

# 3 Formular-Designer und Codeeditor enthüllt

---

38	<b>Das Fallbeispiel – Der DVD-Hüllen-Generator »Covers«</b>
43	<b>Gestalten von Formularen mit dem Windows Forms-Designer</b>
71	<b>Der Codeeditor</b>
79	<b>Smarttags im Editor von Visual Basic</b>
80	<b>Erzwungene Typsicherheit (Option Strict) projektweit einstellen</b>
101	<b>Weitere Funktionen des Codeeditors</b>

---

Wenn es darum geht, Windows Forms-Anwendungen, die auch unter dem Modenamen »Smartclients«<sup>1</sup> gehandelt werden, zu entwickeln – und darauf legt dieses Buch erklärterweise seinen Schwerpunkt – dann sind der Formular-Designer und der Codeeditor die Werkzeuge, die Sie neben den Debugging-Werkzeugen wohl am häufigsten verwenden werden. Darum sei ihnen auch ein eigenes Kapitel gewidmet.

Nun gibt es zwei nahe liegende Ansätze, die wichtigen Werkzeuge einer Entwicklungsumgebung zu erklären: eine theoretische und eine praktische Vorgehensweise. Die theoretische gibt Ihnen einen schnellen Überblick, was für Möglichkeiten es bei dem einen oder anderen Werkzeug gibt, macht Sie aber nicht praxissicher. Die praktische bringt ungleich mehr: Vielleicht unterfordert Sie das Beispiel aus Entwicklersicht ein wenig, das Sie dazu am besten von vorne bis hinten durchexerzieren müssen; aber auf diese Weise lernen Sie auf schnellste Weise die Feinheiten und Möglichkeiten der Werkzeuge kennen, und Sie sind viel besser für die Praxis gerüstet.

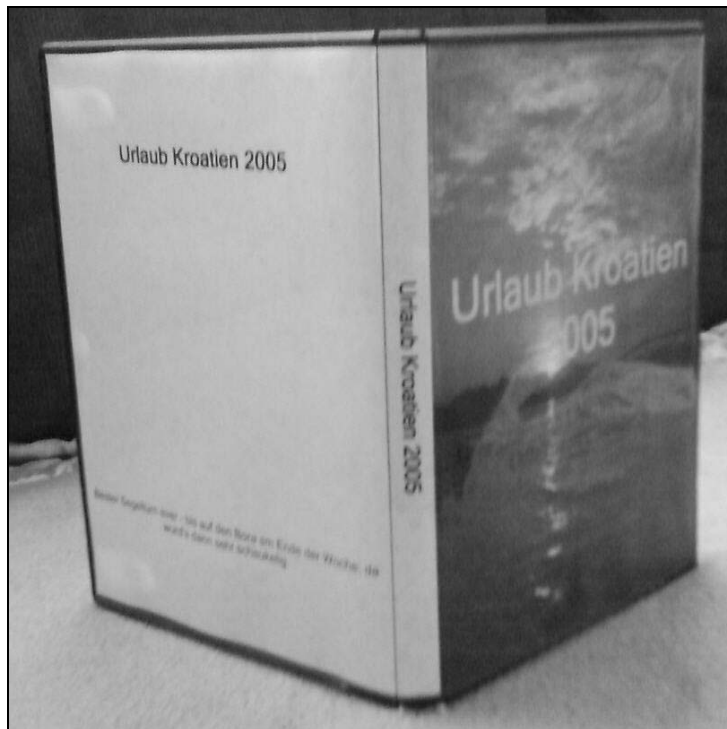
Aus diesem Grund habe ich mich dazu entschieden, die Vorstellung der Möglichkeiten von Formular-Designer und Codeeditor an einem konkreten Beispiel durchzuexerzieren. Das Beispiel geht sogar noch ein wenig über die thematischen Kapitelgrenzen hinaus und liefert Ihnen schon jetzt die eine oder andere Info zu völlig anderen Themen, die mir aber schon zu diesem Zeitpunkt im Kontext sinnvoll erscheinen. Natürlich werden fast alle Punkte im Laufe des Buches noch weiter vertieft – und denken Sie daran, dass es in erster Linie darum geht, Ihnen schnellstmöglich zu so viel Repertoire zu verhelfen, dass Sie ohne das Buch zunächst komplett auswendig lernen zu müssen, schon Ihre ersten kleinen Projekte realisieren können.

---

<sup>1</sup> Die ► Kapitel ab Teil G beschäftigen sich mit diesem Thema im Detail.

## Das Fallbeispiel – Der DVD-Hüllen-Generator »Covers«

Es ist schon eindrucksvoll, wie Zeitschriften, Zeitungen, Tankstellen, Buchclubs und andere Institutionen inzwischen um die Gunst ihrer Kunden werben. Auf diese Weise bin ich in den Genuss einer DVD-Sammlung gekommen, die sich sehen lassen kann, und für die ich vergleichsweise wenig Geld investiert habe. So liegen beispielsweise fast allen großen Fernsehzeitschriften immer mal wieder DVDs einfach so als »Goody« bei – klasse Spielfilme, mehrsprachig, Untertitelt und in überragender Qualität, quasi zum Nulltarif. Doch was verständlicherweise fehlt ist jeweils ein ordentliches Cover, mit dessen Hilfe man diese DVD wie ein Buch in den Schrank stellen könnte, und in meterlangen Regalen – so die Idee – DVDs im Bedarfsfall auch wieder finden kann. Das gilt umso mehr für die Cover von Filmen, die man beispielsweise von Pay-TV-Sendern aufgenommen oder sogar selber gedreht hat. Klar – es gibt die unterschiedlichsten Cover-Designer für wenig Geld auf dem Markt; und jeder, der mit seinem Computer einen CD- oder DVD-Brenner erworben hat, wird ohnehin ein solches Programm sein Eigen nennen können, denn die meisten Brennprogramme, die den Brennern beiliegen, verfügen über teilweise recht leistungsfähige Designer.



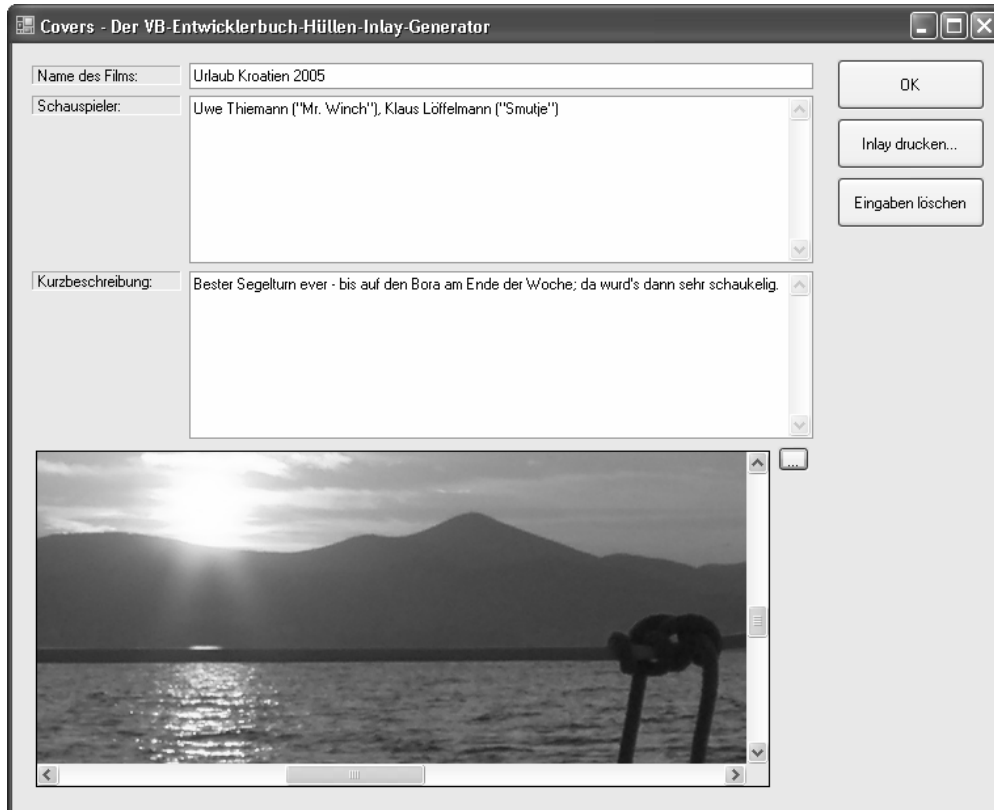
**Abbildung 3.1:** Ein ausgedrucktes Inlay schneiden Sie einfach entlang der Markierungslinien aus und stecken es in das Plastikcover – hier ein Beispiel

Diese allerdings können in meinen Augen und für meine Belange viel zu viel. Wenn ich Zeit hätte, könnte ich damit bestimmt auch Cover gestalten, die vielleicht sogar fast genau so gut sind wie manches Original. Doch habe ich nie oder selten Zeit, und sollte ich sie doch mal haben, werde ich sicherlich keine Cover in meiner freien Zeit gestalten. Was ich brauche – was viele andere vielleicht auch haben wollen – ist ein Programm, mit dem ich Titel, Schauspieler, Kurzbeschreibung eintippen, vielleicht noch ein Bild auswählen kann, und das mir daraus dann ein Cover zaubert. Das geht schnell und reicht völlig.

## Das »Pflichtenheft« von Covers

Ein solches Programm sollte folgendermaßen funktionieren:

- Das Programm sollte einen einfachen, aber dynamisch vergrößerbaren Dialog anbieten, in dem man die Grunddaten des Films in simplen Texteingabefelder erfassen kann: Filmtitel, Schauspieler, Kurzbeschreibung und ein Bild, das auf der Frontseite des Covers abgedruckt werden soll. Abbildung 3.1 zeigt, wie so was in etwa im Ergebnis aussuchen kann; die beiden folgenden Abbildungen vermitteln Ihnen einen Eindruck von der Bedienung des Programms.



**Abbildung 3.2:** Dazu soll »Covers« in der Lage sein: Aus einer simplen Erfassung der Filmdaten druckt es buchstäblich auf Knopfdruck ...

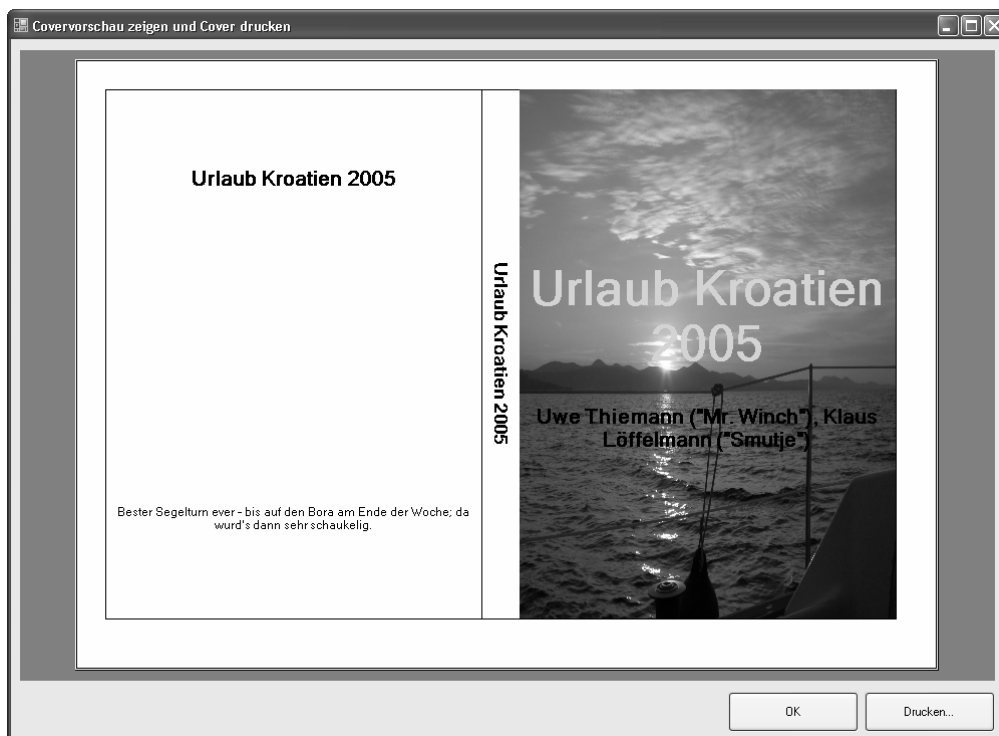


Abbildung 3.3: ... das Inlay für das Film-Cover

- Wenn Sie das Programmfenster von »Covers« vergrößern oder verkleinern, sollen sich alle Steuerelemente automatisch an die Größe des Formulars anpassen. Die Steuerelemente wachsen nach unten im Verhältnis zur Vergrößerung des Formulars mit; beim Verkleinern werden auch die Steuerelemente wieder entsprechend niedriger. Vergrößern Sie das Formular nach rechts, wandern die Schaltflächen auf der rechten Seite quasi mit dem rechten Fensterrand mit. Die anderen Steuerelemente, die für die Dateneingaben vorhanden sind, vergrößern sich dementsprechend.
- Sie können ein Bild in das Formular laden, das dann später automatisch auf die kompletten Ausmaße der Covervorderseite skaliert wird. Sollte das Bild nicht in das Bildelement auf dem Formular passen, erscheinen automatisch Rollbalken, mit denen Sie den Ausschnitt des Bildes einstellen können.
- Mit der Schaltfläche *Eingaben löschen* setzen Sie alle Eingaben wieder zurück. Diese Schaltfläche ist praktisch, wenn Sie direkt hintereinander mehrere Cover drucken wollen.
- Das Programm soll sich alle Eingaben »merken«. Wenn Sie das Programm verlassen und neu starten, sollen sämtliche zuletzt getätigte Eingaben oder Bilddefinitionen wieder so vorzufinden sein, wie sie vor dem letzten Beenden des Programms vorhanden waren.
- Wenn Sie Daten in das Formular eingetragen haben, gelangen Sie durch *Inlay drucken* zur Druckvorschau, die Ihnen das Inlay auf dem Bildschirm so darstellt, wie es später auch gedruckt werden soll. Auch dieses Vorschaufenster soll sich dynamisch vergrößern lassen.

- Vor dem Drucken sollen Sie die Möglichkeit haben, einen Drucker, den Sie verwenden wollen, auszuwählen und einzustellen. Der Drucker soll das Inlay aber immer unabhängig von den Einstellungen im Querformat drucken.

Die folgende Schritt-für-Schritt-Anleitung, die sich über mehrere Abschnitte erstreckt, zeigt, wie Sie diese Software von A–Z mit Visual Studio 2005 und Visual Basic 2005 erstellen. Sie werden erstaunt sein, wie wenig Code dazu nötig ist – das eigentliche Drucken macht dabei noch den größten Teil aus.

---

**BEGLEITDATEIEN:** Natürlich ist das vollständige Projekt in den Begleitdateien zum Buch enthalten – Sie sollten aber in diesem Fall besser vollständig auf sie verzichten und kleinere Codepassagen wirklich abtippen, um ein Gefühl und besseres Verständnis für den Codeeditor zu erlangen. Die vergleichsweise umfangreiche Druckroutine wäre allerdings doch ein wenig zu viel des Guten, und Sie finden sie deswegen als reine Textdatei im Verzeichnis `.|VB 2005 - Entwicklerbuch|B - IDE|03 - Covers|Druckroutine für Covers.txt`.

---

## Erstellen eines neuen Projektes

Und los geht's. Wir beginnen so einfach wie möglich: mit dem Erstellen eines neuen Projektes. Dazu starten Sie Visual Studio, falls Sie es noch nicht getan haben.

---

**HINWEIS:** Um den Speicherort des Projektes, wie im Folgenden beschrieben, schon beim Anlegen festlegen zu können, rufen Sie die *Options*-Dialog über den Menübefehl *Extras/Optionen* auf. Setzen Sie das Häkchen *Neues Objekt beim Erstellen speichern* im Bereich *Projekte und Projektmappen/Allgemein*.

---

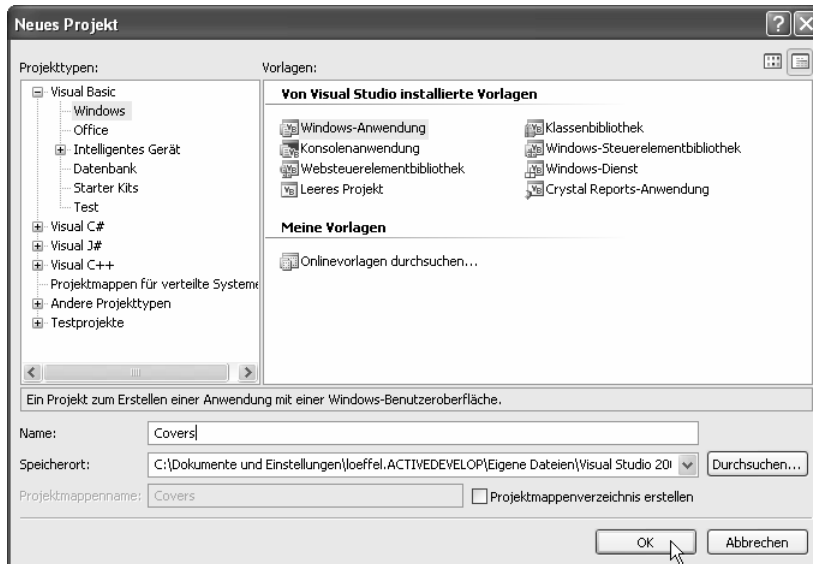
1. Um ein neues Projekt anzulegen, klicken Sie auf der Startseite im linken oberen Bereich auf *Projekt* hinter *Erstellen*: Abbildung 3.4 hilft Ihnen bei der Orientierung.



**Abbildung 3.4:** Wenn Sie ein neues Projekt erstellen wollen, rufen Sie die entsprechende Funktion direkt aus der Startseite auf

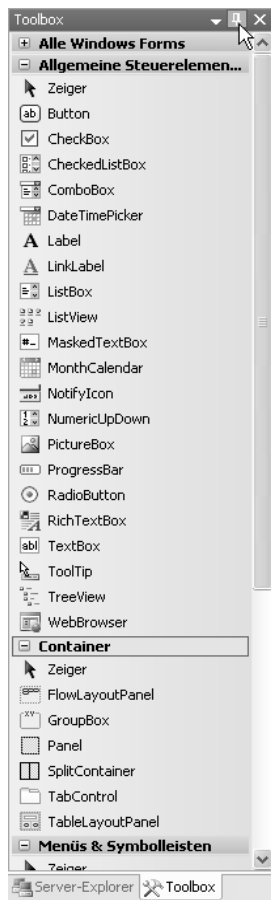
2. Im anschließend erscheinenden Dialog wählen Sie in der Baumstruktur unter *Projekttypen* den Zweig *Visual Basic/Windows*. Unter *Vorlagen* wählen Sie *Windows-Anwendung*. Geben Sie unter *Namen* den Namen Ihres neuen Projektes ein – für unser Beispiel wählen Sie *Covers*. Unter *Speicherort* bestimmen Sie das Verzeichnis, in dem alle Projektdateien gespeichert werden sollen. *Durchsuchen...* bringt Sie dafür zu einem Dialog, der Ihnen bei der Bestimmung des Verzeichnisses helfen kann. Stellen Sie sicher, dass das Häkchen neben *Projektmappenverzeichnis erstellen* nicht zu sehen ist. Abbildung 3.5 hilft Ihnen bei der Orientierung. Klicken Sie anschließend auf *OK*.

**TIPP:** Wenn Sie mehrere Projekte in einer Projektmappe erstellen möchten – Sie möchten beispielsweise eine Anwendung in mehrere Schichten (Datenschicht, Geschäftslogik, Benutzeroberfläche, Steuerelemente, etc.) aufteilen und diese Schichten in verschiedenen Klassenbibliotheken verteilen –, können Sie Ihr Windows-Anwendungsprojekt schon jetzt in einer Projektmappe einordnen. Die Projektmappe bekommt dann ein eigenes Verzeichnis; ihr Windows-Projekt liegt in einem Verzeichnis unterhalb des Projektmappen-Verzeichnisses. In diesem Fall setzen Sie das Häkchen in *Projektmappenverzeichnis erstellen*, und geben Sie im Eingabefeld *Projektmappenname* den Namen der Projektmappe ein.



**Abbildung 3.5:** Mit diesem Dialog richten Sie ein neue Windows-Anwendung ein

# Gestalten von Formularen mit dem Windows Forms-Designer



Wir beginnen unser Fallbeispiel aus den letzten Abschnitten mit dem Entwurf des Formulars. Da wir die Toolbox zu diesem Zweck häufig gebrauchen werden, macht es Sinn, diese ständig im Vordergrund zu haben. An der linken Seite der Visual Studio-IDE finden Sie eine »eingefahrene« Registergruppe, bestehend aus dem Server-Explorer und der Toolbox.

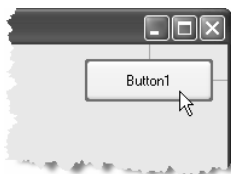
Fahren Sie mit dem Mauszeiger auf das Toolbox-Symbol, worauf sich die Toolbox in den Vordergrund schiebt, und klicken Sie auf das Pin-Symbol im Fenstertitel (links im Bild zu sehen). Dadurch bleibt die Toolbox, wo sie ist, und fährt nicht wieder automatisch in den Hintergrund, wenn der Mauszeiger sie verlässt.

Damit viel Platz für Experimente bleibt, vergrößern Sie das vorhandene Formular, sodass es einen Großteil der Arbeitsfläche einnimmt.

## Positionieren von Steuerelementen

Wie in Abbildung 3.2 zu sehen, verfügt das Formular über drei Schaltflächen, mit denen verschiedene Funktionen ausgelöst werden. Diese Schaltflächen werden wir als erstes auf dem Formular positionieren.

1. Dazu klicken Sie in der Toolbox auf das *Button*-Symbol. Fahren Sie mit dem Mauszeiger in das Formular, und ziehen Sie eine Schaltfläche ungefähr in der Größe auf, wie Sie den Schaltflächen in Abbildung 3.2 entspricht.
2. Ziehen Sie zwei weitere Schaltflächen irgendwo im Formular auf – Größe und Position der Schaltflächen sind dabei zunächst völlig egal. Wir werden gleich die entsprechenden Funktionen kennen lernen, mit denen wir die Schaltflächen angleichen können.
3. Per Drag&Drop verschieben Sie nun die Schaltfläche mit der Aufschrift *Button1* in die rechte, obere Ecke des Formulars. Wenn Sie in der oberen, rechten Ecke ankommen, werden Sie merken, dass die Schaltfläche an einer bestimmten Ecke »anschnappt«, und Sie können zwei Hilfslinien erkennen (siehe Abbildung 3.6), die die Schaltfläche auf Abstand halten.



**Abbildung 3.6:** Ausrichtungslinien helfen Ihnen bei der Positionierung von Steuerelementen am Formularrand oder untereinander

## Ausrichtungslinien (Guidelines) und die Margin/Padding-Eigenschaften von Steuerelementen

Jedes Steuerelement, das Sie im Formular positionieren können, basiert auf einem Grundsteuerelement namens `Control`. Dieses Steuerelement stellt nicht nur einen Grundpool an Funktionalität zur Verfügung, auf denen andere Steuerelemente beispielsweise für das Anzeigen ihrer eigentlichen Inhalte (eine Schaltfläche muss andere Inhalte darstellen als ein `CheckBox`-Steuerelement) aufbauen können – es stellt auch eine grundsätzliche Designerfunktionalität zur Verfügung (der folgende graue Kasten liefert genauere Infos zu diesem Thema).

Das ist der Grund, wieso jedes Steuerelement, das Sie im Formular platzieren können, automatisch und ausnahmslos über eine `Margin`-Eigenschaft verfügt. Diese Eigenschaft bestimmt, wie groß der Abstand zwischen einem Steuerelement und einem anderen Steuerelement sein wird, wenn Sie bei deren Platzierung die durch die Ausrichtungslinien vorgegebenen Abstände akzeptieren (oder mit anderen Worten: ab welchem Abstand die Ausrichtungslinien »anspringen« und das Steuerelement bei diesem Abstand »einschnappt«).

Wenn Steuerelemente als »Träger« anderer Steuerelemente fungieren – ein solches Steuerelement nennt man übrigens `Container`-Steuerelement bzw. `ContainerControl` – kommt eine weitere Eigenschaft ins Spiel: die `Padding`-Eigenschaft. Diese Eigenschaft bestimmt, wie viel Abstand zusätzlich zum äußeren Rand eingehalten werden soll, wenn ein beinhaltendes Steuerelement am Rand eines beinhaltenden Steuerelements angeordnet wird.

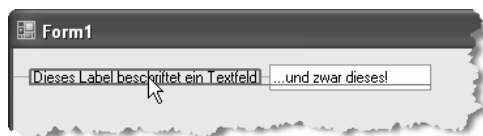
---

**HINWEIS:** Formulare selbst machen dabei offensichtlich noch eine Ausnahme: Beim Anordnen von Steuerelementen am Rand von Formularen werden zusätzlich ein paar Pixel Abstand eingehalten, wenn die Ausrichtungslinien dargestellt werden, möglicherweise um Microsofts Style-Guide für das Gestalten von Formularen zu entsprechen.<sup>2</sup>

---

**TIPP:** Das Einschnappen geschieht übrigens nicht nur an Stellen, die den durch die `Margin`- bzw. die `Padding`-Eigenschaften festgelegten Abständen entsprechen. Visual Studio stellt Ihnen das Einschnappen auch zur Verfügung, um Steuerelemente an Basislinien ihrer Textzeilen exakt auszurichten, wenn diese nebeneinander stehen. Das ist häufig der Fall, wenn Sie ein `Label`-Steuerelement dafür verwenden, ein `TextBox`-Steuerelement im Formular zu beschriften, wie in Abbildung 3.7 zu sehen.

---



**Abbildung 3.7:** Gerade beim Beschriften von `TextBox`-Steuerelementen mit `Label`-Steuerelementen helfen Ihnen Ausrichtungslinien, da sich die Steuerelemente auch an den Textbasislinien einschnappen

---

<sup>2</sup> Die Online-Hilfe meint zu diesem Thema sinngemäß: »Die Ausrichtungslinien entsprechen der Addition von `Padding`- und `Margin`-Eigenschaften«; für Formulare scheint das nicht zu gelten. Eine entsprechende Diskussion finden Sie im Product Feedback Center unter dem IntelliLink `B0301`.

---

**TIPP:** Entwickler, die lieber mit der Rasterfunktionalität arbeiten, die sie aus Visual Studio .NET 2003 bzw. Visual Basic 6.0 kennen, müssen darauf auch in Visual Studio 2005 nicht verzichten. Im *Optionen*-Dialog von Visual Studio, den Sie über das *Extras*-Menü erreichen, können Sie auf der Registerkarte *Windows Forms-Designer* die *LayoutMode*-Eigenschaft von *SnapLines* auf *SnapToGrid* ändern. Möchten Sie zwar im neuen Layout-Modus von Visual Studio 2005 arbeiten, aber ohne die Einschnapp-Funktionalität auskommen, setzen Sie im gleichen Dialog die *SnapToGrid*-Eigenschaft auf *False*.

---

## Wem »gehört« eigentlich der Formular-Designer?

Rein rechtlich gesehen, natürlich Ihnen – schließlich haben Sie Visual Studio und damit auch den Formular-Designer erworben. Aber das meine ich gar nicht. Die eigentliche Frage lautet: In welchem Programmteil des gesamten Visual Studio/Framework-Komplex ist die Designer-Funktionalität eigentlich untergebracht? Die Antwort mag Sie überraschen, denn: Es ist nicht so, wie man vielleicht vermuten würde, dass Visual Studio komplett selbst für die Bearbeitung der Steuerelemente im Designer zuständig ist. Visual Studio fungiert im Grunde nur als Ereignisweiterreicher an die betroffene Komponente. Ein Steuerelement besteht nämlich grundsätzlich aus zwei Teilen: aus dem Steuerelement, das die eigentliche Funktionalität zur Verfügung stellt, die Sie zur Laufzeit Ihres Programms benötigen und aus einer weiteren Komponente, dem Designer. Ja genau. Jede Komponente verfügt streng genommen über ihren eigenen Designer. Aber natürlich haben nicht hunderte von Entwicklern hunderte verschiedene Designer programmiert. Genauso, wie Control alle Grundfunktionalitäten für ein neues, eigentliches Steuerelement zur Verfügung stellt, das diese lediglich weiter ausbaut, gibt es auch eine weitere Komponente namens ControlDesigner, die alle Design-Grundfunktionalitäten beinhaltet. Visual Studio spielt dabei nur die Rolle eines Gastgebers und stellt das Umfeld, damit ein solcher Designer Platz zum Austoben hat. Im Fachterminus lautet das: »Visual Studio ist der *Designer Host*«. Wird ein neues Steuerelement geschaffen, und es bringt keinen eigenen Designer (komplett neu oder auf ControlDesigner basierend spielt dabei keine Rolle), verwendet Visual Studio eben einfach ControlDesigner selbst für dieses Steuerelement – und dessen Basisfunktionen erlauben zumindest das Verschieben, Kopieren, »Einschnappen« oder andere Funktionen, die Sie beim Aufbau eines Formulars benötigen. Für die unter Ihnen, die sich ein wenig mit den .NET-Assemblies auskennen und die es interessiert: Die Designer aller Steuerelemente von .NET befinden sich in der Assembly *System.Design.dll*.

## Angleichen von Größe und Position von Steuerelementen

Nun befinden sich alle drei Steuerelemente wild platziert im Formular – Ziel ist es aber, dass sie alle drei nicht nur wie an einer Schnur ausgerichtet, bündig untereinander stehen, sondern dass sie auch alle drei die gleiche Größe haben. Es gäbe die Möglichkeit, diese Änderungen manuell vorzunehmen, indem Sie die Steuerelemente solange »zurechtzuppeln« und verschieben, bis sie passen. Dank der Ausrichtungslinienfunktionalität ist das kein großer, aber immerhin ein Akt.

Es geht auch einfacher. Sie können mehrere Steuerelemente markieren und anschließend Operationen die Größe und Position betreffend durchführen, bei denen alle Steuerelemente sich an dem als erstes markierten Steuerelement orientieren. Wenn Sie die Steuerelemente markiert haben (wie gesagt: das zuerst markierte ist immer das Referenzstueuerelement), bietet Ihnen Visual Studio bestimm-

te Funktionen an, die Sie im Abschnitt »Funktionen zum Layouten von Steuerelementen im Designer« ab Seite 68 beschrieben finden.

### Selektieren mehrerer Steuerelemente und Bestimmen des Referenzsteuerelements

Um mehrere Steuerelemente im Formular gleichzeitig zu selektieren, halten Sie entweder die **Umschalt**-Taste gedrückt und klicken anschließend alle betroffenen Steuerelemente nacheinander an, oder Sie ziehen mit der Maus einen Rahmen um die Steuerelemente, die selektiert werden sollen. Das Referenzsteuerelement für bestimmte Funktionen, die Sie in Tabelle 3.4 ab Seite 68 beschrieben finden, ist das, welches als erstes selektiert wurde, wenn Sie die Steuerelemente mit gedrückter **Umschalt**-Taste selektieren. Möchten Sie das Referenzsteuerelement nach der Selektion der Steuerelemente ändern, klicken Sie es einfach an (dabei halten Sie die **Umschalt**-Taste *nicht* mehr gedrückt). Anders als bei vielen anderen Selektionsverfahren unter Windows, wird dabei die Selektion nicht aufgehoben und das neu angeklickte Element selektiert, sondern Sie ändern wirklich nur das Referenzsteuerelement. Möchten Sie die Selektion komplett aufheben, klicken Sie einfach in einen freien Bereich innerhalb des Formulars oder auf ein zuvor nicht selektiertes Steuerelement.

---

**TIPP:** Das Referenzsteuerelement erkennen Sie übrigens daran, dass es mit weißen Anfasspunkten versehen ist, während alle anderen markierten Steuerelemente über schwarze Anfasspunkte verfügen.

---

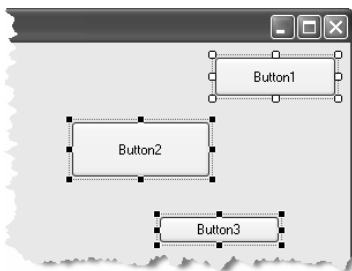
1. Für unser Beispiel selektieren Sie auf diese Weise zunächst alle Schaltflächen im Formular. Wichtig dabei ist, dass die Schaltfläche in der linken, oberen Ecke zum Referenzsteuerelement wird.

---

**HINWEIS:** Für den folgenden Schritt müssen Sie unter Umständen die benötigte Symbolleiste einschalten. Klicken Sie dazu mit der rechten Maustaste auf einen auf einen freien Bereich neben einer schon eingeblendeten Symbolleiste, und wählen Sie aus dem Kontextmenü, das jetzt erscheint, den Eintrag *Layout*.

---

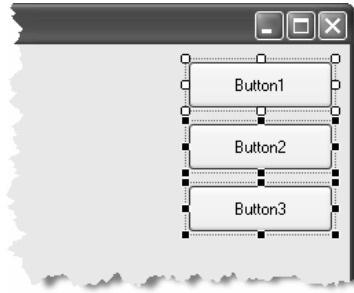
2. Wählen Sie aus der Symbolleiste die Funktion *Größe angleichen* – die Tabelle im Abschnitt »Funktionen zum Layouten von Steuerelementen im Designer« ab Seite 68 hilft Ihnen, das richtige Symbol dafür zu finden.



**Abbildung 3.8:** Hier sehen Sie, dass sich alle Schaltflächen wild platziert im Formular befinden – mit Ausnahme der ersten, die ihre endgültige Position und Größe bereits hat ...

3. Klicken Sie anschließend auf das Symbol *Links ausrichten*, um die Schaltflächen linksbündig untereinander auszurichten. Damit stehen anschließend alle drei Schaltflächen genau ausgerichtet untereinander.
4. Die Abstände zwischen den Schaltflächen passen Sie dann entweder durch Verschieben der Schaltflächen mit der Maus per Drag&Drop und der Unterstützung durch die Ausrichtungslinien an, oder Sie verwenden die Funktion *Vertikalen Abstand angleichen*, und so oft die Funktionen

Vertikalen Abstand vergrößern bzw. Vertikalen Abstand verkleinern, bis Sie mit dem Ergebnis zufrieden sind.

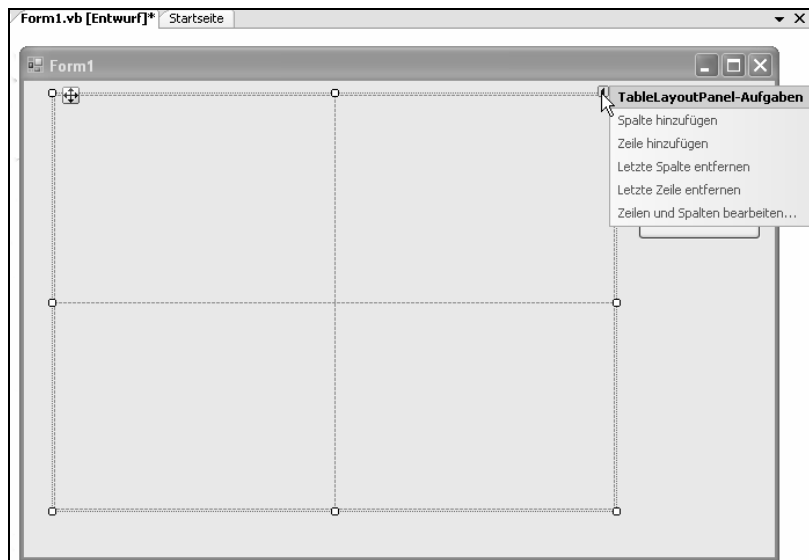


**Abbildung 3.9:** ... und hier nach Anwenden der Funktionen *Größe angleichen*, *Links ausrichten*, *Vertikalen Abstand angleichen* sowie *Vertikalen Abstand verkleinern*

Im günstigsten Fall mussten Sie die Schaltflächen nach dem Aufziehen im Formular nicht einmal mehr »anpacken«. Innerhalb von Sekunden konnten Sie sie mithilfe der Funktionen der Layout-Symbolleiste so anpassen, dass sie mit sauberem Layout im Formular standen.

## Häufige Arbeiten an Steuerelementen mit Smarttags erledigen

Es gibt komplexe Steuerelemente, wie beispielsweise das `TableLayoutPanel` (das wir für den nächsten Schritt unseres Beispiels benötigen), das für das bequeme Einstellen bestimmter Verhaltensweisen so genannte Smarttags anbietet. Smarttags, wie in *Abbildung 3.10* zu sehen, führen zu einer Liste mit Kontextaufgaben, die an das jeweilige Steuerelement gekoppelt ist, und das Ihnen eine Art Abkürzung zu den Eigenschaften bietet und eine gleichzeitig komfortablere Einstellung der Eigenschaften desselben ermöglicht.



**Abbildung 3.10:** Ein Mausklick auf den Smarttag eines Steuerelements öffnet das Aufgabenfenster, das Funktionen zum Erledigen der häufigsten Aufgaben anbietet

1. Für unser Beispiel wählen Sie aus der Toolbox per Mausklick das `TableLayoutPanel` aus. Ziehen Sie dieses Steuerelement im Formular in etwa in der Größe auf, dass es Abbildung 3.10 entspricht.
2. Sofort öffnet sich die hinter dem Smarttag stehende Liste mit Kontextaufgaben. Sie können diese Kontextaufgabenliste jederzeit mit einem Mausklick auf den Smarttag auf- und zuklappen.

---

**HINWEIS:** Nicht jedes Steuerelement verfügt über einen Smarttag mit einer dahinter stehenden Liste mit Kontextaufgaben, aber wenn ein Steuerelement darüber verfügt, dann steht Ihnen diese Liste jederzeit zur Verfügung.

---

## Dynamische Anordnung von Steuerelementen zur Laufzeit

Denken Sie mal einige Jahre zurück und dabei an den Aufwand, den es noch in Visual Basic 6.0 machte, eine Benutzeroberfläche wie den Windows-Explorer zur implementieren. Sie mussten alleine mehrere Mannstunden dafür opfern, den Code dafür zu entwickeln, die Steuerelemente dynamisch neu anzuordnen, falls der Anwender des Programms zur Laufzeit das Fenster vergrößerte oder verkleinerte.

Seit Visual Basic .NET ist das sehr viel einfacher geworden. Geschickt umgesetzt müssen Sie nicht eine einzige Zeile Code schreiben, um Steuerelemente in Abhängigkeit einer Formulargrößenänderung neu zu positionieren und auszurichten. Und mit Visual Basic 2005 ist dies noch ungleich bequemer geworden, denn es stellt neue Container-Steuerelemente zur Verfügung, die ausschließlich diesem Zweck dienen. Folgende Features stehen Ihnen in Visual Basic 2005 zur Verfügung, mit denen Sie diese Problematik – wie gesagt ohne Programmierung – in den Griff bekommen:

- **Anchor-Eigenschaft:** Erlaubt das Verankern jeder Seite eines Steuerelements auf dem Formular. Vergrößern oder verkleinern Sie das Formular, dann »wandern« die Seiten des Steuerelements im gleichen Verhältnis hinter den Seiten des Formulars her, an denen es verankert ist.
- **Dock-Eigenschaft:** Erlaubt das Andocken eines Steuerelements an einen Formularrand oder an ein bereits gedocktes Steuerelement. Dabei wird dafür gesorgt, dass das Steuerelement an den gedockten Seiten immer zum Formularrand oder zum ersten schon gedockten Steuerelement aufschließt.
- **SplitContainer-Steuerelement:** Stellt einen Doppelcontainer für weitere Steuerelemente zur Verfügung. Dessen Besonderheit ist, dass sich die Größe eines Containerteils auf Kosten des anderen Containerteils vergrößern lässt. Denken Sie als Beispiel an den Windows-Explorer. In der Mitte können Sie einen vertikalen Trennbalken mit der Maus packen und nach links oder rechts verschieben – dementsprechend vergrößert bzw. verkleinert sich die linke `TreeView` mit den Verzeichnissen und anderen Hauptelementen (wie Systemsteuerung, Mobile Geräte, etc.) während sich die Elementeliste im rechten Containerteil, die durch eine `ListView` realisiert wird, verkleinert bzw. vergrößert.
- **TableLayoutPanel-Steuerelement:** Stellt einen Container für weitere Steuerelemente zur Verfügung. Die Besonderheit ist, dass dieser Container mehrere Steuerelemente in einer Tabelle anordnet, deren Zeilen und Spalten sich in einstellbaren Verhältnissen vergrößern, wenn das `TableLayoutPanel`-Steuerelement selbst sich ebenfalls vergrößert oder verkleinert. Damit wird es möglich, dass nicht nur bestimmte Steuerelemente sich vergrößern, wenn sich das Formular vergrößert, sondern dass Steuerelemente in bestimmten Verhältnissen anwachsen oder sich verkleinern, im Falle einer Vergrößerung oder Verkleinerung des Formulars.

- **FlowLayoutPanel-Steuer-element:** Ist am besten mit einem Texteditor mit aktiviertem Wortumbruch zu erklären. Wenn ein Wort nicht mehr in eine Zeile passt, wird es automatisch in die nächste Zeile umbrochen. Beim FlowLayoutPanel ist dies genauso, jedoch nicht mit Worten, sondern mit Steuer-elementen. Sie ordnen Steuer-elemente nicht an festen Positionen, sondern nebeneinander bzw. untereinander an. Wenn sich das Formular vergrößert oder verkleinert, versucht das FlowLayoutPanel möglichst viele Steuer-elemente beispielsweise in eine Zeile zu bekommen. Passen sie nicht mehr in eine Zeile, da es zu »eng« wird im Formular, springen sie wie beim Wortumbruch des Editors automatisch in die nächste Zeile.

Die folgenden Abschnitte demonstrieren die wichtigsten dieser Features an unserem Fallbeispiel.

### Verankern von Steuer-elementen mit der Anchor-Eigenschaft

Die Anchor-Eigenschaft, die jedem Steuer-element anheim ist, löst das wichtigste, da häufigste, aller Probleme auf schnelle und elegante Weise. Sie dient dazu, die Seiten von Steuer-elementen mit den Seiten eines Formulars oder des sie beinhaltenden Container-Steuer-elementes zu verankern. Jede verankerte Seite des Steuer-elementes behält den Abstand zum Rand seines Containers, wenn dieser sich vergrößert oder verkleinert, was zur Folge hat, dass das Steuer-element (wenn es an zwei gegenüberliegenden Seiten verankert ist) sich entsprechend vergrößert oder seine Position verändert (wenn es an nur jeweils einer Seite verankert ist).

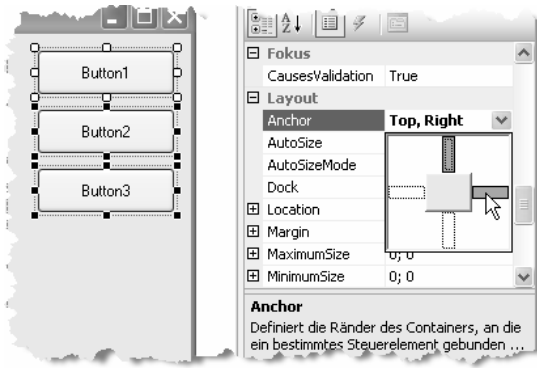
Für unser Beispiel werden wir im Folgenden die drei Schaltflächen so neu verankern, dass sie hinter dem rechten Formularrand mitwandern, wenn der Anwender das Formular in X-Richtung vergrößert oder verkleinert. Das TableLayoutPanel, das später die Eingabefelder, Beschriftungen und das Bild enthält, wird an allen vier Seiten verankert, damit es in allen Richtungen dynamisch mit dem Formular mitwächst und seinerseits wieder dafür sorgt, dass sich später alle in ihm enthaltenen Steuer-elemente proportional anpassen.

1. Markieren Sie alle drei Schaltflächen.
2. Suchen Sie im Eigenschaftenfenster nach der Anchor-Eigenschaft. Klappen Sie die Aufklappliste auf, und klicken Sie mit der Maus auf die kleinen grauen Zwischenräume zwischen den weißen Flächen, um die Verankerungspositionen festzulegen. Orientieren Sie sich dabei am besten an Abbildung 3.11.

---

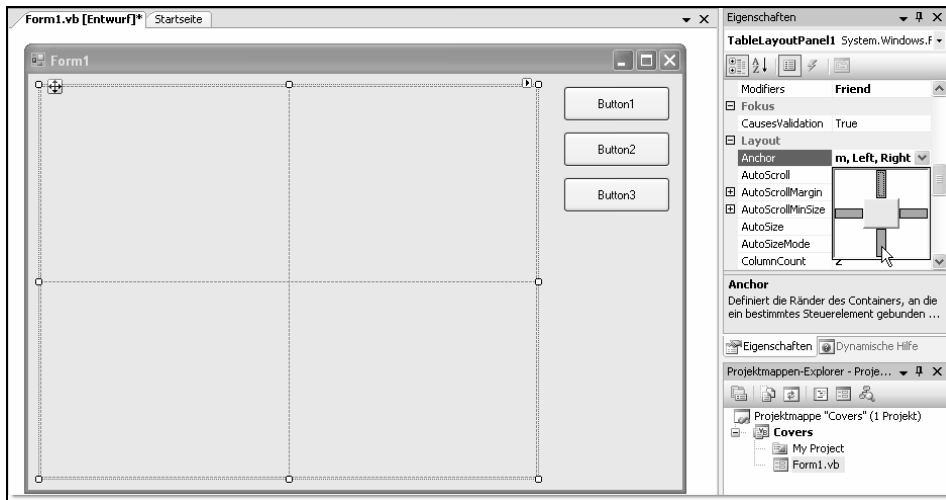
**TIPP:** Sie müssen wie hier im Beispiel nicht Eigenschaften verschiedener Steuer-elemente, die Sie mit gleichen Werten belegen möchten, nacheinander setzen. Selektieren Sie stattdessen die Steuer-elemente, für die der gleiche Wert einer Eigenschaft gelten soll, und weisen Sie mit dem Eigenschaftenfenster diese Eigenschaft allen selektierten Steuer-elementen »in einem Rutsch« zu. Visual Studio findet bei mehreren selektierten Steuer-elementen übrigens automatisch den größten gemeinsamen Nenner in Sachen vorhandene Eigenschaften, zeigt also bei mehreren selektierten Steuer-elementen nur die Eigenschaften im Eigenschaftenfenster an, über die alle selektierten Steuer-elemente verfügen.

---



**Abbildung 3.11:** Setzen Sie die *Anchor*-Eigenschaft von ursprünglich *Top, Left* auf *Top, Right* bewirkt das, dass die Steuerelemente immer den gleichen Abstand zum oberen und rechten Formularrand behalten und beim Vergrößern des Formulars nach rechts quasi hinter dem Rand herlaufen

3. Vergrößern Sie als nächstes das `TableLayoutPanel` so, dass es mit den drei betroffenen Seiten möglichst nah an den Formularrändern liegt.
4. Setzen Sie die *Anchor*-Eigenschaft des `TableLayoutPanel` so, dass es an alle vier Seiten gebunden ist.



**Abbildung 3.12:** Setzen Sie die *Anchor*-Eigenschaft eines Steuerelements für alle vier Seiten, vergrößert sich das Steuerelement in alle Richtungen proportional mit dem Formular

Nachdem Sie alle Einstellungen vorgenommen haben, können Sie schon jetzt im Entwurfsmodus das Formular vergrößern und verkleinern, und Sie werden feststellen, dass sich bereits jetzt alle Steuerelemente an die jeweils neuen Formularausmaße anpassen.

### Proportionales Anpassen von Steuerelementen an Formular-Größenänderungen mit dem `TableLayoutPanel`

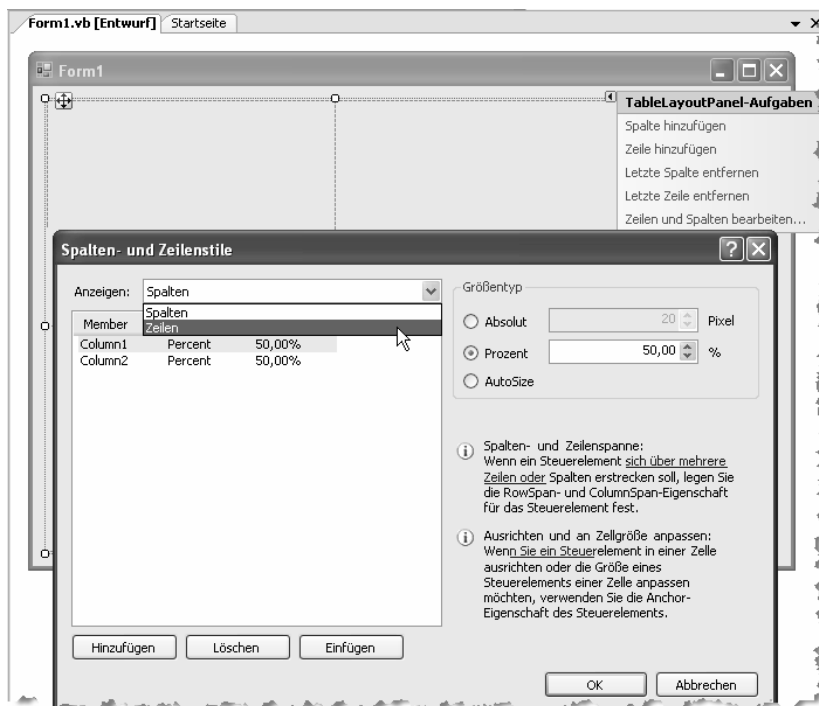
Wenn Sie das `TableLayoutPanel` in Abbildung 3.12 betrachten, sehen Sie, dass es aus insgesamt vier Zellen besteht. Diese Zellen dienen, wie beispielsweise das `Panel`, als Container für jeweils ein weiteres Steuerelement. Das Besondere: Wenn sich das `TableLayoutPanel` an sich vergrößert bzw. verkleinert,

ner, vergrößern bzw. verkleinern sich die in ihm enthaltenen Zellen im Verhältnis. Bei geschickter Anordnung von Steuerelementen in den Zellen reichen Sie die verhältnismäßige Vergrößerung oder Verkleinerung an diese weiter. Ihre Formulare wachsen damit dynamisch, wenn mehr Platz auf dem Bildschirm zur Verfügung steht, und nutzen diesen damit wesentlich besser aus.

## Einstellen der vorhandenen Spalten und Zeilen des TableLayoutPanel

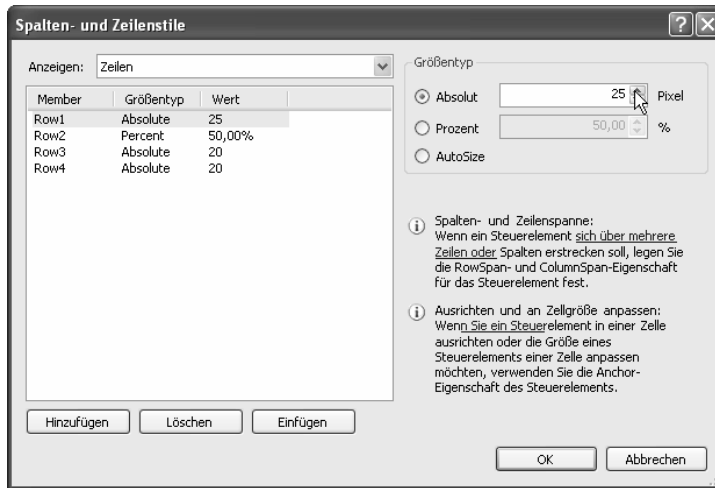
Standardmäßig, also wenn Sie ein TableLayoutPanel das erste Mal im Formular aufziehen, besteht es aus vier Zellen: nämlich aus zwei Spalten und zwei Zeilen. Um die Anzahl an Spalten oder Zeilen zu erhöhen, verwenden Sie entweder – wie bei allen Steuerelementen – das Eigenschaftfenster, oder Sie verwenden die Kontextaufgabenliste, die Sie über den Smarttag des Steuerelements erreichen.

1. Für unser Beispiel klicken Sie auf den Smarttag des sich bereits im Formular befindlichen TableLayoutPanel und klicken anschließend auf *Zeilen und Spalten bearbeiten*. Orientieren Sie sich dafür an Abbildung 3.13.
2. Wählen Sie aus der Aufklappliste *Anzeigen* das Element *Zeilen*.



**Abbildung 3.13:** Wählen Sie aus der Kontextaufgabenliste des *TableLayoutPanel*, die Sie durch Klick auf sein Smarttag öffnen, *Zeilen und Spalten bearbeiten* und unter *Anzeigen* das Element *Zeilen*, um die Anzahl und Eigenschaften der Zeilen einzustellen

3. Klicken Sie zweimal auf *Hinzufügen*, um dem TableLayoutPanel zwei zusätzliche Zeilen hinzuzufügen.



**Abbildung 3.14:** In diesem Dialog konfigurieren Sie die Größenverhältnisse der einzelnen Zellen

4. Klicken Sie auf die erste Zeile der Zeilenliste, und stellen Sie den Größentyp auf *Absolut*. Geben Sie 25 Pixel als Höhe der ersten Zeile ein. Orientieren Sie sich an Abbildung 3.14. Sie bestimmen damit, dass sich diese Zeile nicht dynamisch vergrößert, sondern stets eine Höhe von 25 Pixeln enthält. Da sich in dieser Zeile später das Eingabefeld für den Filmtitel befindet, der nur einzeilig eingegeben werden soll, braucht sich das Eingabefeld nicht in Y-Richtung zu vergrößern, egal, wie groß der Abstand nach unten auch wird. Vielmehr macht es durch diese Einstellung sogar Sinn, den Abstand klein zu halten, damit anderen Zeilen, die Steuerelemente enthalten werden, und die sich vergrößern sollen, mehr Platz für ihre Inhalte zur Verfügung steht.

---

**WICHTIG:** Wenn Sie eine Wertänderung vorgenommen haben, drücken Sie NICHT **Eingabe**, sondern klicken Sie, falls Sie weitere Größentypen oder Werte ändern möchten, einfach auf die nächste Zeile in der Liste, deren Einstellungen Sie ändern möchten. **Eingabe** bewirkt das Auslösen der OK-Schaltfläche, und damit verschwindet der Dialog vom Bildschirm; Sie müssen den Dialog dann erneut aufrufen.

---

**HINWEIS:** Wenn Sie einen Mix aus absoluten und prozentualen Größentypen verwenden, dann werden vom maximal zur Verfügung stehenden Platz (egal ob für Zeilen in der Höhe oder Spalten in der Breite) die absoluten Pixelangaben abgezogen. Der Rest des zur Verfügung stehenden Platzes wird zwischen den anderen Zellen so aufgeteilt, wie es den prozentualen Werteangaben entspricht.

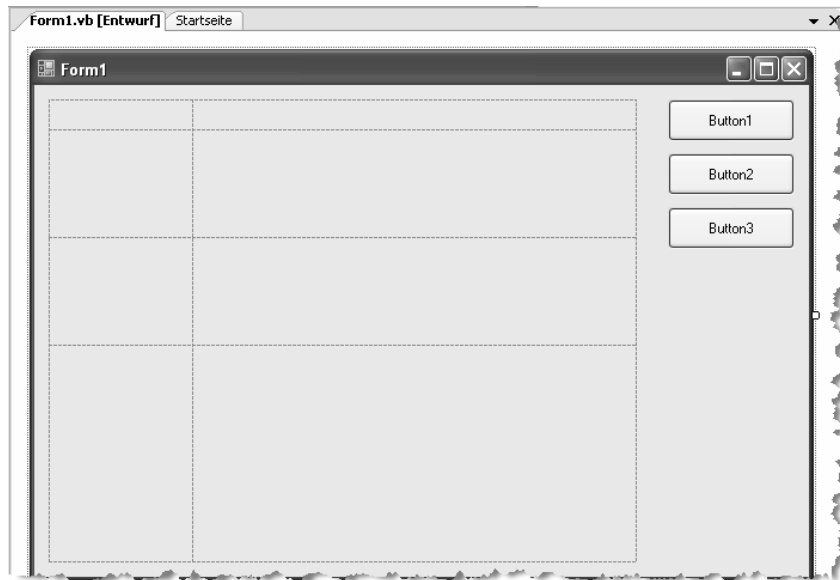
---

5. Für *Row2* (Zeile 2) belassen Sie den Größentyp auf *Prozent*, geben als Wert allerdings 25 % ein.
6. Für *Row3* und *Row4* ändern Sie den Größentyp auf *Prozent* und geben als Wert 25 % und 50 % ein.
7. Wählen Sie anschließend aus der Aufklappliste *Anzeigen* das Element *Spalten*. Klicken Sie die Zeile *Column1* (Spalte 1) an, setzen Sie den Größentyp auf *Absolut* und geben Sie 120 als feste Pixelbreite ein.
8. An *Column1* (Spalte 2) nehmen Sie ebenfalls eine kleine Änderung vor, indem Sie die Prozentangabe für diese Spalte auf 100 % einstellen.

Mit diesen Einstellungen haben Sie nun erreicht, dass die linke Spalte, die die immer gleichlautenden Beschriftungen enthalten wird, stets eine Breite von 120 Pixel behält, während sich die rechte Spalte dynamisch vergrößern oder verkleinern kann.

9. Sie können den Dialog nun mit **Eingabe** bestätigen und damit schließen, da Sie ihn jetzt nicht mehr benötigen.
10. Klappen Sie die noch geöffnete Kontextaufgabenliste durch Klick auf den Smarttag des `TableLayoutPanel` auch wieder zu.

Das Ergebnis sollte anschließend in etwa wie in Abbildung 3.15 ausschauen.

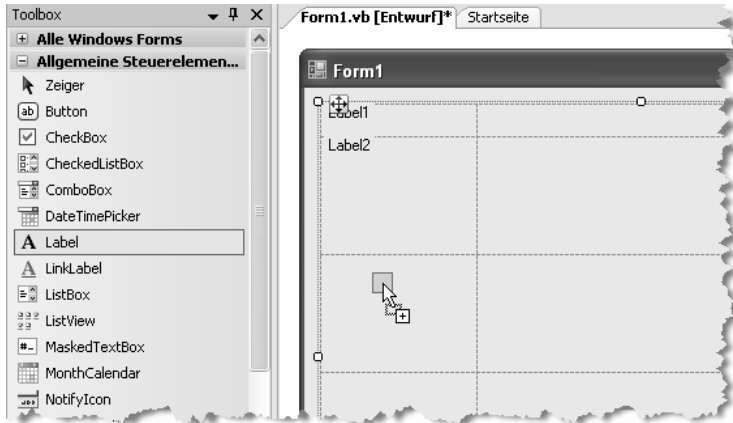


**Abbildung 3.15:** So sollte das vorläufige Ergebnis ausschauen, nachdem Sie die Zeilen- und Spaltenparameter der Zellen bestimmt haben

### **Anordnen von Steuerelementen in den Zellen eines `TableLayoutPanel`**

Ein Blick auf das Ergebnis in Abbildung 3.2 offenbart, was als nächstes auf uns zukommt: die Anordnung der `TextBox`- und `Label`-Steuerelemente für die Eingabe der Daten sowie deren Beschriftung.

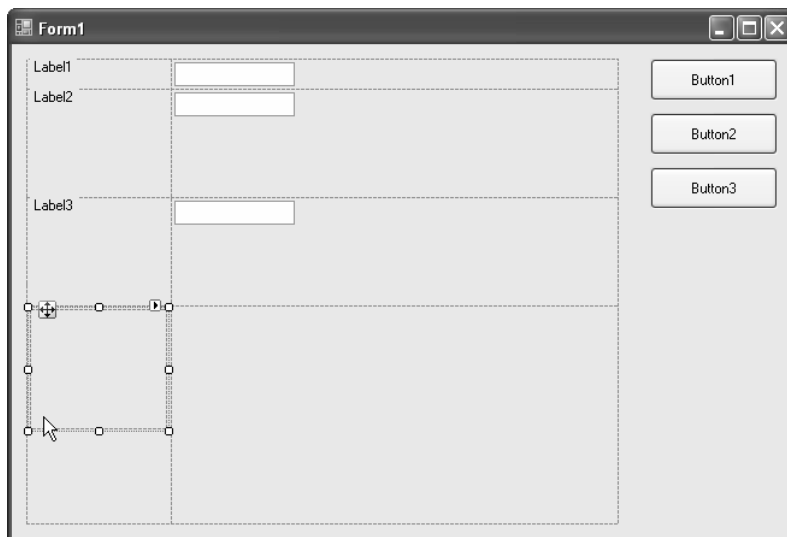
1. Zum Platzieren der Steuerelemente fügen Sie sie per `Drag&Drop` aus der `Toolbox` in die Zellen des `TableLayoutPanel` ein. Abbildung 3.16 hilft Ihnen bei diesem Vorgang.



**Abbildung 3.16:** Ziehen Sie zum Platzieren der Steuerelemente diese einfach per Drag&Drop in die jeweiligen Zellen des TableLayoutPanel

2. Nach diesem Verfahren ziehen Sie drei Label-Elemente in die oberen drei linken Zellen und drei TextBox-Elemente in die oberen drei rechten Zellen.
3. In die untere linke Zelle fügen Sie ein Panel-Steuerelement ein, das später als Träger für die Auslassungsschaltfläche (...) zur Wahl der Grafikdatei sowie für ein weiteres Panel und ein darin geschachteltes PictureBox-Steuerelement dient.

Anschließend sollte sich das Ergebnis in etwa wie in Abbildung 3.17 gestalten.



**Abbildung 3.17:** Nach dem Einfügen aller Steuerelemente sollten Sie in etwa dieses Ergebnis vorfinden

---

**HINWEIS:** Wozu dient das `Panel` in der linken, unteren Ecke (wir benötigen dort schließlich ein Bild und eine Auslassungsschaltfläche für die Auswahl des Bildes)? Das verhält sich folgendermaßen: In jeder Zelle eines `TableLayoutPanel`-`Panel` darf sich nur ein Steuerelement befinden. Das bedeutet jedoch nicht, dass dieses ein Steuerelement nicht als Container für andere Steuerelemente fungieren kann. Auf diese Weise stehen Ihnen Tür und Tor offen für komplexe Gebilde, die jedoch, je komplexer und verschachtelter sie werden, einen entscheidenden Nachteil haben: Das Neuordnen solcher Gebilde zur Laufzeit kostet Zeit – gerade bei langsamen Grafikkarten, weil natürlich die Inhalte aller Steuerelemente bei der kleinsten Größenänderung aktualisiert werden müssen. Sie sollten sich eigentlich mit maximal einem Container pro Zelle begnügen und die Anzahl der Steuerelemente in Containern pro Zelle wirklich so eng wie möglich begrenzen. Nur in Sonderfällen – die folgenden Abschnitte demonstrieren einen solchen – sollten Sie eine weitere Verschachtelungstiefe (im Sinne von »Container enthält Container enthält die eigentlichen Steuerelemente«) in Erwägung ziehen.

---

### Verankern von Steuerelementen im `TableLayoutPanel`

Nun ist die Anordnung, wie Sie sie in Abbildung 3.17 sehen können, noch nicht besonders elegant. Die Beschriftungen »hängen« in der linken, oberen Ecke und die Textbox-Elemente erstrecken sich nicht über die volle Breite des Formulars. Das zu ändern hängt wieder buchstäblich an der `Anchor`-Eigenschaft der einzelnen Steuerelemente.

1. Setzen Sie die `Anchor`-Eigenschaft des oberen `Label`-Steuerelements auf *Left, Right*.
2. Setzen Sie die `Anchor`-Eigenschaften der verbleibenden `Label`-Steuerelemente auf *Left, Top, Right*.
3. Damit die `Label`-Elemente einen schöneren Eindruck machen, selektieren Sie alle, und...
4. ...setzen Sie alle `BorderStyle`-Eigenschaften der `Label` auf `Fixed3D`.
5. Setzen Sie alle `TextAlign`-Eigenschaften der `Label` auf *MiddleRight*.
6. Setzen Sie die `Anchor`-Eigenschaft des oberen `TextBox`-Steuerelements auf *Left, Right*.
7. Bevor wir die `Anchor`-Eigenschaften der zwei unteren `TextBox`-Steuerelemente ändern können, müssen wir diese für den mehrzeiligen Betrieb einstellen. Nur dann kann eine `TextBox` den gesamten zur Verfügung stehenden Bereich einer Zelle ausnutzen und an allen vier Seiten ohne Abstand verankert werden. Selektieren Sie dazu beide unteren `TextBox`-Steuerelemente.
8. Stellen Sie die `Multiline`-Eigenschaft der `TextBox` auf `True`. Damit diese `Textboxes` automatisch Rollbalken erhalten, falls ein Anwender mehr Zeilen eingibt als sichtbarer Platz zur Verfügung steht, setzen Sie die `Scrollbars`-Eigenschaft auf `Vertical`.
9. Selektieren Sie nun zusätzlich die `PictureBox`, indem Sie die Taste **Strg** festhalten und auf das Steuerelement klicken.
10. Stellen Sie jetzt die `Anchor`-Eigenschaft aller selektierten Steuerelemente auf *Top, Bottom, Left, Right*. Das Ergebnis sollte nun ausschauen wie in Abbildung 3.18.

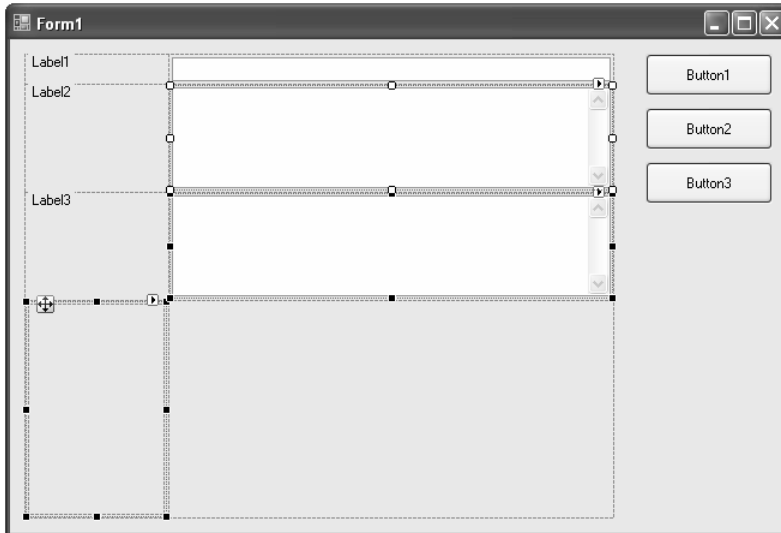


Abbildung 3.18: Nach dem Verankern aller Steuerelemente sollte das Formular wie hier ausschauen

### Verbinden von Zeilen oder Spalten des TableLayoutPanel

Das TableLayoutPanel fungiert neben seiner schon bekannten Funktionalität auch als so genannter *Property Extender*.

#### Was ist ein Property Extender?

Ein Steuerelement oder eine Komponente, die als Property Extender (etwa *Eigenschaftenerweiterer*) ausgelegt ist, ergänzt Steuerelemente, die im gleichen Container wie sie selbst liegen, oder Steuerelemente, die sie beinhaltet (wenn es sich bei dem Property Extender selbst um ein Container-Steuerelement handelt), um weitere Eigenschaften.

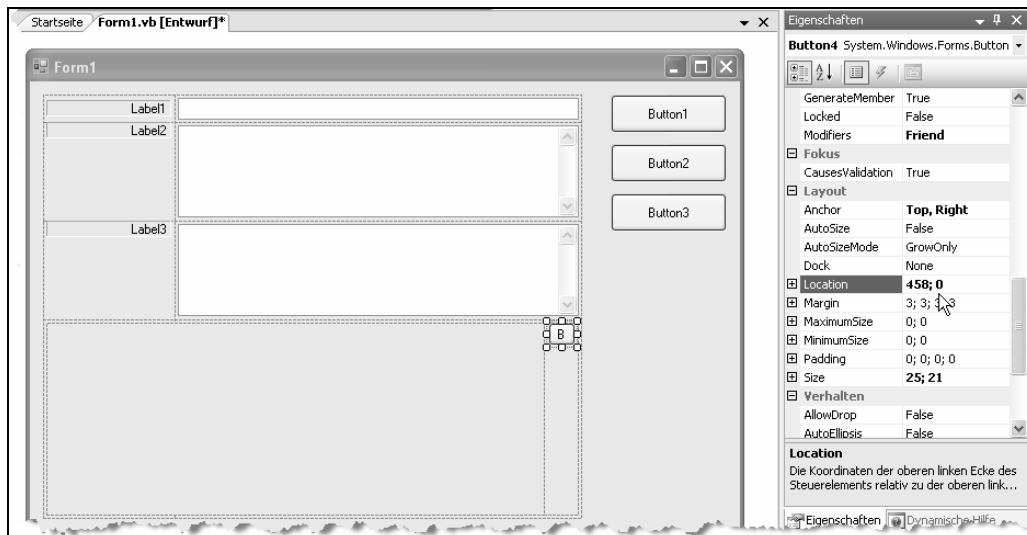
Eine solche Komponente schaut sich dabei an, welche Steuerelemente, die in ihrer Reichweite liegen, für die Erweiterung durch neue Eigenschaften, die im Kontext Sinn machen, in Frage kommen und stattet diese Steuerelemente mit neuen Eigenschaften aus. Diese Eigenschaften existieren aber dann nur für diese erreichbaren Steuerelemente, da sich diese neuen Eigenschaften grundsätzlich auf irgendeine Interaktion mit dem Property-Extender-Steuerelement beziehen sollten.

Steuerelemente außerhalb der Reichweite des Properties Extender oder Steuerelemente in anderen Formularen sind davon nicht betroffen.

Aus diesem Grund haben alle Steuerelemente, die sich innerhalb einer Zelle eines TableLayoutPanel befinden, vier weitere Eigenschaften: Column, ColumnSpan, Row und RowSpan. Mit Column und Row wird festgelegt, in welcher Zeile und Spalte sich das Steuerelement befindet. Interessanter sind ColumnSpan und RowSpan: Diese bestimmen, über wie viele Zeilen und Spalten sich das Steuerelement erstrecken soll.

Das Panel, das später PictureBox und Auslassungsschaltfläche beinhalten soll, erstreckt sich bei genauerer Betrachtung (siehe Abbildung 3.2) über die komplette Spaltenbreite. Aus diesem Grund sollte seine ColumnSpan-Eigenschaft auf 2 (die Anzahl an Spalten im TableLayoutPanel) gesetzt werden.

1. Löschen Sie dazu als erstes die aktuelle Selektion der anderen Steuerelemente, damit Sie nicht versehentlich die ColumnSpan-Eigenschaften der zuletzt selektierten Steuerelemente mit neu setzen (ist mir gerade passiert). Klicken Sie dazu auf einen freien Bereich im Formular.
2. Selektieren Sie das Panel und setzen Sie die ColumnSpan-Eigenschaft auf 2.
3. Für weitere Aufgaben, die in den folgenden Abschnitten besprochen werden, fügen Sie innerhalb des Panel eine Schaltfläche mit vergleichsweise kleinen Ausmaßen sowie ein weiteres Panel ein.
4. Platzieren Sie die Schaltfläche in der rechten oberen Ecke des sie umschließenden Panel. Kontrollieren Sie die Location-Eigenschaft, die die linke obere Ecke des Steuerelements widerspiegelt, dass der Y-Wert nicht negativ sondern am besten 0 ist. Damit liegt die Schaltfläche ganz oben an.



**Abbildung 3.19:** Die untere Zeile des TableLayoutPanel enthält nun ein Panel, das sich über zwei Spalten erstreckt. Es beinhaltet ein weiteres Panel sowie eine Schaltfläche zur späteren Bilddateiauswahl.

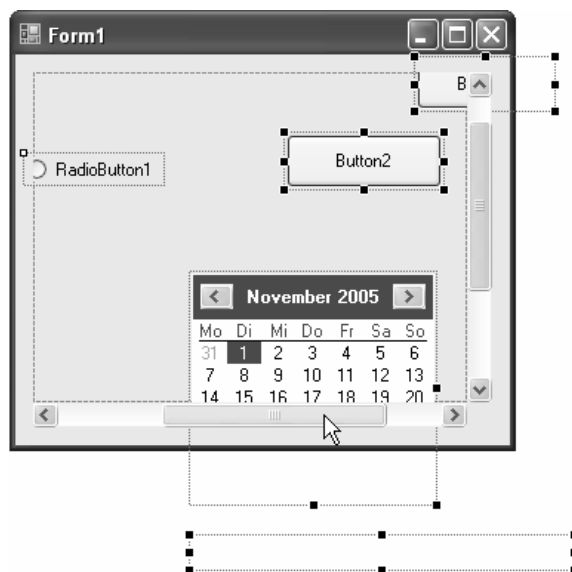
5. Vergrößern Sie das zweite Panel, das Sie gerade eingefügt haben, so, dass es möglichst ohne Abstand an den äußeren Rändern des umgebenden Panel anliegt. Die Eigenschaften Location und Size können Ihnen beim Feintuning helfen.
6. Setzen Sie die Anchor-Eigenschaft der Schaltfläche auf *Top, Right*.
7. Setzen Sie die Anchor-Eigenschaft des zweiten Panel auf *Top, Bottom, Left, Right*.

Sie sollten anschließend ein Ergebnis etwa wie das in Abbildung 3.19 sehen.

## Automatisches Scrollen von Steuerelementen in Containern

Vielleicht stellen Sie sich die Frage, wieso wir neben die Schaltfläche ein weiteres Panel und dort nicht direkt die PictureBox platziert haben. Die Antwort zeigt sich wie von selbst, wenn Sie nochmals einen

Blick auf Abbildung 3.2 werfen: Der Bildausschnitt bei Coverbildern, die größenmäßig nicht in die PictureBox passen, soll gescrollt werden können. Leider stellt die PictureBox eine solche Funktionalität nicht zur Verfügung.

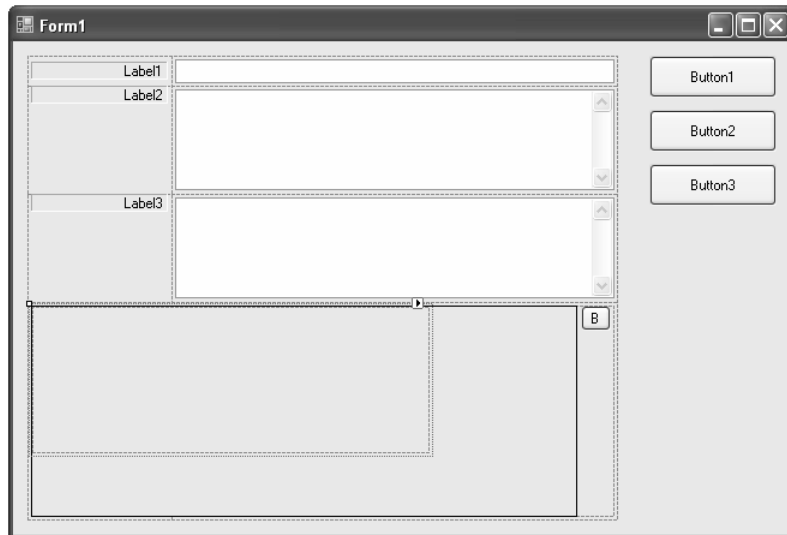


**Abbildung 3.20:** In diesem Beispiel befinden sich alle Steuerelemente in einem Panel, das als Container fungiert. Da nicht alle Steuerelemente in den sichtbaren Ausschnitt passen und seine *AutoScroll*-Eigenschaft gesetzt ist, lässt sich der dargestellte Ausschnitt zur Entwurfs- und Laufzeit mit automatisch zur Verfügung gestellten Rollbalken einstellen.

Allerdings stellen Container-Steuerelemente eine Funktionalität zur Verfügung, mit der wir sehr nah an das rankommen, was wir erreichen wollen. Wenn Sie die *AutoScroll*-Eigenschaft eines Container-Steuerelements auf *True* setzen, dann lassen sich die in ihm enthaltenen, aber durch einen zu kleinen sichtbaren Ausschnitt ausgeblendeten Steuerelemente mit den automatisch erzeugten Rollbalken in das Sichtfenster des Containers »einrollen«. Abbildung 3.20 demonstriert dieses Verhalten recht anschaulich.

Dieses Verhalten nutzen wir nun aus: Die *PictureBox* erlaubt eine Einstellung, die das *PictureBox*-Steuerelement an die Größe des Bildes anpasst – diese Eigenschaft lautet *SizeMode*, und sie muss zu diesem Zweck auf *AutoSize* gestellt werden. Befindet sich die *PictureBox* in einem Panel, dessen *AutoSize*-Eigenschaft gesetzt ist, und wird ein Bild geladen, das die *PictureBox* dazu zwingt sich so weit zu vergrößern, dass sie nicht mehr in das Panel passt, bekommt das Panel ohne weiteres Zutun Rollbalken. Wenn nun nur das Panel über einen Rahmen verfügt und die *PictureBox* keine weiteren sichtbaren Abgrenzungen aufweist, dann schaut es für den Anwender so aus, als ließe sich das Bild in der *PictureBox* scrollen – in Wirklichkeit scrollt aber das Panel die ganze *PictureBox*. Ihr Vorteil: Sie haben nicht nur gerade mehrere Tage Entwicklungsarbeit gespart,<sup>3</sup> Sie brauchten sogar noch *nicht einmal eine einzige* Zeile Code dazu zu schreiben. Gehen wir's also an:

<sup>3</sup> Kein Scherz: Ein Steuerelement mit scrollbarem Inhalt zu entwickeln, ist wirklich keine triviale Sache.



**Abbildung 3.21:** Die geschachtelte Kombination aus Panel, Panel und PictureBox stellt später die Scroll-Funktionalität zur Verfügung – so sollte das vorläufige Ergebnis ausschauen

1. Selektieren Sie das Panel neben der Auslassungsschaltfläche, sofern dieses noch nicht selektiert ist.
2. Setzen Sie dessen BorderStyle-Eigenschaft auf FixedSingle.
3. Setzen Sie dessen AutoScroll-Eigenschaft auf True.
4. Ziehen Sie eine PictureBox im Panel auf.
5. Damit es mit der Maus nicht zu filigran wird: Setzen Sie die Location-Eigenschaft der PictureBox »zu Fuß« auf 0;0.

---

**HINWEIS:** Denken Sie daran, dass sich Koordinaten eines Steuerelements immer auf seinen unmittelbaren Container beziehen.

---

6. Setzen Sie die SizeMode-Eigenschaft der PictureBox auf AutoSize.

Das Ergebnis sollte nun in etwa dem in Abbildung 3.21 entsprechen.

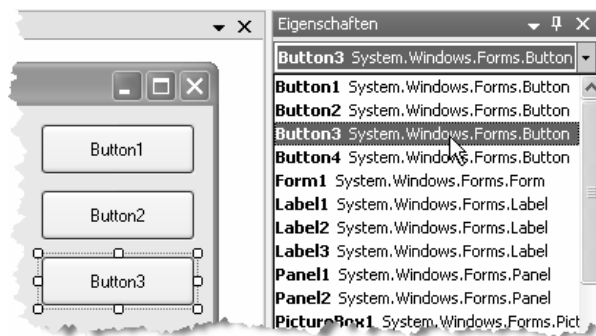
## Selektieren von Steuerelementen, die Sie mit der Maus nicht erreichen

Bei solchen Verschachtelungen, wie Sie sie im vorherigen Abschnitt kennen gelernt haben, wird es ziemlich schwierig, bestimmte Steuerelemente mit der Maus zu selektieren – Sie treffen bisweilen einfach nicht mehr eine Begrenzungslinie des Steuerelements, das Sie eigentlich treffen wollten, und selektieren ein ganz anderes.

In diesem Fall selektieren Sie einfach ein anderes. Drücken Sie dann so oft **Tabulator**, bis Sie das Steuerelement selektiert haben, das Sie auch tatsächlich selektieren wollten. Und falls das auch nicht mehr hilft:

### Selektieren von Steuerelementen mit dem Eigenschaftfenster

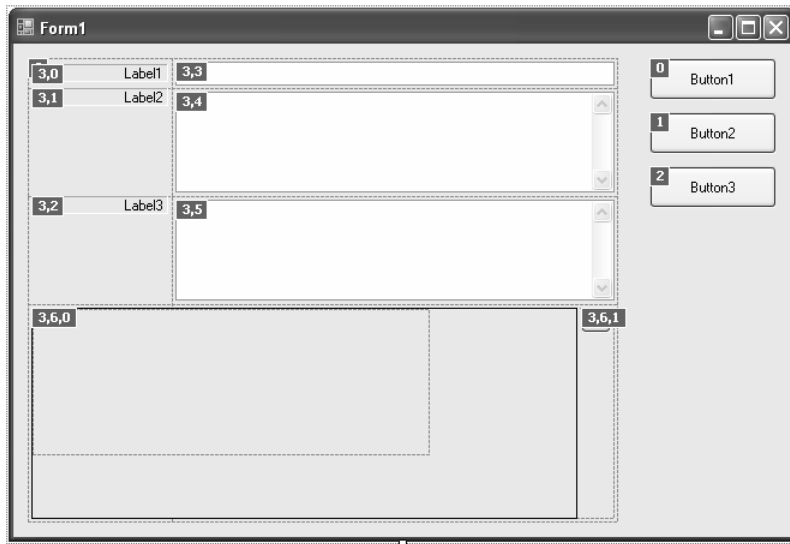
Das Eigenschaftfenster gibt Ihnen in der oberen Zeile Information über das derzeit selektierte Steuerelement. Bei dieser Infozeile handelt es sich allerdings um eine Aufklappliste. Wählen Sie aus dieser Liste ein Steuerelement oder eine Komponente aus, wird diese im Formular-Designer selektiert.



**Abbildung 3.22:** Mithilfe des Eigenschaftfensters können Sie jedes Steuerelement im Designer selektieren

## Festlegen der Tabulatorreihenfolge (Aktivierreihenfolge) von Steuerelementen

Die Tabulatorreihenfolge – die Microsoftterminologie lautet »Aktivierreihenfolge« – bestimmt, welches Steuerelement jeweils als nächstes aktiviert wird, wenn der Anwender es vorzieht, mit der Tastatur zu arbeiten, und mit Tabulator zum jeweils nächsten Element springen will.



**Abbildung 3.23:** Nach dem Aufrufen von *Ansicht/Aktivierreihenfolge* können Sie die Reihenfolge festlegen, mit der Steuerelemente zur Laufzeit per Tabulator zu erreichen sind. Das Formular muss dazu selektiert sein.

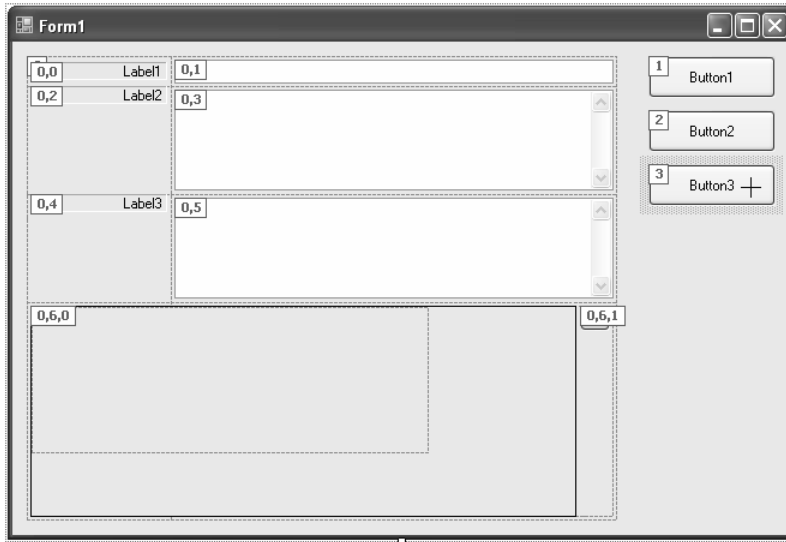
Mit Visual Studio 2005 ist das ruckzuck erledigt. Um beim Beispiel zu bleiben:

1. Klicken Sie auf die Titelzeile des Formulars, um es zu selektieren.
2. Wählen Sie aus dem Menü *Ansicht* den Befehl *Aktivierreihenfolge*. Das Formular ändert seine Darstellung, etwa wie in Abbildung 3.23 zu sehen.
3. Klicken Sie nacheinander die Steuerelemente in der Reihenfolge an, wie Sie durch Tabulator zur Laufzeit des Programms aktiviert werden soll. Beziehen Sie dabei auch die Container mit ein; etwas knifflig wird es beim Bestimmen des ersten Steuerelements – dem `TableLayoutPanel` – dessen Aktivierreihenfolgenordinalnummer wie in Abbildung 3.23 zu sehen, etwas in den Hintergrund gerät (sie liegt hinter der Beschriftung 3,0). Klicken Sie am besten auf den äußersten Rand der linken oberen Ecke, um es »zu erwischen«. Orientieren Sie sich dabei am Ergebnis, das Sie in Abbildung 3.24 sehen können.

---

**TIPP:** Wenn Sie sich im Aktivierreihenfolgenmodus befinden und mit dem Mauszeiger über ein Steuerelement fahren, wird es zur besseren Orientierung mit einem dicken, gerasterten Rahmen gekennzeichnet. In Abbildung 3.24 sehen Sie das bei *Button3*.

---



**Abbildung 3.24:** Die Ordinalnummern aller Steuerelemente, die Sie bereits bestimmt haben, werden mit einem hellen Hintergrund eingefärbt. Drücken Sie **Esc**, wenn Sie mit der Zuweisung fertig sind.

4. Nachdem Sie die letzte Schaltfläche angeklickt haben, drücken Sie **Esc**, um den Aktivierreihenfolgenmodus zu verlassen.

---

**TIPP:** Die Aktivierreihenfolge wird maßgeblich durch die `TabIndex`-Eigenschaft eines Steuerelements festgelegt. Sie können die Aktivierreihenfolge deswegen auch ändern, indem Sie die `TabIndex`-Eigenschaften der Steuerelemente manuell anpassen.

---

Der eigentliche Aufbau des Formulars samt seiner kompletten Größenanpassungs-Funktionslogik ist damit abgeschlossen. Rekapitulieren Sie: Für das, was Sie schätzungsweise in der letzten Stunde erreicht haben, hätten Sie noch in Visual Basic 6.0 sehr, sehr lange programmieren müssen. Mit dem `PictureBox`-Steuerelement, das eine Scroll-Funktionalität für seinen Inhalt angeboten hätte, vielleicht sogar mehrere Tage.

## Über die Eigenschaften `Name`, `Text` und `Caption`

Was für das Beispiel auf Designer-Seite jetzt noch fehlt, sind die Beschriftungen (für die Steuerelemente, die es betrifft) und die Namen der Steuerelemente, unter denen Sie sie beim Programmieren »erreichen« können.

Diese Dinge werden mit den Eigenschaften `Name` (für den Namen eines Steuerelements, mit dem Sie es beim Programmieren ansprechen) und `Text` (für Beschriftungen) festgelegt.

---

**TIPP:** Ausnahmslos jedes Steuerelement verfügt über eine `Text`-Eigenschaft, da diese auch in `Control` implementiert ist, auf das jedes Steuerelement aufbaut. Zwar gibt es Steuerelemente, die ihre `Text`-Eigenschaft nicht im Eigenschaftfenster offen legen, doch ist die `Text`-Eigenschaft bei diesen dennoch vorhanden.

---

---

**WICHTIG:** Die Caption-Eigenschaft, wie Sie sie von Visual Basic 6 beispielsweise für das Festlegen der Beschriftung von Schaltflächen und Label-Steuer-elementen kennen, gibt es in .NET NICHT mehr. Auch hier übernimmt, wie es unter VB6 beispielsweise auch schon immer beim Textbox-Steuer-element der Fall war, die Text-Eigenschaft diese Aufgabe.

---

Und noch ein ...

---

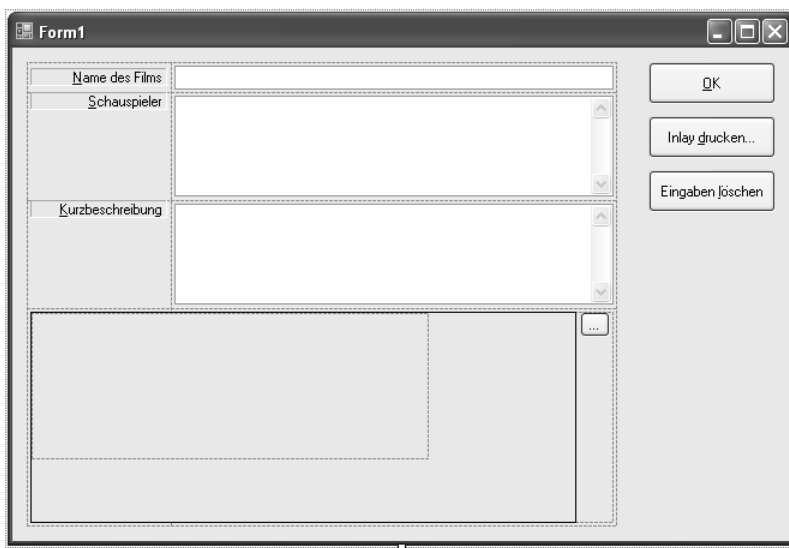
**TIPP:** Wenn Sie Text- und Name-Eigenschaften zuweisen, sollten Sie das nicht wechselweise sondern nacheinander machen. Bestimmen Sie also erst alle Name-Eigenschaften und dann alle Text-Eigenschaften. Der Grund: Wenn Sie das erste Steuer-element angeklickt und dann beispielsweise die Text-Eigenschaft im Eigenschaftenfenster festgelegt haben, brauchen Sie anschließend einfach nur das nächste Steuer-element anzuklicken und einfach drauflos zu tippen. Visual Studio merkt sich, dass Sie beim letzten Gebrauch des Eigenschaftenfensters die Text-Eigenschaft gesetzt hatten, und leitet in dem Moment, in dem Sie nach der Selektion eines neuen Steuer-elementes zu tippen begonnen haben, die Tastatureingaben automatisch an die zuletzt verwendete Eigenschaft im Eigenschaftenfenster weiter.

---

### Schnellzugriffstasten durch die Text-eigenschaft bestimmen

Die Text-Eigenschaft übernimmt bei vielen Steuer-elementen übrigens noch eine weitere Funktion: Buchstaben, vor die Sie das &-Zeichen stellen, markieren Sie als Schnellzugriffstasten. Sie erkennen die Schnellzugriffstasten eines Steuer-elementes dadurch, dass diese unterstrichen dargestellt werden. Zur Laufzeit befähigen Sie den Anwender Ihres Programms dann, dieses Steuer-element anzusteuern oder – bei Schaltflächen beispielsweise – auszulösen, in dem er die Schnellzugriffstaste in Verbindung mit **Alt** betätigt.

Auf diese Weise bestimmen Sie zunächst die Text-Eigenschaften aller Label- und Button- (Schaltflächen-) Steuer-elemente. Orientieren Sie sich dabei am besten an Abbildung 3.25 und folgender Tabelle.



**Abbildung 3.25:** Nehmen Sie bei der Zuweisung von Beschriftungen und Schnellzugriffstasten an die Steuer-elemente diese Grafik zu Hilfe

Steuerelement	Name (Name-Eigenschaft)	Beschriftung (Text-Eigenschaft)
Label (links, oben)	lblNameDesFilms	"Name des &Films"
Label (links, mitte)	lblSchauspieler	"&Schauspieler"
Label (links, unten)	lblKurzbeschreibung	"&Kurzbeschreibung"
Schaltfläche (rechts, oben)	btnOK	"&OK"
Schaltfläche (rechts, mitte)	btnInlayDrucken	"Inlay &drucken..."
Schaltfläche (rechts, unten)	btnEingabenLöschen	"Eingaben &löschen"
Textbox (oben, mitte)	txtNameDesFilms	"" (Leerstring – Löschen Sie die Eingabe)
Textbox (mitte, mitte)	txtSchauspieler	""
Textbox (unten, mitte)	txtKurzbeschreibung	""
Schaltfläche (neben der PictureBox)	btnCoverbildWählen	"..."
PictureBox	picCoverbild	- nicht anwendbar -

**Tabelle 3.1:** Verwenden Sie diese Name- und Text-Eigenschaften für die Steuerelemente im Formular

Sie erkennen, dass ich bei der Benennung von Steuerelementen nach einem bestimmten Schema vorgehe, indem ich im Namen mit einer Drei-Buchstaben-Abkürzung kenntlich mache, um was für ein Steuerelement es sich handelt. Ob Sie für eigene Namensgebungen von Steuerelementen diese Konvention adaptieren oder nicht, bleibt Ihnen natürlich überlassen. Viele Entwickler machen das, einige nicht – Microsoft sagt, man soll es *nicht* tun. Sollten Sie sich dafür entscheiden – im Abschnitt »Namensgebungskonventionen für Steuerelemente« ab Seite 67 finden Sie eine Tabelle, die die Konventionen für die wichtigsten Steuerelemente auflistet.

---

**HINWEIS:** Auch das Formular selbst verfügt über eine Text-Eigenschaft. In diesem Fall bestimmt sie den Text, der in der Titelzeile des Formulars erscheinen soll.

---

1. Selektieren Sie das Formular.
2. Bestimmen Sie im Eigenschaftenfenster als Text-Eigenschaft einen Text, der Ihnen geeignet scheint – beispielsweise »Der VB-Entwicklerbuch-Hüllen-Inlay-Generator« oder Ähnliches.

## Einrichten von Bestätigungs- und Abbrechen-Funktionalitäten für Schaltflächen in Formularen

Wie Sie Schnellzugriffstasten einrichten, haben Sie im letzten Abschnitt bereits erfahren. Es gibt zwei Tasten bei der Formularbedienung, denen eine besondere Funktionalität zukommt, und die Sie mit diesem Verfahren allerdings nicht definieren können: **Eingabe** (in der Regel für die OK-Schaltfläche), um einen Dialog zu bestätigen und **Esc**, um einen Dialog wieder zu verlassen, ohne dass die Eingaben übernommen werden.

VB6-Entwickler werden bei den Schaltflächeneigenschaften vergeblich nach diesen Einstellungsmöglichkeiten suchen – sie wurden in .NET nämlich in das Formular verlagert. Das macht letzten Endes auch mehr Sinn, schließlich kann nur jeweils eine Schaltfläche für das Abbrechen und eine weitere für das Bestätigen eines Dialogs in Frage kommen.

Diese Aufgaben übernehmen also zwei Formulareigenschaften namens `AcceptButton` und `CancelButton`. Beim Setzen dieser Eigenschaften klappen Sie eine Aufklappliste auf, die alle Schaltflächenelemente des Formulars enthält, die für die jeweilige Aufgabe in Frage kommen.

Für unser Beispiel weichen wir ein wenig vom Standard ab. Wir legen die Abbrechen-Funktionalität auf die *OK*-Schaltfläche und die Bestätigen-Funktionalität auf die *Inlay drucken*-Schaltfläche. Das macht mehr Sinn für den Anwender, denn er verlässt ja schließlich auch mit *OK* den Dialog. Ein versehentliches Betätigen von **Eingabe** führt damit nicht zum Dialogende (und damit zum Beenden des Programms).

1. Selektieren Sie das Formular im Designer.
2. Setzen Sie die `AcceptButton`-Eigenschaft auf die Schaltfläche `btnInlayDrucken`.
3. Setzen Sie die `CancelButton`-Eigenschaft auf die Schaltfläche `btnOK`.

## Hinzufügen neuer Formulare zu einem Projekt

Eine Smartclient-Anwendung besteht in den wenigsten Fällen aus nur einem Formular. In unserem Fall ist das genauso wenig der Fall. Wir benötigen ein weiteres Formular, das später die Druckvorschau enthält, und mit dem man den Druckvorgang auslösen kann. Diesem Formular werden dann später zur Laufzeit die zu druckenden Daten übergeben – die Funktion des Druckens und der Vorschau darstellung soll es dann völlig autonom übernehmen.

---

**HINWEIS:** Trennen Sie sich schon beim Designen der Formulare Ihrer Anwendung von der prozeduralen Denkweise, falls Sie es noch nicht getan haben. Es zeugt von einem schlechten Programmierstil, wenn Sie quasi von außen, also beispielsweise aus einem anderen Formular, einem anderen Modul oder einer anderen Klasse auf die Elemente eines Formulars zugreifen. Stellen Sie besser nur wenige öffentliche Methoden im Formular bereit, die ausschließlich dazu da sind, Parameter entgegen zu nehmen, und überlassen Sie dem Formular selbst die Auswertung dieser Parameter. Sie sollten – auch in Vorbereitung auf objektorientierte Programmierung – jedes Formular als eine kleine, in sich geschlossene Anwendung betrachten, der Sie lediglich Daten übergeben können und Resultate von diesem wiederbekommen.

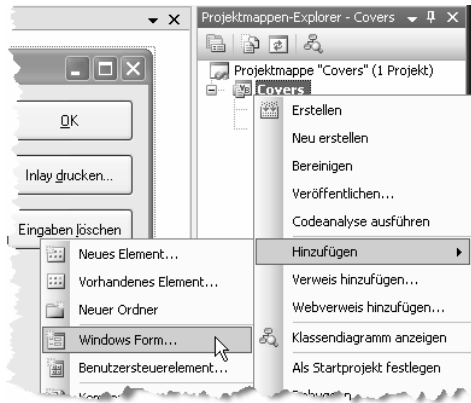
---

**WICHTIG:** Was im Formular passiert (das Setzen oder Abfragen von Eingabefeldern, das Auswerten von Benutzeraktionen) sollte einzig und allein der Formulkasse vorbehalten sein! Vermeiden Sie es, Inhalte von Benutzersteuerelementen von anderen Modulen, Klassen oder Formularen aus direkt zu manipulieren!

---

Um dem Projekt ein weiteres Formular hinzuzufügen, verfahren Sie wie folgt:

1. Öffnen Sie im Projektmappen-Explorer das Kontextmenü des Projektes (nicht der Projektmappe!) mit der rechten Maustaste. Falls Sie keine Projektmappe für Ihr Projekt sehen, öffnen Sie den Dialog *Extras/Optionen*, und wählen Sie im Bereich *Allgemein* die Option *Projektmappe immer anzeigen*.
2. Wählen Sie *Hinzufügen* und *Windows Form*.



**Abbildung 3.26:** Mithilfe des Eigenschaftenfensters können Sie jedes Steuerelement im Designer selektieren

3. Im Dialog, der jetzt erscheint, belassen Sie den vorgegebenen Namen zunächst mit *Form2.vb* so, wie er ist.
4. Klicken Sie auf *Hinzufügen*.
5. Vergrößern Sie das neue Formular, das jetzt sofort angezeigt wird, so, dass es genug Platz für weitere Steuerelemente bereitstellt. Orientieren Sie sich am besten dazu an Abbildung 3.27.
6. Suchen Sie in der Toolbox in der Sektion *Drucken* (die Sie im Bedarfsfall aufklappen müssen) nach dem *PrintPreviewControl*. Ziehen Sie es in ausreichender Größe im Formular auf. Stellen Sie dann die einzelnen Seiten des Steuerelements mithilfe der Ausrichtungslinien so ein, dass sie alle den gleichen Abstand zum oberen, linken und rechten Formularrand haben.
7. Setzen Sie die Anchor-Eigenschaft dieses Steuerelements auf *Top, Bottom, Left, Right*.
8. Fügen Sie zwei Schaltflächen in das Formular nebeneinander ein. Setzen Sie die Anchor-Eigenschaften beider Schaltflächen auf *Bottom, Right*.
9. Weisen Sie Name- und Text-Eigenschaften der Steuerelemente mithilfe der folgenden Tabelle zu.

Steuerelement	Name (Name-Eigenschaft)	Beschriftung (Text-Eigenschaft)
PrintPreviewControl	(lassen Sie es so, wie es heißt)	–
Linker Button	btnOK	&OK
Rechter Button	btnDrucken	&Drucken...
Formular	bleibt, wie sie ist	Covervorschau zeigen und Cover drucken

**Tabelle 3.2:** Name- und Text-Eigenschaften für die Steuerelemente im zweiten (Druck-) Formular

10. Speichern Sie das komplette Projekt ab, indem Sie entweder aus dem Menü *Datei* den Menüpunkt *Alles Speichern* oder das entsprechende Symbol aus der Werkzeugleiste verwenden.



**Abbildung 3.27:** So soll das Druckformular im Designer endgültig aussehen

## Wie geht's weiter?

Der Design-Teil unseres kleinen Softwareprojektes ist abgeschlossen – jetzt geht es darum, die entsprechenden Steuerelemente mit Programmlogik zu füttern. Die beiden folgenden Abschnitte sollen Ihnen als Referenzen zum Nachschlagen dienen – verlieren Sie ruhig einen Blick daran, und wenn Sie dann bereit sind, mit dem Programmieren zu beginnen, und die Software fertig zu stellen, fahren Sie mit dem Abschnitt »Der Codeeditor« ab Seite 71 fort.

## Namensgebungskonventionen für Steuerelemente in diesem Buch

Microsoft erklärt ausdrücklich, dass die Namensgebung von Objektvariablen keinerlei Hinweise darauf enthalten sollen, welchen Typ sie darstellen. Diverse Stil- und Sicherheitsüberprüfungswerkzeuge für Quellcode (beispielsweise FxCop, das bei den »größeren« Versionen von Visual Studio in den Projekteigenschaften implementiert ist) würden das bei der Quellcodeanalyse sogar anmahnen.

Eine Funktion von IntelliSense, so das Argument, reicht aus, um den Typ einer Objektvariablen eindeutig zu ermitteln – Sie brauchen lediglich den Mauszeiger auf die entsprechende Objektvariable zu positionieren, um den Typ der Variablen als Tooltip anzeigen zu lassen.

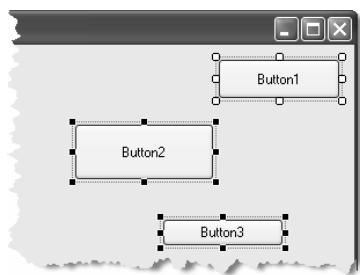
Für die Beispiele in diesem Buch habe ich mich dennoch dazu entschlossen, zumindest bei der Benennung von Objektvariablen die Steuerelemente referenzieren, darauf zu verzichten. Ich finde, dass das das Lesen von Listings auf Papier erleichtert und zum besseren Verständnis beiträgt. IntelliSense steht Ihnen nämlich nur im Codeeditor von Visual Studio zur Verfügung – aber wer weiß: Vielleicht arbeitet Microsoft vielleicht schon an einer Papierversion von IntelliSense?

Komponente	Präfix-/Namenkombination
Label	lblName
Button	btnName oder cmdName <sup>4</sup>
TextBox	txtName
CheckBox	chkName
RadioButton	optName <sup>5</sup> oder rbtName
GroupBox	grbName
PictureBox	picName
Panel	pnlName
ListBox	lstName
ComboBox	cmbName
ListView	lvwName
TreeView	twvName

**Tabelle 3.3:** Ein Vorschlag für Namenskonventionen bei der Namensgebung von Steuerelementen

## Funktionen zum Layouten von Steuerelementen im Designer




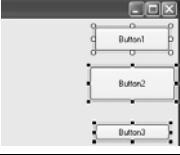

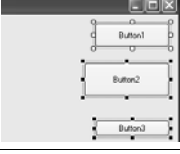

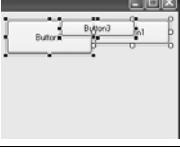

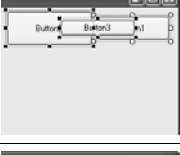

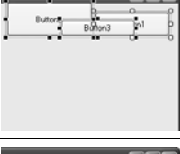

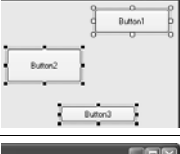

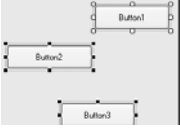
Die folgende Tabelle zeigt Ihnen die Funktionen, die sich hinter den Symbolen der Werkzeugleiste *Layout* befinden. Die folgende Grafik zeigt die Ausgangspositionierung der drei Steuerelemente im Formular. In der Tabelle finden Sie in der rechten Spalte das Ergebnis der Anordnung, nach dem Sie auf das entsprechende Symbol geklickt haben.



**Abbildung 3.28:** Die Ausgangsanordnung der Schaltflächen für die folgende Tabelle. Achten Sie darauf, dass sich bestimmte Funktionen an der oberen rechten Schaltfläche orientieren.

<sup>4</sup> Von `CommandButton` – so hieß eine Schaltfläche offiziell unter VB6. Unter .NET ist dies aber nicht mehr üblich; man sieht es nur noch hier und da von VB6-Portierungen.

<sup>5</sup> Von `OptionButton`, der Name der Optionsschaltfläche unter VB6.

Symbol	Funktion	Ergebnis nach Anwendung auf Steuerelemente in Abbildung
	Links ausrichten	
	Zentrieren	
	Rechts ausrichten	
	Oben ausrichten	
	Mittig ausrichten	
	Unten ausrichten	
	Breite angleichen	
	Höhe angleichen	

Symbol	Funktion	Ergebnis nach Anwendung auf Steuerelemente in Abbildung
	Größe angleichen	
	Horizontalen Abstand angleichen	
	Horizontalen Abstand vergrößern	- n/a -
	Horizontalen Abstand verkleinern	- n/a -
	Horizontalen Abstand entfernen	- n/a -
	Vertikalen Abstand angleichen	
	Vertikalen Abstand vergrößern	- n/a -
	Vertikalen Abstand verkleinern	- n/a -
	Vertikalen Abstand entfernen	- n/a -
	Horizontal im Formular zentrieren	- n/a -
	Vertikal im Formular zentrieren	- n/a -
	In den Vordergrund bringen	- n/a -
	In den Hintergrund bringen	- n/a -

**Tabelle 3.4:** Symbole, mit der Sie erweiterte Funktionalitäten des Ausgabefensters steuern

## Tastaturkürzel für die Platzierung von Steuerelementen

Taste	Aufgabe
<b>Pfeiltasten</b>	Verschiebt das ausgewählte Steuerelement pixelweise in die den Pfeiltasten entsprechende Richtung.
<b>Strg + Pfeiltasten</b>	Verschiebt das ausgewählte Steuerelement zur ersten, in der den Pfeiltasten entsprechenden Richtung liegenden Ausrichtungslinie.
<b>Umschalt + Pfeiltasten</b>	Vergrößert oder verkleinert das ausgewählte Steuerelement pixelweise entsprechend der verwendeten Pfeiltaste. ▶

Taste	Aufgabe
<b>Strg + Umschalt + Pfeiltasten</b>	Vergrößert oder verkleinert das Steuerelement so, dass es – abhängig von der verwendeten Pfeiltaste – mit der entsprechenden Seite an der jeweils nächsten Ausrichtungslinie anliegt.
<b>Alt</b>	Schaltet die Ausrichtungslinienfunktion so lange aus, wie die Taste beim Verschieben von Steuerelementen mithilfe der Maus gedrückt bleibt.

**Tabelle 3.5:** Symbole, mit der Sie erweiterte Funktionalitäten des Ausgabefensters steuern

## Der Codeeditor

Der Codeeditor ist mit Sicherheit das Element in Visual Studio, in dem Sie die mit Abstand meiste Zeit verbringen werden. Doch ist er über die Jahre zu einem solch genialen Werkzeug avanciert, das Ihnen soviel Arbeit abnehmen kann, dass das Arbeiten auch nach Jahren noch Spaß macht und flüssig von der Hand geht.

Das ist aber erst dann der Fall, wenn Sie alle seine Feinheiten und auch die eine oder andere Tücke kennen und zu beherrschen wissen. Und dabei geht es schon los mit der Wahl der richtigen Schriftart, die Ihnen ein ermüdungsfreies Arbeiten ermöglicht:

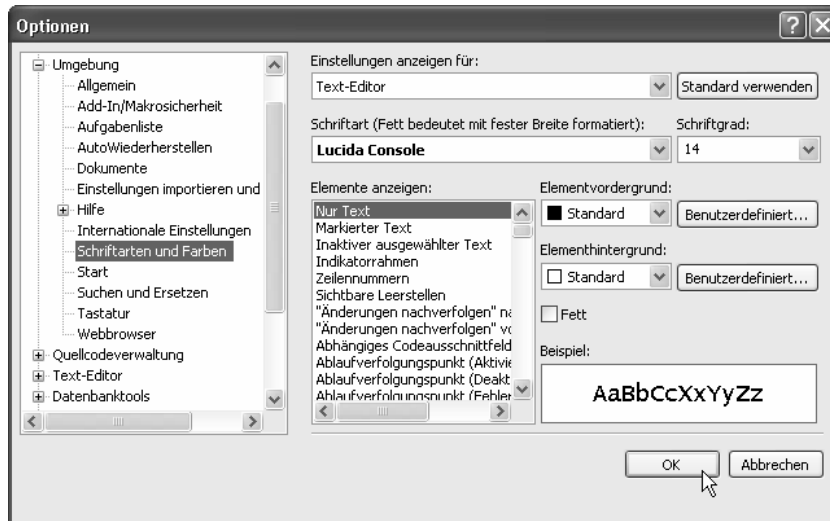
### Die Wahl der richtigen Schriftart für ermüdungsfreies Arbeiten

Sie werden Lachen: Für diesen kleinen Abschnitt habe ich tatsächlich fast zwei Stunden herumtelefoniert. Mich interessierte, ob:

1. Andere befreundete Programmierer die Original-Einstellung der Schriftart des Editors genau so stört wie mich, und
2. welche Schriftart die meisten Entwickler, die mit Visual Studio 2003 .NET oder Visual Studio 2005 arbeiten, bevorzugen.

Das Ergebnis war eine statistische Relevanz für zwei Schriftarten, von der Meinungsforscher nur träumen können:

- Bis zu einer Auflösung von 1024 x 768 Pixel auf einem Monitor – dies betraf besonders die Entwickler, die oft im Außendienst mit Notebooks arbeiten – ist FixedSys die bevorzugte Schrift. Da diese Schrift eine Bitmap-Schriftart ist, spielt die Fontgröße keine Rolle; sie erscheint immer in der gleichen Größe.
- Ab einer Auflösung von 1280 x 1024 Pixel oder bei 1024 x 768 Pixeln im Mehrmonitorbetrieb über 2 Monitore verteilt, war Lucida Console (aber nur ab 14 Punkt) die bevorzugte Größe.
- Die originale Courier-Schrift war bei keinem der befragten 11 Entwickler im Einsatz.



**Abbildung 3.29:** Auf dieser Registerkarte stellen Sie die Fonts unter anderem auch für den Codeeditor von Visual Studio ein

Um die Schriftart für den Editor umzustellen, wählen Sie aus dem Menü *Extras* den Menüpunkt *Optionen*. Wählen Sie die Registerkarte *Schriftarten und Farben*, die Sie im Zweig *Umgebung* finden. Der Aufruf dieses Dialogs dauert beim ersten Mal ein paar Sekunden – also nicht gleich ungeduldig werden und einen Absturz vermuten!

---

**HINWEIS:** Die Screenshots in diesem Buch sind übrigens ebenfalls alle mit der Schriftart Lucidia Console in 14 Punkt Größe entstanden. Für Präsentationen, Schulungen oder Projektbesprechungen am Beamer empfiehlt sich diese Schriftart im Übrigen auch bei kleineren Auflösungen.

---

## Viele Wege führen zum Codeeditor

Es gibt die verschiedensten Möglichkeiten, den Codeeditor ins Leben zu rufen. Der einfachste Weg, Module oder reine Klassendateien zu bearbeiten, läuft natürlich über den Projektmappen-Explorer: Ein Doppelklick auf eine Klassen- oder eine Moduldatei öffnet die Datei im Editor direkt. Um den Klassencode eines Formulars einzusehen, müssen Sie das Formular im Projektmappen-Explorer selektieren und anschließend auf das Symbol *Code anzeigen* klicken, das Sie in der oberen Zeile des Projektmappen-Explorer finden (der Tooltip hilft Ihnen, das richtige Symbol zu erhaschen). Folgende Möglichkeiten gibt es, den Codeeditor ins Leben zu rufen:

- Doppelklick auf eine reine Klassen- oder Moduldatei im Projektmappen-Explorer.
- Bei ausgewählter Formulkasse, Mausklick auf das Symbol *Code anzeigen* im Projektmappen-Explorer.
- Bei einem Compiler-Fehler: Doppelklick auf die entsprechende Fehlermeldung im Ausgabefenster.
- Bei einem Compiler-Fehler: Doppelklick auf die entsprechende Fehlermeldung in der Fehlerliste.
- Bei Kommentaren in der Aufgabenliste: Doppelklick auf den entsprechenden Kommentar.

- Doppelklick auf ein Element in einem Designer. Ein Doppelklick auf eine Schaltfläche in einem Formular bringt Sie beispielsweise zur bereits vorhandenen Ereignisbehandlung für diese Schaltfläche oder öffnet den Editor für das Formular und fügt den Coderumpf für die Ereignisbehandlungsroutine ein.
- Nach dem Auftreten eines Fehlers während der Ausführung einer Anwendung im Debug-Modus.

## IntelliSense – Ihr stärkstes Zugpferd im Coding-Stall

Das Konzept von IntelliSense spart Ihnen beim Entwickeln die meiste Zeit. Warum? IntelliSense liefert Ihnen alle denkbaren Informationen über Objekte und Sprachelemente, die Sie gerade in Bearbeitung haben.

Zur Demonstration implementieren wir nun die ersten Codezeilen unseres Beispiels.

1. Öffnen Sie *Form1* im Designer, indem Sie auf die Formulardatei im Projektmappen-Explorer doppelklicken.
2. Doppelklicken Sie anschließend auf die Schaltfläche *OK*. Visual Studio bringt Sie daraufhin zum Codeeditor für den Klassencode von *Form1* und stellt automatisch den Funktionsrumpf für die Ereignisbehandlung der *OK*-Schaltfläche zur Verfügung. Das ist die Methode, die aufgerufen und ausgeführt wird, wenn der Anwender zur Laufzeit auf die *OK*-Schaltfläche klickt.
3. In die Zeile zwischen *Private Sub...* und *End Sub* geben Sie nun **me.** ein. Sobald Sie den Punkt getippt haben, öffnet sich eine Liste mit allen Elementen, die für das Objekt anwendbar sind (in diesem Fall ist *me* das Formular selbst, da wir uns in der Klassendatei *Form1* befinden; die genaue Bedeutung lernen Sie im Klassenkapitel noch kennen – wir wollen uns an dieser Stelle auf die Editorfähigkeiten konzentrieren). Diese Liste nennt sich »Vervollständigungsliste«.
4. Um das Formular beim Mausklick auf *OK* zu schließen, wollen wir die *Close*-Methode verwenden. Geben Sie nun die ersten Buchstaben von *Close* weiter ein, springt der Auswahlbalken irgendwann auf die gesuchte Methode (siehe Abbildung 3.30).

---

**HINWEIS:** Sie werden feststellen, dass IntelliSense Ihnen nicht nur eine vollständige Elementliste für das Objekt liefert, sondern auch eine Kurzbeschreibung des jeweils ausgewählten Elements als Tooltip – ebenfalls in Abbildung 3.30 zu erkennen.

---

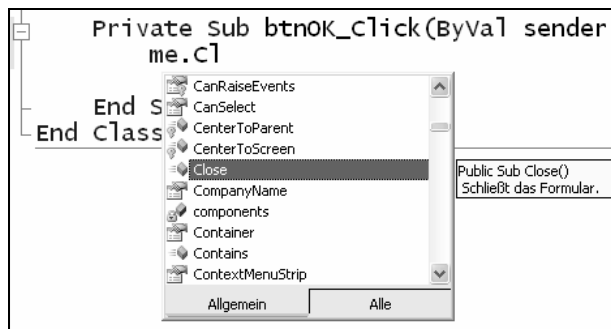
5. Sie brauchen den Methodennamen nun gar nicht weiter einzugeben. Sobald der richtige Methodename markiert ist, drücken Sie einfach **Eingabe** – der Codeeditor fügt die restlichen Buchstaben des Methodennamens dann automatisch ein und stellt den Cursor in die nächste Zeile.

---

**TIPP:** Möchten Sie hinter dem Methodennamen keine neue Zeile beginnen, drücken Sie **Strg+Eingabe**. Damit vervollständigen Sie lediglich den Methodennamen. Oder Leertaste, dann erscheint die Methode gefolgt von einem Leerschritt.

---

Die erste Funktionalität des Programms haben Sie damit schon implementiert: Starten Sie das Programm testweise mit **F5**, und probieren Sie die *OK*-Schaltfläche aus!



**Abbildung 3.30:** Der Punkt nach einem Objektnamen öffnet dank IntelliSense die Vervollständigungsliste, die eine Übersicht liefert, welche Elemente für das Objekt zur Anwendung kommen können

### Filtern von Elementen in der Vervollständigungsliste

Wie Sie in Abbildung 3.30 erkennen können, verfügt die von IntelliSense gezeigte Elementliste über zwei Registerungen, mit denen Sie die Elemente nach Wichtigkeit filtern können. Welche Elemente dabei »wichtig« sind, bestimmt Microsoft – schweigt sich aber über das Selektionsverfahren aus.

Zitat der Online-Hilfe: »Auf der standardmäßig aktivierten Registerkarte *Allgemein* werden Elemente angezeigt, die am häufigsten zum Vervollständigen der geschriebenen Anweisung verwendet werden. Auf der Registerkarte *Alle* sind alle für die automatische Vervollständigung verfügbaren Elemente aufgeführt, einschließlich der Elemente auf der Registerkarte *Allgemein*.«

### Anzeigen der Parameterinfo von Elementen

Für das nächste IntelliSense-Feature werden wir die Funktion »Eingaben löschen« implementieren.

1. Wechseln Sie mit **Strg+Tab** erneut in die Designerdarstellung des Formulars *Form1*.
2. Doppelklicken Sie auf die Schaltfläche *Eingaben löschen*, um den Funktionsrumpf für die Ereignisbehandlungsroutine dieser Schaltfläche einzufügen.
3. Beginnen Sie einzugeben:

```
Dim locDr As DialogResult
```

Sie werden feststellen, dass Ihnen IntelliSense nach dem Schreiben des Schlüsselworts *As* auch wieder die Vervollständigungsliste anbietet. Tippen Sie soviel vom Wort »DialogResult«, bis diese Enumeration in der Liste erscheint und markiert ist. Drücken Sie anschließend **Eingabe**.

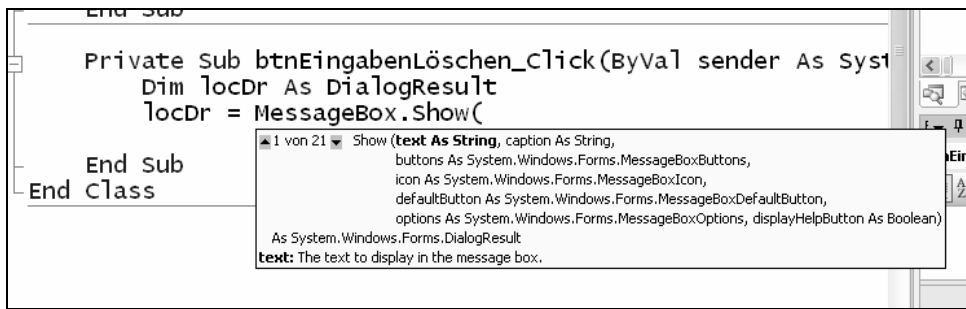
4. Schreiben Sie in die nächste Zeile

```
locDr=
```

Auch hier wird IntelliSense wieder aktiv. Diesmal zeigt es Ihnen alle möglichen Member der Enumeration *DialogResult* an, da es davon ausgeht, dass Sie der Variablen *locDr* eines ihrer Elemente zuweisen wollen. Ignorieren Sie die Liste aber einfach in diesem Fall, und schreiben Sie weiter (das *Show* brauchen Sie dabei dank Vervollständigungsfunktion auch nicht komplett selbst zu schreiben).

```
MessageBox.Show(
```

In dem Moment, in dem Sie die Klammer tippen, zeigt Ihnen IntelliSense eine vollständige Parameterinfo der `MessageBox.Show`-Methode, mit der Sie übrigens ein Meldungsfeld auf dem Bildschirm darstellen können.



**Abbildung 3.31:** Bei Methoden, die über Parameter verfügen, zeigt IntelliSense alle Parameter mit samt Erklärungen der Parameter an. Der jeweils aktuelle Parameter, den Sie gerade eingeben, ist in Fettschrift gekennzeichnet.

### Mehrzeilige Befehlszeilen und die Parameterinfo

Viele von Ihnen wissen bereits aus VB6-Zeiten oder von Erfahrungen mit VBScript, dass Sie eine logische Codezeile in Visual Basic der besseren Übersicht wegen mithilfe des Underscore-Zeichens (»\_«) auf mehreren physischen Zeilen im Editor verteilen können. Am Ende der (physischen) Zeile fügen Sie dazu ein Leerzeichen, gefolgt vom Underscore-Zeichen, ein. Die eigentliche (logische) Zeile schreiben Sie dann ganz normal weiter, als hätten Sie nie einen Zeilenumbruch eingefügt.

Um beim Beispiel zu bleiben:

1. Ergänzen Sie die Zeile (die ja noch nicht vollständig eingegeben wurde), um

```
"Sind Sie sicher?", "Eingaben löschen?", _  
und drücken anschließend Eingabe.
```

Nach dem Zeilenumbruch ist die Parameterinfo verschwunden. Um die Parameterinfo auch in der neuen Zeile wieder darzustellen, drücken Sie einfach **Strg+Umschalt+Leertaste**.

2. Geben sie nun den restlichen Teil der logischen Befehlszeile ein.

```
MessageBoxButtons.YesNo, _  
MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)
```

## Automatische Vervollständigung von Struktur-Schlüsselworten und Codeeinrückung

1. Für die Komplettierung der Methode zum Löschen der Eingabefelder geben Sie bitte die folgende Zeile an:

```
If locDr = Windows.Forms.DialogResult.Yes Then
```

Sobald Sie **Eingabe** nach dem Eingeben dieser Zeile betätigt haben, fügt der Editor automatisch ein entsprechendes `End If` zwei Zeilen darunter ein und platziert die Schreibmarke zwischen den beiden Zeilen.

## 2. Wenn Sie nun die restlichen Zeilen

```
txtKurzbeschreibung.Text = ""  
txtNameDesFilms.Text = ""  
txtSchauspieler.Text = ""  
picCoverbild.Image = Nothing  
myBilddateiname = ""
```

dazwischen eingeben, werden Sie feststellen, dass egal in welcher Spalte Sie zu tippen beginnen, die Zeilen sich immer der von If/End If vorgegebenen Struktur anpassen und entsprechend eingerückt formatiert werden.

---

**HINWEIS:** Das funktioniert auch bei geschachtelten Strukturen; Sie behalten auf diese Weise immer den Überblick, in welcher Strukturverschachtelungsebene Sie sich gerade befinden.

---

## Fehlererkennung im Codeeditor

Visual Basic verfügt über einen so genannten Hintergrund-Compiler (*Background Compiler*). Dieser Hintergrund-Compiler leistet einiges an Vorarbeit für den eigentlichen Compiler, der ja erst dann aktiv wird, wenn Sie ein Projekt erstellen (und das führt dazu, dass Anwendungen, die Sie in Visual Basic entwickeln, wesentlich kürzere Turn-Around-Zeiten<sup>6</sup> aufweisen, als die, die Sie in anderen .NET-Sprachen entwickeln. Dieser Hintergrund-Compiler spart Ihnen eine ganze Menge Zeit beim Entwickeln, denn im Gegensatz zu anderen Sprachen wie C++ oder C# (oder auch zum alten VB6) entdeckt der Hintergrund-Compiler syntaktische Fehler bereits nachdem Sie eine Codezeile vollständig eingegeben haben.<sup>7</sup>

### Einfache Fehlerkennzeichnung im Editor

Wenn Sie die letzten Änderungen am Code aufmerksam nachvollzogen haben, bemerkten Sie sicherlich einen Fehler in der letzten Zeile, die Sie eingegeben haben. Dieser Fehler war auch gar nicht schwer zu bemerken, denn schließlich hat der Codeeditor diese Zeile gekennzeichnet wie in Abbildung 3.32 zu sehen.



```
If locDr = Windows.Forms.DialogResult Then  
    txtKurzbeschreibung.Text = ""  
    txtNameDesFilms.Text = ""  
    txtSchauspieler.Text = ""  
    picCoverbild.Image = Nothing  
    myBilddateiname = ""  
End If
```

Der Name "myBilddateiname" wurde nicht deklariert.

**Abbildung 3.32:** Fehler, die der Hintergrundcompiler feststellt, werden direkt im Editor noch vor dem eigentlichen Kompilierungsvorgang beim Erstellen des Projektes gekennzeichnet

<sup>6</sup> Als »Turn-Around-Zeit« (etwa: »Wenden-Zeit«, wie das Wenden beim Autofahren) bezeichnet man die Zeitspanne, die vom Starten des Compilers über das Kompilieren eines Projektes bis zum eigentlichen Anwendungsstart vergeht.

<sup>7</sup> Hintergrundcompiler gibt es zwar auch in diesen Sprachen, aber die sind lange nicht so konsequent wie in Visual Basic .NET bzw. Visual Basic 2005, und bei ihnen passiert es oft, dass der eigentliche Compiler erst beim Erstellen des Projekts syntaktische Fehler oder nicht deklarierte Variablen findet. In Visual Basic .NET bzw. 2005 passiert das äußerst selten – quasi nie.

Diese betroffene Variable, die wir später im gesamten Formular für die Speicherung des Dateinamens benötigen, wurde nicht deklariert. Also machen wir das als nächstes.

## Editorunterstützung bei Fehlern zur Laufzeit

Direkt unterhalb der Klassendefinition fügen Sie die im Folgenden in Fettschrift formatierte Zeile ein:

```
Public Class Form1
```

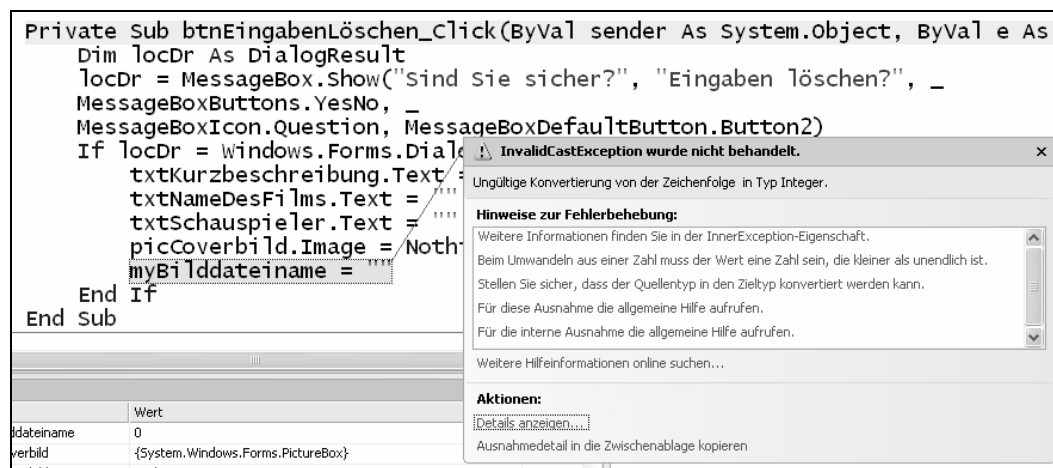
```
    Dim myBilddateiname As Integer
```

Sie sehen, dass der Fehler nun nicht mehr durch Unterschlängelung markiert ist.<sup>8</sup>

Nun probieren wir das Programm in seinem derzeitigen Zustand aus.

- Starten Sie das Programm mit **F5**.
- Geben Sie ein paar Zeilen in die Eingabefelder an.
- Klicken Sie auf *Eingabe löschen*.
- Bestätigen Sie das Meldungsfeld mit *Ja*.

Statt die Eingabefelder löschen, bricht das Programm mit folgender Meldung ab (Abbildung 3.33):

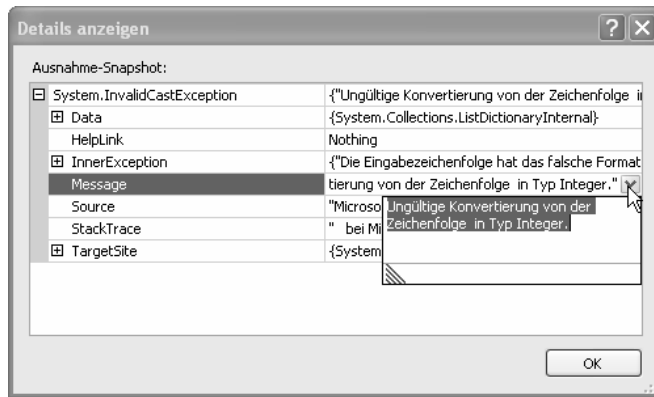


**Abbildung 3.33:** Tritt während der Anwendungsausführung im Debug-Modus ein Fehler (eine so genannte »Ausnahme« – engl.: *Exception*) auf, ruft Visual Studio den Editor auf, unterlegt die betroffene Stelle gelb und zeigt einen entsprechenden Hinweis an

Fehler, die zur Laufzeit in einer .NET-Anwendung auftreten und dafür verantwortlich sind, dass ein Programm nicht fortgesetzt werden kann, werden als Ausnahmen (engl: *Exception* – sprich: »Ixäpschen«) bezeichnet. In diesem Fall ist uns ein Fehler beim Deklarieren der Variablen

<sup>8</sup> Kleiner Hinweis am Rande: Bevor Sie nun eine E-Mail schreiben, da Sie glauben, einen Fehler gefunden zu haben – warten Sie erst die nächsten Absätze ab! ;-)

myBilddateiname passiert,<sup>9</sup> der schließlich zu dieser Ausnahme geführt hat. .NET versuchte zur Laufzeit, den Leerstring der Variablen zuzuweisen, die aber dummerweise als Integervariable deklariert wurde – das führte zu einer `InvalidCastException` (etwa: *Ausnahme wegen ungültiger Typkonvertierung*).



**Abbildung 3.34:** Mithilfe des Ausnahmedetail-Dialogs erfahren Sie Genaueres über die Umstände, die zur Ausnahme führten

Falls Sie in einem solchen Fall genauere Information zum Ausnahmenumstand benötigen, klicken Sie unten im Dialog unter *Aktionen* auf *Details anzeigen...* Der Editor zeigt Ihnen anschließend einen weiteren Dialog (siehe Abbildung 3.34), mit dem Sie diese erweiterten Informationen abrufen können.

3. Für das weitere Nachvollziehen des Beispiels, klicken Sie im Dialog auf *OK*.
4. Schließen Sie den darunter liegenden Dialog mit einem Mausklick auf die Schließschaltfläche in der rechten oberen Ecke.
5. Beenden Sie das Debuggen: Wählen Sie dazu aus dem Menü *Debuggen* den Befehl *Debuggen beenden* oder klicken Sie in der Symbolleiste auf das entsprechende Symbol zum Beenden des Debuggens.

### Fehlerverbesserungsvorschläge des Editors bei Typkonflikten – erzwungene Typsicherheit

Passiert wäre das nicht, wenn wir in unsere Anwendung von vornherein Typsicherheit erzwungen hätten. In diesem Fall hätten wir den Fehler bereits gemerkt, noch bevor wir das Programm gestartet hätten – der Editor hätte uns darauf aufmerksam gemacht.

Seit Visual Basic .NET 2002 kennt Visual Basic die Anweisung `Option Strict [On|Off]`. Schalten Sie `Option Strict` mit `On` ein, sorgt schon der Hintergrund-Compiler dafür, dass Sie nur gleiche Typen zuweisen können – oder optional strikt dafür sorgen, dass eine saubere Typkonvertierung (beispielsweise von `Integer` zu `String`) stattfindet.

Sobald Sie die Anweisung

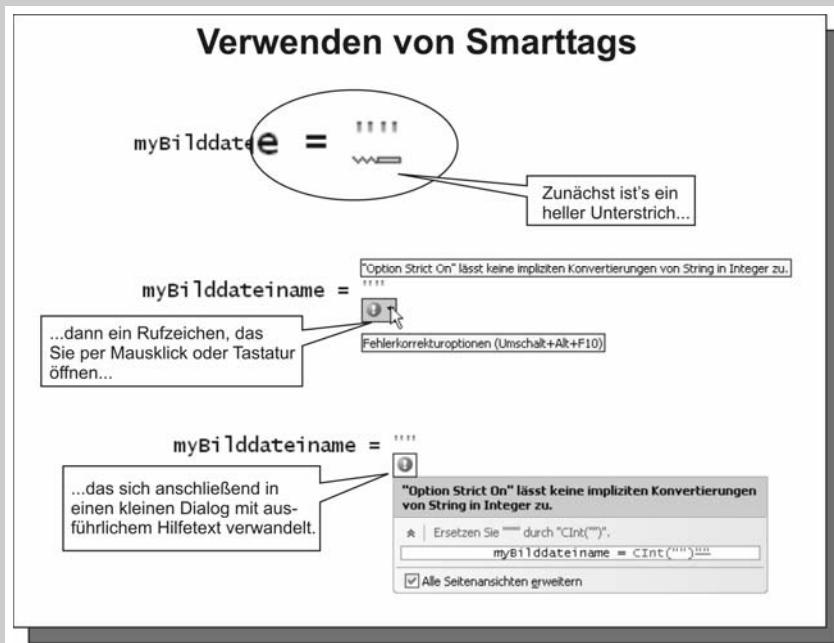
```
Option Strict on
```

<sup>9</sup> O.K., o.k. – mir, nicht Ihnen.

ganz oben in den Code einfügen (noch vor der Klassendefinitionsanweisung `Class Form1`) und anschließend zurück zur betroffenen Zeile scrollen, sehen Sie, dass nicht nur der vermutlich zur Ausnahme führende Teil unterschlängelt ist, sondern am Ende auch einen gelben Balken aufweist.

## Smarttags im Editor von Visual Basic

Einen Smarttag erkennen Sie zunächst als kleinen, hellen Unterstrich in einer Codezeile im Visual Basic-Editor, an der es seiner Meinung nach irgendetwas zu verbessern oder anzumerken gibt.



Fahren Sie mit dem Mauszeiger auf diesen Unterstrich, wandelt er sich in ein kleines Symbol mit der Form eines Ausrufezeichens, das Ihnen verschiedene Arten von Unterstützung anbietet. Smarttags können dabei ganz unterschiedliche Formen zusätzlicher Unterstützung anbieten – nicht nur Hilfestellungen bei Typkonflikten geben, wie hier im Beispiel zu sehen.

## Autokorrektur für intelligentes Kompilieren

In vielen Fällen verbirgt sich hinter dem Smarttag ein Dialog, der Ihnen einen Korrekturvorschlag für den erkannten »Fehler« unterbreitet – etwa, wie in der Abbildung zu sehen. Solche Hilfedialoge, die sich hinter Smarttags verbergen, nennt Microsoft übrigens »*Autokorrektur für intelligentes Kompilieren*«. Falls die Autokorrektur für intelligentes Kompilieren Recht behält, und es sich tatsächlich um den vermuteten Fehler handelt, brauchen Sie noch nicht einmal die Tastatur zu bemühen: Klicken Sie einfach auf den blauen Korrekturvorschlag, und der Editor nimmt die Korrektur im Programmcode vor.

---

**WICHTIG:** In unserem Beispiel greift die Autokorrektur für intelligentes Kompilieren an der Stelle des Fehlers leider nicht, weil es sich um einen Folgefehler handelt. Aber Sie sehen, dass wir auf jeden Fall einen kompletten Turn-Around-Lauf gespart hätten, wenn `Option Strict` von vorne herein eingeschaltet gewesen wäre. Aus diesem Grund sollten Sie `Option Strict` projektweit einschalten und sich die Anweisung vor jeder Codedatei sparen. Außerdem sollten Sie die Visual Studio-Optionen so einstellen, dass `Option Strict` grundsätzlich beim Anlegen jedes neuen Projektes eingeschaltet ist. Wie das geht, zeigt der folgende graue Kasten.

---

Um den Fehler zu beheben, ändern Sie die Zeile

```
Private myBilddateiname As Integer  
einfach in  
Private myBilddateiname As String
```

## Erzwungene Typsicherheit (Option Strict) projektweit einstellen

Sie können dafür sorgen, dass Typsicherheit grundsätzlich in einem Projekt erzwungen wird, ohne dass Sie dazu `Option Strict On` über jede Codedatei schreiben müssen. Dazu öffnen Sie im Projektmappen-Explorer das Kontextmenü des entsprechenden Projekts (nicht der Projektmappe!) und wählen *Eigenschaften*. Die Projekteigenschaften öffnen sich nun als Dokumentenfester in der Dokumentenregisterkartengruppe, und Sie können auf der Registerkarte *Kompilieren* `Option Strict` global mit *On* einschalten.

### Erzwungene Typsicherheit für alle folgenden neuen Projekte

Möchten Sie, dass diese Einstellung grundsätzlich gilt, wenn Sie ein neues Projekt anlegen, müssen Sie den Optionsdialog von Visual Studio bemühen. Rufen Sie ihn mit *Extras | Optionen* auf, und wählen Sie den Bereich *Projekte und Projektmappen*. Auf der Registerkarte *VB-Standard* nehmen Sie die Einstellungen für `Option Strict` vor.

In ► Kapitel 6 finden Sie im Abschnitt »Erzwungene Typsicherheit und Deklarationszwang von Variablen« ein weiteres Beispiel für `Option Explicit` und `Option Strict`.

## XML-Dokumentationskommentare für IntelliSense bei eigenen Objekten und Klassen

Wie Ihnen IntelliSense beim Finden der richtigen Klassen, Objekte, Methoden und anderer Elemente bei der Entwicklung von Anwendungen helfen kann, haben Sie bereits kennen gelernt. Doch damit ist noch lange nicht das Ende der Fahnenstange erreicht.

Zur Demonstration müssen wir ein wenig mehr vorbereitenden Aufwand betreiben. Dazu implementieren wir im Folgenden eine Methode, die eine Bilddatei aus einer Datei in ein Image-Objekt lädt. Auch hier soll die eigentliche Funktionsweise nicht von primärem Interesse sein – es geht schließlich immer noch um die Erarbeitung der Funktionalitäten der Codeeditor-Fähigkeit.

1. Fügen Sie zu diesem Zweck die folgenden Zeilen in den Klassencode von *Form1* ein:

```
Function CoverbildAusDateinamen(ByVal CoverbildDateiname As String) As Image
    Dim locImage As Image
    If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <> "" Then
        locImage = Image.FromFile(CoverbildDateiname)
        Return locImage
    End If
    Return Nothing
End Function
```

2. Wechseln Sie mit **F7** zum Entwurfsmodus (zum Designer-Dokumentenfenster). Übrigens: Mit **F7** wechseln Sie zwischen Designer- und Codedarstellung eines Formulars.
3. Doppelklicken Sie auf die Auslassungsschaltfläche neben der PictureBox, um den Codeeditor zu öffnen und die Ereignisbehandlungsroutine für diese Schaltfläche zu bearbeiten.
4. Fügen Sie in die Stub<sup>10</sup> (in den Funktionsrumpf, also zwischen Private Sub btnCoverbildWählen ... und End Sub) folgende Zeilen ein:

```
Dim locOfd As New OpenFileDialog

With locOfd
    locOfd.CheckFileExists = True
    locOfd.DefaultExt = "*.bmp"
    locOfd.Filter = "JPeg-Bilder (*.jpg)|*.jpg|Windows Bitmap (*.bmp)|*.bmp|Alle Dateien (*.*)|*.*"

    Dim locDr As DialogResult = locOfd.ShowDialog()
    If locDr = Windows.Forms.DialogResult.Cancel Then
        Return
    End If

    myBilddateiname = locOfd.FileName
End With
```

5. Zwischen den letzten beiden Zeilen ergänzen Sie nun eine weitere Zeile, die Sie bitte noch nicht komplett eingeben:

```
picCoverbild.Image = CoverbildAusDateinamen(
```

Sobald Sie die Klammer getippt haben, sehen Sie, dass IntelliSense auch bei selbst geschriebenen Methoden (und anderen Elementen wie Eigenschaften, etc.) greift. Doch schauen Sie sich Abbildung 3.35 an. Fällt Ihnen was auf?



```
myBilddateiname = locOfd.FileName
picCoverbild.Image = CoverbildAusDateinamen(
End With
End Sub
```

CoverbildAusDateinamen (CoverbildDateiname As String) As System.Drawing.Image

**Abbildung 3.35:** Auch bei selbst geschriebenen Methoden funktioniert IntelliSense – natürlich fehlen dabei zunächst noch die Erklärungen

---

<sup>10</sup> Sprich: »Stap«.

Genau wie bei eingebauten Methoden wird IntelliSense zwar aktiv – doch die sonst so hilfreichen Erklärungen finden wir hier natürlich nicht. Woher sollen sie auch stammen! Noch in Visual Basic 2003 war ohne Tools von Drittherstellern in Sachen IntelliSense bei selbst entwickelten Methoden an dieser Stelle Schluss. Doch mit Visual Basic 2005 ist das anders, wie Sie gleich sehen werden:

6. Bevor Sie nämlich die Zeile nun komplettieren, positionieren Sie die Schreibmarke oberhalb der Zeile

```
Function CoverbildAusDateinamen(ByVal CoverbildDateiname As String) As Image
```

7. Tippen Sie zwei Hochkomma (**Umschalt+#**).

8. Sobald Sie das dritte Hochkomma getippt haben, wird der Codeeditor aktiv und greift Ihnen unter die Arme.

```
''' <summary>
''' |
''' </summary>
''' <param name="CoverbildDateiname"></param>
''' <returns></returns>
''' <remarks></remarks>
Function CoverbildAusDateinamen(ByVal CoverbildDateiname As String) As
    Dim locImage As Image
    If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <>
        locImage = Image.FromFile(CoverbildDateiname)
        Return locImage
    End If
    Return Nothing
End Function
```

**Abbildung 3.36:** Der Editor fügt ein XML-Skelett über der Methode ein, in der Sie ihre Dokumentation nur zu ergänzen brauchen

9. Für die folgenden Schritte orientieren Sie sich auch an Abbildung 3.37. Geben Sie zwischen `<summary>` und `</summary>` eine Funktionsbeschreibung ein.

```
''' <summary>
''' Lädt ein Bild, so vorhanden, aus einer Datei und liefert das
''' Bild als Image (oder Nothing) zurück.
''' </summary>
''' <param name="CoverbildDateiname">Name der Coverbild-Datei.</param>
''' <returns>Image-Objekt, das das Bild der angegebenen Datei enthält.</returns>
''' <remarks>Erstellt von Klaus Löffelmann am 26.10.2005</remarks>
Function CoverbildAusDateinamen(ByVal CoverbildDateiname As String) As Image
    Dim locImage As Image
    If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <> "" Then
        locImage = Image.FromFile(CoverbildDateiname)
        Return locImage
    End If
    Return Nothing
End Function
```

**Abbildung 3.37:** Komplettieren Sie die XML-Tags etwa nach dieser Vorlage. Denken Sie beim mehrzeiligen Verteilen eines Textes an die Leerzeichen (blaues Kästchen in der Abbildung).

---

**HINWEIS:** Denken Sie daran, dass Sie beim Verteilen von Funktionsbeschreibungen über mehrere Zeilen ein Leerzeichen entweder *hinter* das letzte Wort der vorherigen oder *vor* das erste Wort der nächsten Zeile setzen, damit die Wörter bei der späteren Anzeige als Tooltip nicht zusammenlaufen.

---

10. Zwischen `<param name="CoverbildDateiname">` und `</param>` geben Sie die Bedeutung des Parameters *CoverbildDateiname* an.

---

**HINWEIS:** Sollten Sie es zu einem späteren Zeitpunkt mit Methoden oder Eigenschaften zu tun haben, die mehrere Parameter entgegennehmen, werden an dieser Stelle weitere `param`-Tags aufgelistet, zwischen denen Sie die Beschreibungen der weiteren Parameter platzieren können.

---

11. Zwischen `<returns>` und `</returns>` geben Sie die Bedeutung des Rückgabewertes an.  
12. Optional geben Sie zwischen `<remarks>` und `</remarks>` eine Bemerkung ein.  
13. Wenn Sie alle Eingaben abgeschlossen haben, kehren Sie zur noch nicht vollständig eingegebenen Zeile

```
picCoverbild.Image = CoverbildAusDateinamen(
```

zurück. Löschen Sie alle Zeichen, bis nur noch »Cov« von »CoverbildAusDateinamen(« übrig bleibt, und drücken Sie anschließend **Strg+Leertaste**. Sie sehen nun, dass die Funktion nicht nur in der Vervollständigungsliste zu sehen ist (das war sie zuvor auch schon), sondern nunmehr auch die Funktionsbeschreibung enthält, die Sie mithilfe der XML-Tags angegeben haben (siehe Abbildung 3.38).



**Abbildung 3.38:** Wenn Sie Ihre Methoden- und Eigenschaftenprozeduren mit entsprechenden XML-Tags versehen haben, zeigt IntelliSense sowohl eine Funktionsbeschreibung ...

14. Drücken Sie **Strg+Eingabe**, um den Funktionsnamen zu vervollständigen.  
15. Tippen Sie die Klammer, wird IntelliSense wieder aktiv (siehe Abbildung 3.39). Sie sehen, dass IntelliSense nun nicht nur den Parameternamen und den Parametertyp, sondern auch die Parameterdokumentation zeigt.

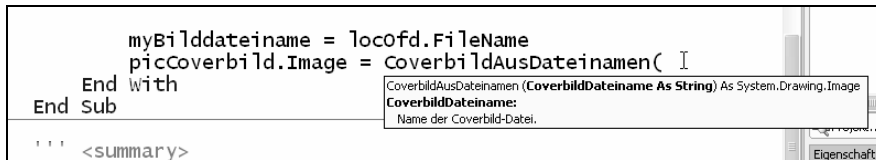


Abbildung 3.39: ... als auch entsprechende Parametererklärungen an

16. Vervollständigen Sie die Zeile, sodass diese komplett wie folgt lautet:

```
picCoverbild.Image = CoverbildAusDateinamen(myBilddateiname)
```

---

**HINWEIS:** Übrigens: So ganz nebenbei haben Sie mit nur ein paar Zeilen Code die komplette Bilddarstellung im Formular implementiert – und dank der Einstellungen, die Sie zuvor im Designer vorgenommen haben, läuft die Bildauschnittseinstellung mit Rollbalken ebenfalls schon. Den Beweis dazu können Sie selbst antreten: Starten Sie das Programm mit **F5**, klicken Sie auf die Auslassungsschaltfläche, und wählen Sie im Dialog, der jetzt gezeigt wird, eine Bilddatei aus.

---

## Hinzufügen neuer Codedateien zum Projekt

Für unser Beispiel benötigen wir eine Datenstruktur, mit der die Eingaben, die der Anwender im Hauptformular getätigt hat, an das Druckformular übergeben werden.

Das Programm verwendet also diese Datenstruktur in diesem Beispiel, um die einzelnen Datenfelder in dieser Datenstruktur – nennen wir Sie *CoverInhalt* – zunächst zwischenspeichern, und sie dann in einem Rutsch an das Druckformular zu übergeben.

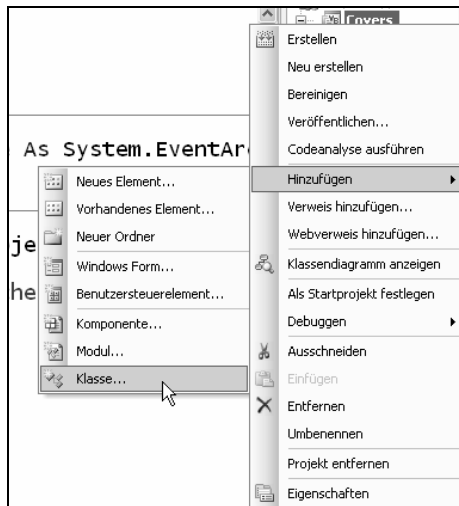
Auf diese Weise sind die beiden Aufgaben – Datenerfassung und Drucken – sauber voneinander getrennt. Der Hauptdialog sorgt in eigener Regie für die Datenerfassung, der Druckdialog selbstständig für das Drucken der Daten. Der Hauptdialog muss so nicht auf den Druckdialog direkt zugreifen, und dort irgendwelche Variablen manipulieren, sondern sagt dem Druckdialog nur mithilfe einer einzelnen öffentlichen Funktion, wie dieser das Cover in der Vorschau darstellen oder auf einem Drucker ausdrucken soll.

Eleganterweise bringen wir diese Datenstruktur in einer eigenen Codedatei unter: Der Klassendatei, die den gleichen Namen wie die Datenstruktur (die Klasse) selbst bekommen soll: *CoverInhalt*.

---

**HINWEIS:** Visual Basic 6 Programmierer kennen Datenstrukturen der einfachsten Ausführung in Form von benutzerdefinierten Typen. Ohne den Klassen- oder Umstiegsteil des Buches vorwegzunehmen: Die einfachste vorstellbare Klasse, wie Sie sie gleich kennen lernen werden, entspricht in etwa der Definition eines neuen Typen mit `Type`.

---



**Abbildung 3.40:** So fügen Sie eine neue Klassendatei zum Projekt hinzu

1. Um eine neue Klassendatei zu einem Projekt hinzuzufügen, verfahren Sie auf fast genau dieselbe Art und Weise, wie Sie es beim Hinzufügen einer Formular-Klassendatei schon getan haben. Rufen Sie das Kontextmenü des Projektes *Covers* (nicht der Projektmappe!) im Projektmappen-Explorer auf.
2. Wählen Sie *Hinzufügen | Klasse*.
3. Im jetzt erscheinenden Dialog geben Sie den Namen der Klassendatei (ohne Dateiendung) ein – für unser Beispiel **CoverInhalt**.
4. Klicken Sie auf **OK** oder drücken Sie **Eingabe**.

Der Codeeditor wird geöffnet; die Klassendefinition ist bereits vorgegeben. Geben Sie nun zwischen `Public Class CoverInhalt` und `End Class` die folgenden Zeilen ein:

```
Public FilmTitel As String
Public Schauspieler As String
Public Beschreibung As String
Public Coverbild As Image
```

Mit der Einführung dieser neuen Klasse haben wir die Möglichkeit, die Daten aus dem Hauptformular auszulesen und dem Druckformular zu übergeben. Genau das werden wir als nächstes implementieren.

---

**BEGLEITDATEIEN:** Damit dieser Abschnitt für Sie nicht in ein endloses Getippe ausartet, werden wir es uns mit dem Code für das Drucken einfach machen, und diesen aus einer Textdatei in das Formular hineinkopieren. Sie finden sie deswegen als reine Textdatei im Verzeichnis `.\VB 2005 - Entwicklerbuch\B - IDE\03 - Covers\Druckroutine für Covers.txt`.

---

1. Öffnen Sie die Textdatei `\B - Ein- und Umstieg\Druckroutine für Covers.txt` mit dem Notepad von Windows.
2. Drücken Sie **Strg+A** (alles markieren), **Strg+C** (in die Zwischenablage kopieren).
3. Schließen Sie das Notepad.

4. Wechseln Sie zurück zu Visual Studio.
5. Klicken Sie im Projektmappen-Explorer auf *Form2.vb* und anschließend in der Symbolleiste des Projektmappen-Explorers auf das Symbol *Code anzeigen*.
6. Drücken Sie **Strg+A** (alles markieren), **Strg+V** (Zwischenablageinhalt einfügen). Damit ist der Code für das Druckformular vollständig. Die genaue Funktionalität soll uns an dieser Stelle noch nicht interessieren; es würde bedeuten, zu viele Themen vorwegnehmen zu müssen, deren ausführliche Erklärung für ein genaueres Verständnis erforderlich wäre.
7. Speichern Sie alle Änderungen, und schließen Sie das Dokument *Form2.vb*.

Um den Code zu implementieren, der eine neue Instanz der Klasse *CoverInhalt* bildet, sie mit Daten aus den Eingabefeldern füttert und die Klasse zur Weiterverarbeitung dem Druckformular übergibt, verfahren Sie wie folgt:

1. Wechseln Sie mit **Strg+Tabulator** zum Entwurfsmodus von *Form1.vb* (also zu *Form1.vb [Entwurf]*).
2. Doppelklicken Sie auf die Schaltfläche *Inlay drucken*, um dafür zu sorgen, dass die Stub für die Ereignisbehandlungsroutine von *btnInlayDrucken\_Click()* im Code eingefügt wird.
3. Geben Sie den folgenden Code ein (die Kommentare dienen nur zur Groberklärung des Codes, und müssen natürlich nicht mit eingegeben werden).

```
'Coverinhalt-Klasse in ein Objekt instanzieren
Dim locCoverInhalt As New CoverInhalt

'Dem CoverInhalt-Objekt die Daten zuordnen
locCoverInhalt.FilmTitel = txtNameDesFilms.Text
locCoverInhalt.Schauspieler = txtSchauspieler.Text
locCoverInhalt.Beschreibung = txtKurzbeschreibung.Text
locCoverInhalt.Coverbild = picCoverbild.Image

'Die Druckvorschau aufrufen, und das CoverInhalt-Objekt
'mit den Daten übergeben.
Dim locCoverDruckenForm As New Form2
locCoverDruckenForm.ShowDialogDarstellen(locCoverInhalt)
```

## Code umgestalten (Refactoring)

Visual C# 2005 wurde leider in viel größerem Umfang mit Refactoring-Werkzeugen bedacht als Visual Basic 2005. Im Grunde genommen gibt es in Visual Basic 2005 nur eine einzige Funktion, die das automatische Umgestalten von Code erlaubt – das Umbenennen von Namen von Methoden, Eigenschaften, Objekten oder Ereignissen. Doch dazu später mehr.

Was bedeutet Refactoring genau?

Stellen Sie sich vor, Sie entwickeln eine relativ umfangreiche Funktion, und Sie stellen nach einer Weile fest, dass nur diese eine Funktion bereits aus 1000 Zeilen Code besteht. Sie müssen (oder sollten zumindest) sich dann eingestehen, dass Sie das Problem, das Sie lösen wollen, besser auf mehrere Unterfunktionen verteilt hätten. Aber dazu ist es ja nicht zu spät. Wenn Sie aus einem Teil der Funktion nun eine neue Unterfunktion generieren, dann betreiben Sie bereits aktives Refactoring.

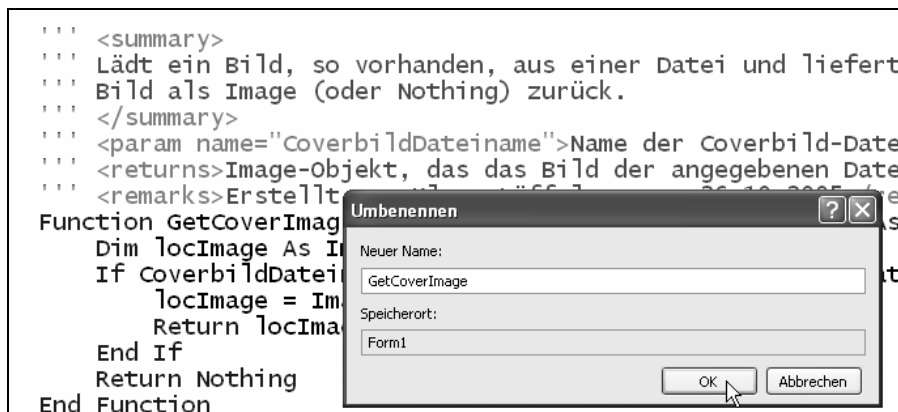
Am schlimmsten ist es bei solchen Aktionen, wenn sich Namen, die Sie beispielsweise Methoden gegeben haben, als falsch oder nicht ausreichend aussagekräftig entpuppen. Gerade wenn Sie im Team arbeiten, sind Sie auf dieses Problem sicherlich schon das ein oder andere Mal gestoßen. Und jeder, der eine Funktion aus solchen Gründen umbenennen musste, weiß, was die Umbenennung für eine Arbeit macht, weil die Funktion doch viel öfter referenziert wird, als man es eigentlich erwartet hätte.

Suchen & Ersetzen kommt für diesen Zweck auch nicht in Frage, denn stellen Sie sich vor: Sie möchten eine Eigenschaft namens »bindControl« in »BoundControl« umbenennen. Es gibt aber auch eine Methode namens »UnbindControl«; an die Sie aber überhaupt nicht mehr denken, und die Sie aus Versehen und ohne es zu merken, ebenfalls umbenennen, nämlich in »UnboundControl«.

Noch ernster wird es bei der Umbenennung von Variablen, die Sie in verschiedenen Gültigkeitsbereichen mehrfach verwenden. Wenn Sie beispielsweise eine Methode in zwei Klassen implementiert haben, dann möchten Sie unter Umständen, dass sich nur der Name der Methode in einer Klasse ändert – und überall dort, wo Sie ihn verwendet haben. Mit Suchen & Ersetzen wäre das fehlerfrei zu tun fast ein Ding der Unmöglichkeit.

Hier kommt das einzige Refactoring-Werkzeug von Visual Basic 2005 ins Spiel – das Umbenennen-Werkzeug. Das Umbenennen von Namen über das Refactoring bezieht sich immer nur auf das Element, das Sie umbenennen, und es benennt nicht nur das Element, sondern auch alle Referenzen um. Um das auszuprobieren, machen Sie Folgendes:

1. Suchen Sie im Code von *Form1.vb* die Funktion *CoverbildAusDateinamen*.
2. Klicken Sie mit der rechten Maustaste auf den Funktionsnamen, und aus dem Kontextmenü, das der Editor nun öffnet, wählen Sie *Umbenennen...*
3. Im Dialog, der jetzt dargestellt wird, geben Sie einen neuen Namen für die Funktion ein – beispielsweise **GetCoverImage**.



**Abbildung 3.41:** Mit dem Umbenennen beispielsweise einer Methode (einer Funktion) ändern Sie nicht nur deren Namen, sondern auch alle ihre Referenzen

4. Klicken Sie auf *OK*.



**HINWEIS:** Bei umfangreichen Projekten mit vielen Formulardateien oder Klassendateien kann sich das Refactoring von Klassen- und Formulklassen durch das Umbenennen ihrer Codedatei als störend erweisen. Sie können das »Dateinamen-Umbenennen-Refactoring« im *Optionen*-Dialog von Visual Studio ausschalten. Rufen Sie diesen Dialog dazu aus dem *Extras*-Menü auf, und wählen Sie das Register *Windows Forms-Designer*. Wie in Abbildung 3.42 zu sehen, setzen Sie die `EnableRefactoringOnRename`-Eigenschaft auf `False`.<sup>11</sup>

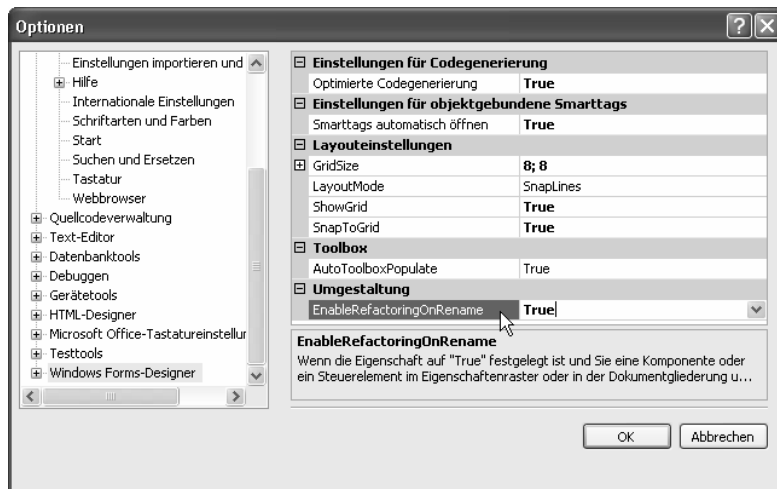
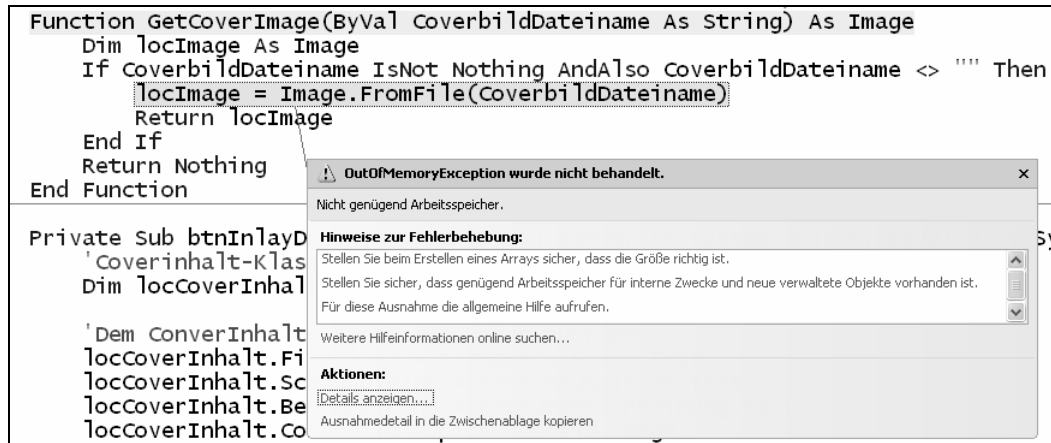


Abbildung 3.42: Hier schalten Sie das Refactoring durch Umbenennen von Codedateien aus

## Die Bibliothek der Codeausschnitte (Code Snippets Library)

Unser Beispielprogramm kann sich bis zu diesem Zeitpunkt schon sehen lassen. Allerdings läuft es noch nicht wirklich fehlerfrei und ist nicht gegen »groben Unfug« geschützt. Warum? Nun, Sie könnten beispielsweise versuchen, statt einer Bilddatei eine Textdatei als Coverbild zu laden. Einen solchen Versuch quittiert das Programm direkt mit einer Ausnahme – einer Laufzeitfehlermeldung –, die es aber nicht abfängt:

<sup>11</sup> Die in VS 2005 enthaltenen Tools zum Refactoring gehen auf die Werkzeuge einer Firma namens Developer Express zurück. Wenn Sie das vollständige Tool zum Refactoring benutzen wollen (auch für VB.NET), können Sie dies unter dem IntelliLink *B0302* als Testversion oder kostenpflichtige Vollversion beziehen.



**Abbildung 3.43:** Das Laden einer kleinen Textdatei führt zu einer »Zu-wenig-Speicher-Ausnahme«? Das lässt Platz für Vermutungen, wird aber wohl an den internen Grafikfiltern liegen, die Textbytes fälschlicherweise als Größenangaben für eine zu ladende Grafik interpretieren. So tritt der Fehler bereits beim Speicherreservieren für die Grafik und nicht erst beim Lesen der »falschen« Bytefolgen auf.

In einem professionellen Programm darf so etwas natürlich nicht passieren – schon gar nicht, wenn die ausgelöste Ausnahme, wie hier im Beispiel, den Anwender auf eine völlig falsche Fährte lockt (siehe Bildunterschrift).

Schön wäre es überdies, wenn dieses Fehlverhalten nicht nur nicht zum Abbruch des Programms führte, sondern den Fehler einer zuständigen Stelle obendrein noch meldete! Zum Beispiel, indem es versucht, eine E-Mail an die E-Mail-Adresse eines Administrators zu schicken.

Ich würde Ihnen gerne erklären, wie das funktioniert. Aber wissen Sie was? Ich kann's nicht. Ich müsste dazu stundenlang recherchieren, und ob ich die Infos zur Programmierung eines solchen Features selbst dann überhaupt finden würde, wäre fraglich... ;-)

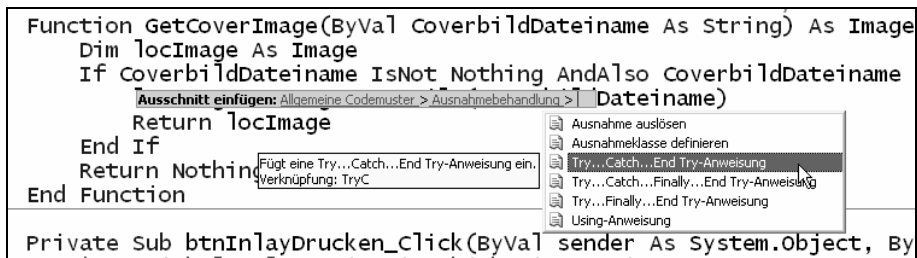
Aber wissen Sie noch was: Das muss ich auch gar nicht. Denn Visual Basic 2005 verfügt über eine Codeausschnittsbibliothek, die für jeden Geschmack etwas Passendes bereitstellt. Zum Beispiel, wie man einen Coderumpf zum Abfangen eines Fehlers implementiert. Und das geht so:

1. Schließen Sie zunächst den Dialog mit der Ausnahme, der immer noch auf dem Bildschirm zu sehen sein sollte.
2. Stoppen Sie das Programm mit einem Klick auf das *Debuggen beenden*-Symbol (der Tooltip des Symbols hilft Ihnen, das richtige zu finden).
3. Bewegen Sie die Schreibmarke in die Methode `GetCoverImage`, und zwar so, dass sie sich genau vor der Zeile
 

```
locImage = Image.FromFile(CoverbildDateiname)
```

 befindet.
4. Rufen Sie mit der rechten Maustaste das Kontextmenü auf, und wählen Sie *Ausschnitt einfügen*.
5. Sie sehen nun eine Liste mit Ordnern, die die verschiedenen Codeausschnitt-Oberbegriffe enthalten. Doppelklicken Sie auf *Allgemeine Codemuster*.

6. Doppelklicken Sie auf *Ausnahmebehandlung*.



**Abbildung 3.44:** Beim Einfügen eines Codeausschnittes (Code Snippet) sehen Sie die verschiedenen Kategorien hierarchisch in blau gefärbt nebeneinander stehen. Ein Tooltip gibt Ihnen genauere Infos zum ausgewählten Ausschnitt. **Achten Sie dabei auch auf die dort ausgewiesene Verknüpfungszeichenfolge für »das nächste Mal«!**

7. In der Liste, die sich daraufhin öffnet, doppelklicken Sie auf *Try...Catch...End Try-Anweisung*.

---

**HINWEIS:** Merken Sie sich am besten dabei gleich die Verknüpfung, die der Tooltip anzeigt, für das nächste Mal, wenn Sie den Codeausschnitt häufiger benötigen. Sie können diese Verknüpfung dann später verwenden, um mit weniger Aufwand den Codeausschnitt einzufügen.

---

Der Editor fügt nun den kompletten Rumpf zum Abfangen eines Fehlers ein, den Sie im Prinzip nur ein wenig umgestalten müssen, etwa wie in den folgenden Codezeilen zu sehen:

```
If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <> "" Then
    Try
        'Versuche das hier ohne Fehler, und...
        locImage = Image.FromFile(CoverbildDateiname)
        '...wenn kein Fehler auftrat, liefere das Ergebnis zurück:
        Return locImage
    Catch ex As Exception
        'Beim auftreten eines Fehlers, landet man hier.
    End Try
End If
```

---

**HINWEIS:** Denken Sie dabei bitte auch daran, *ApplicationException* in *Exception* umzuwandeln, damit nicht nur Ausnahmen vom Typ *ApplicationException*, sondern alle denkbaren Ausnahmen abgefangen werden können. Mehr zu diesem Thema, das insbesondere für VB6-Umsteiger wichtig ist, erfahren Sie in ► Kapitel 5.

---

Mit dieser Codeänderung haben wir den Fehler auf alle Fälle schon mal abgefangen. Jetzt müssen wir im Catch-Block nur noch dafür sorgen, auf ihn auch entsprechend zu reagieren – zum Beispiel in dem wir eine E-Mail an einen zuständigen Administrator versenden.

### Einfügen von Codeausschnitten mithilfe von Verknüpfungen

Auch dafür kommen uns die Codeausschnitte zu Hilfe. In der Kategorienliste *Konnektivität und Netzwerk* findet sich ein Eintrag namens *E-Mail-Nachricht erstellen*, der übrigens die Verknüpfungszeichenfolge *conEmail* trägt. Wenn Sie, wie in diesem Fall, die Verknüpfungszeichenfolge kennen,

brauchen Sie sie lediglich an die Stelle im Code einfügen und Tabulator drücken, an dem der ganze Ausschnitt erscheinen soll. Und so wird ...

```
Function GetCoverImage(ByVal CoverbildDateiname As String) As Image
    Dim locImage As Image
    If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <> "" Then
        Try
            'Versuche das hier ohne Fehler, und...
            locImage = Image.FromFile(CoverbildDateiname)
            '...wenn kein Fehler auftrat, liefere das Ergebnis zurück:
            Return locImage
        Catch ex As Exception
            'Beim auftreten eines Fehlers, landet man hier.
            conEmail
        End Try
    End If
    Return Nothing
End Function
```

... nach dem Drücken von **Tabulator** hinter *conEmail* die Funktion folgendermaßen abgeändert:

```
Function GetCoverImage(ByVal CoverbildDateiname As String) As Image
    Dim locImage As Image
    If CoverbildDateiname IsNot Nothing AndAlso CoverbildDateiname <> "" Then
        Try
            'Versuche das hier ohne Fehler, und...
            locImage = Image.FromFile(CoverbildDateiname)
            '...wenn kein Fehler auftrat, liefere das Ergebnis zurück:
            Return locImage
        Catch ex As Exception
            'Beim Auftreten eines Fehlers, landet man hier.

            Dim message As New MailMessage("absender@adresse", "an@adresse", _
                "Betreff", "Nachrichtentext")
            Dim emailClient As New SmtplibClient("E-Mail-Servername")
            emailClient.Send(message)
        End Try
    End If
    Return Nothing
```

**Abbildung 3.45:** Der eingefügte Codeausschnitt ist hier zu sehen. Mit Tabulator springen Sie bequem von Parameter zu Parameter und ändern diese in einem Rutsch.

Sie brauchen nun nur mit Tabulator von Parameter zu Parameter zu springen, und die entsprechend gültigen Werte einzutragen, bis sich im Catch-Block beispielsweise folgendes Bild ergibt:

```
.
.
.
    Catch ex As ApplicationException
        'Beim Auftreten eines Fehlers, landet man hier.

        Dim message As New MailMessage("covers@loeffelmann.de", "klaus@loeffelmann.de", _
            "Fehler bei der Programmausführung", ex.Message)
        Dim emailClient As New SmtplibClient("192.168.0.1")
        emailClient.Send(message)
```

End Try

---

**HINWEIS:** Damit – und das sei nur der Vollständigkeit halber erwähnt – dieses Beispiel auf Ihrem System laufen kann, müssen Sie natürlich einen entsprechend konfigurierten SMTP-(Mail-)Server im Netzwerk zur Verfügung haben. Tragen Sie dann die für Sie gültigen Daten anstelle der hier im Listing abgedruckten E-Mail-Daten ein, auch die hier angegebene TCP/IP Nummer (192,168,0.1) müssen Sie durch die TCP/IP Nummer oder den Hostnamen (z.B. smtp.web.de) Ihres SMTP Servers ersetzen. Falls Sie auf SMTP-Server zugreifen müssen, die keine offene Relay-Funktion unterstützen,<sup>12</sup> ergänzen Sie im Bedarfsfall noch folgende Zeile (fett im folgenden Listingauszug), mit der Sie die Anmeldeinformationen übergeben.

---

```
Catch ex As Exception
    'Beim Auftreten eines Fehlers, landet man hier.

    Dim message As New MailMessage("covers@loeffelmann.de", "klaus@loeffelmann.de", _
        "Fehler bei der Programmausführung", ex.Message)
    Dim emailClient As New SmtpClient("192.168.0.1")
    emailClient.Credentials = New Net.NetworkCredential("Username", "Passwort")
    emailClient.Send(message)
End Try
```

## Einstellen des Speicherns von Anwendungseinstellungen mit dem Settings-Designer

Viele Anwendungen müssen beim Beenden Einstellungen speichern. Früher war das ein vergleichsweise großer Aufwand, denn Anwendungen mussten sich komplett selbst um die so genannte Serialisierung<sup>13</sup> ihrer Einstellungsdaten kümmern.

Programmierte Methoden innerhalb der Anwendung hatten dafür zu sorgen, die wichtigen Daten so aufzubereiten, dass sie im richtigen Format abgespeichert werden konnten. Das Konvertieren in das richtige Format (beispielsweise numerische Werte in Zeichenketten beim Serialisieren oder Zeichen-

---

<sup>12</sup> Ein Weiterleiten von und an beliebige E-Mail-Adressen ohne Anmeldung, und das sollte bei den meisten SMTP-Servern (hoffentlich!) nicht gegeben sein, es sei denn sie erlauben das Relaying aufgrund Verwendung der integrierten Sicherheit in Active Directory-Netzwerken. In diesen Fällen übernimmt die Anmeldung an das Netzwerk beim Starten von Windows auch das implizite Anmelden am SMTP-Server im Bedarfsfall.

<sup>13</sup> Serialisierung nennt man den Vorgang, bei dem ein Objekt, das innerhalb einer Anwendung Daten im Hauptspeicher speichert, diese durch einen Datenstrom in einen Zielspeicher (seriell) ablegt, entweder zu dem Zweck, die Daten für die Weiterverwendung durch ein Objekt gleicher »Bauart« bereitzustellen oder dauerhaft auf einem Datenträger zu speichern. Beim entgegengesetzten Vorgang entsteht aus einem Datenstrom wieder die ursprüngliche Instanz des Objektes – es entspricht also dem Laden der Daten in den Hauptspeicher.

ketten in numerische Werte beim Deserialisieren) stellte dabei den größten Aufwand dar, weil falsche Typkonvertierungen (Datum liegt als Zeichenkette vor, es wurde aber beispielsweise versucht, die Zeichenkette in eine numerische Variable zu konvertieren) in vergleichsweise großem Aufwand abgefangen und ausgeschlossen werden mussten.

In .NET 2.0 bzw. Visual Studio 2005 geht das ungleich einfacher. Interaktiv können Sie mit einem speziellen Designer Einstellungsvariablen einrichten, auf die Sie dann von Ihrer Anwendung aus zugreifen und diese speziell zum Abspeichern von Anwendungseinstellungen verwenden können. Und das Tolle: Eine Visual Basic-Anwendung kümmert sich automatisch, dass die Inhalte dieser Variablen, wenn Sie es wünschen, beim Programmende gesichert und beim nächsten Programmstart automatisch wieder gestartet werden.

## Einrichten von Settings-Variablen

Der Designer zum Einrichten der Settings-Variablen verbirgt sich in den Projekteigenschaften.

1. Um diesen Designer also aufzurufen, wählen Sie aus dem Kontextmenü des Projekts *Covers* (nicht der Projektmappe!) im Projektmappen-Explorer den Menüpunkt *Eigenschaften*.
2. Wählen Sie die Registerkarte *Einstellungen*, indem Sie auf die entsprechende Registerzunge an der linken Seite klicken.

Für unser Beispielprogramm möchten wir, dass die Eingaben, die der Anwender in den Texteingabefeldern zur Laufzeit vorgenommen hat, zum Programmende gespeichert und, wenn das Programm erneut startet, wieder geladen und in die Eingabefelder geschrieben werden. Unsere erste Settings-Variable nennen wir daher `LetzterFilmtitle`, und sie soll vom Typ `String` sein, da sie Zeichenketten speichert.

3. Klicken Sie, wie in Abbildung 3.46 zu sehen, in die ersten Namens-Zelle der Settings-Tabelle, und geben Sie als Variablennamen `LetzterFilmname` ein. Drücken Sie **Tabulator**.

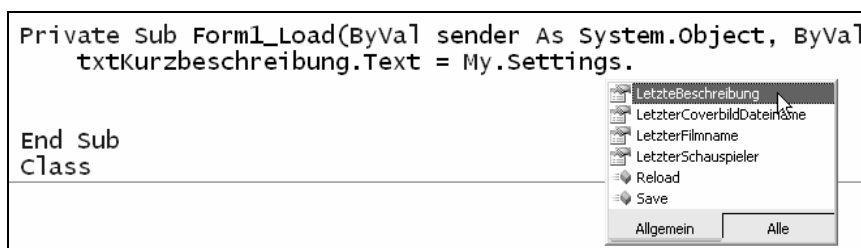


**Abbildung 3.46:** Den Settings-Designer (Einstellungs-Designer) finden Sie in den Projekteigenschaften, die Sie aus dem Kontextmenü des Projektes erreichen

4. In der nächsten Spalte würden Sie in der Aufklappliste den gewünschten Variablentyp auswählen, was Sie in diesem Fall nicht machen müssen, da String für Texteingaben bereits der passende ist. Drücken Sie daher einfach **Tabulator**.
5. Im *Bereich* wählen Sie aus, ob die Settings-Variable schreibgeschützt oder durch das Programm veränderbar sein soll. *Benutzer* wählen Sie, wenn Sie den Variableninhalt zur Laufzeit verändern wollen, und der neue Inhalt nach Programmende auch gesichert werden soll; *Anwendung* wählen Sie, wenn es sich um eine Konstante handeln soll, die nur Sie zur Entwurfszeit einstellen können, die man aber zur Laufzeit nicht verändern darf. Lassen Sie auch hier die Einstellung *Benutzer* so, wie sie ist.
6. Das Feld Wert lassen Sie frei. Hier könnten Sie einen Standardwert einfügen, den die *Settings*-Variable beim ersten Start der Anwendung unter dem Benutzerkonto eines Benutzers haben würde.
7. Drücken Sie **Tabulator**, um in die nächste Zeile zu gelangen, die vom Settings-Designer automatisch angelegt wird.
8. Auf diese Weise legen Sie weitere Variablen (alle vom Typ String und dem Bereich *Benutzer*) namens *LetzterSchauspieler*, *LetzteBeschreibung* und *LetzterCoverbildDateiname* an.
9. Klicken Sie auf das *Ausgewählte Elemente Speichern*-Symbol, um die Änderungen zu übernehmen.

### Verwenden von Settings-Variablen im Code

1. Wechseln Sie nun mit **Strg+Tabulator** zum Entwurfsmodus von *Form1.vb*.
2. Doppelklicken Sie irgendwo ins Formular – am besten unterhalb der drei Schaltflächen, damit Sie nicht versehentlich doch ein anderes Steuerelement erwischen.
3. Platzieren Sie die Schreibmarke zwischen `Private Sub Form1_Load` und `End Sub`.
4. Beginnen Sie zu schreiben:  
`txtKurzbeschreibung.Text = My.Settings.`
5. In dem Moment, in dem Sie den Punkt hinter `My.Settings` getippt haben, wird IntelliSense aktiv und Sie bekommen, etwa wie in Abbildung 3.47 zu sehen, alle Settings-Variablen aufgelistet, die Sie gerade eingerichtet haben.



**Abbildung 3.47:** Den Zugriff auf die Settings-Variablen nehmen Sie über *My.Settings* – IntelliSense hilft Ihnen im Codeeditor anschließend beim Finden der richtigen Settings-Variablen

6. Wählen Sie für diesen Fall aus der Vervollständigungsliste LetzteBeschreibung aus.
7. Ergänzen Sie nach diesem Schema den Code um folgende Zeilen, sodass sich folgender Gesamtcodeblock ergibt:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    txtKurzbeschreibung.Text = My.Settings.LetzteBeschreibung  
    txtNameDesFilms.Text = My.Settings.LetzterFilmname  
    txtSchauspieler.Text = My.Settings.LetzterSchauspieler  
  
    Dim locImage As Image = GetCoverImage(My.Settings.LetzterCoverbildDateiname)  
    picCoverbild.Image = locImage  
End Sub
```

---

**HINWEIS:** Sie werden übrigens feststellen, dass IntelliSense in der Vervollständigungsliste so lange, wie Sie noch keine Buchstaben zur genauen Identifizierung des zu vervollständigenden Wortes eingegeben haben, immer das Element in der Liste markiert, das Sie am häufigsten verwendet haben. Wenn Sie also beispielsweise zum dritten Mal My und anschließend den Punkt eingegeben haben, wird Settings automatisch selektiert.

---

Was macht Form1\_Load nun genau?

Nun, durch den Zusatz `Handles MyBase.Load` wird bestimmt, dass `Form1_Load` dann automatisch aufgerufen wird, wenn das Framework das Formular darstellt – und das ist beim Programmstart der Fall. `Form1_Load` macht dann nichts weiter, als die Programmeinstellungen, die sich bereits in den `Settings`-Variablen befinden, in die Textfelder zu übertragen. Da Bilder übrigens nicht direkt in `Settings`-Variablen gespeichert werden können, bedienen wir uns eines Tricks: Wir speichern einfach den letzten bekannten Dateinamen, und versuchen das Bild beim Programmstart einfach wieder aus der gleichen Quelle zu laden. Da unsere `Coverbild`-Ladefunktion Fehler dabei abfangen kann, können wir uns die Eventualität leisten, dass das Bild nicht mehr an seiner ursprünglichen Stelle zu finden ist.

Damit das Konzept aufgeht, benötigen wir nun noch den umgekehrten Fall: Wenn das Formular geschlossen wird, müssen die Texte in den Textfeldern in die `Settings`-Variablen übertragen werden, damit dafür gesorgt werden kann, dass die Inhalte der `Settings`-Variablen (und damit der Feldinhalte) beim Programmende gesichert werden.

Der beste Zeitpunkt, genau dafür zu sorgen, ist, wenn das Formular im Begriff ist, sich zu schließen. Zu diesem Zeitpunkt sind die Inhalte der Steuerelemente nämlich noch alle erhalten. Dieses Ereignis trägt den Namen `FormClosing`,<sup>14</sup> und die entsprechende Ereignisbehandlungsroutine werden wir im Folgenden implementieren:

1. Wählen Sie mit **Strg+Tabulator** das Formular `Form1.vb` im Entwurfsmodus aus.
2. Klicken Sie auf die Titelzeile des Formulars, um es zu selektieren.
3. Klicken Sie im Eigenschaftfenster auf das Symbol *Ereignisse* (das Blitz-Symbol), um die Ereignisse anzeigen zu lassen.
4. In der Rubrik *Verhalten* finden Sie das `FormClosing`-Ereignis; auf diesen Eintrag doppelklicken Sie nun.

---

<sup>14</sup> Mehr zum Thema »Schließen von Formularen« erfahren Sie in ► Kapitel 27.

- Der Editor hat nun die Stub für das FormClosing-Ereignis eingefügt, das Sie nur noch um die entsprechenden Codezeilen zu ergänzen brauchen:

```
Private Sub Form1_FormClosing(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
    My.Settings.LetzteBeschreibung = txtKurzbeschreibung.Text
    My.Settings.LetzterFilmmame = txtNameDesFilms.Text
    My.Settings.LetzterSchauspieler = txtSchauspieler.Text
    My.Settings.LetzterCoverbildDateiname = myBilddateiname
End Sub
```

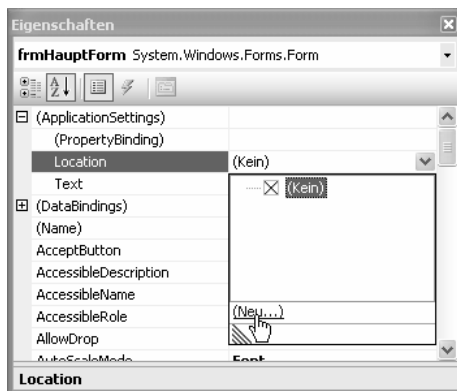
## Verknüpfen von Settings-Werten mit Formular- oder Steuerelementeigenschaften

Settings-Werte können übrigens auch dazu verwendet werden, Eigenschaften von Formularen und/oder deren Steuerelementen zu speichern. Dabei werden die entsprechenden Eigenschaftswerte sogar direkt an die Settings-Werte gebunden. Und was haben Sie davon?

Ein Beispiel: Angenommen Sie möchten, dass zur Laufzeit Ihres Programms beim Öffnen eines Formulars dieses automatisch an der letzten Position dargestellt wird. In diesem Fall könnten Sie Settings-Werte zur Speicherung der Location-Eigenschaft verwenden, die die Position des Formulars bestimmt. Sie müssten innerhalb des Load-Ereignisbehandlers des Formulars die Werte für die entsprechende Eigenschaft aus den Settings lesen, und sie umgekehrt beim Schließen des Formulars in FormClosing wieder in die Settings übernehmen.

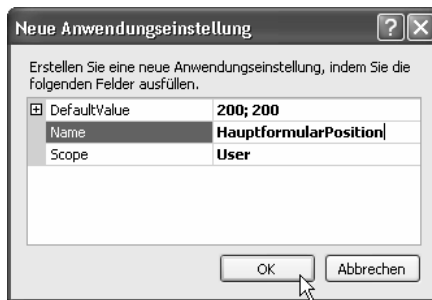
Doch diesen Vorgang können Sie auch automatisieren – Sie binden die entsprechende Eigenschaft einfach an einen Settings-Wert, und das funktioniert folgendermaßen:

- Öffnen Sie Form1 durch Doppelklick auf die entsprechende Klassendatei im Projektmappen-Explorer.
- Klicken Sie auf den Titelbalken des Formulars, um das Formular selbst zu selektieren.
- Suchen Sie im Eigenschaftenfenster (auf das Zurückschalten auf die Eigenschaftenliste achten!) nach dem Eintrag (*ApplicationSettings*), und öffnen Sie den Zweig durch Klick auf das davor stehende +-Symbol.
- Klicken Sie auf den Eintrag *Location*, und öffnen Sie die Aufklappliste, wie in Abbildung 3.48 zu sehen.



**Abbildung 3.48:** An dieser Stelle steuern Sie das Binden von Formulareigenschaften an die Anwendungseinstellungen (Application Settings)

5. In der Liste, die sich nun öffnet, klicken Sie auf (Neu...).
6. Visual Studio öffnet nun einen weiteren Dialog, in dem Sie ohne in die Anwendungseinstellungstabelle wechseln zu müssen, direkt einen neuen Settings-Wert mit dem für die Eigenschaft automatisch richtigem Typ einrichten können. Geben Sie dazu zunächst eine Standardposition für das Formular unter DefaultValue ein (denken Sie daran, die beiden Zahlen mit einem Semikolon und nicht mit einem Komma zu trennen!), und bestimmen Sie ferner den Namen für den Settings-Wert – beispielsweise HauptformularPosition.



**Abbildung 3.49:** An dieser Stelle steuern Sie das Binden von Formulareigenschaften an die Anwendungseinstellungen (Application Settings)

7. Beenden Sie den Dialog mit OK.

Wenn Sie das Programm nun starten, öffnet sich das Hauptformular des Programms beim ersten Mal an Position 200; 200. Verschieben Sie anschließend das Formular und schließen es, dann wird es beim nächsten Start automatisch wieder an der Stelle erscheinen, an der es auch beim letzten Beenden positioniert war.

---

**HINWEIS:** Die Größe des Formulars können Sie leider nicht auf diese Art und Weise an Settings-Werte binden. Die `Size`-Eigenschaft ist aus framework-internen Gründen nämlich nicht an die Anwendungseinstellungen bindbar. Zwar ließe sich die Formulargröße auch indirekt durch die `ClientSize`-Eigenschaft an einen Settings-Wert binden, doch wenn Sie das machen, werden bei der Wiederherstellung der Größe die `Anchor`-Einstellungen der anderen Steuerelemente nicht korrekt berücksichtigt. In diesem Fall haben Sie also nur die Möglichkeit, wie in einem der vorherigen Absätze schon beschrieben, die Eigenschaftenzuweisung für die Größe des Formulars durch `Size` in den Ereignisbehandlungsroutinen `Load` und `FormClosing` manuell zu erledigen.

---

### Und wo werden die Settings-Daten abgelegt?

Im persönlichen Anwendungsdatenverzeichnis unter *Lokale Einstellungen* auf dem Windows-Installationslaufwerk in weiteren Unterordnern in Abhängigkeit von Ihrem Benutzernamen sowie dem in den Assembly-Infos hinterlegten Firmennamen und dort in weiteren Unterverzeichnissen, die sich aus Laufzeitumgebung (Debug- oder Nicht-Debug-Modus), dem Anwendungsnamen und der Anwendungsversion ergeben. Alles klar?

Oder besser, da verständlicher: In meinem Fall lautet das Verzeichnis:

```
C:\Dokumente und Einstellungen\loeffel.ACTIVEDEVELOP\Lokale
Einstellungen\Anwendungsdaten\AndereFirma\Covers.vshost.exe_Ur1_ohvc1ojvckpztoiylkw1hs3ba2acq2v4i\1.0.0.0
```

Warum?

- Mein Anmeldename in unserem Active Directory-Netzwerk lautet *loeffel*. Unsere Domäne *Active-Develop* (mein Geburtsdatum ist übrigens der 24.7.69, aber Sie werden sich mit diesem Wissen dennoch nicht bei uns anmelden können ...
- Als Firmennamen habe ich im Assembly-Infodialog *Andere Firma* eingegeben. Diesen Dialog erreichen Sie, indem Sie aus dem Kontextmenü des Projektes im Projektmappen-Explorer *Eigenschaften* aufrufen. Wählen Sie das Register *Anwendung* (das ist die vorgewählte Eigenschaftenseite), und klicken Sie auf die Schaltfläche *Assemblyinformationen*. Das vorvorletzte Verzeichnis im Pfad entsteht aus dem Namen, den Sie unter *Firma* bestimmt haben.
- Das nächste Verzeichnis wird durch den Programmnamen bestimmt. Es entsteht aus dem Laufzeitprogrammnamen, der variieren kann. Wenn Sie das Programm im Debug-Modus starten, ist es nämlich nicht das Programm selbst, das gestartet wird, sondern ein Host-Prozess, der dafür sorgt, dass Sie Programmcode auch während des Debuggens verändern können<sup>15</sup> und das dafür sorgt, dass die Geschwindigkeit beim Debuggen einigermaßen erträglich bleibt. Dieser Hostprozess hat als Anwendungsnamen eine Kombination aus eigentlichem Anwendungsnamen (*Covers*) und *.vshost.exe*. Diesem Block wird die Zeichenkette *Url\_* sowie eine weitere Kennung angehängt – deren genaue Bedeutung beim Entstehen dieser Zeilen noch nicht zu ermitteln war.
- Schließlich folgt die Versionsnummer des Programms als weiteres Unterverzeichnis, in dem sich schließlich die *user.config*-Datei befindet – eine XML-Datei die die eigentlichen Einstellungen speichert, wie im folgenden Beispiellisting zu sehen:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userSettings>
    <Covers.My.MySettings>
      <setting name="LetzterFilmname" serializeAs="String">
        <value>Terminator III - Rise of the Machines</value>
      </setting>
      <setting name="LetzterSchauspieler" serializeAs="String">
        <value>Arnold Schwarzenegger, Kristanna Loken, Nick Stahl, Claire Danes</value>
      </setting>
      <setting name="LetzteBeschreibung" serializeAs="String">
        <value>Kristanna versucht einen Kranwagen einzuparken, was nicht wirklich klappt. Arnie
          hilft nach, legt ihn aber dann aufs Dach.</value>
      </setting>
      <setting name="LetzterCoverbildDateiname" serializeAs="String">
        <value>C:\Dokumente und Einstellungen\All Users\Dokumente\
          Eigene Bilder\Beispielbilder\Arnie.jpg</value>
      </setting>
      <setting name="PrintFormPosition" serializeAs="String">
        <value>303, 123</value>
      </setting>
    </Covers.My.MySettings>
  </userSettings>
</configuration>
```

<sup>15</sup> Dabei handelt es sich übrigens um das vieldiskutierte Edit & Continue-Feature, das es seit Visual Basic .NET 2002 nicht mehr und erst mit Visual Basic 2005 wieder gibt.

---

**HINWEIS:** Benutzerdaten und Anwendungsdaten, die aus Settings-Einstellungen hervorgehen, werden übrigens nicht in den gleichen Konfigurationsdateien gespeichert. Wenn Sie im Settings-Designer unter *Bereich* den Eintrag *Anwendung* gewählt und damit eine Nur-Lesen-Settings-Eigenschaft bestimmt haben, die für alle Benutzer gilt, werden diese Einstellungen in der Datei *appname.exe.config* gespeichert – wobei dabei *appname* dem Namen Ihrer Anwendung Ihres Projektes entspricht. Diese Datei befindet sich wiederum im *bin*-Unterverzeichnis des Verzeichnisses, das dem Namen der Konfiguration für die Erstellung des Projektes entspricht. Standardmäßig gibt es die Konfigurationseinstellungen *Debug* und *Release* – die sich in ihren Parametern zunächst nicht unterscheiden, außer, dass das Kompilat durch die unterschiedlichen Konfigurationsnamen auch in unterschiedlichen Unterverzeichnissen abgespeichert wird.

---

Haben Sie also beispielsweise Ihr Projekt im Hauptverzeichnis von Laufwerk »D:« unter dem Namen *Covers* erstellt, finden Sie die ausführbaren Dateien (und auch die *appname.exe.config*) im Verzeichnis *d:\covers\*. Mehr zu den Konfigurationseinstellungen finden Sie im anschließenden grauen Kasten.

## Über die Konfigurationseinstellungen »Debug« und »Release« sowie die Geschwindigkeiten der Codeausführung

Gerade Entwickler, die ihre ersten Gehversuche mit der Visual Studio-IDE absolvieren, neigen anfangs dazu, das Konfigurationsmanagement von Projekt-Compilerinstellungen zu missverstehen. Sie stellen oft fest, dass die Geschwindigkeit der Codeausführung in der *Debug*-Konfigurationseinstellung wohl nicht der echten entsprechen kann, und sie sind dann enttäuscht, wenn auch das Umstellen auf *Release* keine Geschwindigkeitsrekorde erzielt.

Aber das kann es auch gar nicht. Denn sie haben nur die vordefinierten Konfigurationseinstellungen von *Debug* auf *Release* umgestellt, und abgesehen davon, dass sich diese Einstellungen von vorne herein 1. überhaupt nicht unterscheiden (außer durch den Namen) und 2. genauso gut auch »Margarine« und »Plattenspieler« heißen könnten, dient die Konfigurationsumschaltung auch gar nicht dazu, die Ausführungsgeschwindigkeit in irgendeiner Form zu beeinflussen. Die Konfigurationsnamen *Debug* und *Release* implizieren das allerdings auf unglückliche Weise.

Wenn Sie wissen wollen, wie schnell ihr Programm später beim Kunden wirklich zu laufen in der Lage ist, dann starten Sie es über das Menü *Debuggen* und den Befehl *Starten ohne Debuggen* – oder drücken Sie einfach die Tastenkombination **Strg+F5**.

Mit den Konfigurationseinstellungen, die Sie übrigens tatsächlich um die Einstellungen »Plattenspieler« oder »Margarine« ergänzen könnten, legen Sie hingegen beispielsweise fest, auf welche Plattformen das Kompilat abzielen soll (x86, egal welcher Prozessor – »Any«, etc.), welche Projekte innerhalb einer Projektmappe kompiliert werden sollen oder nicht und Ähnliches. Und im Übrigen auch, in welchen Unterverzeichnissen des Projektes die ausführbaren Dateien generiert werden.

---

**WICHTIG:** Sobald Sie ein Programm im Debug-Modus (also mit **F5**) starten, wird es immer langsamer laufen, als wenn Sie es ohne zu debuggen starten (mit **Strg+F5**).

---

## Herzlichen Glückwunsch!

Sie haben soeben Ihr erstes funktionsfähiges (und wie ich finde auch recht brauchbares) Windows-Framework-2.0-Programm fertig gestellt. Und nun: Viel Spaß beim Arbeiten mit Covers. Testen Sie es! Produzieren Sie damit die Hüllen Ihrer Urlaubsvideos. Drucken Sie bis der Drucker qualmt!

## Weitere Funktionen des Codeeditors

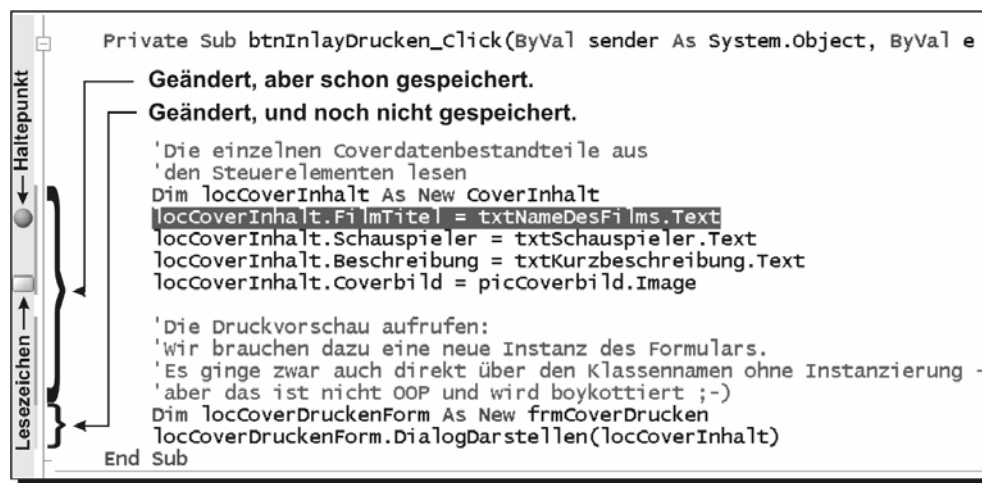
Nun hat Ihnen das Durcharbeiten des Beispiels schon viele der Funktionen des Editors nahe gebracht – nur leider nicht alle. Einige, von denen ich meine, dass Sie Ihnen beim Entwickeln von Projekten ebenfalls gute Dienste leisten können, finden Sie in den folgenden Abschnitten beschrieben.

### Aufbau des Codefensters

Das Codefenster ist in drei vertikale Bereiche aufgeteilt; von links nach rechts gesehen sind das der so genannte **Indikatorrand** (der graue Bereich an der äußersten linken Seite), der **Auswahlrand** (die weiße, recht schmale Spalte links daneben) sowie der eigentliche **Codebereich**, der den Programmtext enthält.

Der Indikatorrand dient dazu, Haltepunkte, Lesezeichen oder Verknüpfungen aufzunehmen. Möchten Sie beispielsweise, dass Ihr Programm zu Testzwecken an einer bestimmten Programmzeile unterbrochen wird, setzen Sie mit **F9** einen Haltepunkt in der Zeile, vor der dann anschließend im Indikatorrand ein roter Haltepunkt zu sehen ist.

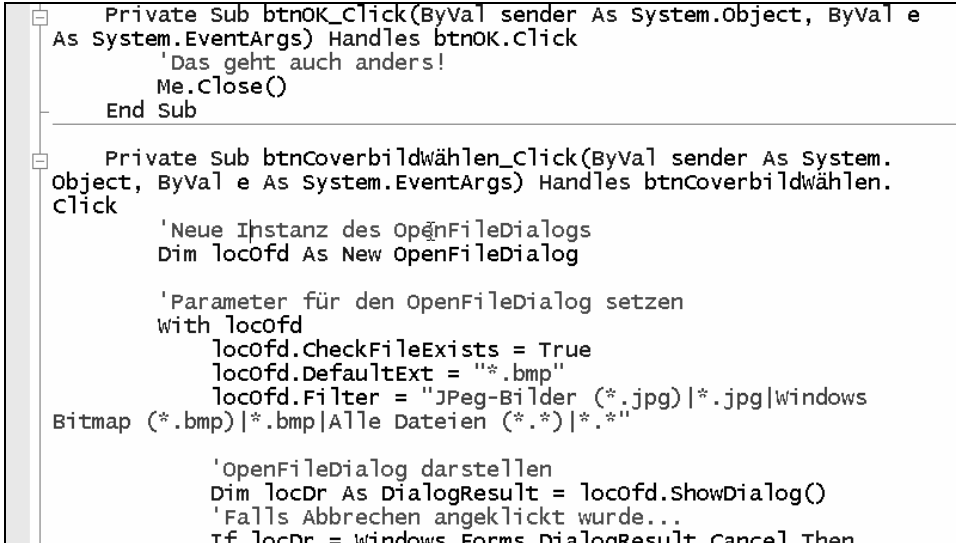
Sie können im Indikatorrand ebenfalls erkennen, welche Änderungen Sie seit dem Öffnen einer Codedatei am Code durchgeführt haben, und welche dieser Änderungen wiederum schon gespeichert wurden. Diese Codezustandsanzeige wird durch entsprechende Balken im Indikatorrand hervorgehoben. Abbildung 3.50 verdeutlicht die Funktionsweise des Indikatorrands.



**Abbildung 3.50:** Der Indikatorrand hält Sie über den Speicher- und Änderungszustand einer Codedatei auf dem Laufenden und zeigt Elemente wie Lesezeichen oder Haltepunkte

## Automatischen Zeilenumbruch aktivieren/deaktivieren

Mit der Tastenfolge (Achtung: Sie drücken die beiden Tastenkombinationen *nacheinander*) **Strg+E**, **Strg+W** können Sie Zeilen des Codes, die nicht in den sichtbaren Bereich passen, automatisch umbrechen lassen (siehe Abbildung 3.51).



```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
    'Das geht auch anders!
    Me.Close()
End Sub

Private Sub btnCoverbildwählen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCoverbildwählen.Click
    'Neue Instanz des openFileDialog
    Dim locOfd As New OpenFileDialog

    'Parameter für den openFileDialog setzen
    with locOfd
        locOfd.CheckFileExists = True
        locOfd.DefaultExt = "*.bmp"
        locOfd.Filter = "Jpeg-Bilder (*.jpg)|*.jpg|windows Bitmap (*.bmp)|*.bmp|Alle Dateien (*.*)|*.*"

    'OpenFileDialog darstellen
    Dim locDr As DialogResult = locOfd.ShowDialog()
    'Falls Abbrechen angeklickt wurde...
    If locDr = Windows.Forms.DialogResult.Cancel Then
```

Abbildung 3.51: Mit dem automatischen Zeilenumbruch werden auch lange Zeilen auf einen Blick erkennbar

Doch aufgepasst: Eine derart umbrochene Zeile entspricht nicht dem Codezeilenumbruch von Visual Basic mit dem »\_«-Zeichen am Zeilenende. Die Gefahr ist groß, eine durch den Editor umbrochene Zeile mit Tabulatoren oder Leerzeichen bündig zu formatieren – Sie würden dadurch aber Leerzeilen in die eigentliche Codezeile einfügen. Sie schalten mit der gleichen Tastenkombination den automatischen Zeilenumbruch auch wieder aus.

## Navigieren zu vorherigen Bearbeitungspositionen im Code

Um schnell zu einer vorherigen Bearbeitungsposition zu gelangen, können Sie die Navigationsschaltfläche der Symbolleiste verwenden, oder Sie verwenden alternativ die Tasten **Strg + -**, um rückwärts bzw. **Strg + Umschalt + -**, um vorwärts zu navigieren.

## Rechteckige Textmarkierung

Wenn Sie die Taste **Alt** auf der Tastatur gedrückt halten, können Sie mit dem Mauszeiger einen rechteckigen Textausschnitt markieren, etwa wie in Abbildung 3.52 zu sehen.

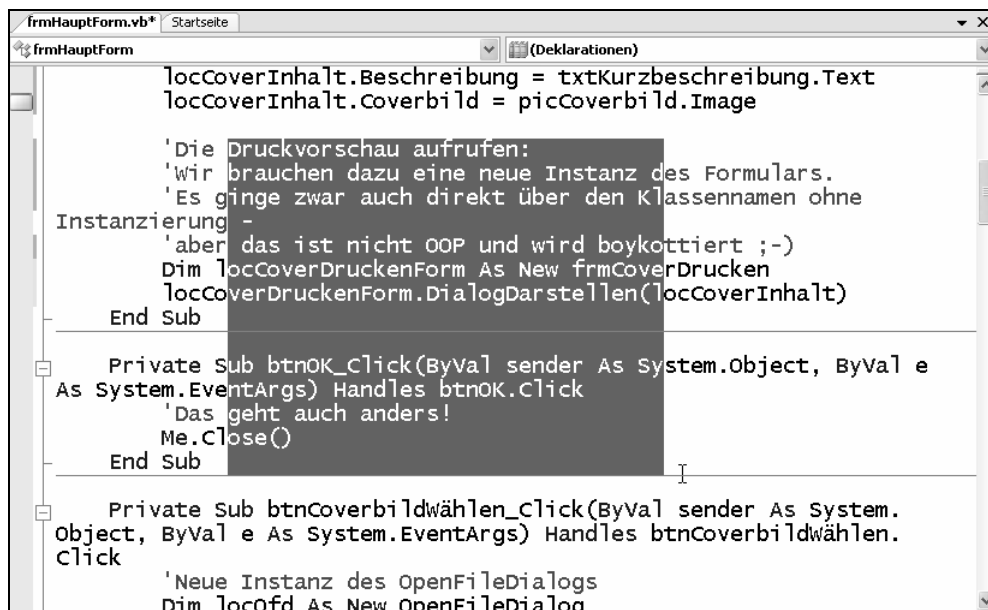


Abbildung 3.52: Der Editor von Visual Studio .NET erlaubt die rechteckige Ausschnittsmarkierung von Text bei gedrückter Alt-Taste

## Gliederungsansicht

Der Codeeditor erlaubt Ihnen, bestimmte Codeteile ein- und auszublenden. Standardmäßig sind in Visual Basic .NET Prozeduren (Sub, Function), Klassen und Eigenschaften mit einem kleinen Gliederungszeichen versehen – mit einem Mausklick auf dieses Plus-Zeichen (siehe Abbildung 3.53) können Sie den Code ausblenden.

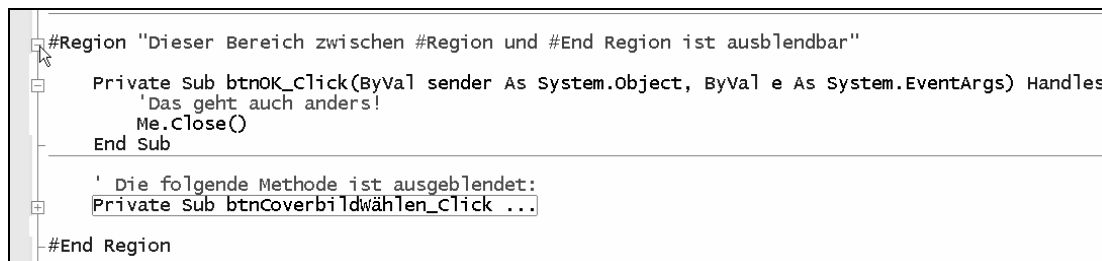


Abbildung 3.53: Dieser Codeabschnitt demonstriert den Einsatz der Gliederungsfunktion im Codeeditor von Visual Basic .NET

Mit den Funktionen im Menü *Bearbeiten/Gliederung* können Sie die Gliederungsansicht steuern.

Möchten Sie Teile des Quellcodes ausblenden, die weniger Codezeilen als eines der Standardelemente umfassen, dann markieren Sie den Codeteil, den Sie ausblenden wollen und wählen aus dem Menü *Bearbeiten* den Befehl *Gliedern/Aktuelles Element umschalten*.

Codeteile, die sich über mehrere Objekte erstrecken, können Sie auch mit den Direktiven

```
#Region
```

und

```
#End Region
```

(ebenfalls in Abbildung 3.53 zu sehen) gliedern.

## Suchen und Ersetzen, Suche in Dateien

Sie erreichen die Suchen- bzw. die Ersetzenfunktion, indem Sie aus dem Menü *Bearbeiten* den Menüpunkt *Suchen und Ersetzen* abrufen. Hier öffnet sich ein weiteres Menü, das verschiedene Suchoptionen zur Verfügung stellt.

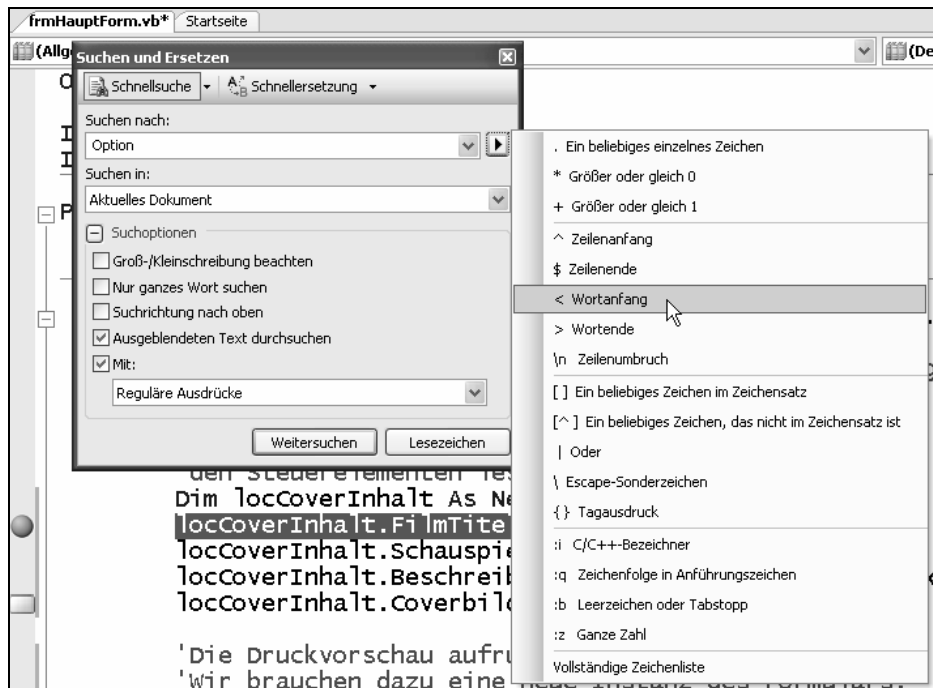
Mit der Schnellsuchfunktion finden Sie in der aktuellen Datei bzw. in allen geöffneten Dateien einen beliebigen Suchtext. Diese Funktion arbeitet bei Bedarf auch mit regulären Ausdrücken; Sie können den zu durchsuchenden Bereich genauer spezifizieren; Sie können nach Sonderzeichen suchen und haben sogar die Möglichkeit, alle Stellen, an denen ihr Suchbegriff vorkommt, durch Lesezeichen zu markieren (siehe auch Abbildung 3.54).

---

**HINWEIS:** Reguläre Ausdrücke sind leistungsfähige Werkzeuge – allerdings braucht man ein wenig Einarbeitungszeit, um sie nutzen zu können. Im ► Kapitel 21 über reguläre Ausdrücke finden Sie ausführliche Beschreibungen zu diesem Thema. Auch wenn Sie nicht mit regulären Ausdrücken programmieren werden – allein für die reine Anwendung in Visual Studio (oder auch in Word 2003, das diese Funktion ebenfalls bietet) lohnt sich die Lektüre dieses Kapitels.

---

Das Schnellersetzen erlaubt es nicht nur, nach einem Begriff zu suchen, sondern diesen auch durch einen anderen Begriff zu ersetzen. Auch hier haben Sie die Möglichkeit, mit regulären Ausdrücken zu arbeiten.



**Abbildung 3.54:** Mit der Suchfunktion können Sie nach regulären Ausdrücken suchen und alle Stellen, an denen der Suchbegriff vorkam, durch Lesezeichen markieren lassen

## Suchen in Dateien



**Abbildung 3.55:** Mit der Suche in Dateien finden Sie alle Vorkommnisse eines Suchbegriffs in den Codedateien eines Projektes oder einer gesamten Projektmappe ...

Die Schnellsuchenfunktion wird nur auf die aktuelle Datei oder alle geöffneten Dateien angewendet. Wenn Sie eine Referenzliste aller Dateien innerhalb eines Projektes oder einer Projektmappe erhalten wollen, die einen bestimmten Suchbegriff beinhalten, bedienen Sie sich der Suche in Dateien.

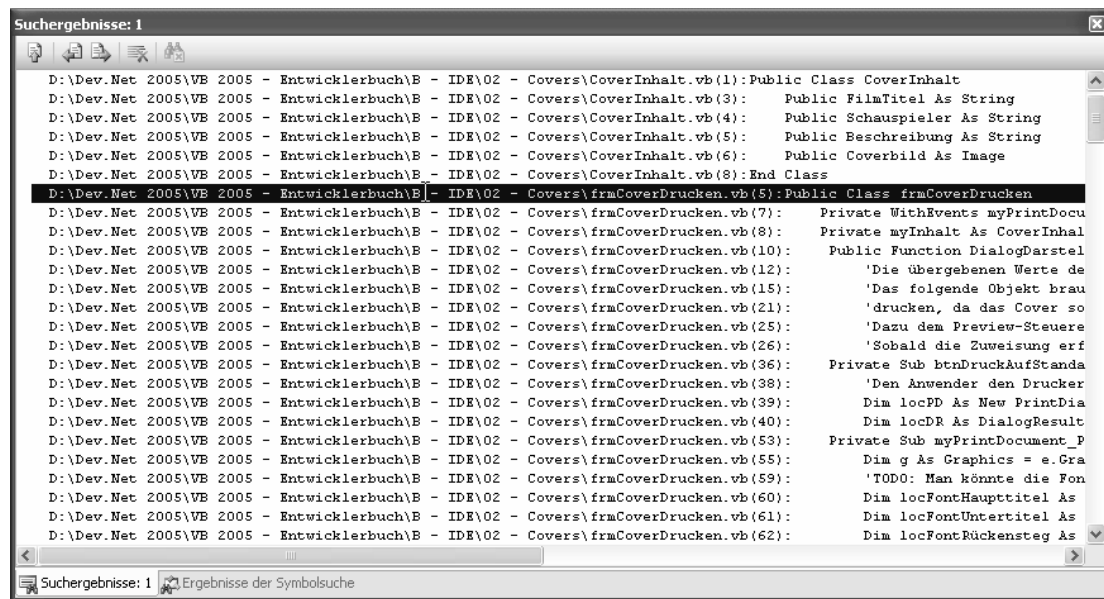


Abbildung 3.56: ... die dann in einer Dateiliste angezeigt werden. Ein Doppelklick auf eine Zeile der Ergebnisliste öffnet dann die Datei im Editor und positioniert den Cursor auf dem gesuchten Begriff.

Die Abbildungsunterschriften der beiden Abbildungen veranschaulichen die Funktionsweise der Suche in den Dateien.

## Inkrementelles Suchen

Mit der inkrementellen Suche brauchen Sie den Suchen-Dialog erst gar nicht zu bemühen. Durch die Tastenkombination **Strg+I** schalten Sie die inkrementelle Suche ein. Beginnen Sie anschließend direkt, die gesuchte Zeichenfolge einzutippen – währenddessen springt die Einfügemarke automatisch an die erste Stelle, die dem bis dorthin eingetippten Suchtext entspricht. Den bis dahin eingegebenen Suchbegriff zeigt die Visual Studio-IDE in der Statuszeile an.

Mit erneutem Drücken der Tastenkombination **Strg+I** finden Sie die nächste Stelle, die dem bisher eingegebenen Suchbegriff entspricht. Mit **Esc** beenden Sie die inkrementelle Suche.

## Gehe zu Zeilennummern

Möchten Sie direkt zu einer bestimmten Zeile springen, deren Zeilennummer Ihnen bekannt ist, wählen Sie aus dem Menü *Bearbeiten* den Menüpunkt *Gehe zu*, oder Sie drücken einfach die Tastenkombination **Strg+G**. Im Dialog, der jetzt erscheint, tippen Sie die Nummer der Zeile ein, zu der Sie die Einfügemarke bewegen wollen.



**Abbildung 3.57:** Mit diesem Dialog gelangen Sie zu jeder Zeile durch Eingabe der Zeilennummer – die aktuelle Zeilennummer wird vorgegeben. Zeilennummern brauchen nicht eingeschaltet zu sein.

## Lesezeichen

Lesezeichen ermöglichen Ihnen, sich bestimmte Stellen zu merken, die Sie später noch überprüfen und bearbeiten wollen. Um ein Lesezeichen in einer Zeile zu setzen oder ein vorhandenes wieder zu löschen, verwenden Sie die Tastenreihenfolge **Strg+K, Strg+K**. Ein gesetztes Lesezeichen wird im Indikatorrand des Codefensters durch eine kleine blaue Marke angezeigt. Mit **Strg+K, Strg+N** verschieben Sie die Einfügemarke zum nächsten Lesezeichen, mit **Strg+K, Strg+P** zum vorherigen. Mit **Strg+K, Strg+L** löschen Sie alle Lesezeichen.

---

**WICHTIG:** Das Löschen geschieht ohne weiteres Nachfragen – seien Sie also vorsichtig mit dieser Tastenkombination!

---

Alternativ können Sie alle beschriebenen Funktionen auch aus dem Menü *Textmarken* aufrufen, das Sie im Menü *Bearbeiten* finden.

Sie können übrigens, wenn Sie mit der Suchfunktion nach Begriffen in Ihren Dateien suchen, alle Stellen, an denen der Suchbegriff vorkommt, mit Lesezeichen markieren. Dazu wählen Sie im Schnellsuchendialog einfach die Schaltfläche *Lesezeichen*.

---

**TIPP:** Alternativ zu Lesezeichen können Sie auch eine Zeile für die Anzeige zur Nachbearbeitung in der Aufgabenliste markieren. Zu diesem Zweck fahren Sie mit dem Cursor in die entsprechende Zeile, und wählen aus den Menüs *Bearbeiten/Textmarken* die Funktion *Verknüpfung für Aufgabenliste hinzufügen* aus. Im Indikatorrand wird anschließend ein entsprechendes Symbol aufgerufen. Um eine solche Kennzeichnung wieder zu löschen, rufen Sie die gleiche Funktion abermals auf.

---

