestellten Kurztasten in der Benutzerschnittstelle

Natürlich können Sie die Reihenfolge der Steuerelemente für das Drücken der Tabulator-Taste über die Eigenschaft `TabIndex` festlegen, welche in jedem Steuerelement vorhanden ist. Außerdem haben wir in Listing 12.18 die Eigenschaft `ToolTip` verwendet, um kleine Hilfstexte auf den Steuerelementen darzustellen, wenn Sie mit der Maus darüber verweilen.

# Menüs

Viele Windows-Applikationen werden durch hierarchisch angeordnete Menüs gesteuert. Wir unterscheiden zwischen *Popup-Menüs*, die am oberen Rand eines Fensters angeordnet sind (Hauptmenü) und den *Kontext-Menüs*, die dann erscheinen, wenn Sie mit der linken Maustaste auf ein Steuerelement klicken. Ein Menü enthält im Allgemeinen mehrere `MenuItem`-Elemente.

Das folgende Beispiel (Listing 12.19) zeigt den Code für ein einfaches Hauptmenü eines noch einfacheren Textverarbeitungsprogramms. In diesem Menü wurden alle gängigen Befehle einer Anwendung dargestellt, jedoch hier nicht immer implementiert.

```
<Window x:Class="TestMenue.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Ein Menü" Height="300" Width="400"
    >
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Window.Resources>
      // Bild für Befehl "Neu"
      <Path x:Key="imgNeu" Fill="White" Stroke="Black" StrokeThickness="1"
            Data="M 3,3 L 3,15 14,15 14,3 Z M 10,3 L 14,7" />

      // Bild für Befehl "Beenden"
      <Path x:Key="imgBeenden" Fill="Red" Stroke="Black" StrokeThickness="1"
        Data="M 7,13 L 2,7 5,7 5,0 9,0 9,7 12,7 z" />
    </Window.Resources>

    <DockPanel>
      <Menu DockPanel.Dock="Top">
        <MenuItem Header="_Datei">
          <MenuItem Header="_Neu..." Command="New" InputGestureText="Strg+N" Icon="{StaticResource
imgNeu}" />
          <Separator />
          <MenuItem Header="_Öffnen..." />
          <MenuItem Header="_Speichern" />
          <MenuItem Header="Speichern _unter..." />
          <Separator />
          <MenuItem Header="_Beenden" Click="OnBeenden" Icon="{StaticResource imgBeenden}" />
        </MenuItem>

        <MenuItem Header="_Bearbeiten">
          <MenuItem Header="_Ausschneiden" Command="Cut" InputGestureText="Strg+X" />
          <MenuItem Header="_Kopieren" Command="Copy" InputGestureText="Strg+C" />
          <MenuItem Header="_Einfügen" Command="Paste" InputGestureText="Strg+V" />
          <Separator />
          <MenuItem Header="_Alles markieren" Command="SelectAll" InputGestureText="Strg+A" />
        </MenuItem>

        <MenuItem Header="_Hilfe">
          <MenuItem Header="_Hilfe..." />
          <Separator />
          <MenuItem Header="_Über..." />
        </MenuItem>
      </Menu>

      <TextBox Name="text" AcceptsReturn="True" TextWrapping="Wrap" VerticalScrollBarVisibility="Auto"
/>
    </DockPanel>
</Window>
```

**Listing 12.19** Ein einfaches Hauptmenü

Damit das Hauptmenü an der oberen Fensterkante anliegt, benutzen wir als Layout-Element ein `DockPanel` und setzen die Eigenschaft `DockPanel.Dock` auf den Wert `Top`. Nun können wir `MenuItem`-Elemente in das Hauptmenü einfügen. Jeder Menüpunkt wird über mehrere wichtige Eigenschaften bestimmt. Zunächst wird die Eigenschaft `Header` benutzt, um den Text für den Menüpunkt festzulegen. In diesem Text können Sie wiederum ein Unterstreichungszeichen benutzen, um die Kurztaste (in Kombination mit der Alt-Taste) für den Menüpunkt festzulegen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Da ein Menüpunkt normalerweise einen Befehl in der Anwendung auslöst, müssen wir eine Verbindung mit den entsprechenden Ereignismethoden herstellen. Die kann entweder über die Eigenschaft `Command` mit einem `CommandBinding`-Objekt oder durch direktes Binden des `Click`-Ereignisses an die Ereignismethode realisiert werden. Im Beispiel wurden beide Methoden benutzt (Listing 12.20). Die benötigten `CommandBinding`-Objekte werden im Konstruktor-Code des Fensters implementiert. Die einzelnen Ereignismethoden, die an die Menübefehle gebunden sind, führen sehr einfachen Code aus und müssen nicht weiter erläutert werden.

Wenn Sie den Text für die Kurztaste des Menübefehls ändern wollen, benutzen Sie hierzu die Eigenschaft `InputGestureText`.

Horizontale Trennelemente zwischen zwei Menüpunkten werden durch ein `Separator`-Element dargestellt.

Oft möchten wir im linken Bereich des Menüpunktes ein kleines Bild (Icon) angezeigen, welches dann auch auf den Schaltflächen der Werkzeugleiste benutzt werden kann. Hier müssen wir etwas vorgreifen, denn wir benutzen »Ressourcen«, um diese Bilder zu definieren. Im Block `Window.Resources` deklarieren wir zwei `Path`-Elemente, denen wir über die Eigenschaft `Key` einen Namen zuweisen.

Nun können wir die definierten Grafiken über die Eigenschaft `Icon` des `MenuItem`-Elements zuweisen. Die Bilder sollten die Größe 17x17 Einheiten nicht überschreiten, ansonsten wird der Menüpunkt sehr hoch bzw. breit.

**HINWEIS** Sie können natürlich auch Pixel-Grafiken (BMP, JPG, o.ä.) über die `Icon`-Eigenschaft in einem Menüpunkt darstellen. Verwenden Sie dann ein `Image`-Element für die Eigenschaft `MenuItem.Icon`. Ggf. müssen Sie auch hier die Größe der Grafik mit einem `Viewbox`-Element anpassen.

Nach dem Hauptmenü deklarieren wir noch ein `TextBox`-Element mit dem Namen »text«, das die Textverarbeitungsmöglichkeit realisieren soll. Die Befehle *Einfügen*, *Kopieren*, *Ausschneiden* und *Alles markieren* werden auf diese `TextBox` angewendet, um eine möglichst einfache Implementierung zu erhalten.

Starten Sie nun die Anwendung, und »spielen« Sie etwas mit den Befehlen in den Menüs *Datei* und *Bearbeiten*. Beobachten Sie, wie die Menüpunkte durch die implementierten `CommandBinding`-Elemente ein- und ausgeschaltet werden. Benutzen Sie auch die möglichen Kurztasten, die wir für das Menü definiert haben. Die laufende Anwendung ist in Abbildung 12.18 zu sehen.

```
Imports System
Imports System.Windows
Imports System.Windows.Input

Namespace TestMenue
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
            text.Focus()

            ' Befehl: Neu
            Dim cbNeu As CommandBinding
            cbNeu = New CommandBinding(ApplicationCommands.[New])
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        AddHandler cbNeu.Executed, AddressOf OnNeu
        AddHandler cbNeu.CanExecute, AddressOf OnNeuCanExecute
        Me.CommandBindings.Add(cbNeu)

        ' Befehl: Ausschneiden
        Dim cbAusschneiden As CommandBinding
        cbAusschneiden = New CommandBinding(ApplicationCommands.Cut)
        AddHandler cbAusschneiden.Executed, AddressOf OnAusschneiden
        AddHandler cbAusschneiden.CanExecute, AddressOf OnAusschneidenCanExecute
        Me.CommandBindings.Add(cbAusschneiden)

        ' Befehl: Einfügen
        Dim cbEinfuegen As CommandBinding
        cbEinfuegen = New CommandBinding(ApplicationCommands.Paste)
        AddHandler cbEinfuegen.Executed, AddressOf OnEinfuegen
        AddHandler cbEinfuegen.CanExecute, AddressOf OnEinfuegenCanExecute
        Me.CommandBindings.Add(cbEinfuegen)

        ' Befehl: Kopieren
        Dim cbKopieren As CommandBinding
        cbKopieren = New CommandBinding(ApplicationCommands.Copy)
        AddHandler cbKopieren.Executed, AddressOf OnKopieren
        AddHandler cbKopieren.CanExecute, AddressOf OnAusschneidenCanExecute
        Me.CommandBindings.Add(cbKopieren)
    End Sub

    Private Sub OnNeu(ByVal Sender As Object, ByVal e As RoutedEventArgs)
        ' Text löschen
        text.Clear()
    End Sub

    Private Sub OnNeuCanExecute(ByVal sender As Object, _
                                ByVal e As CanExecuteRoutedEventArgs)
        ' Neu-Befehl nur ausführen, wenn TextBox einen Text enthält
        e.CanExecute = (text.Text.Length > 0)
    End Sub

    Private Sub OnBeenden(ByVal sender As Object, _
                          ByVal e As RoutedEventArgs)
        'Fenster schließen
        Me.Close()
    End Sub

    Private Sub OnAusschneiden(ByVal sender As Object, _
                               ByVal e As RoutedEventArgs)
        ' Befehl: Ausschneiden
        text.Cut()
    End Sub

    Private Sub OnAusschneidenCanExecute(ByVal sender As Object, _
                                         ByVal e As CanExecuteRoutedEventArgs)
        e.CanExecute = (text.Text.Length > 0)
    End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private Sub OnEinfuegen(ByVal sender As Object, _
                                ByVal e As RoutedEventArgs)
            ' Befehl: Einfügen
            text.Paste()
        End Sub

        Private Sub OnEinfuegenCanExecute(ByVal sender As Object, _
                                          ByVal e As CanExecuteRoutedEventArgs)
            ' Einfügen-Befehl nur ausführen, wenn Text in der Zwischenablage ist
            e.CanExecute = Clipboard.ContainsText
        End Sub

        Private Sub OnKopieren(ByVal sender As Object, _
                               ByVal e As RoutedEventArgs)
            ' Befehl: Kopieren
            text.Copy()
        End Sub

        Private Sub OnAllesMarkieren(ByVal sender As Object, _
                                     ByVal e As RoutedEventArgs)
            'Befehl: Alles markieren
            text.SelectAll()
        End Sub
    End Class
End Namespace
```

**Listing 12.20** Implementierung der Befehle im Hauptmenü


**Abbildung 12.18** Hauptmenü mit Bildern und Kurztasten

Die Benutzung von kontextsensitiven Menüs funktioniert ganz ähnlich wie die Deklaration eines Hauptmenüs. Die gewünschten Menüpunkte werden in dem Steuerelement als `ContextMenu`-Element deklariert, in dem sie benutzt werden sollen. Listing 12.21 zeigt die Erweiterung des letzten Beispiels. Durch Anklicken der `TextBox` mit der rechten Maustaste erscheint das Menü mit den beiden Befehlen *Einfügen* und *Neu*.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
…
        Private Sub OnSuchenStart(ByVal sender As Object, _
                                  ByVal e As RoutedEventArgs)
            Dim iPos As Integer
            iPos = text.SelectionStart + text.SelectionLength
            Dim strText As String
            strText = text.Text
            iPos = strText.IndexOf(textSuchen.Text, iPos)
            If iPos >= 0 Then
                text.Select(iPos, textSuchen.Text.Length)
            Else
                text.Select(0, 0)
            End If
        End Sub
…
```

**Listing 12.21** Erweiterter XAML-Code für ein kontextsensitives Menü im TextBox-Element

Der implementierte Code bleibt unverändert, da die Menüpunkte aus dem Kontextmenü über `CommandBinding`-Elemente an die entsprechenden Ereignismethoden gebunden sind. Auch die Aktivierung und Deaktivierung der Menüpunkte funktioniert durch diese Bindung wie erwartet. Natürlich können Sie auch in einem Kontextmenü kleine Bildchen auf der linken Seite darstellen, in dem Sie die `Icon`-Eigenschaft entsprechend setzen.

# Werkzeugleisten (Toolbars)

Ein ebenfalls häufig verwendetes Element in Benutzerschnittstellen von Windows-Anwendungen ist die Werkzeugleiste (Toolbar). Die WPF-Werkzeugleiste kann nicht nur Schaltflächen, sondern auch andere Steuerelemente, wie `TextBox`- oder `ComboBox`-Elemente, enthalten. Wir wollen das letzte Beispiel mit dem Menü nun mit einer Werkzeugleiste erweitern. Dazu implementieren wir den folgenden XAML-Code zur Definition von `Menu`- und `TextBox`-Element aus dem vorherigen Beispiel (Listing 12.22):

```
...
<ToolBarTray DockPanel.Dock="Top">
  <ToolBar>
    <Button Command="New" Content="{StaticResource imgNeu}" />
    <Separator />
    <Label>_Suchen:</Label>
    <TextBox Name="textSuchen" Width="100" />
    <Button Click="OnSuchenStart">
      <AccessText>_Start</AccessText>
    </Button>
    <Separator />
    <Button Click="OnBeenden" Content="{StaticResource imgBeenden}" />
  </ToolBar>
</ToolBarTray>
...
```

**Listing 12.22** Die Anwendung mit einer Werkzeugleiste

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wir beginnen mit einem ToolBarTray-Element, welches das Layout der gesamten Werkzeugleiste kontrolliert. Damit die Werkzeugleiste direkt unter dem Hauptmenü erscheint, setzen wir die Eigenschaft DockPanel.Dock wieder auf den Wert Top. Im Layout-Element platzieren wir jetzt ein ToolBar-Element, in dem dann die verschiedenen Steuerelemente der Werkzeugleiste positioniert werden können.

Zunächst benötigen wir eine Schaltfläche für den *Neu*-Befehl und verwenden das bereits definierte Bildchen aus den vorhandenen Windows-Ressourcen. Danach kommt eine senkrechte Trennlinie (Separator).

Wir wollen den Text in unserer TextBox nach einem bestimmten Text durchsuchen können. Dazu erzeugen wir im ToolBar-Element nun erst ein Label-Element mit dem Text *Suchen*, dann eine TextBox mit einer festen Breite und schließlich eine Schaltfläche mit der Aufschrift *Start*. Diese Schaltfläche ist mit der Ereignismethode OnSuchenStart (Listing 12.23) verbunden. Nach einer weiteren Trennlinie wird noch eine Schaltfläche deklariert, mit der das Programm beendet werden kann.

Auch für die Werkzeugleiste können wir die bereits existierenden CommandBinding-Elemente oder Ereignismethoden verwenden.

Wir müssen also nur die Ereignismethode OnSuchenStart implementieren (Listing 12.23). Damit wir ein mehrfaches Auftreten des eingegebenen Suchtextes ermitteln können, starten wir die Suche am Ende des selektierten Bereichs (Variable iPos) im TextBox-Element mit dem Namen »text«. Wenn kein Text selektiert ist, entspricht dies der Cursor-Position in der TextBox. Eine Referenz auf den Text wird aus dieser TextBox in eine String-Variable übernommen und mit der Methode IndexOf nach dem Suchtext aus der TextBox der Werkzeugleiste durchsucht. Das Ergebnis der Suchmethode IndexOf wird wieder als Startposition für die nächste Suche verwendet. Somit kann der Suchtext auch mehrfach vorkommen. Die einzelnen Positionen werden dann nacheinander selektiert. Wenn der Suchtext nicht gefunden wird, wird der Cursor in TextBox »text« an Position 0 gesetzt (Abbildung 12.19).

```
...
 Private Sub OnSuchenStart(ByVal sender As Object, _
                           ByVal e As RoutedEventArgs)
            Dim iPos As Integer
            iPos = text.SelectionStart + text.SelectionLength
            Dim strText As String
            strText = text.Text
            iPos = strText.IndexOf(textSuchen.Text, iPos)
            If iPos >= 0 Then
                text.Select(iPos, textSuchen.Text.Length)
            Else
                text.Select(0, 0)
            End If
        End Sub
...
```

**Listing 12.23** Der Zusatzcode für den Suchen-Befehl

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

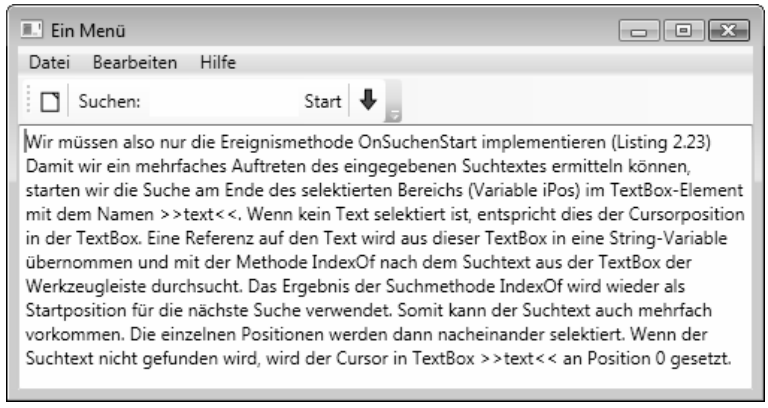



**Abbildung 12.19** Der Text-Editor mit einer Werkzeugleiste

Mehrere Werkzeugleisten können nacheinander in einem `ToolBarTray`-Element deklariert werden (siehe Abbildung 12.20):

```
<ToolBarTray DockPanel.Dock="Top">
  <ToolBar>
  <!-- Die erste Toolbar -->
  </ToolBar>
  <!-- Die zweite Toolbar -->
  <ToolBar>
    <Button Command="Cut">Ausschneiden</Button>
    <Button Command="Copy">Kopieren</Button>
    <Button Command="Paste">Einfügen</Button>
  </ToolBar>
</ToolBarTray>
```

Die einzelnen Werkzeugleisten können Sie nebeneinander oder untereinander anordnen. Eine Verschiebung an den rechten, linken oder unteren Rand des Fensters ist in dieser Anwendung jedoch nicht möglich.

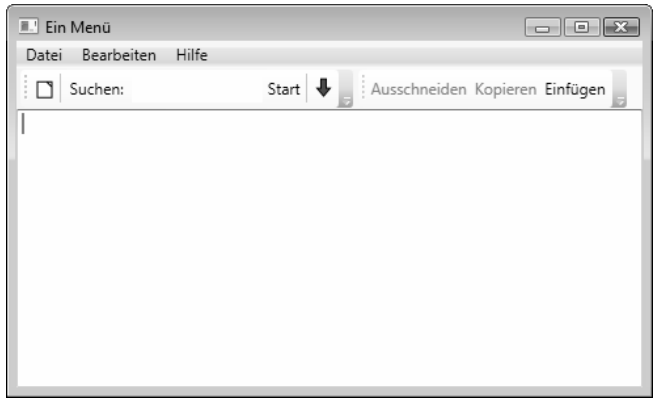



**Abbildung 12.20** Zwei Werkzeugleisten in der Anwendung

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Zusammenfassung

Steuerelemente sind in jedem Framework für Benutzerschnittstellen vorhanden, so auch in Windows Presentation Foundation. Auch hier werden Steuerelemente über Klassen implementiert, die über Eigenschaften und Methoden verfügen. Diese nutzt der Programmierer, um die Steuerelemente zu kontrollieren.

Wichtig für Steuerelemente sind die Konzepte *Routed Events* (weitergeleitete Ereignisse) und *Routed Commands* (weitergeleitete Befehle). Diese Konzepte erleichtern die Programmierung von hierarchischen Benutzerschnittstellen erheblich. Auch die Weitergabe von Eigenschaften stellt eine große Erleichterung für den Softwareentwickler dar.

Denken Sie auch daran, dass WPF-Steuerelemente eine Trennung von Design und Logik durchführen, sodass das Aussehen von Steuerelementen sehr leicht geändert werden kann. Diese Möglichkeit betrachten wir in einem späteren Kapitel noch genauer.
Die einzelnen Steuerelemente von WPF wurden hier nicht im Detail durchgesprochen. Das würde den Rahmen dieses Buches leider sprengen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 13

# Layout

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Layout ist ein wichtiges und sehr umfangreiches Feature von Windows Presentation Foundation (WPF). Ohne die Möglichkeiten des Layouts wäre es nur sehr schwer möglich, Applikationen mit Dialogfeldern oder Fenstern, die in der Größe änderbar sind, zu programmieren. In herkömmlichen Windows-Frameworks (z.B. MFC, Windows Forms) werden die einzelnen Steuerelemente eines Dialogfeldes oft mit fest vorgegebenen Positionen und Größen angelegt. Dieser Weg wird in WPF selten verwendet, da es die Möglichkeit gibt, alle Steuerelemente eines Fensters mithilfe von Layout-Steuerelementen so zu verwalten, dass eine variable Position und Größe ohne wesentlichen Programmieraufwand erzielbar sind.

In Windows Presentation Foundation wird das Layout der Steuerelemente in einem Fenster über so genannte Panels gesteuert. Je nach Art des verwendeten Panels werden die enthaltenen Steuerelemente angeordnet. Die Position des jeweiligen Elements wird durch das Panel bestimmt, welches das Element enthält. Einige Panel-Elemente verwalten auch die Größe ihrer Kindelemente. In WPF stehen diverse Steuerelemente zur Verfügung, die das Layout beeinflussen (Tabelle 13.1):

| Panel | Anwendung |
|---|---|
| `Canvas` | Einfaches Element für die genaue Positionierung von Kindelementen. |
| `StackPanel` | Ordnet die Kindelemente vertikal oder horizontal an. |
| `DockPanel` | Ordnet die Kindelemente am Rand des Elternelements an. |
| `TabPanel` | Ordnet die Kindelemente als einzelne umschaltbare Seiten an. |
| `WrapPanel` | Ordnet die Kindelemente in einer Zeile an. Wenn der Platz in der Zeile nicht reicht, findet ein Zeilenumbruch statt. |
| `Viewbox` | Darstellung von Grafiken. |
| `Grid` | Ordnet die Kindelemente in einer Tabelle mit Zeilen und Spalten an. |

**Tabelle 13.1** Die vorhandenen Panel-Typen

Wir wollen die Möglichkeiten dieser Layout-Steuerelemente nun im Einzelnen betrachten.

**BEGLEITDATEIEN** Unter .*Samples**Chapter13*\ finden Sie die Beispieldateien für dieses Kapitel.

# Das StackPanel

Das Panel, welches am einfachsten zu benutzen ist, heißt `StackPanel`. Mit einem `StackPanel` können Sie die Kindelemente entweder nebeneinander oder untereinander anordnen. Bei einer vertikalen Anordnung der Elemente im `StackPanel` werden die Standardhöhen der Kindelemente für die Positionierung verwendet. Dies bedeutet, dass die Höhe eines Elements aus der Höhe des Inhalts bestimmt wird, es sei denn, Sie geben explizit einen Wert für die Höhe des Elements an. In einem horizontalen `StackPanel` werden die Standardbreiten der Kindelemente angewendet. Somit ergibt sich die Breite aus der Breite des Inhaltselements. Natürlich können Sie auch hier die Breite selbst angeben. In diesem Fall werden die Eigenschaften innerhalb der Hierarchie nicht weitergegeben. Die Kindelemente, die in einem Layout-Panel angelegt werden, müssen nicht vom gleichen Typ sein. Sie können also unterschiedliche Steuerelemente im Panel beliebig mischen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Im folgenden Beispiel werden verschiedene Steuerelemente (`TextBlock`, `Button`, `CheckBox`,…) untereinander angeordnet.

```
<Window x:Class="StackPanel1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Vertikales StackPanel" Height="300" Width="300"
    >
  <StackPanel>
    <TextBlock>Hier steht ein Text!</TextBlock>
    <Button>Die erste Schaltfläche</Button>
    <CheckBox>Wählen Sie hier aus</CheckBox>
    <Button>Klicken Sie hier!</Button>
    <TextBlock>Auch hier steht was...</TextBlock>
    <TextBox>Eingabe hier!</TextBox>
  </StackPanel>
</Window>
```

**Listing 13.1** Ein StackPanel mit vertikaler Anordnung der Elemente

Das `StackPanel` in Listing 13.1 ordnet die Steuerelemente folgendermaßen an:

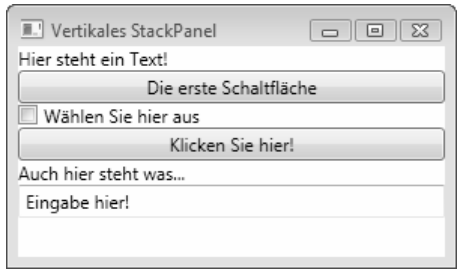



**Abbildung 13.1** Vertikale Anordnung im StackPanel

Im Beispiel aus Listing 13.1 müssen wir beachten, dass das `StackPanel` die Breite und Höhe des Anwendungsfensters annimmt. Weiterhin nehmen die Kindelemente im `StackPanel` wiederum genau diese Breite an. Für die jeweilige Höhe der einzelnen Elemente wird die Standardhöhe verwendet, die sich aus der Höhe des Inhalts eines Kindelementes ergibt. Die Standardhöhe kann für die verschiedenen Elemente durchaus einen unterschiedlichen Wert annehmen.

Wird nun das Fenster in der Breite verändert, dann passt sich das `StackPanel` an die neue Breite an. Die Kindelemente des `StackPanel` passen sich dann ebenfalls an diese Breite an. Verkleinern Sie dagegen die Höhe des Fensters, so werden die Kindelemente des `StackPanel` ebenfalls verkleinert und irgendwann einfach abgeschnitten bzw. nicht mehr dargestellt.

Mit einer kleinen Änderung im Listing 13.1 können Sie das `StackPanel` dazu bewegen, seine Kindelemente in horizontaler Reihenfolge anzuordnen:

```
<!-- Wie in Listing 1.1 -->
<StackPanel Orientation="Horizontal">
<!-- Weiter wie in Listing 1.1 -->
```

**Listing 13.2** Horizontale Anordnung im StackPanel

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

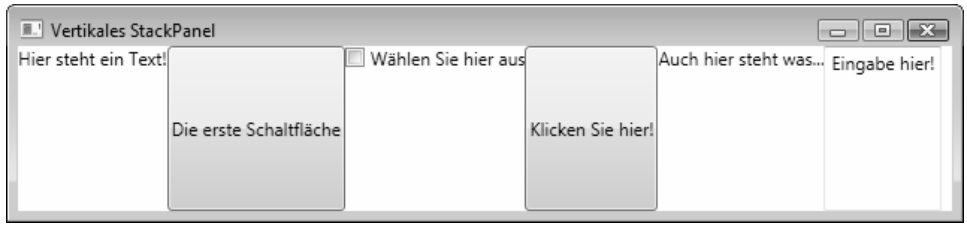



**Abbildung 13.2** Ein horizontales StackPanel

In Abbildung 13.2 ist deutlich zu sehen, dass die Höhen der Kindelemente nun durch die Höhe des `StackPanel` (welches das Fenster vollständig ausfüllt) bestimmt werden. Die Breite der einzelnen Elemente ist durch ihre Standardbreite gegeben. Sie können gut erkennen, dass sich diese Breite aus der Breite des Inhalts des Elements ergibt. So ist die erste Schaltfläche ganz links etwas breiter als die zweite Schaltfläche, da die Texte unterschiedliche Breiten haben. Wird die Breite des Fensters und damit des `StackPanel` verkleinert, werden die Elemente rechts nach und nach abgeschnitten und nicht mehr dargestellt. Dieses Verhalten können Sie jedoch durch eine explizite Angabe von Breiten bzw. Höhen für die Kindelemente beeinflussen.

# Das DockPanel

Ein `DockPanel`-Element ermöglicht es, das Gesamt-Layout eines Teils der Benutzerschnittstelle zu definieren. Sie können angeben, an welchem Rand (oben, unten, rechts oder links) ein Kindelement angeordnet werden soll. Werden mehrere Elemente am gleichen Rand angelegt, so werden diese Elemente neben- oder untereinander, ähnlich wie in einem `StackPanel`, angeordnet.

**HINWEIS** Wenn nicht anders angegeben, werden beim Andocken an das Elternelement die jeweiligen Standardbreiten und -höhen verwendet. Sie können natürlich explizit Breiten und Höhen für die Kindelemente angeben, die dann auch entsprechend verwendet werden.

```
<Window x:Class="DockPanel1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="DockPanel #1" Height="300" Width="300"
    >
  <DockPanel>
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

**Listing 13.3** Einfaches Docking mit dem DockPanel

Wenn sie das Beispiel aus Listing 13.3 ausführen, werden Sie sehen, dass die Schaltfläche, die zuletzt erzeugt wird (Schaltfläche mit dem Inhalt *Rechts*), den Bereich des `DockPanel` vollständig auffüllt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 13.3** Das DockPanel aus Listing 13.3

Für jede Schaltfläche wird angegeben, an welchem Rand des `DockPanel` sie angedockt werden soll. Hierzu wird die spezielle Eigenschaft `Dock` aus dem `DockPanel`-Element verwendet:

```
DockPanel.Dock="Right"
```

Im Beispiel aus Listing 13.1 werden die Schaltflächen, die oben und unten angedockt werden, in der Breite des `DockPanel` dargestellt. Die beiden Schaltflächen rechts und links werden in die verbleibende Höhe eingepasst. Die linke Schaltfläche hat die Standardbreite und die rechte Schaltfläche füllt schließlich den verbleibenden Rest des `DockPanel`-Elements auf.

**HINWEIS** Wenn Sie die Größe des Fensters aus Listing 13.1 ändern, bleibt die Anordnung der Schaltflächen im `DockPanel` erhalten. Nur die jeweiligen Breiten und Höhen, die nicht fest vorgegeben wurden, werden entsprechend der Fenstergröße neu berechnet und dargestellt.

Wenn Sie das Auffüllen durch das letzte Kindelement verhindern, müssen Sie das Attribut `LastChildFill` auf `False` setzen (Listing 13.4). Die letzte Schaltfläche, die im `DockPanel` dargestellt wird, erhält dann einfach die Standardbreite. Ein Teil des `DockPanel`-Elements bleibt dann frei.

```
<Window x:Class="DockPanel2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="DockPanel #2" Height="300" Width="300"
    >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

**Listing 13.4** DockPanel ohne »Auffüllen«

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 13.4** Ein nicht vollständig gefülltes DockPanel

Im nächsten Beispiel soll die Schaltfläche *Links* nun nicht mehr die gesamte Höhe des `DockPanel`-Elements ausfüllen. In diesem Fall (Listing 13.5) müssen wir die Höhe für diese Schaltfläche explizit angeben. Die Schaltfläche wird dann in der Mitte des zur Verfügung stehenden Bereiches dargestellt.

```
<Window x:Class="DockPanel3.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="DockPanel #3" Height="300" Width="300"
    >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left" Height="22">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

**Listing 13.5** Eine Schaltfläche mit fest definierter Höhe im DockPanel


**Abbildung 13.5** Die Schaltfläche steht in der Mitte des Bereiches, der im DockPanel zur Verfügung steht

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Um die Position der Schaltfläche zu modifizieren, können wir die Attribute `HorizontalAlignment` und `VerticalAlignment` entsprechend einsetzen. Im folgenden Code-Beispiel soll die Schaltfläche mit dem Text *Links* nun weiter unten, aber direkt oberhalb der Schaltfläche mit dem Text *Unten* angeordnet werden (Listing 13.6 und Abbildung 13.6).

```
<Window x:Class="DockPanel4.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="DockPanel #4" Height="300" Width="300"
    >
    <DockPanel LastChildFill="False">
      <Button DockPanel.Dock="Top">Oben</Button>
      <Button DockPanel.Dock="Bottom">Unten</Button>
      <Button DockPanel.Dock="Left" Height="22" VerticalAlignment="Bottom">Links</Button>
      <Button DockPanel.Dock="Right">Rechts</Button>
    </DockPanel>
</Window>
```

**Listing 13.6** Die Schaltfläche ist jetzt unten angeordnet

Sie erkennen sicherlich schon, wie leistungsfähig selbst einfache Layout-Elemente sind, wenn die Darstellung der Kindelemente über die verschiedenen Attribute angepasst wird.


**Abbildung 13.6** Die kleine Schaltfläche ist unten links angeordnet

Die Darstellung der Kindelemente im `DockPanel` ist natürlich von der Reihenfolge abhängig, in der die Elemente erzeugt werden. Gehen Sie vom Beispiel in Listing 13.4 aus. Nun sollen zuerst die beiden Schaltflächen auf der rechten und der linken Seite erzeugt werden. Wir erhalten dann den in Abbildung 13.5 gezeigten neuen Aufbau des Fensters.

```
<Window x:Class="DockPanel5.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="DockPanel #5" Height="300" Width="300"
    >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Left">Links</Button>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Button DockPanel.Dock="Right">Rechts</Button>
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
  </DockPanel>
</Window>
```

**Listing 13.7** Die Reihenfolge der Definition ist entscheidend für den Aufbau


**Abbildung 13.7** Zuerst werden die beiden Schaltflächen rechts und links erzeugt

**HINWEIS** Die Elemente in einem `DockPanel` überlappen sich niemals. Wenn einzelne Elemente in der geforderten Größe nicht dargestellt werden können, wird dieser Teil einfach abgeschnitten.

Die Kindelemente im `DockPanel` entscheiden teilweise selbst, in welcher Größe sie sich darstellen müssen. Wenn eine Schaltfläche oben oder unten im `DockPanel` platziert wird, so entscheidet das `DockPanel` über die Breite der Schaltfläche oder Sie müssen die Breite des Kindelements extra angeben. Das Kindelement, im Beispiel die Schaltfläche, entscheidet aber ggf. selbst, wie hoch das Element sein soll. Wird die Schaltfläche rechts oder links angedockt, so verhält es sich umgekehrt. Die Höhe wird vom `DockPanel` bestimmt und die Breite wird vom Kindelement vorgegeben. Hierbei spielt dann der Text auf der Schaltfläche eine entscheidende Rolle. Wenn möglich, wird die Breite so gesetzt, dass der gesamte Text auf der Schaltfläche sichtbar ist. Wenn dies nicht möglich ist, wird ein Teil des Kindelementes abgeschnitten.

# Das Grid

Das `Grid` ist ein sehr mächtiges Layout-Steuerelement. Es bietet die Möglichkeit, die Kindelemente in einer oder mehreren Zeilen und Spalten anzuordnen. Sie werden sehen, dass die Höhe der Zeilen und die Breite der Spalten im `Grid` sehr leicht konfigurierbar sind.

Das einfachste `Grid`-Element enthält nur eine Zelle, also eine Zeile und eine Spalte. In dieser `Grid`-Zelle kann ein Kindelement platziert werden. Dieses Element wird standardmäßig in der Mitte der Zelle dargestellt (Listing 13.8).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
<Grid>
  <Button Width="100" Height="30">Test</Button>
</Grid>
```

**Listing 13.8** Das einfachste Grid mit einer Zelle

Um nun mehrere Spalten und Zeilen im `Grid` darzustellen, müssen diese in den Bereichen `Grid.ColumnDefinitions` bzw. in den `Grid.RowDefinitions` angelegt werden. Im Beispiel aus Listing 13.9 wird ein `Grid` mit zwei Spalten und drei Zeilen erzeugt. In den einzelnen Zellen werden `TextBlock`-Elemente für die Ausgabe von Texten benutzt. Beachten Sie hierbei, dass die `TextBlock`-Elemente links oben mit der Ausgabe von Texten beginnen. Diese Standardeinstellung können Sie über die Attribute `HorizontalAlignment` und `VerticalAlignment` des `TextBlock`-Elements ändern. In diesem Beispiel wollen wir mithilfe des `Grid`-Attributs `ShowGridLines="True"` die Trennlinien zwischen den Zellen im `Grid` darstellen (Abbildung 13.8). Für jedes `TextBlock`-Element wird über die Attribute `Grid.Column` und `Grid.Row` (aus der `Grid`-Klasse) festgelegt, in welcher Zeile und Spalte es dargestellt werden soll.

```
<Window x:Class="Grid.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Grid #1"
    Height="300" Width="300">

  <Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>

    <TextBlock Grid.Column="0" Grid.Row="0">Name:</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="0">Vorname:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="1">Heckhuis</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="1">Jürgen</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="2">Löffelmann</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="2">Klaus</TextBlock>
  </Grid>
</Window>
```

**Listing 13.9** Ein Grid mit mehreren Spalten und Zeilen

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

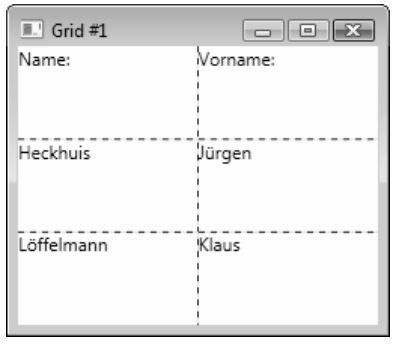



**Abbildung 13.8** Ein Grid mit mehreren Spalten und Zeilen

Beachten Sie bitte auch das Verhalten des `Grid`-Elements beim Ändern der Größe des Fensters. Das `Grid` füllt in diesem Beispiel immer das gesamte Fenster aus. Zeilen und Spalten werden entsprechend in der Höhe und der Breite angepasst, so dass die Höhen- und Breitenverhältnisse im `Grid`-Element erhalten bleiben.

Nun ist es im Normalfall nicht damit getan, einfach nur einige Zeilen und Spalten zu definieren. Im nächsten Beispiel werden darum die Breiten und Höhen der Spalten und Zeilen im `Grid`-Element konfiguriert. Dies wird ebenfalls in den jeweiligen `RowDefinition`- und `ColumnDefinition`-Elementen durchgeführt.

Es gibt drei Möglichkeiten, die Höhe und Breite der Zellen zu definieren. Um Spalten (und Zeilen) mit einer festen Breite (Höhe) zu erzeugen, wird im `ColumnDefinition`-Element (`RowDefinition`) einfach die gewünschte Pixelanzahl angegeben:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="35" />
</Grid.ColumnDefinitions>
```

Wird in einer `ColumnDefinition` der Wert `Auto` angegeben, so wird die Breite der Spalte aus der maximalen Breite der Kindelemente bestimmt, die in der Spalte enthalten sind:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="Auto" />
</Grid.ColumnDefinition>
```

Die dritte Variante der Breitenangabe in einer `ColumnDefinition` erstellt Spalten, deren Breite sich am vorhandenen Platz im Elternfenster orientiert. In diesem Fall wird als Parameter ein »*« verwendet:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

Im letzten Beispiel wird ein `Grid` mit einer Spalte erzeugt, welche die gesamte Breite des Elternelements bekommt. Wird das Elternelement in der Breite verändert, so wird die Spaltenbreite entsprechend angepasst. Das ist aber noch nicht alles. Betrachten Sie folgendes Beispiel:

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="2*" />
  <ColumnDefinition Width="4*" />
  <ColumnDefinition Width="0.5*" />
<Grid.ColumnDefinitions>
```

Hier erhalten Sie ein `Grid` mit vier Spalten, die sich an die Breite des Elternelements anpassen. Das besondere ist jedoch, dass die Breiten der einzelnen Spalten in dem Verhältnis angelegt werden, die in den `Width`-Attributen angegeben sind. Die Summe aller Breiten ist »7.5*« und entspricht der Gesamtbreite des `Grid`-Elements. Wenn die Gesamtbreite im `Grid` nun zum Beispiel 300 Pixel beträgt, dann haben die einzelnen Spalten folgende Breiten:

| Spalte-Nr. | Breitenangabe | Breite in Pixel |
|---|---|---|
| 1 | * | 40 |
| 2 | 2* | 80 |
| 3 | 4* | 160 |
| 4 | 0.5* | 20 |
| **Summe:** | **7.5*** | **300** |

Wird die Breite dieses `Grid`-Elements geändert, z.B. durch eine Änderung der Fenstergröße, so bleibt das Breitenverhältnis der einzelnen `Grid`-Spalten jedoch erhalten.

Alles was hier über die Definition von Spaltenbreiten geschrieben wurde, gilt ebenfalls für die Definition von Zeilenhöhen im `Grid`. Das entsprechende Definitionselement heißt dann `RowDefinition`. Auch hier ist es möglich, feste, automatisch angepasste und im Verhältnis angepasste Höhen für eine Zeile anzugeben. In Listing 13.10 wird ein vollständiges Beispiel mit einem konfigurierten `Grid` gezeigt. Interessant ist hier die Breitenermittlung für die erste Spalte ganz links. Für diese Spalte wird die Einstellung `Auto` verwendet. Somit wird die Breite des größten Elements dieser Spalte benutzt. Hier ist das die Breite der Schaltfläche in der zweiten Zeile.

```
<Window x:Class="Grid2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Grid #2" Height="300" Width="300"
    >
  <Grid ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="2*" />
      <RowDefinition Height="50" />
      <RowDefinition />
    </Grid.RowDefinitions>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="0.5*" />
      <ColumnDefinition Width="50" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Button Grid.Column="0" Grid.Row="0">Test</Button>
    <Button Grid.Column="0" Grid.Row="1">Ein großer Button</Button>
    <TextBlock Grid.Column="0" Grid.Row="2">Auto</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="2">*</TextBlock>
    <TextBlock Grid.Column="2" Grid.Row="2">0.5*</TextBlock>
    <TextBlock Grid.Column="3" Grid.Row="2">Fest 50</TextBlock>
    <TextBlock Grid.Column="4" Grid.Row="2">
      Keine<LineBreak />Angabe
    </TextBlock>
  </Grid>
</Window>
```

**Listing 13.10** Ein Grid-Element mit diversen Breiten- und Höhendefinitionen


**Abbildung 13.9** Das Grid aus Listing 13.10

Oft ist es erforderlich, dass ein Element im `Grid` mehrere Zeilen oder Spalten belegt. Hierfür stehen in der `Grid`-Klasse die beiden Eigenschaften `Grid.RowSpan` und `Grid.ColumnSpan` zur Verfügung. Mit diesen beiden Eigenschaften ist es sehr leicht möglich, ein `Grid`-Element zur beliebigen Positionierung von anderen Elementen zu benutzen und über mehrere Zellen zu verteilen. Die linken Ränder der `Grid`-Spalten haben dann in etwa das Verhalten von definierten Tabulatorpositionen.

Das Beispiel in Listing 13.11 zeigt die Positionierung von mehreren `TextBlock`-Elementen in einem `Grid`. Für die einzelnen `TextBlock`-Elemente werden an verschiedenen Stellen die Attribute `Grid.ColumnSpan` und `Grid.RowSpan` verwendet, damit die längeren Textinhalte auch vollständig ausgegeben werden. Abbildung 13.10 zeigt die Ausgabe von Listing 13.11 in einem Fenster. Die Attribute `Grid.RowSpan` und `Grid.ColumnSpan` können natürlich bei beliebigen Kindelementen angewendet werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Beachten Sie, dass bei einem `TextBlock`-Element der Text nur dann über mehrere Zeilen (`Grid.RowSpan`) verteilt wird, wenn das Attribut `TextWrapping="Wrap"` benutzt wird.

```
<Window x:Class="Grid3.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Grid #3" Height="300" Width="300"
    >
    <Grid ShowGridLines="true">
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="30" />
        <RowDefinition Height="30" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="20" />
        <ColumnDefinition Width="20" />
        <ColumnDefinition Width="20" />
        <ColumnDefinition Width="20" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Grid.Column="0" Grid.Row="0">Name:</TextBlock>
      <TextBlock Grid.Column="0" Grid.Row="1">Vorname:</TextBlock>
      <TextBlock Grid.Column="0" Grid.Row="2">PLZ und Ort:</TextBlock>
      <TextBlock Grid.Column="0" Grid.Row="3">Strasse und Nr.:</TextBlock>
      <TextBlock Grid.Column="0" Grid.Row="4">Telefon:</TextBlock>
      <TextBlock Grid.Column="0" Grid.Row="5">Angaben:</TextBlock>

      <TextBlock Grid.Column="1" Grid.Row="0" Grid.ColumnSpan="5">Heckhuis</TextBlock>
      <TextBlock Grid.Column="2" Grid.Row="1" Grid.ColumnSpan="5">Jürgen</TextBlock>
      <TextBlock Grid.Column="3" Grid.Row="2" Grid.ColumnSpan="5">12345 Wpfstadt</TextBlock>
      <TextBlock Grid.Column="4" Grid.Row="3" Grid.ColumnSpan="5">Avalonstrasse 22</TextBlock>
      <TextBlock Grid.Column="5" Grid.Row="4">01234-567890</TextBlock>
      <TextBlock Grid.Column="1" Grid.Row="5" Grid.ColumnSpan="4" Grid.RowSpan="3" TextWrapping="Wrap">
        Hier können weitere Angaben zur Person eingetragen werden. Diese Angaben sind natürlich
freiwillig.
      </TextBlock>
    </Grid>
</Window>
```

**Listing 13.11** Ein Grid zur Positionierung von Texten, die mehrere Spalten und Zeilen überspannen

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**WICHTIG** Bedenken Sie, dass die vorgestellten Layout-Elemente in einer beliebigen Hierarchie verwendet werden können. Es ist möglich, in der Zelle eines komplexen `Grid`-Elementes ein `DockPanel` oder `StackPanel` zu platzieren. Ebenso kann man in eine `Grid`-Zelle ein weiteres `Grid` einfügen und konfigurieren.

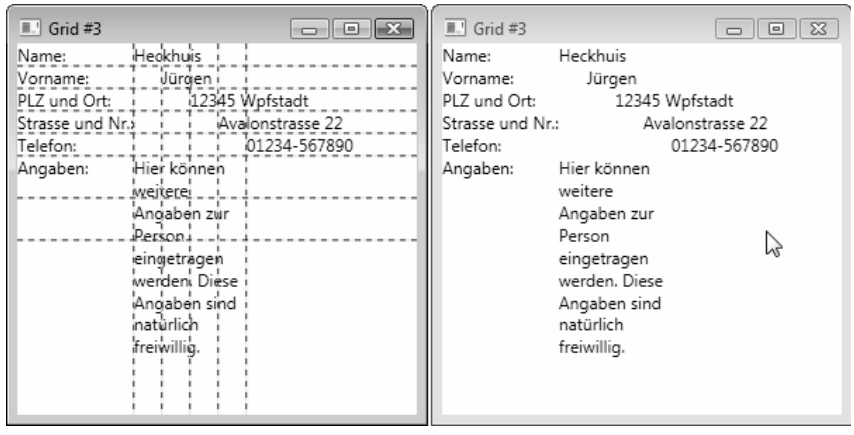



**Abbildung 13.10**
Das Grid aus Listing 13.11 – mit und ohne Trennlinien

# Das GridSplitter-Element

Beim Einsatz eines `Grid`-Elementes taucht natürlich sofort die Frage auf, ob die Breite der Spalten bzw. die Höhe der Zeilen zur Laufzeit einfach änderbar sind. Um das zu ermöglichen, können Sie ein oder mehrere `GridSplitter`-Elemente einsetzen. Eine einfache Anwendung des `GridSplitter`-Elements zeigt Listing 13.12:

```
<Window x:Class="GridSplitter.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="GridSplitter" Height="120" Width="300"
    >
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
        <Button Grid.Column="0">Button Nr. 1</Button>
        <Button Grid.Column="1">Button Nr. 2</Button>
        <GridSplitter Grid.Column="1" Width="5" Background="Blue" HorizontalAlignment="Left"/>
    </Grid>
</Window>
```

**Listing 13.12** Grid mit einem GridSplitter-Element

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der Bereich des `GridSplitter`-Elements ist in Abbildung 13.11 gut erkennbar in blau sichtbar. Wenn Sie die Maus über dieses Element bewegen, ändert sich auch der Mauszeiger entsprechend. Bei einer Verschiebung des `GridSplitter`-Elements wird in diesem Fall eine Schaltfläche größer und die andere kleiner. Im Beispiel haben wir ein vertikales Trennelement, welches nach der `Grid`-Spalte mit dem Index »0« eingefügt werden soll.

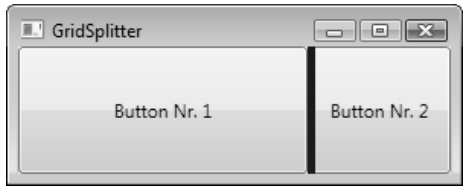



**Abbildung 13.11** Das Grid mit nach rechts verschobenem GridSplitter-Element

Welcher Wert für die Eigenschaft `Grid.Column` angegeben werden muss, hängt allerdings auch noch von der Eigenschaft `HorizontalAlignment` des `GridSplitter`-Elements ab. Der Standardwert für diese Eigenschaft ist `Right`. In diesem Fall wird `Grid.Column` auf den Wert 0 gesetzt, wie in Listing 13.12 gezeigt wurde. Das `GridSplitter`-Element ist dann ein Teil der linken Grid-Zelle. Wird die Eigenschaft `HorizontalAlignment` auf den Wert `Left` eingestellt, dann ist das Trennelement ein Teil der linken Grid-Zelle und die Eigenschaft `Grid.Column` muss auf den Wert 1 gesetzt werden, damit die Trennung zwischen den beiden Schaltflächen erfolgt.

```
<GridSplitter Grid.Column="1" Width="5" Background="Blue" HorizontalAlignment="Left"/>
```

**HINWEIS** Beachten Sie bitte, dass ein `GridSplitter`-Element das Element in der Zelle (hier die Schaltfläche) teilweise überdeckt. Das ist besser erkennbar, wenn man die Breite des Trennelements vergrößert. Die Ereignisbearbeitung für die Schaltflächen wird aber nur dann ausgelöst, wenn Sie in den Bereich klicken, der als Schaltfläche auch tatsächlich sichtbar ist.

Das nächste Beispiel (Listing 13.13) zeigt den Einsatz mehrerer `GridSplitter`-Elemente, um alle Zeilen und Spalten eines `Grid`-Steuerelements in der Breite und in der Höhe zu modifizieren.

```
<Window x:Class="GridSplitter2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="GridSplitter #2" Height="250" Width="300"
    >
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
      </Grid.RowDefinitions>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
      <Button Grid.Column="0" Grid.Row="0">1 - 1</Button>
      <Button Grid.Column="0" Grid.Row="1">1 - 2</Button>
      <Button Grid.Column="0" Grid.Row="2">1 - 3</Button>
      <Button Grid.Column="1" Grid.Row="0">2 - 1</Button>
      <Button Grid.Column="1" Grid.Row="1">2 - 2</Button>
      <Button Grid.Column="1" Grid.Row="2">2 - 3</Button>
      <Button Grid.Column="2" Grid.Row="0">3 - 1</Button>
      <Button Grid.Column="2" Grid.Row="1">3 - 2</Button>
      <Button Grid.Column="2" Grid.Row="2">3 - 3</Button>
      <GridSplitter Width="6" Grid.Column="0" Grid.Row="0" Grid.RowSpan="3" Background="Blue"
                    ResizeDirection="Columns" />
      <GridSplitter Width="6" Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" Background="Blue"
                    ResizeDirection="Columns" />
      <GridSplitter Height="6" Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="3" Background="Red"
                    ResizeDirection="Rows" HorizontalAlignment="Stretch" VerticalAlignment="Bottom" />
      <GridSplitter Height="6" Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="3" Background="Red"
                    ResizeDirection="Rows" HorizontalAlignment="Stretch" VerticalAlignment="Bottom" />
    </Grid>
</Window>
```

**Listing 13.13** Ein Grid mit horizontalen und vertikalen GridSplitter-Elementen

Nach der Definition von Spalten und Zeilen für das `Grid`-Element wird für jede Zelle eine Schaltfläche angelegt. Danach werden zuerst zwei vertikale und dann zwei horizontale `GridSplitter`-Elemente definiert. Für die vertikalen Trennelemente wird die Eigenschaft `ResizeDirection` auf den Wert `Columns` gesetzt. Dies ist die Standardeinstellung für diese Eigenschaft. Die horizontalen Trennelemente verwenden den Wert `Rows` für diese Eigenschaft. Hier müssen außerdem die Eigenschaften `HorizontalAlignment` und `VerticalAlignment` angepasst werden. Für die vertikalen Trennelemente können dagegen die Standardeinstellungen benutzt werden. Da die Trennelemente über das gesamte `Grid` hinweg sichtbar und anwendbar sein sollen, müssen die Eigenschaften `Grid.RowSpan` und `Grid.ColumnSpan` entsprechend auf den Wert 3 gesetzt werden. Das Resultat zeigt Abbildung 13.12. Sie können nun Breite und Höhe aller Zellen im `Grid` mit der Maus beeinflussen.

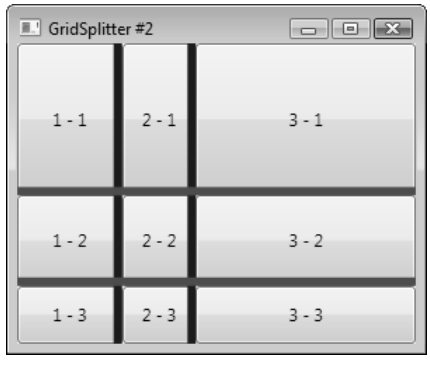



**Abbildung 13.12** Vertikale und horizontale GridSplitter-Elemente im Einsatz

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Das UniformGrid

Im `UniformGrid`-Element haben alle Spalten die gleiche Breite und alle Zeilen die gleiche Höhe. Die Kindelemente werden der Reihe nach in die `Grid`-Zellen eingefügt. Es beginnt mit Zeile 1, Spalte 1. Danach wird das Element in Zeile 1, Spalte 2 eingefügt. Das geht so weiter, bis die erste Zeile komplett mit den Kindelementen belegt ist. Nun wird mit der zweiten Zeile im `UniformGrid` fortgefahren. Im `UniformGrid`-Element gibt es keine Eigenschaften, die eine direkte Adressierung einer Zelle (wie beim normalen `Grid`-Element) ermöglichen.

```
<Window x:Class="UniformGrid.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="UniformGrid" Height="300" Width="300"
    >
  <UniformGrid Columns="4" Rows="3">
    <Button>1, 1</Button>
    <Button>2, 1</Button>
    <Button>3, 1</Button>
    <Button>4, 1</Button>
    <Button>1, 2</Button>
    <Button>2, 2</Button>
    <Button>3, 2</Button>
    <Button>4, 2</Button>
    <Button>1, 3</Button>
    <Button>2, 3</Button>
    <Button>3, 3</Button>
    <Button>4, 3</Button>
  </UniformGrid>
</Window>
```

**Listing 13.14** Ein UniformGrid mit vier Spalten und drei Zeilen

Listing 13.14 zeigt ein `UniformGrid`-Element mit vier Spalten und drei Zeilen aus Abbildung 13.13. Fügt man im Beispiel aus Listing 13.14 weitere Kindelemente in das Grid ein, so werden diese nicht dargestellt, da alle Zeilen und Spalten schon belegt sind. Das `UniformGrid` verhält sich jedoch anders, wenn eine feste Breite und Höhe für das Element vorgegeben wird. In diesem Fall werden auch die Kindelemente dargestellt, welche die vorgegebene Spalten- bzw. Zeilenanzahl überschreiten.

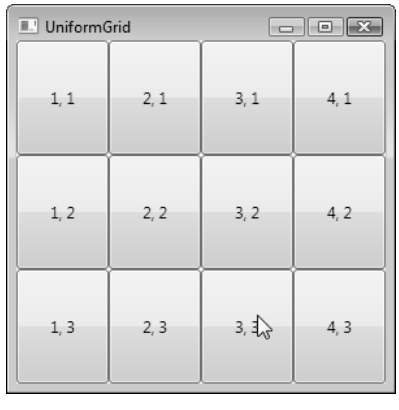



**Abbildung 13.13** Ein UniformGrid mit gleicher Zeilenhöhe und Spaltenbreite für alle Zellen

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Das Canvas-Element

Viele Layout-Anforderungen können mit den bereits vorgestellten Panels erfüllt werden. Was nun noch fehlt, ist ein Element, welches eine präzise Positionsvorgabe für seine Kindelemente ermöglicht. Oft ist es erforderlich, einzelne Grafikelemente genau an einer bestimmten Position auszugeben. Die Position der Grafiken soll nicht durch das automatische Layout beeinflusst werden. In diesem Fall müssen Sie ein `Canvas`-Element benutzen.

Eigentlich verhält sich das `Canvas`-Element sehr einfach. Es ermöglicht die genaue Positionierung von Elementen in Abhängigkeit von den Ecken des Elements. Ein `Canvas`-Element funktioniert etwa so, wie man früher mit Visual Basic 6.0 oder den Microsoft Foundation Classes (MFC) die Steuerelemente in einem Dialogfeld fest positioniert hat. Bei der Positionierung im `Canvas` müssen wir also die Koordinaten relativ zu den Ecken des `Canvas`-Elements angeben. Hierzu gibt es vier »angehängte« Eigenschaften (attached properties) in der `Canvas`-Klasse: `Top`, `Left`, `Bottom` und `Right`. Das folgende Beispiel (Listing 13.15) zeigt die Anwendung des `Canvas` bei der Positionierung von vier Schaltflächen.

```
<Window x:Class="Canvas.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Canvas" Height="300" Width="300"
    >
    <Canvas>
      <Button Canvas.Left="20" Canvas.Top="20">Button 1</Button>
      <Button Canvas.Right="20" Canvas.Top="20">Button 2</Button>
      <Button Canvas.Left="20" Canvas.Bottom="20">Button 3</Button>
      <Button Canvas.Left="200" Canvas.Top="200">Button 4</Button>
    </Canvas>
</Window>
```

**Listing 13.15** Vier Schaltflächen im Canvas-Element

Beachten Sie bei diesem Beispiel die Bewegung der Schaltflächen beim Verändern der Fenstergröße. Je nachdem, an welche Eigenschaft im `Canvas` die Position des Kindelements gebunden wird, bleibt die Position relativ zur linken, oberen Fensterecke gleich (*Button 1* und *Button 4*) oder sie verschiebt sich (*Button 2* und *Button 3*).

Hier drängt sich sofort ein Vergleich zur `Anchor`-Eigenschaft bei WindowsForms-Steuerelementen auf. Das `Canvas`-Element verhält sich jedoch etwas anders: Wenn man z.B. `Canvas.Left` und `Canvas.Right` an die Schaltfläche bindet, so wird nicht – wie in WindowsForms – das Steuerelement beim Aufziehen des Fensters verbreitert, sondern es wird eine der beiden Eigenschaften einfach ignoriert und die Größe der Schaltfläche bleibt unverändert. Um das `Anchor`-Verhalten nachzubauen, müssen Sie zusätzlich ein `Grid`-Element verwenden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

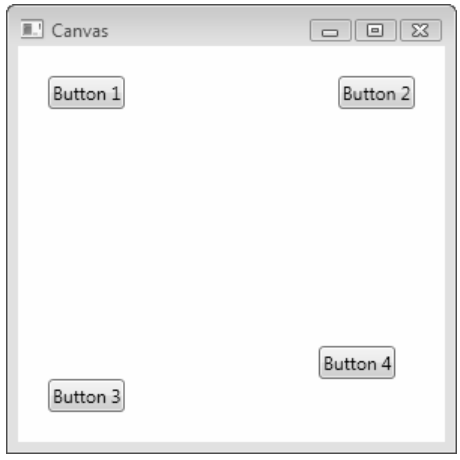



**Abbildung 13.14** Vier Schaltflächen im Canvas-Element

Das `Canvas`-Element dient in erster Linie zur exakten Positionierung von Grafiken. Wenn ein Kindelement im `Canvas` zu groß ist, so wird der überstehende Teil normalerweise nicht abgeschnitten. Dies geschieht erst, wenn die Eigenschaft `ClipToBounds` des `Canvas` auf `True` gesetzt wird.

# Das Viewbox-Element

Um Grafiken in der Größe anzupassen, gibt es das `Viewbox`-Element. Es ist möglich, eine beliebige Grafik automatisch an die Größe der `Viewbox` anzupassen. Dabei können verschiedene Varianten gewählt werden. Einerseits kann das Seitenverhältnis der Grafik beibehalten werden oder die Grafik so skaliert werden, dass sie das gesamte Elternelement ausfüllt. Dieses Verhalten der `Viewbox` wird mit der Eigenschaft `Stretch` gesteuert.

```
<Window x:Class="ViewBox1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Viewbox" Height="300" Width="300"
    >
    <Viewbox Stretch="None">
      <!-- Stretch="Fill" oder Stretch="Uniform" oder Stretch="UniformToFill" -->
      <Canvas Width="30" Height="30">
        <Ellipse Canvas.Left="1" Canvas.Top="1"
                 Width="14" Height="14"
                 Fill="Red" Stroke="Black" />
        <Ellipse Canvas.Left="1" Canvas.Bottom="1"
                 Width="14" Height="14"
                 Fill="Green" Stroke="Black" />
        <Ellipse Canvas.Right="1" Canvas.Top="1"
                 Width="14" Height="14"
                 Fill="Blue" Stroke="Black" />
        <Ellipse Canvas.Right="1" Canvas.Bottom="1"
                 Width="14" Height="14"
                 Fill="Yellow" Stroke="Black" />
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
      </Canvas>
    </Viewbox>
</Window>
```

**Listing 13.16** Benutzung der Viewbox

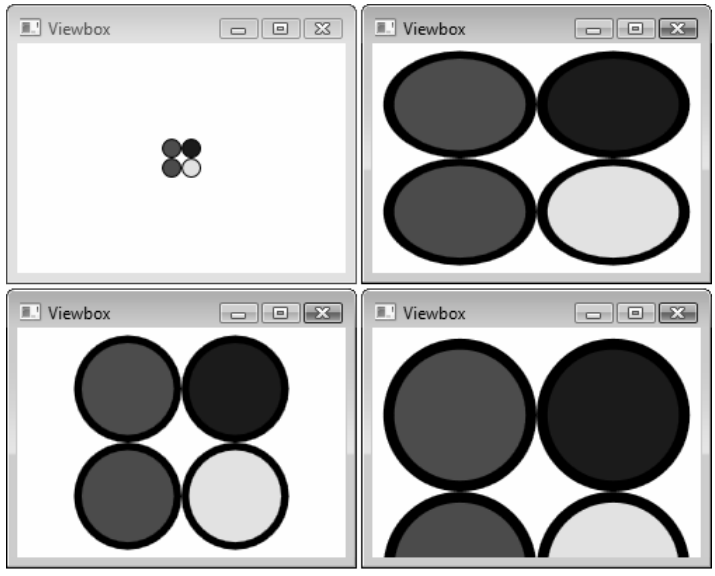



**Abbildung 13.15** Grafikausgabe mit der Viewbox

Abbildung 13.15 zeigt die verschiedenen Möglichkeiten mit dem `Viewbox`-Element. Für das Fenster oben links wurde das `Stretch`-Attribut auf `None` gesetzt. In diesem Fall wird die Grafik in der Originalgröße ausgegeben. Im Fenster rechts oben wurde das Attribut `Stretch` auf `Fill` eingestellt. In der unteren Reihe wurde links der Wert `Uniform` und für das rechte Fenster der Wert `UniformToFill` verwendet. Beim Wert `Uniform` wird das Seitenverhältnis der Grafik beibehalten und diese wird vollständig ausgegeben. Wird die Größe des Fensters geändert, dann wird, je nach eingestelltem `Stretch`-Attribut, die Ausgabe der Grafik angepasst.

In einer `Viewbox` kann man nicht nur ein `Canvas`-Element skalieren, sondern auch alle anderen WPF-Elemente. Die Qualität beim Vergrößern oder Verkleinern von beliebigen Elementen ist dabei sehr gut, da alles als Vektorgrafik dargestellt wird (Abbildung 13.16). Bei der Skalierung von Pixel-Grafiken können jedoch Qualitätsverluste auftreten, wenn die Vergrößerungswerte sehr groß werden oder die Grafik eine geringe Auflösung hat.

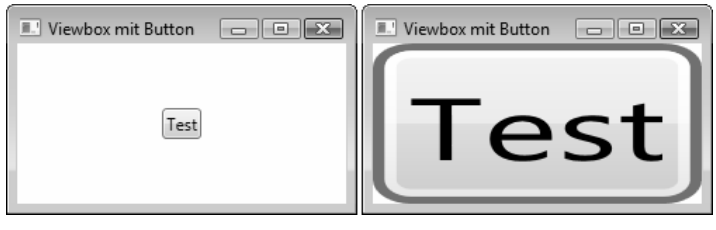



**Abbildung 13.16** Ein Button-Element in der Viewbox, links: Stretch="None", rechts: Stretch="Fill"

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Text-Layout

Beim Layout von Texten müssen verschiedene Eigenschaften berücksichtigt werden. Einerseits muss der Umbruch der Texte korrekt sein. Andererseits gibt es viele Parameter bei einer Schriftart, welche die Größe und die Darstellung eines Textes stark beeinflussen. Das einfachste Element, welches Texte darstellen kann, ist das `TextBlock`-Element.

Um einfache, kurze Texte darzustellen, kann das `TextBlock`-Element folgendermaßen benutzt werden:

```
<TextBlock>Hallo, Welt!</TextBlock>
```

Der Text wird im oben gezeigten Beispiel ohne weitere Formatierung dargestellt. Dennoch bietet das `TextBlock`-Element viele Formatierungsmöglichkeiten an. Zunächst kann man eine Schriftart für die Textdarstellung bestimmen. Dabei ist es natürlich auch möglich, die Größe der Schrift und den Schriftschnitt (fett, kursiv,…) festzulegen:

```
<TextBlock FontFamily="Courier New" FontSize="24" FontWeight="Bold">Guten Morgen!</TextBlock>
```

Eine weitere wichtige Darstellungsart des `TextBlock`-Elements erlaubt den Umbruch von längeren Textzeilen. Es gibt zwei Varianten des Zeilenumbruchs. Mit dem Wert `Wrap` für das Attribut `TextWrapping` erreicht man einen »vollständigen« Umbruch. D.h., wenn ein Wort nicht mehr in die Zeile passt, wird an einer Buchstabengrenze umgebrochen. Wird dem Attribut dagegen der Wert `WrapWithOverflow` zugewiesen, so wird ein nicht in die Zeile passendes Wort einfach abgeschnitten. Die überstehenden Buchstaben werden nicht dargestellt. Die beiden Varianten werden im Listing 13.17 vorgestellt.

```
<StackPanel>
  <TextBlock FontSize="24" FontStyle="Italic" FontFamily="Courier New" TextWrapping="WrapWithOverflow">
    Guten Morgen, alle miteinander hier im Raum!
  </TextBlock>

  <TextBlock FontSize="24" FontWeight="Bold" FontFamily="Arial" TextWrapping="Wrap">
    Auf Wiedersehen, meine Damen und Herren!
  </TextBlock>
</StackPanel>
```

**Listing 13.17** Das TextBlock-Element mit TextWrapping-Möglichkeiten

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 13.17** TextBlock mit TextWrapping-Möglichkeiten

In Abbildung 13.17 können Sie die Wirkungsweise der verschiedenen Zeilenumbruchsvarianten sehen. Das erste `TextBlock`-Element schneidet das Wort »miteinander« einfach ab und stellt den Rest des Wortes nicht dar (`TextWrapping="WrapWithOverflow"`). Im zweiten `TextBlock` wird das längere Wort »Wiedersehen« einfach im Wort selbst umgebrochen (`TextWrapping="Wrap"`). Auch wenn ein Zeilenumbruch aktiviert ist, versucht ein `TextBlock`-Element den Text zunächst in einer Zeile auszugeben.

Ein `TextBlock`-Element kann jedoch nicht nur Texte umbrechen, sondern auch andere Steuerelemente, wie z.B. Schaltflächen oder Eingabefelder. In Listing 13.18 werden mehrere Elemente mithilfe eines `TextBlocks` zeilenweise angeordnet. Die einzelnen Elemente im `TextBlock` werden im Prinzip genauso angeordnet, als wären es normale Wörter.

```
<TextBlock FontSize="24" TextWrapping="Wrap">
  <Button>Test</Button>
  <Button>Noch ein Test</Button>
  <CheckBox>Ein neues Control</CheckBox>
  <TextBox>Hier Eingabe</TextBox>
</TextBlock>
```

**Listing 13.18** Steuerelemente im TextBlock-Element mit Zeilenumbruch

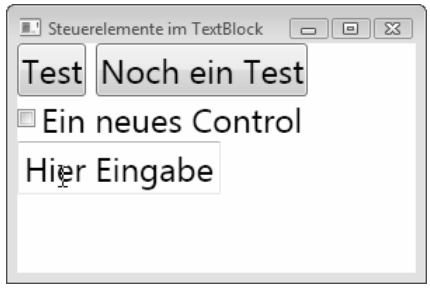



**Abbildung 13.18** Steuerelemente im TextBlock mit Zeilenumbruch

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Innerhalb eines Textes können sie im `TextBlock`-Element verschiedene Effekte in beliebigen Kombinationen anwenden. Dazu zählen fette und kursive Schrift, sowie unterstrichener Text und ein Text als Hyperlink. Die verschiedenen Möglichkeiten können Sie im Listing 13.19 sehen.

```
<TextBlock FontSize="20" FontFamily="Arial" TextWrapping="Wrap">
  <Italic>Ein schöner Text</Italic>
  <Bold>in schöner Schrift</Bold>
  <Underline>mit einem Strich darunter.</Underline>
  <LineBreak /><LineBreak />
  <Hyperlink>Klicken Sie bitte hier!</Hyperlink>
</TextBlock>
```

**Listing 13.19** Verschiedene Effekte im TextBlock-Element

An dieser Stelle soll auch erwähnt werden, dass mehrfache Leerzeichen hintereinander vom XAML-Compiler entfernt werden. Leerzeichen werden im Allgemeinen für die Formatierung des XAML-Codes benutzt. Um dies zu zeigen, ändern wir die dritte Zeile in Listing 13.19 durch Einfügen vieler Leerzeichen:

```
  <Bold>   in     schöner    Schrift   </Bold>
```

Trotz dieser Modifikation im XAML-Code wird sich die Ausgabe dieser Zeile nicht ändern, wenn das Programm ausgeführt wird. Die »überzähligen« Leerzeichen werden vom XAML-Compiler einfach entfernt. Natürlich gibt es viele Fälle, in denen der Programmierer dies verhindern möchte. Hierzu muss das `xml:space`-Attribut auf den Wert `preserve` gesetzt werden, wie in Listing 13.20 gezeigt wird.

```
<TextBlock FontSize="20" FontFamily="Arial" TextWrapping="Wrap">
  <Italic>Ein schöner Text</Italic>
  <Bold xml:space="preserve">   in     schöner     Schrift    </Bold>
  <Underline>mit einem Strich darunter.</Underline>
  <LineBreak /><LineBreak />
  <Hyperlink xml:space="preserve">Klicken Sie     bitte     hier!</Hyperlink>
</TextBlock>
```

**Listing 13.20** Das Attribut `xml:space` wird benutzt

In Listing 13.20 kann man sehen, dass das Attribut `xml:space` für jedes Element im `TextBlock` erneut gesetzt werden muss. Wenden Sie dieses Attribut im `TextBlock`-Element selbst an, werden auch die Leerzeichen, die an den Zeilenanfängen zum Formatieren des XAML-Codes stehen, mit in die Ausgabe einbezogen.

**HINWEIS** Beachten Sie, dass bei Anwendung des Attributs `xml:space="preserve"` auch Zeilenumbrüche im XAML-Code in der Ausgabe umgesetzt werden. Um einen Zeilenumbruch ohne das Attribut `xml:space="preserve"` zu erzielen, können Sie das Element `<LineBreak />` beliebig oft in den Text einsetzen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 13.19** Mehrere Leerzeichen mit dem Attribut `xml:space`

Im `TextBlock`-Element ist es ebenfalls möglich, die Anordnung des Textes mit dem Attribut `TextAlignment` zu beeinflussen. Es sind vier Varianten vorhanden: `Left`, `Right`, `Center` und `Justify`. Listing 13.21 zeigt die Auswirkungen beim `TextAlignment`. Bei der Darstellung des Textes wird mit der gewählten Anordnung auch der Zeilenumbruch (`TextWrapping`) im `TextBlock`-Element berücksichtigt.

```
<Window x:Class="TextBlock4.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="TextBlock mit TextAlignment" Height="300" Width="300"
    >
    <StackPanel Orientation="Vertical">
      <StackPanel Orientation="Horizontal">
        <Border BorderBrush="Blue" BorderThickness="1">
          <TextBlock Width="140" TextAlignment="Left" TextWrapping="Wrap">
            Das ist ein Test mit dem TextAligment-Attribut, welches den Text <Bold>links</Bold>
                anordnet.
          </TextBlock>
        </Border>
        <Border BorderBrush="Blue" BorderThickness="1">
          <TextBlock Width="140" TextAlignment="Right" TextWrapping="Wrap">
            Das ist ein Test mit dem TextAligment-Attribut, welches den Text <Bold>rechts</Bold>
                anordnet.
          </TextBlock>
        </Border>
      </StackPanel>
      <StackPanel Orientation="Horizontal">
        <Border BorderBrush="Blue" BorderThickness="1">
          <TextBlock Width="140" TextAlignment="Center" TextWrapping="Wrap">
            Das ist ein Test mit dem TextAligment-Attribut, welches den Text <Bold>zentriert</Bold>
                anordnet.
          </TextBlock>
      </Border>
      <Border BorderBrush="Blue" BorderThickness="1">
        <TextBlock Width="140" TextAlignment="Justify" TextWrapping="Wrap">
          Das ist ein Test mit dem TextAligment- Attribut, welches den Text im <Bold>Blocksatz</Bold>
              anordnet.
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
          </TextBlock>
        </Border>
      </StackPanel>
    </StackPanel>
</Window>
```

**Listing 13.21** Die Benutzung des Attributs TextAlignment

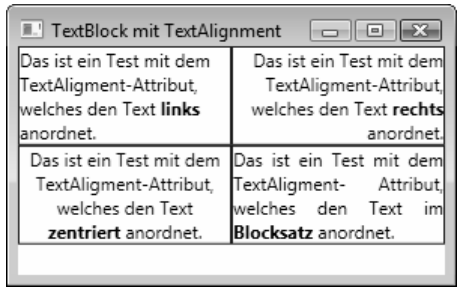



**Abbildung 13.20** Die Ausgabe mit verschiedenen Werten für das TextAlignment

# Das WrapPanel

Ein weiteres Panel, welches WPF-Elemente in einer ganz bestimmten Art und Weise anordnet, ist das `WrapPanel`. In diesem Steuerelement werden die Kindelemente zunächst horizontal nebeneinander angeordnet, bis die maximal erlaubte Breite erreicht wird. Dann findet, wie bei Texten, ein Zeilenumbruch statt. Das bedeutet, dass alle weiteren Elemente in der nächsten Zeile ausgegeben werden.

Im Listing 13.22 wird ein `WrapPanel` mit mehreren nebeneinander angeordneten Schaltflächen vorgestellt. Wird das linke Fenster in Abbildung 13.21 verkleinert, reicht der vorhandene Platz für die Schaltflächen nicht mehr aus. Nach und nach werden nun die Schaltflächen in weiteren Steuerelementzeilen umgebrochen und ausgegeben. Alle Kindelemente bleiben sichtbar, bis die Breite oder Höhe des Fensters generell für die vollständige Darstellung des Inhalts zu klein wird.

```
<WrapPanel>
  <Button>Button 1</Button>
  <Button>Hier klicken</Button>
  <Button>Test 3</Button>
</WrapPanel>
```

**Listing 13.22** `WrapPanel` in horizontaler Ausrichtung

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de


**Abbildung 13.21** Das `WrapPanel` wird Schritt für Schritt in der Breite verkleinert

Auch das `WrapPanel` erlaubt zwei Orientierungen. Wird die Eigenschaft `Orientation` auf `Horizontal` gesetzt, so bekommen sie das oben gezeigte Verhalten. Wird dagegen `Orientation` auf `Vertical` eingestellt, so werden die einzelnen Kindelemente untereinander angeordnet, bis der untere Rand des Elternelements erreicht ist. Dann werden die weiteren Elemente in einer neuen Spalte rechts daneben ausgegeben.

# Standard-Layout-Eigenschaften

Alle Steuerelemente in Windows Presentation Foundation haben einige Standardeigenschaften, d.h., Eigenschaften, die Sie mit jedem Steuerelement anwenden können. Hierbei handelt es sich um grundlegende Eigenschaften, welche die einzelnen Elemente aus der Basisklasse `FrameworkElement` erben.

## Width- und Height-Eigenschaft

Die beiden Eigenschaften `Width` und `Height` geben eine Breite und eine Höhe für ein Steuerelement vor. Verwenden Sie eines oder beide dieser Eigenschaften, so fixieren Sie die Größe dieses Elements. Dies hat verschiedene Auswirkungen. Es ist z.B. schwieriger, eine Benutzerschnittstelle mit vielen in Breite und Höhe fixierten Elementen zu lokalisieren, da die verschiedenen Landessprachen oft unterschiedliche Textbreiten benötigen. Außerdem ist es schwieriger, Dialogfelder zu erstellen, die in der Größe veränderbar sind. Wenn es geht, sollten Sie also die Definition von festen Breiten und Höhen vermeiden.

Bei der Benutzung der Eigenschaften `Width` und `Height` wird das Layout-System von WPF jedoch immer versuchen, Ihre Wünsche auszuführen. Wenn ein Element nicht vollständig darstellbar ist, z.B. weil das Fenster oder der Bildschirm zu klein ist, wird es an der entsprechenden Stelle einfach abgeschnitten.

Eine vorgegebene Breite oder Höhe eines Elements wird ggf. an die Kindelemente weitergegeben.

## MinWidth-, MaxWidth-, MinHeight- und MaxHeight-Eigenschaft

Mit diesen vier Eigenschaften können Sie die minimal erlaubte Breite und Höhe und die maximal erlaubte Breite und Höhe für ein Steuerelement definieren. Die Benutzung dieser Eigenschaften macht eigentlich nur dann einen Sinn, wenn Sie kein fest definiertes Layout mit `Width` und `Height` benutzen.

Mit den Min- und Max-Eigenschaften für Breite und Höhe ist es möglich, die Größenänderungen, welche das Layout-System von WPF vornehmen kann, einzuschränken. Dies ist eine einfache Möglichkeit, die Größe von Steuerelementen zu kontrollieren.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wenn Sie für eine der Eigenschaften einen »unmöglichen« Wert angeben, z.B. `MinHeight="1000000"`, so wird das Element zwar dargestellt, aber an der passenden Stelle abgeschnitten. Das Programm wird in solchen Fällen nicht mit einer Fehlermeldung abgebrochen. Daran ändert auch die zusätzliche Definition einer sehr kleinen maximalen Höhe nichts (z.B. `MaxHeight="50"`). In diesem Fall wird die zu kleine maximale Höhe ignoriert und die `MinHeight`-Eigenschaft für die Darstellung des Elements benutzt.

## HorizontalAlignment- und VerticalAlignment-Eigenschaft

Die Eigenschaft `TextAlignment` haben Sie schon beim `TextBlock`-Element kennen gelernt. Ganz ähnlich funktionieren die Eigenschaften `HorizontalAlignment` und `VerticalAlignment`. Hierbei wird aber nicht der Text rechts-, linksbündig oder zentriert angeordnet, sondern die Steuerelemente werden entsprechend rechts, links oder mittig angeordnet. Es ist also sehr einfach, eine Schaltfläche ohne viele komplizierte Berechnungen mitten in einem Elternelement zu platzieren.

Die vier Möglichkeiten bei der Eigenschaft `HorizontalAlignment` sind `Left`, `Right`, `Center` und `Stretch`. Die Eigenschaft `VerticalAlignment` erlaubt `Top`, `Bottom`, `Center` und `Stretch`.

In einem `StackPanel` oder `WrapPanel` mit der Einstellung `Orientation="Vertical"` werden alle Kindelemente von oben nach unten angelegt. Das erste Element wird am oberen Rand des Elternelementes ausgegeben. Nun können Sie für das Panel zusätzlich die Eigenschaft `VerticalAlignment="Bottom"` setzen. Die Kindelemente werden jetzt so angeordnet, dass das letzte Element mit dem unteren Rand des Elternelementes abschließt.

**HINWEIS** Wenn die Eigenschaften `VerticalAlignment="Bottom"` bzw. `HorizontalAlignment="Right"` gesetzt werden, bleibt so die Reihenfolge der Elemente erhalten. Das letzte Kindelement schließt nur mit dem unteren bzw. rechten Rand ab.

Die Auswahl `Stretch` ist für beide Eigenschaften der Standardwert. In diesem Fall versucht das Element, den gesamten Platz auszunutzen, der zur Verfügung steht.

## Margin-Eigenschaft

Die Eigenschaft `Margin` bestimmt die Breite des Randes, der um das Element herum frei gelassen werden soll. Diese Eigenschaft kann benutzt werden, wenn ein Element sein Elternelement nicht vollständig ausfüllen soll. `Margin`-Werte können in unterschiedlicher Weise angegeben werden. Wenn Sie eine Zahl angeben (z.B. `Margin="10"`), so wird dieser Wert für alle vier Ränder (oben, unten, rechts und links) des Kindelements herangezogen. Mit der Angabe von zwei Zahlen für die Eigenschaft (z.B. `Margin="10 20"`) definieren Sie mit der ersten Zahl die Breite des rechten und linken Randes. Die zweite Zahl in der `Margin`-Eigenschaft gibt an, wie breit die Ränder oben und unten sein sollen. Wenn Sie vier Werte angeben, können Sie vier unterschiedliche Werte für alle Ränder definieren. Die Reihenfolge der Zahlen ist dann: Linker, oberer, rechter und unterer Rand.

Das Beispiel in Listing 13.23 zeigt die Benutzung der verschiedenen `Margin`-Arten. Dort wird ein `Grid` mit vier Spalten definiert. Die Zellen werden durch gestrichelte Linien dargestellt (`ShowGridLines="True"`).

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

**HINWEIS** Die Angabe von drei Zahlen in der `Margin`-Eigenschaft ist nicht erlaubt und führt zum Abbruch des Programms.

```
<Window x:Class="ShowMargin.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Margin" Height="300" Width="500"
    >
    <Grid ShowGridLines="True">
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>

      <Button Grid.Column="0">Ohne Margin</Button>
      <Button Grid.Column="1" Margin="15">Margin 1</Button>
      <Button Grid.Column="2" Margin="10,30">Margin 2</Button>
      <Button Grid.Column="3" Margin="10,20,30,40">Margin 4</Button>
    </Grid>
</Window>
```

**Listing 13.23** Verwendung der Eigenschaft `Margin`

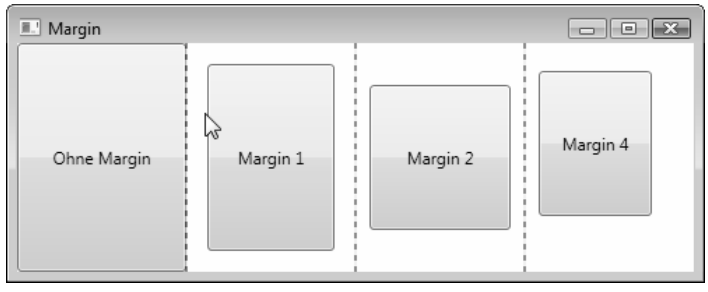



**Abbildung 13.22** Verwendung unterschiedlicher `Margin`-Definitionen

Wird das Fenster mit dem in Listing 13.23 gezeigten Programm in der Größe geändert, so werden die Breiten der Ränder der einzelnen Schaltflächen unverändert bleiben. Die Größen der Schaltflächen werden aber entsprechend angepasst. Verkleinern Sie das Fenster nun immer weiter, so wird irgendwann ein Punkt erreicht werden, an dem der definierte Rand und die Standardgröße des Kindelements nicht mehr in den vorhandenen Bereich passen. Dann wird das jeweilige Kindelement (hier die Schaltflächen) nicht mehr vollständig dargestellt. Es kann sogar passieren, dass ein Element gar nicht mehr dargestellt wird. D.h., die Darstellung des Randes hat immer Vorrang vor der Ausgabe des Kindelementes.

## Padding-Eigenschaft

Die Eigenschaft `Padding` beeinflusst den Abstand zwischen dem Rand eines Steuerelements und den darin enthaltenen Daten (`Content`). In Abbildung 13.23 wird der Unterschied zwischen den beiden Eigenschaften `Margin` und `Padding` dargestellt.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

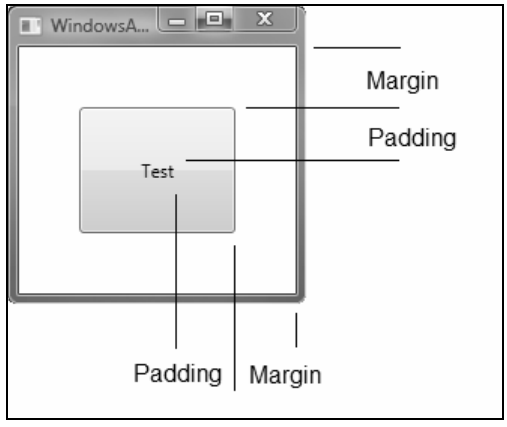



**Abbildung 13.23** `Margin` und `Padding`

Die `Padding`-Eigenschaft finden Sie nur in solchen WPF-Elementen, die einen Inhalt darstellen können. Wenn Sie die Werte für das `Padding` zu groß wählen, ist es möglich, dass der Inhalt des Elements gar nicht oder nur teilweise dargestellt wird. Auch bei der `Padding`-Eigenschaft können Sie eine Zahl, zwei oder vier Zahlen als Parameter angeben. Das Verhalten der `Padding`-Parameter ist identisch zu dem der `Margin`-Angaben.

# Zusammenfassung

In diesem Kapitel haben Sie die verschiedenen Layout-Panels von WPF kennen gelernt. Mit WPF werden die Positionen und Größen der einzelnen Elemente in einem Fenster normalerweise nicht mehr statisch festgelegt, sondern es werden Panels verwendet, die eine dynamische Anordnung der Elemente beim Vergrößern oder Verkleinern des Fensters ermöglichen. WPF enthält viele leistungsfähige Panels: `StackPanel`, `DockPanel` und `WrapPanel`. Weiterhin steht das `Grid`-Element für ein sehr flexibles Layout zur Verfügung.

In den Layout-Steuerelementen gibt es bestimmte Standardeigenschaften, die es ermöglichen, die Art des Layouts zu steuern.

Außerdem gibt es ein WPF-Element, welches das genaue Positionieren von Elementen auf absoluter Basis ermöglicht. In diesem `Canvas`-Element können einzelne Grafikelemente exakt arrangiert werden.

Für das Layout von Texten wurde in WPF natürlich auch gesorgt. Das `TextBlock`-Element lässt viele Formatierungsmöglichkeiten zu und stellt auch einen Zeilenumbruch zur Verfügung.
Sie können alle Layout-Elemente hierarchisch ineinander verschachteln, so dass auch komplexe Anordnungen von Steuerelementen in einem Fenster mit änderbarer Größe automatisch angeordnet werden können.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Kapitel 14

# Grafische Grundelemente

**In diesem Kapitel:**





"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Ohne die grafischen Grundelemente wie Linien, Rechtecke, Ellipsen u.v.m. geht in einem fensterorientierten System eigentlich gar nichts. WPF hat einiges in diesem Bereich zu bieten, denn das *P* in WPF steht ja bekanntlich für *Presentation*. In diesem Kapitel möchte ich Ihnen die Möglichkeiten vorstellen, welche Sie mit diesen Grafikelementen haben. In der zweiten Hälfte des Kapitels werden wir uns mit den komplexen Grafikelementen beschäftigen, dazu gehören die Pfade, Polygone und Bezierkurven.

**BEGLEITDATEIEN** Unter .*Samples**Chapter14*\ finden Sie die Beispieldateien für dieses Kapitel.

# Grundlagen

Grafische Grundelemente können mit WPF überall in der Benutzerschnittstelle verwendet werden. Eine Trennung von Steuerelementen und grafischen Elementen ist hier nicht vorhanden. Grafische Elemente, wie Linien und Ellipsen, sind genau so in einer Benutzerschnittstelle einsetzbar, wie Schaltflächen oder Textboxen. Auch das Programmierparadigma ist identisch, wie das Beispiel in Listing 14.1 zeigt.

```
<Window x:Class="TextKomplex.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Text mit Grafik" Height="150" Width="650"
    >
    <DockPanel>
      <StackPanel Height="30" DockPanel.Dock="Top" Orientation="Horizontal">
        <TextBlock FontSize="20">Mit WPF</TextBlock>
        <Ellipse Width="20" Fill="Red" />
        <TextBlock VerticalAlignment="Center">kann man Grafik</TextBlock>
        <Rectangle Width="30" Fill="Blue" />
        <TextBlock VerticalAlignment="Center">einfach mischen. Und einen</TextBlock>
        <Button Width="120">
          <StackPanel Orientation="Horizontal">
            <Ellipse Width="30" Height="10" Stroke="Blue" Fill="Green" />
            <TextBlock>Hallo</TextBlock>
            <Ellipse Width="30" Height="10" Stroke="Blue" Fill="Red" />
          </StackPanel>
        </Button>
        <TextBlock VerticalAlignment="Center">kann man auch einfügen!</TextBlock>
        <Ellipse Width="30" Fill="Yellow" Stroke="Black" />
      </StackPanel>
    </DockPanel>
</Window>
```

**Listing 14.1** Texte und grafische Elemente

Im Beispiel aus Listing 14.1 wird in einem `DockPanel` ein `StackPanel` mit horizontaler Orientierung platziert. Im `StackPanel` können nun sowohl Texte (`TextBlock`) als auch grafische Elemente (`Ellipse`, `Rectangle`) in beliebiger Anordnung dargestellt werden. Auch Steuerelemente, wie hier eine Schaltfläche (`Button`), können eingesetzt werden. Die Schaltfläche selbst kann dann wieder aus grafischen Elementen und Texten bestehen. Das Ergebnis kann in Abbildung 14.1 betrachtet werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

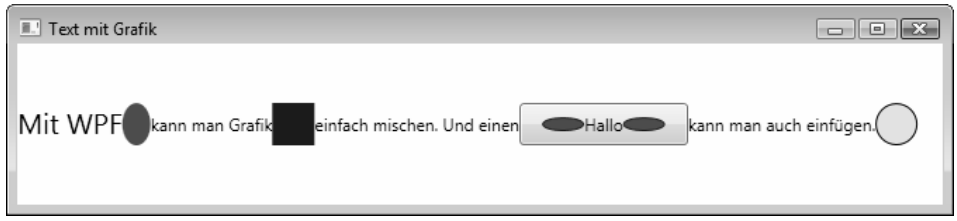



**Abbildung 14.1** Texte und Grafik gemischt

Die grafischen Ausgaben können allerdings nicht nur in XAML definiert, sondern auch durch normalen Programmcode erzeugt werden. Hierbei werden Sie jedoch schnell feststellen, dass eine hierarchische Darstellung der einzelnen Elemente in XAML meistens viel übersichtlicher und einfacher zu lesen ist. In Listing 14.2 wird das Fenster aus Abbildung 14.1 nur durch VB-Code erzeugt.

```
Imports System
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Media
Imports System.Windows.Shapes


Namespace TextKomplex
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
            InitMyLayout()
        End Sub

        Private Sub InitMyLayout()
            Dim dock As New DockPanel

            Dim stack As New StackPanel
            stack.Orientation = Orientation.Horizontal
            stack.Height = 30

            Dim t1 As New TextBlock
            t1.FontSize = 20
            t1.Text = "Mit WPF"
            stack.Children.Add(t1)

            Dim e1 As New Ellipse
            e1.Width = 20
            e1.Fill = Brushes.Red
            stack.Children.Add(e1)

            Dim t2 As New TextBlock
            t2.Text = "kann man Grafik"
            t2.VerticalAlignment = Windows.VerticalAlignment.Center
            stack.Children.Add(t2)
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            Dim r1 As New Rectangle
            r1.Width = 30
            r1.Fill = Brushes.Blue
            stack.Children.Add(r1)

            Dim t3 As New TextBlock
            t3.Text = "einfach mischen. Und einen"
            t3.VerticalAlignment = Windows.VerticalAlignment.Center
            stack.Children.Add(t3)

            Dim btn As New Button
            btn.Width = 120

            Dim pa As New StackPanel
            pa.Orientation = Orientation.Horizontal

            Dim ebtn1 As New Ellipse
            ebtn1.Width = 30
            ebtn1.Height = 10
            ebtn1.Fill = Brushes.Green
            ebtn1.Stroke = Brushes.Blue
            pa.Children.Add(ebtn1)

            Dim tbtn As New TextBlock
            tbtn.Text = "Hallo"
            pa.Children.Add(tbtn)

            Dim ebtn2 As New Ellipse
            ebtn2.Width = 30
            ebtn2.Height = 10
            ebtn2.Fill = Brushes.Red
            ebtn2.Stroke = Brushes.Blue
            pa.Children.Add(ebtn2)
            btn.Content = pa

            stack.Children.Add(btn)

            Dim t4 As New TextBlock
            t4.Text = "kann man auch einfügen."
            t4.VerticalAlignment = Windows.VerticalAlignment.Center
            stack.Children.Add(t4)

            Dim e2 As New Ellipse
            e2.Width = 30
            e2.Fill = Brushes.Yellow
            e2.Stroke = Brushes.Black
            stack.Children.Add(e2)

            dock.Children.Add(stack)
            Me.Content = dock
        End Sub
    End Class
End Namespace
```

**Listing 14.2** Definition von Hierarchien mit VB-Code ist oft aufwendig

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Die Darstellung von grafischen Elementen auf der Schaltfläche im obigen Beispiel ist ebenfalls sehr einfach. Hierbei sollten Sie beachten, dass WPF sehr flexibel ist, was die Hierarchie der einzelnen Elemente anbelangt. So ist es z. B. ohne weiteres möglich, ein `Grid`-Element auf einer Schaltfläche zu platzieren, um dann die einzelnen Zellen zur Darstellung von grafischen Elementen auszunutzen. Ein grafisches Element kann auch mehrere Zellen im `Grid` belegen. Listing 14.3 zeigt ein Beispiel.

```
<Window x:Class="ButtonKomplex.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Eine Super-Schaltfläche" Height="300" Width="400" Background="LightGray"
    >
  <Button Width="250" Height="150">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition />
      </Grid.ColumnDefinitions>

      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition Height="3*"/>
        <RowDefinition />
      </Grid.RowDefinitions>

      <Rectangle Name="rect1" Grid.Column="0" Grid.Row="0" Fill="Red" Stroke="Black" />
      <Rectangle Name="rect2" Grid.Column="2" Grid.Row="2" Fill="Red" Stroke="Black" />

      <TextBlock Name="text1" Grid.Column ="2" Grid.Row="0"
                 FontSize="14" VerticalAlignment="Top" Text="Klicken!" />

      <TextBlock Name="text2" Grid.Column ="0" Grid.Row="2"
                 FontSize="14" VerticalAlignment="Bottom" Text="Klicken!" />

      <Image  Grid.Column="1" Grid.Row="1"
              Source="D:\Avalon-Buch\Buch-Dateien\Kap04\Astro.jpg" />

      <Ellipse Name="ellip" Grid.Column="0" Grid.Row="0"
               Grid.ColumnSpan="3" Grid.RowSpan="3"
               Width="230" Height="130" Stroke="Blue" StrokeThickness="5" />
    </Grid>
  </Button>
</Window>
```

**Listing 14.3** Eine Schaltfläche mit grafischen Elementen

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

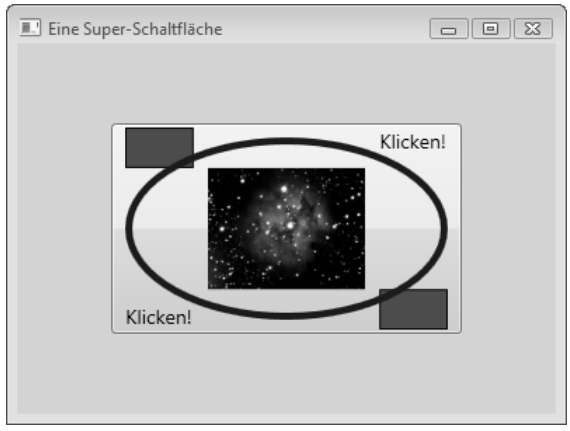



**Abbildung 14.2** Die grafische Schaltfläche aus Listing 14.3

Die Schaltfläche in Abbildung 14.2 wird nicht über eine Bitmap, also eine pixelorientierte Grafik realisiert, sondern besteht weiter aus den einzelnen vektororientierten, grafischen Grundelementen, die z. B. auch auf Ereignisse reagieren können.

Bei vielen Benutzeroberflächen-Technologien müssen Steuerelemente von Grund auf neu programmiert werden, wenn sich die Darstellung ändert. Oft ist das Verhalten beim Neuzeichnen dieser Steuerelemente nicht effizient und sie werden öfter gezeichnet, als eigentlich notwendig ist. Dies führt manchmal zu »flackernden« Benutzerschnittstellen, mit denen das Arbeiten nicht sehr angenehm ist.

WPF geht hier einen anderen Weg. Die grafischen Elemente werden ebenso wie die Steuerelemente in einer baumähnlichen Struktur verwaltet. Grafiken können dadurch genauso wie z. B. eine Schaltfläche durch Änderung ihrer Eigenschaften modifiziert werden. Der Programmierer muss sich jedoch nicht um das korrekte Neuzeichnen des Fensters oder des Fensterausschnitts kümmern.

Das Beispiel aus Listing 14.3 soll so erweitert werden, dass die Schaltfläche das `Click`-Ereignis bedient:

```
<Button Width="250" Height="150" Click="OnClicked">
```

Die Elemente, die aus dem VB-Code heraus verändert werden sollen, sind in Listing 14.3 bereits mit den Objektnamen versehen. Es muss also nur noch die Methode für das `Click`-Ereignis implementiert werden (Listing 14.4). Beim Anklicken der Schaltfläche ändert sich sofort das Aussehen derselben. Es wird keine Methode zum Neuzeichnen der Grafiken aufgerufen.

```
Imports System
Imports System.Windows

Namespace ButtonKomplex
    Partial Public Class Window1
        Inherits System.Windows.Window

        Public Sub New()
            InitializeComponent()
        End Sub
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        Private Sub OnClicked(ByVal sender As Object, ByVal e As RoutedEventArgs)
            rect1.Fill = Brushes.Yellow
            rect2.Fill = Brushes.Yellow
            text2.FontStyle = FontStyles.Italic
            ellip.Stroke = Brushes.Red
            ellip.StrokeThickness = 8

            Dim d As New DoubleCollection
            d.Add(4)
            d.Add(1)
            d.Add(3)
            d.Add(3)
            ellip.StrokeDashArray = d
        End Sub
    End Class
End Namespace
```

**Listing 14.4** Veränderung der Objekte aus VB-Code

Im nächsten Beispiel soll geprüft werden, wie die Behandlung der Ereignisse solcher Grafikelemente funktioniert. In einem Fenster werden mit XAML mehrere Rechtecke (`Rectangle`) erzeugt. Jedes Rechteck beinhaltet eine Methode für die Verarbeitung des `Click`-Ereignisses. Beim Anklicken wird das jeweilige Rechteck um 10 Einheiten vergrößert. Hierbei stellen sich nun drei interessante Fragen:

- Vergrößert sich der Bereich für das `Click`-Ereignis entsprechend der Vergrößerung des Rechtecks?
- Wie kann man das angeklickte Rechteck identifizieren?
- Was passiert, wenn Rechtecke angeklickt werden, die so groß sind, dass sie übereinander liegen?

Listing 14.5 zeigt den entsprechenden VB- und XAML-Code und Abbildung 14.3 zeigt eine mögliche Ausgabe im Fenster.

```
<!-- XAML: Window1.xaml -->
<Window x:Class="TestEreignis.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Test Ereignisse" Height="300" Width="500"
    >
  <Canvas>
    <Rectangle Canvas.Left="10" Canvas.Top="30" Fill="Blue"
               Width="40" Height="40" MouseLeftButtonDown="OnClicked" />
    <Rectangle Canvas.Left="110" Canvas.Top="30" Fill="Red"
               Width="40" Height="40" MouseLeftButtonDown="OnClicked" />
    <Rectangle Canvas.Left="210" Canvas.Top="30" Fill="Black"
               Width="40" Height="40" MouseLeftButtonDown="OnClicked" />
    <Rectangle Canvas.Left="310" Canvas.Top="30" Fill="Green"
               Width="40" Height="40" MouseLeftButtonDown="OnClicked" />
    <Rectangle Canvas.Left="410" Canvas.Top="30" Fill="Yellow"
               Width="40" Height="40" MouseLeftButtonDown="OnClicked" />
  </Canvas>
</Window>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
Imports System
Imports System.Windows

Namespace TestEreignis
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnClicked(ByVal sender As Object, ByVal e As RoutedEventArgs)
            Dim r As Rectangle = CType(sender, Rectangle)
            r.Width += 10
            r.Height += 10
        End Sub
    End Class
End Namespace
```

**Listing 14.5** Mehrere Grafikelemente mit einer Ereignismethode

In der Ereignismethode können Sie (wie z.B. aus Windows Forms bekannt) einfach das Objekt mit dem Namen `sender` in ein `Rectangle`-Objekt konvertieren. Danach können Sie, wie gewohnt, auf die Eigenschaften des Elements zugreifen und diese ändern. Im Beispiel wird nicht explizit eine Neuzeichnungsmethode aufgerufen. WPF zeichnet das Element nach der Änderung der Eigenschaften `Width` und `Height` neu. Mit der Vergrößerung des Rechtecks erweitert sich auch der Bereich für das `MouseLeftButtonDown`-Ereignis ohne weiteren Programmieraufwand.

Wenn die einzelnen Rechtecke größer werden und sich überlagern, so erfolgt die Darstellung in der Reihenfolge der Definition aus dem XAML-Code. Zuerst wird also das blaue Rechteck, dann das rote und zuletzt das gelbe Rechteck gezeichnet.

Beim Experimentieren mit dem Beispiel können Sie feststellen, dass bei überlagerten Rechtecken nur das `Click`-Ereignis für das jeweils ganz oben liegende Rechteck ausgelöst wird. Das `Click`-Ereignis wird nicht an die eventuell (optisch) darunter liegenden Rechtecke weitergeleitet, da alle Rechtecke in der Element-Hierarchie (logischer Baum) in der gleichen Ebene liegen.

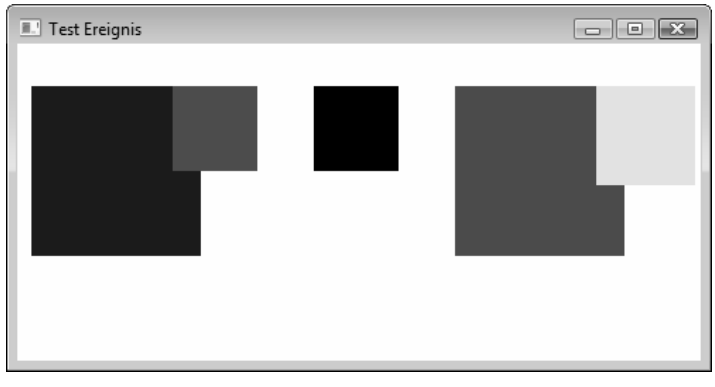



**Abbildung 14.3** Alle Rechtecke werden in einer Ereignismethode behandelt

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Die Grafik-Auflösung

Unsere Rechner sind in den letzten Jahren nicht nur schneller geworden, auch die Grafikkarten sind im Laufe der Zeit immer weiter verbessert worden. Mein erster Computer, ein Radio Shack TRS-80 Level II, hatte einen Z80-Prozessor mit 0,9 MHz Taktfrequenz. Die Grafikauflösung auf einem Schwarz/Weiß-Monitor betrug sagenhafte 128x48 »Klötze« (Nein, da fehlt keine Null hinten!). Aber auch bei diesen Auflösungen konnten verschiedene Programme schon 3D-Darstellungen von einfachen Objekten auf den Bildschirm zaubern. Eigentlich kaum vorstellbar.

In der heutigen Zeit sind die Grafikkarten wesentlich leistungsfähiger. Aber nicht jede Software nutzt alle Möglichkeiten der Grafikhardware aus. Grafiken, die in beliebiger Weise vergrößer- oder verkleinerbar sind, kann man mit Windows schon seit langer Zeit programmieren. Die Grundlagen hierfür finden Sie in der GDI.DLL von Windows. Diese Grafik-Bibliothek ist im Jahr 2001 durch GDI+ noch wesentlich erweitert worden. Trotzdem ist es nicht möglich, mit vertretbarem Aufwand und minimalen Einschränkungen eine vollständig skalierbare Benutzerschnittstelle (hier ist auch die Größe der Steuerelemente gemeint) für Windows-Anwendungen zu erstellen.

Wie bereits weiter oben erwähnt, gibt es diese Trennung von Grafik- und Benutzerschnittstellenelementen in WPF nicht mehr. Eine in der Größe änderbare Benutzerschnittstelle sollte also möglich sein. Wie sieht es aber mit der Darstellungsqualität aus? Wenn Rastergrafiken (Bitmaps) vergrößert werden, erscheinen sehr schnell die unansehnlichen »Klötze« auf dem Bildschirm, die das Betrachten der Grafiken eher zu einem unangenehmen Erlebnis machen. Aber auch hier hat sich in WPF Einiges getan! Alle Elemente der Benutzerschnittstelle werden aus den vektororientierten Grafikelementen erstellt. Es werden keine Rastergrafiken verwendet.

Im folgenden Beispiel (Listing 14.6) können Sie eine Schaltfläche sehen, die mit einer `LayoutTransform` in unterschiedliche Größen transformiert wurde. Die Schaltfläche bleibt hierbei immer exakt die gleiche, nur die Größe wir durch *Zoomen* geändert. Die hierfür verwendete Operation ist eine `LayoutTransform`. Diese Art der Transformation werden wir im Laufe dieses Kapitels noch genauer kennen lernen. Im Moment genügt es, wenn Sie wissen, dass diese Transformation die gesamte Benutzerschnittstelle beeinflussen kann. Dabei muss die Transformation nicht für jedes Element neu angegeben werden. Im Beispiel wird die `LayoutTransformation` nur für die Schaltfläche definiert. Alle Elemente, die in der Hierarchie unter der Schaltfläche liegen (`Ellipse`, `TextBlock`, `Rectangle` und `Line`) werden automatisch entsprechend skaliert.

```
<Window x:Class="ButtonGross.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Schaltfläche transformieren" Height="300" Width="600"
    >
    <StackPanel>
      <Button Height="50" Width="250" Margin="10">
        <Button.LayoutTransform>
          <ScaleTransform ScaleX="1" ScaleY="1" />
        </Button.LayoutTransform>

        <StackPanel Orientation="Horizontal">
          <Ellipse Width="50" Height="25" Fill="Red" Stroke="Black" />
          <TextBlock FontSize="20">Klicken!</TextBlock>
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
        <Rectangle Width="50" Height="25" Stroke="Blue" RadiusX="8" RadiusY="8" />
        <Line X1="0" Y1="0" X2="50" Y2="25" Stroke="Green" StrokeThickness="3" />
      </StackPanel>
    </Button>
  </StackPanel>
</Window>
```

**Listing 14.6** Eine Schaltfläche mit einer LayoutTransformation

In Abbildung 14.4 werden mehrere der Schaltflächen dargestellt. Hierbei wurden die Skalierungsfaktoren (`ScaleX` und `ScaleY`) für die `LayoutTransform` auf die Werte 1, 2, 3 und 5 eingestellt. Die Schaltfläche wird Schritt für Schritt größer. Trotzdem bleibt die Qualität der Grafik sehr hoch. Das Bild wird nicht unscharf und enthält keine »Klötze«. Alle Grafikelemente, die Schaltfläche und der Text werden nicht als Rastergrafik vergrößert, sondern mit dem jeweiligen Zoomfaktor neu gezeichnet. Dabei werden auch die Linienstärken der einzelnen Grafikelemente entsprechend vergrößert, sodass die gesamte Darstellung immer noch »ausgewogen« aussieht.

Genauso, wie die Schaltfläche in Abbildung 14.4 skaliert wurde, können Sie jedes Element der Benutzerschnittstelle vergrößern oder verkleinern. Aber damit noch nicht genug. Ebenso einfach können Sie ein Steuerelement oder ein Grafikelement drehen, kippen oder nur an eine andere Stelle verschieben. Auch diese Operationen können mit einer `LayoutTransform` durchgeführt werden.

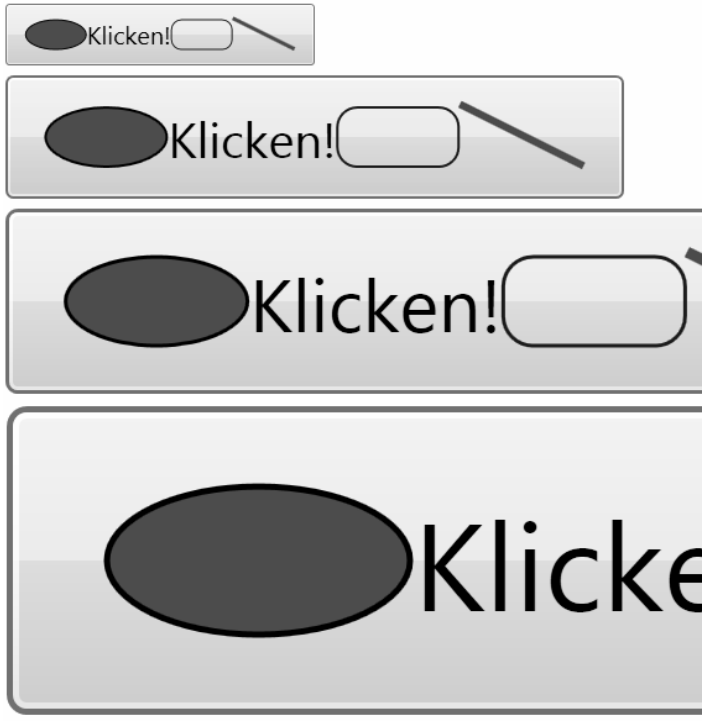



**Abbildung 14.4** Die vergrößerte Schaltfläche

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Da Sie alle Elemente in einer beliebigen Größe darstellen können, gibt es im Grunde genommen keine feste Beziehung zwischen den Koordinaten der Grafikobjekte und den physikalischen Pixeln des Bildschirms. Ganz abgesehen davon können alle Koordinaten als Fließkommazahlen angegeben werden. In WPF werden die Koordinaten in so genannten *geräteunabhängigen Pixeln* angegeben. Ein geräteunabhängiger Pixel entspricht 1/96 Zoll. Anders ausgedrückt entsprechen 96 Pixel einer Länge von 25,4 mm auf dem Bildschirm. Die Anzahl der tatsächlichen, physikalischen Pixel, die hierzu benötigt werden, hängt von der Auflösung des Bildschirms ab.

Der »krumme« Wert von 96 DPI (Dotch per Inch) pro Zoll kommt daher, dass in Windows die Auflösung des Standard-Displays 96 DPI ist. Für das Standard-Display gilt somit: 1 logischer Pixel = 1 echter Pixel! Für andere Wiedergabegeräte werden alle Koordinaten entsprechend umgerechnet.

# Die grafischen Grundelemente

Beginnen wir mit dem Zusammenspiel der grafischen Grundelemente (Shapes), der Stifte (Pens) und der Pinsel (Brushes). Es gibt diverse Klassen, welche die grafischen Grundelemente zur Verfügung stellen. Diese Klassen beinhalten verschiedene Eigenschaften, um das Aussehen zu beeinflussen.

- `Rectangle`
- `Ellipse`
- `Line`
- `Polyline`
- `Polygon`
- `Path`

Für diese grafischen Objekte kann man Füllfarben in Form von Pinseln (`Brush`) auswählen. Der äußere Rand kann durch einen Stift (`Pen`) definiert werden. Der einfachste Pinsel ist ein `SolidColorBrush`, der nur aus einer vorgegebenen Farbe besteht. Andere Pinsel, die wir später noch kennen lernen werden, können dagegen komplexe Farbverläufe darstellen. Listing 14.7 zeigt, wie Sie Ellipsen, Rechtecke und abgerundete Rechtecke zeichnen können. Die Füllfarbe wird mit der Eigenschaft `Fill` bestimmt, die Randfarbe mit der Eigenschaft `Stroke`. Die Dicke der Randlinie wird mit der Eigenschaft `StrokeThickness` angegeben. Schließlich können Sie beliebige gestrichelte Linien erzeugen. Hierzu geben Sie mit der Eigenschaft `StrokeDashArray` an, wie viele Einheiten des Striches gezeichnet und wie viele Einheiten nicht gezeichnet werden sollen. Sie können hier auch mehrere Zahlen angeben, sodass Sie sehr komplizierte Strichmuster erzeugen können (Abbildung 14.5).

```
<Window x:Class="Shapes1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Grafische Grundelemente" Height="350" Width="300"
    >
    <Canvas>
      <Ellipse Canvas.Left="10" Canvas.Top="10" Width="80" Height="60" Fill="Red" />
      <Rectangle Canvas.Left="100" Canvas.Top="10" Width="80" Height="60" Fill="Blue" />
      <Rectangle Canvas.Left="190" Canvas.Top="10" Width="80" Height="60"
                 RadiusX="15" RadiusY="15" Fill="Green" />
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
      <Ellipse Canvas.Left="10" Canvas.Top="80" Width="80" Height="60" Fill="Red" Stroke="Black" />
      <Rectangle Canvas.Left="100" Canvas.Top="80" Width="80" Height="60" Fill="Blue" Stroke="Red" />
      <Rectangle Canvas.Left="190" Canvas.Top="80" Width="80" Height="60"
                 RadiusX="15" RadiusY="15" Fill="Green" Stroke="Aquamarine" />

      <Ellipse Canvas.Left="10" Canvas.Top="150" Width="80" Height="60" Fill="Red" Stroke="Black"
               StrokeThickness="3" />
      <Rectangle Canvas.Left="100" Canvas.Top="150" Width="80" Height="60" Fill="Blue" Stroke="Red"
                 StrokeThickness="7" />
      <Rectangle Canvas.Left="190" Canvas.Top="150" Width="80" Height="60" RadiusX="15" RadiusY="15"
                 Fill="Green" Stroke="Aquamarine" StrokeThickness="10" />

      <Ellipse Canvas.Left="10" Canvas.Top="220" Width="80" Height="60" Fill="Red" Stroke="Black"
               StrokeThickness="3" StrokeDashArray="2 2" />
      <Rectangle Canvas.Left="100" Canvas.Top="220" Width="80" Height="60" Fill="Blue" Stroke="Red"
                 StrokeThickness="7" StrokeDashArray="2 3 4 2" />
      <Rectangle Canvas.Left="190" Canvas.Top="220" Width="80" Height="60" RadiusX="15" RadiusY="15"
                 Fill="Green" Stroke="Aquamarine" StrokeThickness="10" StrokeDashArray="1 1"/>
    </Canvas>
</Window>
```

**Listing 14.7** Verschiedene Ellipsen und Rechtecke

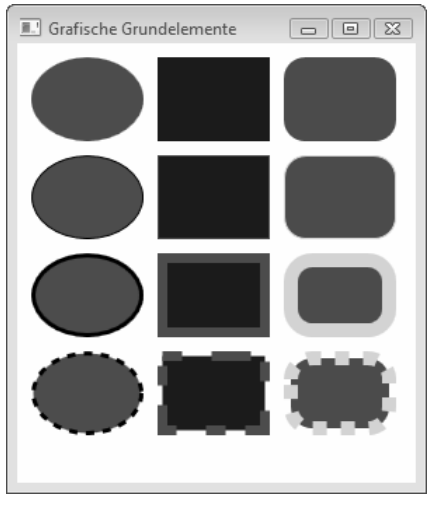



**Abbildung 14.5** Verschiedene Ellipsen und Rechtecke

# Rechteck und Ellipse

Die grafischen Grundelemente `Rectangle`, `Ellipse`, `Line`, `Polygon`, `Polyline` und `Path` sind von der Basisklasse `Shape` abgeleitet. Die meisten Eigenschaften der grafischen Grundelemente kommen aus dieser Basisklasse.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Wie Sie bereits gelernt haben, wird der Rand eines grafischen Elements über die Eigenschaft `Stroke` der `Shape`-Klasse angegeben, welche vom Typ `Brush` ist. Intern wird an dieser Stelle ein `Pen`-Objekt benutzt, wie man das eigentlich auch erwarten würde. Damit die XAML-Syntax aber nicht zu kompliziert wird, werden die Eigenschaften des `Pen`-Objektes über eigene Eigenschaften im `Shape`-Objekt abgebildet.

`Rectangle` und `Ellipse` haben wir schon in einem Beispiel kennen gelernt (Listing 14.7). Dort wurde die Positionierung der grafischen Elemente mit einen `Canvas`-Objekt durchgeführt. Die grafischen Elemente können aber auch genauso in einen `DockPanel`, `StackPanel`, `WrapPanel` oder einem `Grid` (Listing 14.8) positioniert werden.

```
<Window x:Class="Shapes2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Grafik im Grid" Height="200" Width="300"
    >
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="2*" />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="2*" />
        <RowDefinition />
      </Grid.RowDefinitions>

      <Ellipse Grid.Column="0" Grid.Row="0" Fill="Blue" />
      <Rectangle Grid.Column="1" Grid.Row="0" Fill="Red" />
      <Rectangle Grid.Column="0" Grid.Row="1" Fill="Green" />
      <Canvas Grid.Column="1" Grid.Row="1">
        <Ellipse Canvas.Left="10" Canvas.Top="10" Width="70" Height="35"
                 Fill="White" Stroke="Black" StrokeThickness="3" />
      </Canvas>
    </Grid>
</Window>
```

**Listing 14.8** Die Positionierung von grafischen Elementen mit einem Grid

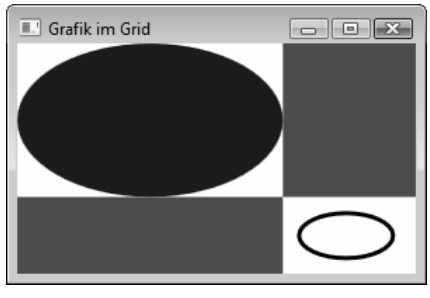



**Abbildung 14.6** Grafische Elemente im Grid

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In Abbildung 14.6 können Sie das Ergebnis des Programms aus Listing 14.8 sehen. In diesem `Grid` ist die erste Zeile doppelt so hoch wie die zweite Zeile. Dies gilt auch für die Spaltenbreiten. Die blaue Ellipse und die beiden Rechtecke füllen jeweils die gesamte Zelle im `Grid` aus, da explizit keine Größen (`Width`, `Height`) angegeben wurden. Die kleine weiße Ellipse wurde mithilfe eines `Canvas`-Elementes positioniert. Das `Canvas`-Element füllt in diesem Fall die gesamte Grid-Zelle aus. Im `Canvas` selbst kann nun die Ellipse beliebig angeordnet werden.

## Einfache Transformationen

Da wir gerade mit Grafiken zu tun haben, stellt sich natürlich auch die Frage, ob es möglich ist, ein schräg liegendes Rechteck oder eine »schräge« Ellipse zu zeichnen. Bisher wurden die Ellipsen und Rechtecke immer so dargestellt, dass die beiden Hauptachsen des Grafikobjektes parallel zu den Achsen des Koordinatensystems lagen. Das soll nun geändert werden.

Die meisten Elemente in WPF enthalten zwei wichtige Transformationen, die als Eigenschaften in den Klassen vorhanden sind: `LayoutTransform` und `RenderTransform`. Die `LayoutTransform`-Eigenschaft wird in erster Linie benutzt, um Teile der Benutzerschnittstelle zu vergrößern oder zu verkleinern. Bei der Anwendung einer `LayoutTransform` werden die einzelnen Elemente in der Benutzerschnittstelle durch das Layout-System von WPF ggf. neu arrangiert. Diese Transformation werden wir später noch genauer kennen lernen.

Die Eigenschaft `RenderTransform` beeinflusst dagegen nur den Inhalt eines Steuerelements. Diese Transformation können wir benutzen, um unsere grafischen Objekte zu drehen. Abbildung 14.7 zeigt das Ergebnis aus Listing 14.9.

```
<Window x:Class="Transform1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Gedrehte Objekte" Height="350" Width="400"
    >
    <Canvas>
      <Rectangle Canvas.Left="100" Canvas.Top="50" Width="200" Height="50" Fill="Red" />

      <Ellipse Canvas.Left="100" Canvas.Top="50" Width="200" Height="50" Fill="Green">
        <Ellipse.RenderTransform>
          <RotateTransform Angle="30" />
        </Ellipse.RenderTransform>
      </Ellipse>

      <Rectangle Canvas.Left="100" Canvas.Top="50" Width="200" Height="50" Fill="Blue">
        <Rectangle.RenderTransform>
          <RotateTransform Angle="60" />
        </Rectangle.RenderTransform>
      </Rectangle>

      <Ellipse Canvas.Left="100" Canvas.Top="50" Width="200" Height="50" Fill="Violet">
        <Ellipse.RenderTransform>
          <RotateTransform Angle="90" />
        </Ellipse.RenderTransform>
      </Ellipse>
    </Canvas>
</Window>
```

**Listing 14.9** Gedrehte Rechtecke und Ellipsen

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Bei der Ausführung einer Rotation ist nicht nur der Rotationswinkel von Bedeutung, sondern auch der Punkt, um den die einzelnen Objekte gedreht werden sollen. Im Beispiel aus Listing 14.9 ist das der obere, linke Eckpunkt der Rechtecke (bzw. Ellipsen). Sie können jedoch einen beliebigen Drehpunkt für die Rotation mit den Eigenschaften `CenterX` und `CenterY` im `RotateTransform`-Objekt angeben (Listing 14.10).

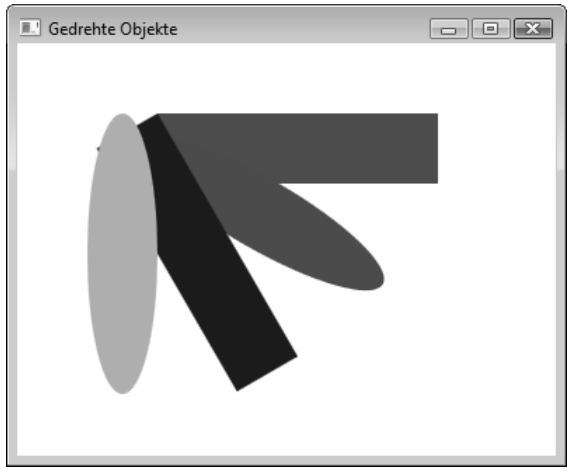



**Abbildung 14.7** Gedrehte Rechtecke und Ellipsen

Die Transformationen im XAML-Code können folgendermaßen angepasst werden:

```
<Rectangle.RenderTransform>
  <RotateTransform Angle="60" CenterX="100" CenterY="25" />
</Rectangle.RenderTransform>
```

Nun liegt der Drehpunkt genau in der Mitte des ersten, roten Rechtecks und die ausgegebene Grafik wird in Abbildung 14.8 gezeigt.

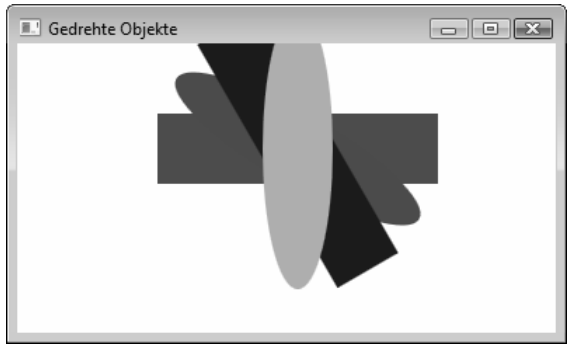



**Abbildung 14.8** Der Drehpunkt liegt in der Mitte des blauen Rechtecks

Das Thema »Transformationen« ist hiermit aber noch nicht abgeschlossen. In den Eigenschaften `LayoutTransform` und `RenderTransform` stecken noch viel mehr Möglichkeiten, die dann an den entsprechenden Stellen vorgestellt werden.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Die Linie

Das einfachste grafische Element ist eine Linie (`Line`). Für eine Linie wird ein Anfangspunkt (`X1`, `Y1`) und ein Endpunkt (`X2`, `Y2`) definiert. Eine Füllfarbe ist natürlich nicht erforderlich. Mit der `Stroke`-Eigenschaft wird die Farbe der Linie festgelegt. Die Linienstärke wird mit der Eigenschaft `Thickness` gesetzt und die Strichelung mit `DashArray`. Bei Linienobjekten können Sie außerdem das Ende der Linien beeinflussen (Listing 14.10). Diese Möglichkeit ist besonders bei dicken Linien wichtig. In Abbildung 14.9 können Sie erkennen, dass das Linienende über den Anfangs- und Endpunkt der Linie hinausragt.

```
<Window x:Class="Linecap.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Linienenden definieren" Height="160" Width="300"
    >
    <Canvas>
      <Line X1="30" Y1="30" X2="250" Y2="30" Stroke="Black" StrokeThickness="20" />
      <Line X1="30" Y1="60" X2="250" Y2="60" Stroke="Black" StrokeThickness="20"
            StrokeStartLineCap="Round" StrokeEndLineCap="Round" />
      <Line X1="30" Y1="90" X2="250" Y2="90" Stroke="Black" StrokeThickness="20"
            StrokeStartLineCap="Triangle" StrokeEndLineCap="Triangle" />
    </Canvas>
</Window>
```

**Listing 14.10** Linienende definieren

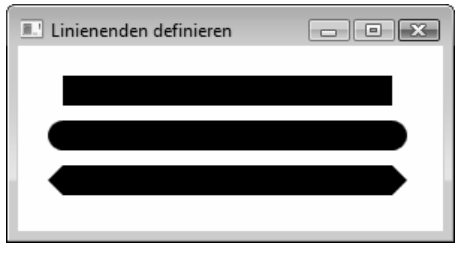



**Abbildung 14.9** Verschiedene Linienenden

# Die Polylinie

Mit dem Element `Polyline` können Sie mehrere Linien, die miteinander verbunden sind, zeichnen. Statt für jedes Liniensegment den Anfangs- und den Endpunkt anzugeben, können Sie hier einfach die einzelnen »Eckpunkte« der Gesamtlinie als Zahlenfeld mit der Eigenschaft `Points` definieren. Der Aufwand ist somit wesentlich geringer, wie auch das Beispiel »Fieberkurve« in Listing 14.11 und Abbildung 14.10 zeigt. Alle anderen, bereits bekannten Eigenschaften können Sie ganz normal benutzen.

```
<Window x:Class="Fieberkurve.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Fieberkurve" Height="180" Width="250"
    >
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
    <Canvas>
      <Polyline Stroke="Red" StrokeThickness="3"
                Points="10,110 20,50 30,20 60,25 75,20 100,120 115,50 135,60 165,65 200,90" />
    </Canvas>
</Window>
```

**Listing 14.11** Eine Linie aus vielen Teilstücken


**Abbildung 14.10** Ein `Polyline`-Element als »Fieberkurve«

Das `Polygon`-Element ist dem `Polyline`-Element sehr ähnlich. Der einzige Unterschied ist, dass ein `Polygon`-Element immer geschlossen wird, d.h., der Anfangspunkt der ersten Linie wird mit dem Endpunkt der letzten Linie verbunden. Sie können das letzte Beispiel nehmen, und den XAML-Code geringfüging ändern (Listing 14.12). Das Ergebnis zeigt Abbildung 14.11.

```
<Canvas>
  <Polygon Stroke="Red" StrokeThickness="3"
           Points="10,110 20,50 30,20 60,25 75,20 100,120 115,50 135,60 165,65 200,90" />
</Canvas>
```

**Listing 14.12** Eine geschlossene Kurve mit Polygon

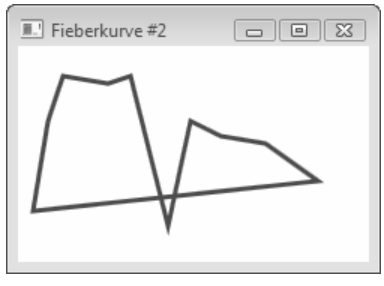



**Abbildung 14.11** Geschlossene Kurve mit `Polygon`

Das automatische Schließen der Kurve vereinfacht viele Zeichenoperationen, da wir uns den Anfangspunkt der gesamten Linie nicht mehr merken müssen. Nun gibt es bei Polygonen noch eine weitere Besonderheit. Da Polygone immer geschlossen werden, kann man sie mit einer Füllfarbe ausfüllen. Aber das ist nicht ganz so einfach, wie es sich im ersten Moment anhört. Wenn wir den Polygon in Abbildung 14.11 betrachten, stellen wir fest, dass es gleich mehrere Flächenteile gibt, die ausgefüllt werden müssen.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Im ersten Versuch wollen wir das letzte Beispiel nur um eine `Fill`-Eigenschaft erweitern:

```
<Canvas>
  <Polygon Stroke="Red" StrokeThickness="3" Fill="Blue"
           Points="10,110 20,50 30,20 60,25 75,20 100,120 115,50 135,60 165,65 200,90" />
</Canvas>
```

**Listing 14.13** Der `Polygon` soll gefüllt worden

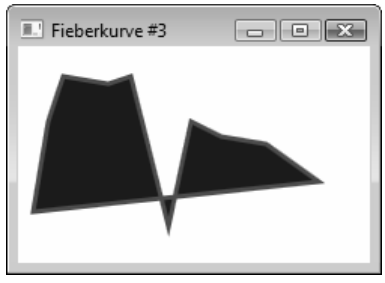



**Abbildung 14.12** `Polygon` mit `Fill`-Eigenschaft

Die Ausgabe von Listing 14.13 zeigt Abbildung 14.12: Alle Teilflächen des Polygons sind ausgefüllt worden. Für das Füllen von Polygonen gibt es zwei Möglichkeiten, die über die Eigenschaft `FillRule` eingestellt werden. Der Standardwert für diese Eigenschaft ist `EvenOdd`. Das bedeutet, dass nur solche Flächen gefüllt werden, von denen aus eine ungerade Anzahl von Linien bis nach außen überquert werden müssen. Schauen Sie sich einfach das folgende Beispiel in Listing 14.14 an:

```
<Window x:Class="FillRule1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Füllen" Height="175" Width="150"
    >
  <Canvas>
    <Polygon Stroke="Red" StrokeThickness="6" Fill="Blue" FillRule="EvenOdd"
             Points="20,70 20,20 120,20 120,120 20,120 20,70 40,70 40,40 100,40 100,100 40,100 40,70" />
  </Canvas>
</Window>
```

**Listing 14.14** Die Füllregel EvenOdd

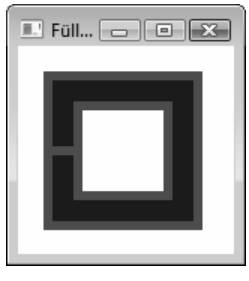



**Abbildung 14.13** Die Füllregel EvenOdd

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

In einem `Polygon`-Element wurden in Abbildung 14.13 zwei Rechtecke mit der Füllregel `EvenOdd` ineinander gezeichnet. Dabei wurde das innere Rechteck nicht mit der Füllfarbe blau ausgefüllt. Wenn wir nun eine Hilfslinie aus dem inneren Rechteck nach außen (bis in die Unendlichkeit) legen (Abbildung 14.14, Linie 1), dann werden zwei Linien des Polygons überquert. Da »Zwei« bekanntermaßen eine gerade Zahl ist, wird die innere Fläche nicht gefüllt. Wenn Sie dagegen aus dem Bereich zwischen dem inneren und dem äußeren Rechteck eine Linie bis in die Unendlichkeit zeichnen (Abbildung 14.14, Linie 2 oder 3), werden eine oder drei Polygon-Linien überschritten. Wir haben eine ungerade Zahl, d.h., die Fläche wird mit blauer Farbe ausgefüllt.

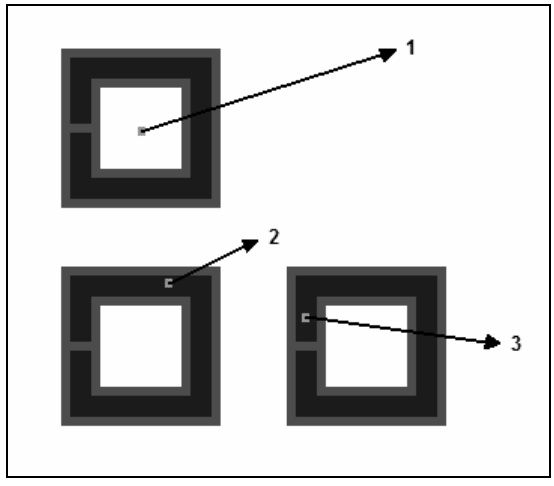



**Abbildung 14.14** Die Füllregel EvenOdd

Die zweite Füllregel `NonZero` ist etwas schwerer verständlich. Für diese Regel benutzen wir ein etwas anderes Beispiel, welches in Listing 14.15 dargestellt ist.

```
<Window x:Class="FillRule2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Füllen" Height="200" Width="350"
    >
    <Canvas>
      <Polygon Stroke="Red" StrokeThickness="4" Fill="Blue" FillRule="EvenOdd"
               Points="20,50 20,100 70,100 70,20 130,20 130,70 50,70 50,130 100,130 100,50" />
      <Polygon Stroke="Red" StrokeThickness="4" Fill="Blue" FillRule="NonZero"
               Points="170,50 170,100 220,100 220,20 280,20 280,70 200,70 200,130 250,130 250,50" />
    </Canvas>
</Window>
```

**Listing 14.15** Die Füllregeln EvenOdd und NonZero

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

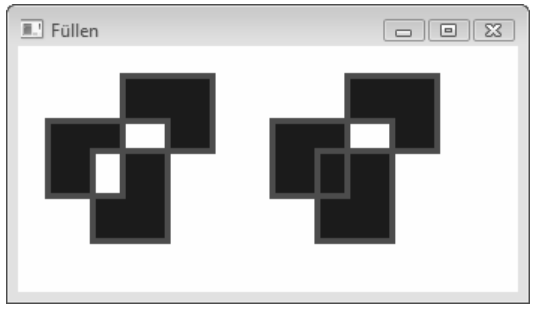



**Abbildung 14.15** Die Füllregeln EvenOdd und NonZero

In Abbildung 14.15 sehen Sie auf der linken Seite einen Polygon mit der Füllregel `EvenOdd`, rechts dagegen ist der gleiche Polygon mit der Regel `NonZero` dargestellt. Das Ergebnis mit `EvenOdd` ist klar, wenn wir wieder die Hilfslinien gedacht nach außen ziehen.

Die Füllregel `NonZero` liefert bei einer ungeraden Anzahl von Linienüberquerungen das gleiche Ergebnis beim Füllen wie die Regel `EvenOdd`. Wenn jedoch eine gerade Anzahl von Linienüberquerungen notwendig ist, wird die Teilfläche nur dann gefüllt, wenn die Anzahl der Linien, die in eine bestimmte Richtung relativ zur Grafik verlaufen, ungleich der Anzahl der Linien in die andere Richtung sind. Abbildung 14.16 stellt dieses Verhalten etwas genauer dar. Die Zeichenrichtung für die entscheidenden Linien ist dort durch Pfeile dargestellt.

Die Hilfslinie Nr. 1 in Abbildung 14.16 überquert zwei Linien, welche in die gleiche Richtung führen. Da nun in die gegensätzliche Richtung gar keine Linie verläuft, wird der Teilbereich mit der angegebenen Füllfarbe ausgefüllt. Bei der Hilfslinie Nr. 2 werden zwei Linien überquert, von denen eine nach rechts und die andere Linie nach links verläuft. Damit ist die Anzahl der Linien, die jeweils in eine Richtung verläuft, gleich und die Teilfläche wird nicht gefüllt.

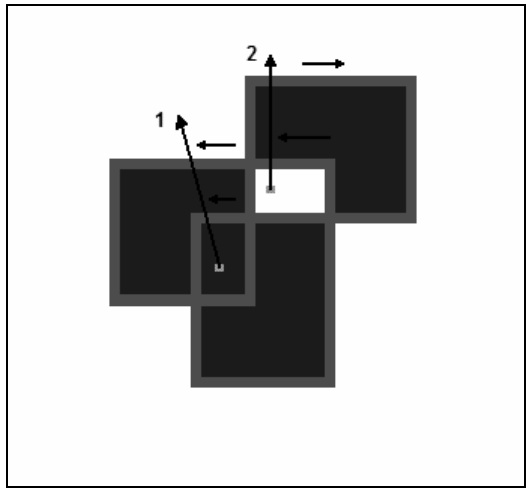



**Abbildung 14.16** Die Füllregel NonZero

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Das Path-Element

Ein weiteres grafisches Grundelement ist das Path-Element. Es ist das leistungsfähigste 2D-Grafikobjekt in WPF. Mithilfe des Path-Elementes können Sie sehr komplexe Figuren aufbauen und darstellen. Das Path-Element enthält die Eigenschaft Data, über die Sie ein oder mehrere grafische Grundelemente definieren können. Die Elemente, die Sie dort benutzen können, heißen RectangleGeometry, EllipseGeometry, LineGeometry, usw. Mehrere Grafikelemente können in einem Path mit einer GeometryGroup zusammengefasst werden. Listing 14.16 zeigt ein Beispiel.

```
<Window x:Class="Path1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Ein grafischer Pfad" Height="300" Width="300"
    >
    <Grid>
      <Path Stroke="Red" StrokeThickness="3" Fill="Yellow" >
        <Path.Data>
          <GeometryGroup>
            <RectangleGeometry Rect="0, 0, 150, 100" />
            <EllipseGeometry Center="75, 50" RadiusX="75" RadiusY="50" />
            <LineGeometry StartPoint="0, 0" EndPoint="150,100" />
            <LineGeometry StartPoint="0, 100" EndPoint="150, 0" />
          </GeometryGroup>
        </Path.Data>
      </Path>
    </Grid>
</Window>
```

**Listing 14.16** Mit einem Path-Objekt komplexere Grafiken definieren

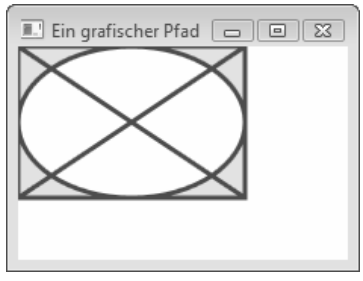



**Abbildung 14.17** Das Path-Element mit Ellipse, Rechteck und Linien

Für das Path-Element steht ebenfalls die Eigenschaft Fill zur Verfügung, um eine Füllfarbe für die gesamte Figur zu definieren. Im Element GeometryGroup dagegen befindet sich die Eigenschaft FillRule, die wir aus den vorherigen Beispielen kennen und die sich auch genau so verhält. In Abbildung 14.17 wird die innere Ellipse nicht gelb ausgefüllt, weil die Eigenschaft FillRule standardmäßig auf EvenOdd gesetzt ist und die Anzahl der Linien, die von innen nach außen durchquert werden müssen, gerade ist.

Innerhalb eines Path-Elements können Sie sehr komplexe Figuren definieren. In Listing 14.17 wird eine PathGeometry definiert. Diese kann mehrere Figuren beinhalten, die bei Bedarf auch geschlossen werden können. Jede Figur wiederum besteht aus einem Startpunkt und mehreren Segmenten, welche dann die

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

gewünschte Figur aufbauen. Die Segmente (Tabelle 14.1) können Linien, Bögen oder auch komplexe Kurven sein. Diese einzelnen Segmente werden fortlaufend aneinander gezeichnet. In der Eigenschaft `PathFigure.Segments` können Sie beliebig viele, unterschiedliche Segmente für eine Figur definieren.

```
<Window x:Class="Path2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Mehrere Teile" Height="300" Width="300"
    >
    <Canvas>
      <Path Fill="Blue" Stroke="Black" StrokeThickness="3">
        <Path.Data>
          <PathGeometry>
            <PathGeometry.Figures>
              <PathFigureCollection>
                <PathFigure StartPoint="50,70" IsClosed="True">
                  <PathFigure.Segments>
                    <PathSegmentCollection>
                      <LineSegment Point="100,40" />
                      <LineSegment Point="200,70" />
                      <LineSegment Point="250,100" />
                      <LineSegment Point="250,150" />
                      <ArcSegment Point="200,200" Size="50,50" SweepDirection="Clockwise"/>
                    </PathSegmentCollection>
                  </PathFigure.Segments>
                </PathFigure>
              </PathFigureCollection>
            </PathGeometry.Figures>
          </PathGeometry>
        </Path.Data>
      </Path>
    </Canvas>
</Window>
```

**Listing 14.17** Eine komplexe Figur mit einem `Path`-Element

| Segmenttyp | Verhalten |
|---|---|
| `LineSegment` | Einfache gerade Linie |
| `PolyLineSegment` | Mehrere gerade Linien |
| `ArcSegment` | Bogensegment |
| `BezierSegment` | Kubische Bezierkurve |
| `QuadraticBezierSegment` | Quadratische Bezierkurve |
| `PolyBezierSegment` | Mehrere kubische Bezierkurven |
| `PolyQuadraticBezierSegment` | Mehrere quadratische Bezierkurven |

**Tabelle 14.1** Die möglichen Segmenttypen in einem `Path`-Element

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## Hit-Testing mit dem Path-Element

Wenn wir komplexe Grafiken mit dem `Path`-Element erzeugen, ergibt sich schnell die Frage, ob wir einfach feststellen können, wann in eine geschlossene Kurve hineingeklickt wurde. Eine einfache Demo kann den Sachverhalt klären. Wir erweitern hierzu das Beispiel aus Listing 14.17 und implementieren das `MouseDown`-Ereignis für das `Path`-Element:

```
<Path Fill="Blue" Stroke="Black" StrokeThickness="3" MouseDown="OnMouseDownPath">
```

Die Ereignismethode wird in VB implementiert und gibt einfach nur eine `MessageBox` aus, wenn das `Path`-Element mit dem Mausklick getroffen wurde.

```
Imports System
Imports System.Windows
Imports System.Windows.Input

Namespace Path4
    Partial Public Class Window1
        Inherits System.Windows.Window

        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnMouseDownPath(ByVal sender As Object, ByVal e As MouseEventArgs)
            MessageBox.Show("Im Path-Element!")
        End Sub
    End Class
End Namespace
```

Wenn Sie das Beispielprogramm laufen lassen, merken Sie sofort, dass nur Klicks innerhalb des `Path`-Elements die Meldung erscheinen lassen. Dadurch ist das Hit-Testing in diesem Fall sehr einfach zu implementieren.

Genauso einfach funktioniert das übrigens auch mit `Polygon`-Elementen. Hier werden beim Hit-Testing im Polygon die Einstellungen der `FillRule`-Eigenschaft berücksichtigt. Wenn ein Bereich des Polygons mit der gewählten Füllfarbe gezeichnet wird, dann wird auch das `MouseDown`-Ereignis in diesem Bereich ausgelöst (siehe Abbildung 14.15). In den Bereichen, die in Abbildung 14.15 nicht in blau eingefärbt sind, wird auch das `MouseDown`-Ereignis nicht ausgelöst. In den `Polygon`-Elementen wird nur das Mausereignis implementiert, das dann im Code verarbeitet werden kann:

```
<Polygon Stroke="Red" StrokeThickness="4" Fill="Blue" FillRule="EvenOdd" MouseDown="OnMouseDown"
         Points="20,50 20,100 70,100 70,20 130,20 130,70 50,70 50,130 100,130 100,50" />
<Polygon Stroke="Red" StrokeThickness="4" Fill="Blue" FillRule="NonZero" MouseDown="OnMouseDown"
         Points="170,50 170,100 220,100 220,20 280,20 280,70 200,70 200,130 250,130 250,50" />
```

Es gibt jedoch auch komplizierte Fälle beim Hit-Testing, welche die Implementierung von etwas mehr Code erfordern. In den oben gezeigten Beispielen werden nur die Mausklickpositionen für den Hit-Test herangezogen. Wir wollen ein Beispiel nun so abändern, dass ein kreisförmiger Bereich um die Klickposition herum als Testbereich verwendet wird.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

Der XAML-Code mit einem Path-Element für das erweiterte Hit-Testing wird in Listing 14.18 gezeigt. Das MouseDown-Ereignis wird in diesem Beispiel im Hauptfenster Window1 »eingehängt«, damit wir das Ereignis auch dann auswerten können, wenn nicht direkt in das Path-Element geklickt wurde. Im Code (Listing 14.19) wird in der Mausereignis-Methode zunächst die Position des Klicks ermittelt. Dann wird ein EllipseGeometry-Objekt angelegt, in dem ein Kreis von 20 Einheiten Durchmesser definiert wird. Nun kann die statische Methode HitTest aus der VisualTreeHelper-Klasse benutzt werden, um im Trefferfall eine Rückrufmethode mit dem Namen HitResult aufzurufen. Dort wird das Hit-Testergebnis ausgewertet. Mit der Eigenschaft IntersectionDetail aus der Klasse GeometryHitTestResult können wir dann ermitteln, ob der angelegte Kreis vollständig, teilweise oder gar nicht im definierten Path-Element liegt. Nachdem ein Treffer erzielt wurde, können Sie die Suche fortführen (HitTestResultBehavior.Continue) oder abbrechen (HitTestResultBehavior.Stop). Eine Steuerung der Suche über die Aufzählung HitTestResultBehavior ist besonders dann interessant, wenn mehrere grafische Objekte übereinander liegen und ermittelt werden soll, ob ein ganz bestimmtes Grafikobjekt angeklickt wurde. Bei der Benutzung von Continue wird die Rückrufmethode für das nächste getroffene Objekt erneut aufgerufen. Dort findet dann die weitere Verarbeitung der Treffer-Informationen statt. Eine Verwendung von Stop beendet die laufende Trefferanalyse und die Rückrufmethode wird nicht mehr aufgerufen, auch wenn es noch weitere Trefferobjekte geben sollte.

```
<Window x:Class="HitTesting.Window1"
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    Title="Hit-Testing" Height="300" Width="300" MouseDown="OnMouseDown"
    >
    <Grid>
      <Path Name="path" Stroke="Black" StrokeThickness="3" Fill="Red">
        <Path.Data>M 50,50 h 130 l 50,50 v 100 h -100 v -50 h 50 Z</Path.Data>
      </Path>
    </Grid>
</Window>
```

**Listing 14.18** Erweitertes Hit-Testing

```
Imports System
Imports System.Windows
Imports System.Windows.Input
Imports System.Windows.Media

Namespace HitTesting
    Partial Public Class Window1
        Inherits System.Windows.Window
        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub OnMouseDown(ByVal sender As Object, _
                                ByVal e As MouseEventArgs)
            ' Mausposition ermitteln
            Dim pt As Point = e.GetPosition(DirectCast(sender, UIElement))
            ' Fläche für Trefferbereich festlegen
            Dim newHitTestArea As EllipseGeometry = New EllipseGeometry(pt, 20.0, 20.0)
            ' Rückrufprozedur für einen Treffer einrichten
```

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

```
            VisualTreeHelper.HitTest(path, Nothing, _
                                     New HitTestResultCallback(AddressOf HitResult), _
                                     New GeometryHitTestParameters(newHitTestArea))
        End Sub

        Public Function HitResult(ByVal result As HitTestResult) As HitTestResultBehavior

            ' Ermitteln des Hit-Test-Ergebnisses
            Select Case DirectCast(result, GeometryHitTestResult).IntersectionDetail
                Case IntersectionDetail.FullyInside
                    MessageBox.Show("Vollständig innen!")
                    Return HitTestResultBehavior.Continue
                Case IntersectionDetail.FullyContains
                    MessageBox.Show("Voll getroffen!")
                    Return HitTestResultBehavior.Stop
                Case IntersectionDetail.Intersects
                    MessageBox.Show("Teilweise getroffen!")
                    Return HitTestResultBehavior.Continue
            End Select
            Return HitTestResultBehavior.Stop
        End Function
    End Class
End Namespace
```

**Listing 14.19** Das erweiterte Hit-Testing

# Zusammenfassung

In diesem Kapitel haben Sie erfahren, wie vielfältig die verschiedenen Grafikelemente von Windows Presentation Foundation eingesetzt werden können. Entscheidend bei der Darstellung einer Benutzerschnittstelle ist die einfache Kombination von Steuerelementen und grafischen Elementen.

Viele interessante Vereinfachungen werden durch Anwendung der `Path`-Klasse ermöglicht. Die Definition und Zusammenfassung von grafischen Figuren wird mit einem `Path`-Element stark vereinfacht.
Entscheidende Design-Möglichkeiten werden aber durch überall anwendbare Transformationen geschaffen. Es ist ohne großen Programmieraufwand mit WPF möglich, einerseits moderne Layouts für Spielprogramme zu definieren, aber andererseits auch eine vollständig in der Größe einstellbare Benutzerschnittstelle zu erstellen, die einen hohen Komfort beim Arbeiten mit dieser Applikation ermöglicht.

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

# Stichwortverzeichnis



## A




"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## B



## C



## D



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## E



## F



## G



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## H



## I



## J



## K



## L



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## M



## N



## O



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## P



## Q



## R



## S



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## T



## U



## V



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de

## W



## X



## Z



"Visual Basic 2008 - Neue Technologien - Crashkurs", © 2008 Microsoft Press, 978-3-86645-518-4

Dieses Buch und viele andere zum Thema .NET-Programmierung finden Sie auf www.microsoft-press.de

Informationen zum Autor und die Möglichkeit Feedback zu geben finden Sie unter activedevelop.de