

4 Tipps & Tricks für das angenehme Entwickeln zuhause und unterwegs

-
- 109 **Der Einsatz mehrerer Monitore**
 - 112 **Zurücksetzen der Fenstereinstellungen**
 - 112 **Sichern, Wiederherstellen oder Zurücksetzen aller Visual Studio-Einstellungen**
 - 116 **Wieviel Arbeitsspeicher darf's denn sein?**
 - 117 **Testen Ihrer Software unterwegs und zuhause – Microsoft Virtual PC und Microsoft Virtual Server**
 - 122 **Hilfe zur Selbsthilfe**
 - 127 **Erweitern Sie die Codeausschnittsbibliothek um eigene Codeausschnitte**
-

So Sie das letzte Kapitel durchgearbeitet haben, wissen Sie, wie sehr Ihnen Visual Studio 2005 die tägliche Entwicklungsarbeit erleichtern kann. Doch es kann nicht alles. Sie können mit ein wenig Investition die Ergonomie Ihres Arbeitsplatzes enorm verbessern und Ihr eigenes Wohlbefinden und damit nicht zuletzt auch Ihren täglichen Output steigern.

Auch einiges Wissenswertes zur IDE, was im vorherigen Kapitel nicht berücksichtigt wurde, trägt dazu bei.

Der Einsatz mehrerer Monitore

Die wohl beste Errungenschaft von Windows 98 – die Älteren unter Ihnen werden sich an dieses Betriebssystem noch erinnern – war seine Möglichkeit, den Windows-Desktop auf mehrere Monitore zu erweitern – entsprechende Hardware vorausgesetzt. Für mich war das »damals« zu meinen Visual Basic 6.0-Zeiten der einzige Grund, nicht auf Windows NT4 (damals mit dem neu erschienenen Internet-Explorer 4.0 und der entsprechenden Desktoperweiterung, die das generelle Feeling von Windows 98 brachte) zu wechseln – denn NT4 sah dieses Feature wegen des anderen Treibermodells zu diesem Zeitpunkt noch nicht vor.

Unter Visual Basic 6.0 war das auch nicht unbedingt notwendig. Sah man sich schon damals in der finanziellen Lage, einen Monitor mit 1280 × 1024 Punkten Auflösungsfähigkeit zu erwerben, und

hatte man auch die entsprechende Grafikkarte zur Verfügung, deren RAMDAC diese Auflösung mit mehr als flimmernden 60 Herz darstellen konnte, gab's unter VB6 keine Probleme.

Heute ist das anders. 1024 × 768 ist meiner Meinung für ein wirklich produktives Arbeiten mit Visual Studio 2005 ohnehin zu wenig – ich denke, darüber brauchen wir uns gar nicht zu unterhalten. 1280 × 1024 ist in meinen Augen buchstäblich die Obergrenze, 1600 × 1200 macht erst bei deutlich mehr als 20 Zoll TFT Sinn, will man nicht nur 10 Zentimeter entfernt vor dem Monitor kleben.

Aber es geht auch besser, flexibler und billiger – mit mehreren Monitoren an einem Rechner. Wenn Ihr Arbeitsplatzrechner nicht deutlich älter als 3 Jahre ist, könnten Sie nämlich das Glück haben, dass Ihre Grafikkarte bereits über zwei Anschlüsse für Monitore verfügt. Und dann beschaffen Sie sich für wenig Geld einfach einen zweiten, z.B. 15"-Monitor, der nur für die Aufnahme der Toolfenster dient. Auf dem linken Hauptmonitor platzieren Sie nur Toolbox und die Dokumentenregisterkartengruppe, und Sie ersparen sich damit ein lästiges ständiges Umpositionieren bzw. Auf- und Zuklappen der Toolfenster.

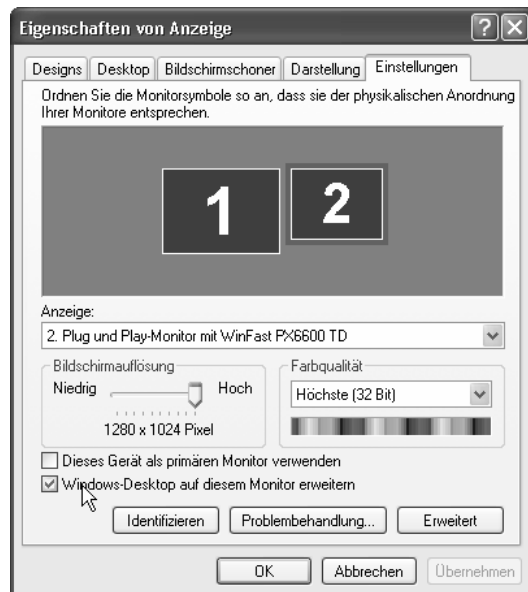


Abbildung 4.1: Mit dem Dialog *Eigenschaften von Anzeige* erweitern Sie auf der entsprechenden Registerkarte den Windows-Desktop auf mehrere Monitore

Zwei Grafikkarten in einem Rechner?

Das geht auch, für den Fall, dass Ihre primäre Grafikkarte nur über einen Monitorausgang verfügt. Allerdings sollten Sie darauf achten, nicht neuere Modelle gleicher Hersteller zu mischen, da diese oft über Treiber mit gleichem Namen verfügen, die sich deswegen dann ins Gehege kommen. Und: Eine Karte die Sie dazu kaufen, sollte dann auch eine PCI-Karte sein, weil in den meisten Rechnern nur ein AGP-Slot bzw. ein PCI-Express-Slot zu finden ist. PCI-Karten bekommen Sie selbst heutzutage noch bei jedem größerem EDV-Zubehör-Versandhändler.

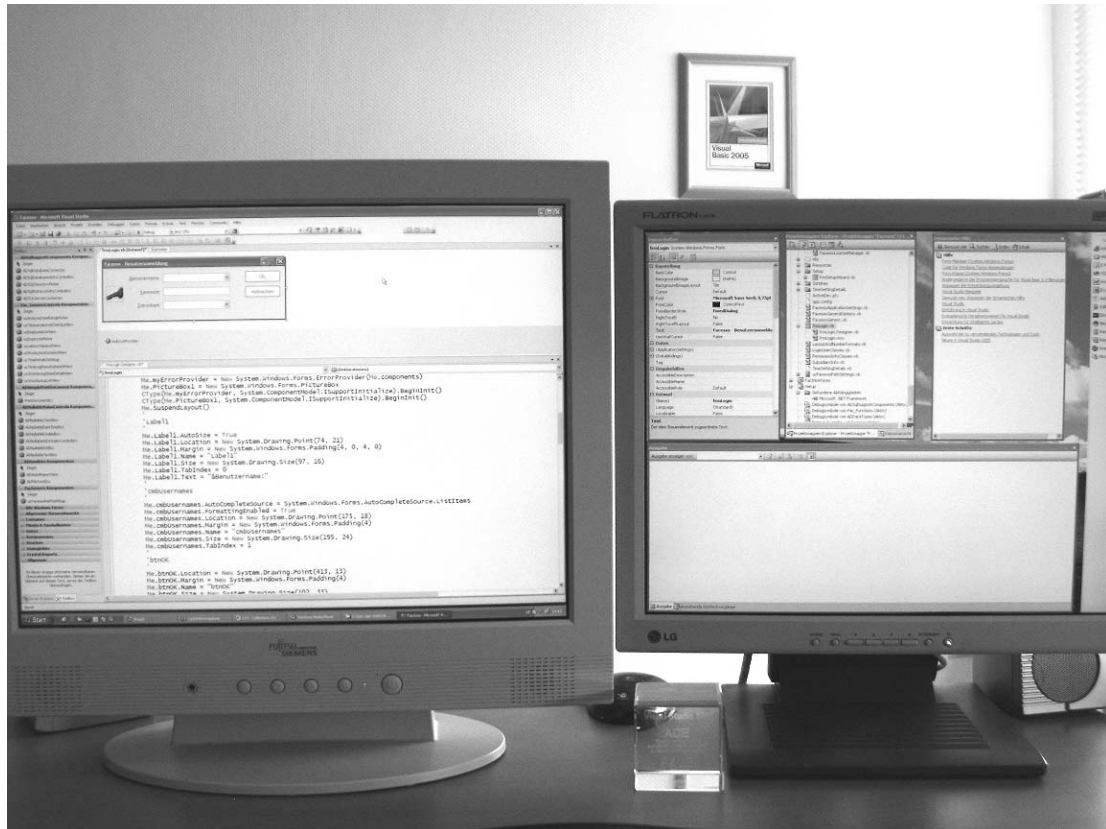


Abbildung 4.2: Die Visual Studio-Oberfläche verteilt auf zwei Monitoren – das lässt richtig Platz für kreative Ergüsse und spart viel Zeit!

Und die Bildschirmdarstellung auf Notebooks?

Auch hier gilt: Je mehr Auflösung, desto besser. Allerdings sollte das Display des Notebooks selbst entsprechend groß sein. An ein längeres konzentriertes Arbeiten ist mit einer 1600 x 1200er Auflösung auf einem 15“-Display nicht zu denken – auch wenn Sie auf beiden Augen volle Sehkraft haben. Günstige 17-Zöller befinden sich inzwischen im Handel, und als ideal ausgeglichen zwischen »hoher Auflösung« und »noch zu erkennen« würde ich eine Wide-Screen-Auflösung mit 1650 x 1050 Pixel empfehlen.

Modernere Notebooks bieten aber auch wie neuere Grafikkarten die Möglichkeit, externe Monitore anzuschließen. Sollten Sie also ein größeres Projekt bei einem Kunden vor Ort erledigen müssen, böte sich ein zweites preiswertes TFT-Display mehr als an. Und da 15-Zoll TFTs auch nicht allzu viel wiegen, sind sie für längere »Draußen«-Einsätze allemal geeignet.

TIPP: Falls Sie die Entscheidung treffen müssen, ein neues Notebook auch für Vor-Ort-Entwicklungseinsätze zu erwerben, sollten Sie auf eine schnelle Grafikkarte Wert legen, die vor allen Dingen über einen eigenen Grafikspeicher verfügt.

Abgesehen davon, dass »Shared Graphics«-Systeme sich aus dem immer knappen Arbeitsspeicher Ihres Rechners bedienen (aus 512 MByte werden so ruckzuck 486 MByte Speicher, die Ihnen nur noch zur Verfügung stehen), sind diese auch spürbar langsamer. Und da es sich bei Visual Studio schon um eine recht grafikintensive Anwendung handelt, wäre ein Sparen an der Grafikausstattung ein Sparen an der falschen Stelle.

Zurücksetzen der Fenstereinstellungen

In diesem Zusammenhang: Wenn Sie sich, ob mit einem, zwei oder drei Monitoren, durch Einblenden, Verschieben, Andocken und Entdocken von Toolfenstern ins Fensterchaos gestürzt haben – es gibt immer wieder einen einfachen und schnellen Weg zurück zur ursprünglichen Fensteranordnung: Wählen Sie einfach aus dem Menü *Fenster* den Befehl *Fensterlayout zurücksetzen*, und Sie finden anschließend sämtliche Fenstereinstellungen so vor, wie Sie sie auch direkt nach der Installation von Visual Studio vorgefunden haben.

Sichern, Wiederherstellen oder Zurücksetzen aller Visual Studio-Einstellungen

Nachdem Sie sich zum ersten Mal Ihre persönliche IDE-Umgebung so eingerichtet haben, wie Sie sie wirklich haben wollten, wissen Sie, wie viel Aufwand das ist. Visual Studio bietet Ihnen eine einfache Möglichkeit, alle Einstellungen von Visual Studio, die Sie einmal vorgenommen haben, in einer Datei zu speichern, sodass Sie sie beispielsweise auch auf andere Computer übertragen können oder einen Entwicklungsrechner, den Sie neu installieren mussten, schnell wieder in seinen VS-Ausgangszustand zurückversetzen können.

Sie können alle Visual Studio-Einstellungen aber auch in den Ausgangszustand zurückversetzen, die das Programm direkt nach der Installation aufwies.

Sichern der Visual Studio Einstellungen

Um die Visual Studio-Einstellungen zu sichern, damit Sie sie auf andere Rechner übertragen oder für die Wiederherstellung desselben Rechners verwenden können, verfahren Sie folgendermaßen:

- Wählen Sie aus dem Menü *Extras* den Menüpunkt *Einstellungen importieren und exportieren*.
- Visual Studio zeigt Ihnen nun einen Assistenten, etwa wie in Abbildung 4.3 zu sehen.

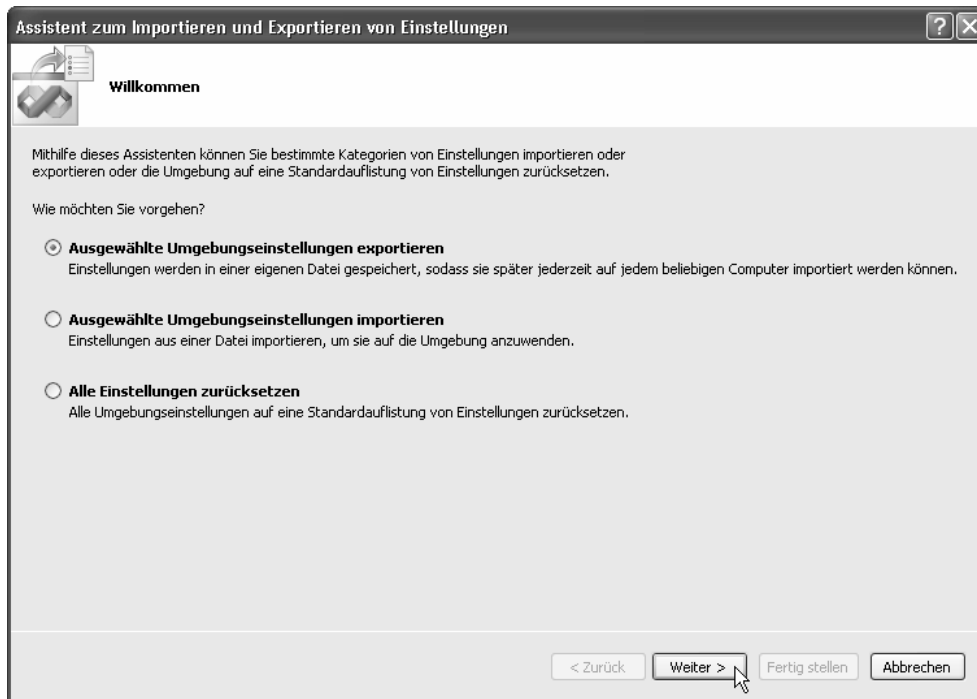


Abbildung 4.3: Mit diesem Assistenten können Sie einige oder alle Einstellungen von Visual Studio in einer Datei speichern, um sie beispielsweise auf andere Rechner zu übertragen oder als Backup zu archivieren

- Wählen Sie den ersten Punkt der Liste *Ausgewählte Umgebungseinstellungen exportieren*, und klicken Sie auf *Weiter*.

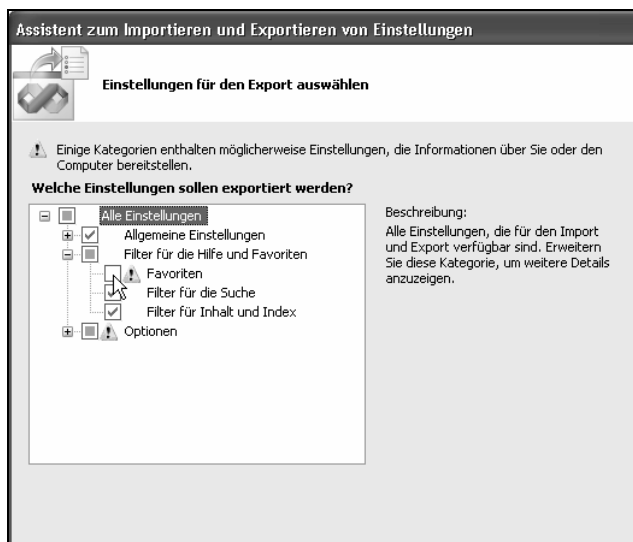


Abbildung 4.4: Wählen Sie, welche Einstellungen Sie exportieren möchten. Die mit dem Ausrufezeichen versehenen könnten übrigens persönliche Informationen enthalten.

- Wählen Sie die Einstellungen, die Sie exportieren möchten, und klicken Sie anschließend auf *Weiter*.

HINWEIS: Die mit dem Ausrufezeichen versehenen Einstellungstypen könnten u.U. persönliche Informationen enthalten. Seien Sie sich dessen bewusst, wenn Sie Visual Studio-Einstellungen anderen Entwicklern Ihres Teams auf diese Weise zur Verfügung stellen.

TIPP: Wenn Sie alle Einstellungen exportieren wollen, wählen Sie *Alle Einstellungen*. Das könnte u.U. persönliche Einstellungen beinhalten.

- Bestimmen Sie im folgenden Assistentenschritt Dateinamen und Zielordner.
- Klicken Sie auf *Fertigstellen*, um das Speichern der Einstellungen zu starten. Dieser Vorgang kann einige Zeit in Anspruch nehmen.

Wiederherstellen von Visual Studio-Einstellungen

Um wie im vorherigen Abschnitt beschriebene Einstellungen in eine Visual Studio-Installation zu importieren, verfahren Sie folgendermaßen:

- Wählen Sie aus dem Menü *Extras* den Menüpunkt *Einstellungen importieren und exportieren*.
- Visual Studio zeigt Ihnen nun einen Assistenten, etwa wie in Abbildung 4.3 zu sehen, und dort wählen Sie *Ausgewählte Umgebungseinstellungen importieren*.

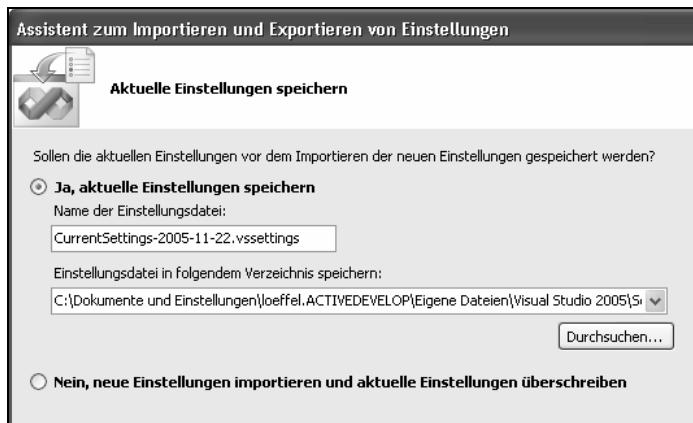


Abbildung 4.5: Sie können vor dem Importieren gespeicherter Einstellungen die aktuellen Einstellungen sichern

- Sie können vor dem Importieren gespeicherter Einstellungen die aktuellen Einstellungen sichern – dazu wählen Sie im Dialog, den Sie auch in Abbildung 4.5 sehen, den ersten Punkt und geben den Dateinamen und den Speicherort ein. Wählen Sie anderenfalls die zweite Option.
- Klicken Sie auf *Weiter*.
- Im nächsten Schritt haben Sie zwei grundsätzliche Möglichkeiten: Sie können aus einem ganzen Pool von Standardeinstellungen für die unterschiedlichsten Zwecke wählen oder eine zuvor ausgewählte Einstellungsdatei auswählen und diese damit für das Importieren festlegen. Um eine zu-

vor gespeicherte Einstellungsdatei für das Importieren auszuwählen, klicken Sie auf die Schaltfläche *Durchsuchen*, die Sie auch in Abbildung 4.6 sehen können.

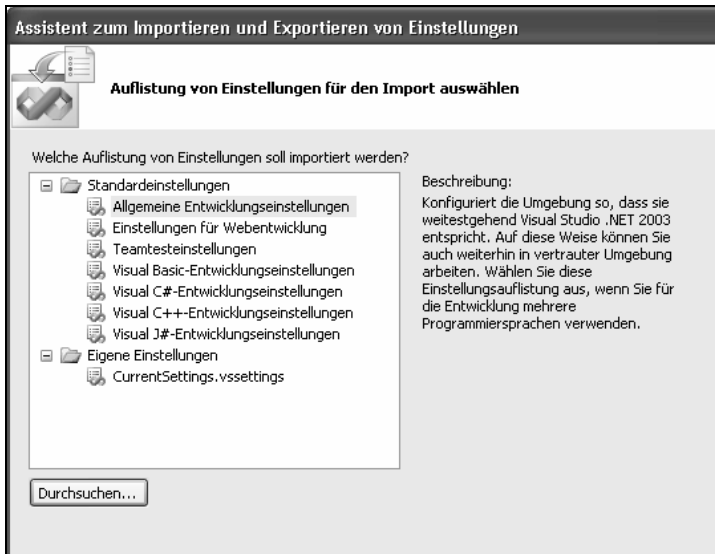


Abbildung 4.6: Entscheiden Sie sich an dieser Stelle entweder für eine Reihe standardmäßig vorhandener Einstellungsvorgaben oder für eine Einstellungsdatei, die zuvor exportierte Einstellungsdaten enthält

- Klicken Sie anschließend auf *Fertigstellen*, um den Importvorgang der Einstellungen zu starten. Dieser Vorgang kann eine Weile dauern.

Zurücksetzen von Visual Studio in den Originalzustand

Sie können alle Visual Studio Einstellungen in den Ausgangszustand zurückversetzen, den es nach der Installation aufwies. Dazu verfahren Sie wie folgt:

- Wählen Sie aus dem Menü *Extras* den Menüpunkt *Einstellungen importieren und exportieren*.
- Visual Studio zeigt Ihnen nun einen Assistenten, etwa wie in Abbildung 4.3 zu sehen.
- Wählen Sie die Option *Alle Einstellungen zurücksetzen*, und klicken Sie auf *Weiter*.
- Sie können vor dem Zurücksetzen aller Einstellungen die aktuellen Einstellungen sichern – dazu wählen Sie im Dialog, den Sie auch in Abbildung 4.5 sehen, den ersten Punkt und geben den Dateinamen und den Speicherort ein. Wählen Sie anderenfalls die zweite Option.
- Klicken Sie auf *Weiter*.
- Entscheiden Sie sich im nächsten Assistentenschritt für eine Reihe standardmäßig vorhandener Einstellungsvorgaben, wie Sie sie auch vor dem ersten Start von Visual Studio ausgewählt haben (siehe Abbildung 4.7).
- Klicken Sie anschließend auf *Fertigstellen*.



Abbildung 4.7: Entscheiden Sie sich an dieser Stelle für eine Reihe standardmäßig vorhandener Einstellungsvorgaben, wie Sie sie auch vor dem ersten Start von Visual Studio ausgewählt haben

Wieviel Arbeitsspeicher darf's denn sein?

Um es ganz klar zu sagen: Die Speichervoraussetzungen, die Microsoft empfiehlt, reichen meiner Meinung nach mit 192 MByte auch auf Windows 2000-Systemen bei weitem nicht aus.¹ Es sind im wahren Wortsinn tatsächlich Mindestanforderungen.

512 MByte sind meiner Meinung nach nötig aber auch ausreichend, wenn Sie ausschließlich mit Visual Studio 2005 arbeiten. Falls Sie Datenbankanwendungen entwickeln möchten und dazu auf SQL Express zurückgreifen, das auf dem gleichen Computer installiert sein soll, sollten Sie am besten 768 MByte oder gleich 1 GByte Hauptspeicher für ein zügiges Arbeiten in Betracht ziehen. Ich arbeite mit 2 GB und denke manchmal immer noch: Mehr wäre auch nicht schlimm.

Wenn Sie sogar Testinstallationen mithilfe von Virtual PC oder ähnlichen Tools auf demselben Rechner durchführen möchten (siehe auch »Testen Ihrer Software unterwegs und zuhause – Microsoft Virtual PC und Microsoft Virtual Server« ab Seite 117), dann sollten sogar 2 GByte im Rechner sein. Und obwohl Speicher vergleichsweise preiswert geworden ist: Mehr als 4 GByte machen unter der 32-bit-Version von Windows XP übrigens keinen Sinn, da Windows XP mit herkömmlichen Mitteln nicht mehr als 3 GByte pro Prozess² zur Verfügung stellen kann, und selbst die bekommen Sie nur

¹ Im Übrigen finde ich, dass selbst 256 MByte auf Windows-XP-Systemen ohnehin nur dann ausreichen, wenn Sie nichts anderes außer Windows-XP installieren. Aber dann müssten Sie Ihre Berichte mit WordPad und Ihre Kalkulationen mit dem Taschenrechner durchführen ...

² 1 GByte wird dann als Kernel-Speicher – also für das gemeinsam benutzte Betriebssystem – verwendet; und auch von dem kann u.U. nicht alles verwendet werden, da sich beispielsweise der eigene Grafikspeicher der Grafikkarte ebenfalls in diesem Bereich befindet.

dann, wenn Sie die *boot.ini*-Datei von Windows XP modifizieren.³ Außerdem können die meisten handelsüblichen Motherboards für 32-Bit-Windows-Systeme ebenfalls nicht mehr als 4 GByte Hauptspeicher verwalten.

Testen Ihrer Software unterwegs und zuhause – Microsoft Virtual PC und Microsoft Virtual Server

Dass Ihre entwickelte Software in Ihrer Entwicklungsumgebung läuft, bedeutet nicht notwendigerweise, dass sie das auch auf einem frisch installierten PC beim Kunden macht. Das Testen Ihrer Software gehört deswegen neben der Entwicklung zu Ihren wichtigsten Aufgaben – sollte es zumindest.

Doch dieses Testen ist unter Umständen nicht minder aufwändig als die Softwareentwicklung, denn:

- Läuft Ihre Software auch auf »frischen« Rechnern ohne Visual Studio-Entwicklungsumgebung, die schließlich dafür sorgt, dass ohnehin alle von Ihrer Software benötigten Komponenten vorhanden sind?
- Wenn Sie Ihre Software unter XP entwickelt haben, läuft Sie auch unter Windows 2000? Wie sieht es aus mit Vista oder Windows 2003?
- Als Entwickler haben Sie sicherlich andere Rechte in einer Domäne als ein »einfacher« Benutzer? Vielleicht entwickeln Sie gar nicht in einer Active-Directory-Umgebung, doch Ihre Software soll vielleicht genau dort eingesetzt werden!

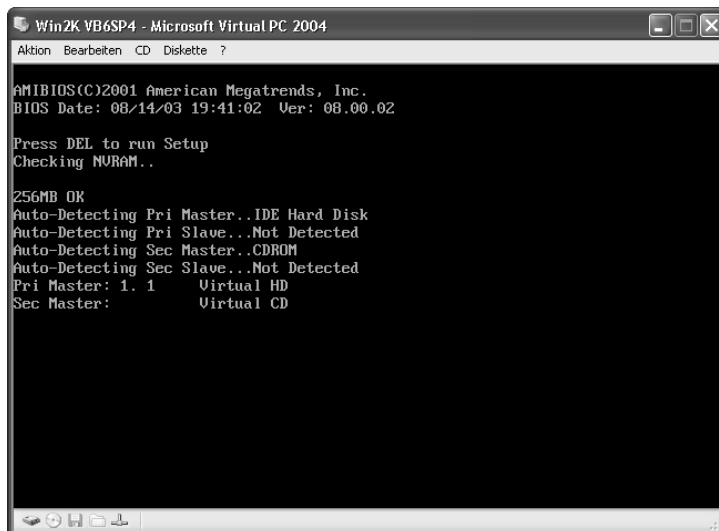


Abbildung 4.8: Ein virtueller PC beginnt mit seiner Arbeit ...

Sie ahnen es: All diese Konstellationen nachzustellen, würde einen ganzen Rechnerpark erfordern. Doch das ist nicht notwendig, denn exakt für diese Testkonfigurationen gibt es Software, die einen

³ Googeln Sie dazu am besten mal nach +“/3GB Switch“ +“Windows XP“, um die vielen verfügbaren Hinweise zum Thema zu bekommen.

kompletten Computer mit Bios, virtuellen Festplatten, die in Dateien gemapt werden, emulierten Grafik- und Soundkarten und umgeleiteter Tastatur- und Maussteuerung mit nahezu 100%iger Kompatibilität in ein Windows-Fenster packen – entsprechend viel Rechenpower und ausreichend Speicher vorausgesetzt. Microsoft selbst stellt dazu zwei Tools zur Verfügung – Microsoft Virtual PC und Microsoft Virtual Server.

Microsoft Virtual PC

Virtual PC installieren Sie am besten genau aus den zuvor genannten Gründen, nämlich eben um Ihre Testaufgaben als Entwickler am einfachsten und elegantesten erledigen zu können.



Abbildung 4.9: ... um Sekunden später ein voll funktionsfähiges und durchaus schnell laufendes Windows 2000 zu präsentieren, mit Windows XP als Gastgeber, und mit allen Rechten und Pflichten

Sie können mehrere virtuelle Maschinen nebeneinander installieren und diese untereinander sogar virtuell vernetzen. Sie können eine virtuelle Maschine genau wie einen richtigen PC an einer Domäne anmelden – der Domänencontroller wird den Unterschied nicht einmal bemerken. Abbildung 4.8 und Abbildung 4.9 demonstrieren dieses Konzept, wie ich finde, auf eindrucksvolle Weise.

Dabei gehen die Möglichkeiten in einem Virtual PC teilweise über die des Gastsystems hinaus. Besonders einfach lässt sich beispielsweise ein Rückgängig-Datenträger konfigurieren. Alle Änderungen während einer Sitzung werden dann zunächst nur auf diesem gespeichert. Beim Ausschalten bzw. Herunterfahren des Virtual PCs werden Sie dann gefragt, ob Sie die gemachten Änderungen endgültig übernehmen wollen: ideal für Tests oder Trainings, wo es verlässlich einen immer gleichen Ausgangszustand herzustellen gilt. Wie oft habe ich mir das für Windows selbst gewünscht!

HINWEIS: Bedenken Sie aber auch, dass Sie für jede virtuelle Maschine genauso die entsprechenden Softwarelizenzen benötigen, als handelte es sich bei diesen um »echte« PCs. Microsoft greift Ihnen aber gerade als Softwareentwickler mit dem MSDN-Abo⁴ unter die Arme. Oder als einfacher Microsoft-Partner mit dem sehr zu empfehlenden Action Pack. Das Gleiche gilt auch für den Arbeitsspeicher, der vom Gastgeberrechner abgezackt wird. Möchten Sie einen virtuellen PC mit Windows XP und 512 MByte Arbeitsspeicher installieren, sollte Ihr System, das den VPC hostet, über 512 MByte zzgl. der Menge an Arbeitsspeicher verfügen, um selbst zügig zu laufen. Bei mehreren VPCs, die gleichzeitig laufen sollen, weil Sie Ihre Software vielleicht in einem simulierten Active-Directory-Netzwerk testen wollen, sind Sie dabei schnell im Gigabyte-Bereich, was den Arbeitsspeicherbedarf anbelangt.

Wie in Abbildung 4.9 oben links zu sehen, steuern Sie alle VPC-Installationen mit einem zentralen Kommandocenter, über das Sie auch deren Einstellungen wie Speicherzuordnungen, virtuelle Festplatten oder Weiteres vornehmen. Eine kostenlose Testversion finden Sie schnell auf der Microsoft Webseite.

Ein etwas anderes Konzept verfolgt Virtual Server 2005 von Microsoft.

⁴ Hinweise zu den verschiedenen MSDN-Abos gibt es unter dem IntelliLink *B0401*.

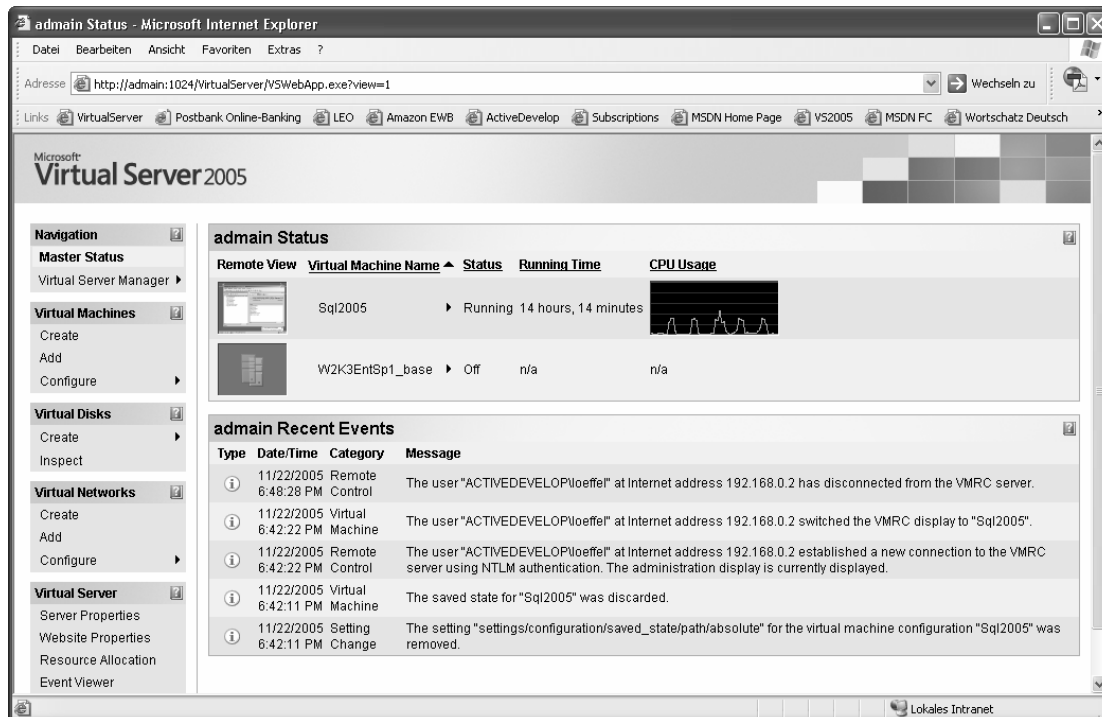


Abbildung 4.10: Alle auf einem Server installierten virtuellen Server werden von jedem beliebigen Client des Netzwerkes aus auf denkbar einfachste Weise administriert: über den Internet-Explorer ...

Virtual Server 2005

Wie Virtual PC dient auch Virtual Server 2005 in erster Linie dazu, ganze Computersystemplattformen auf einem entsprechend ausgestatteten Windows Server (oder auch Windows XP) zu simulieren. Doch das dient vor allem dazu, mehrere physisch vorhandene Server in einem Server in Form von virtuellen Servern zu konsolidieren.

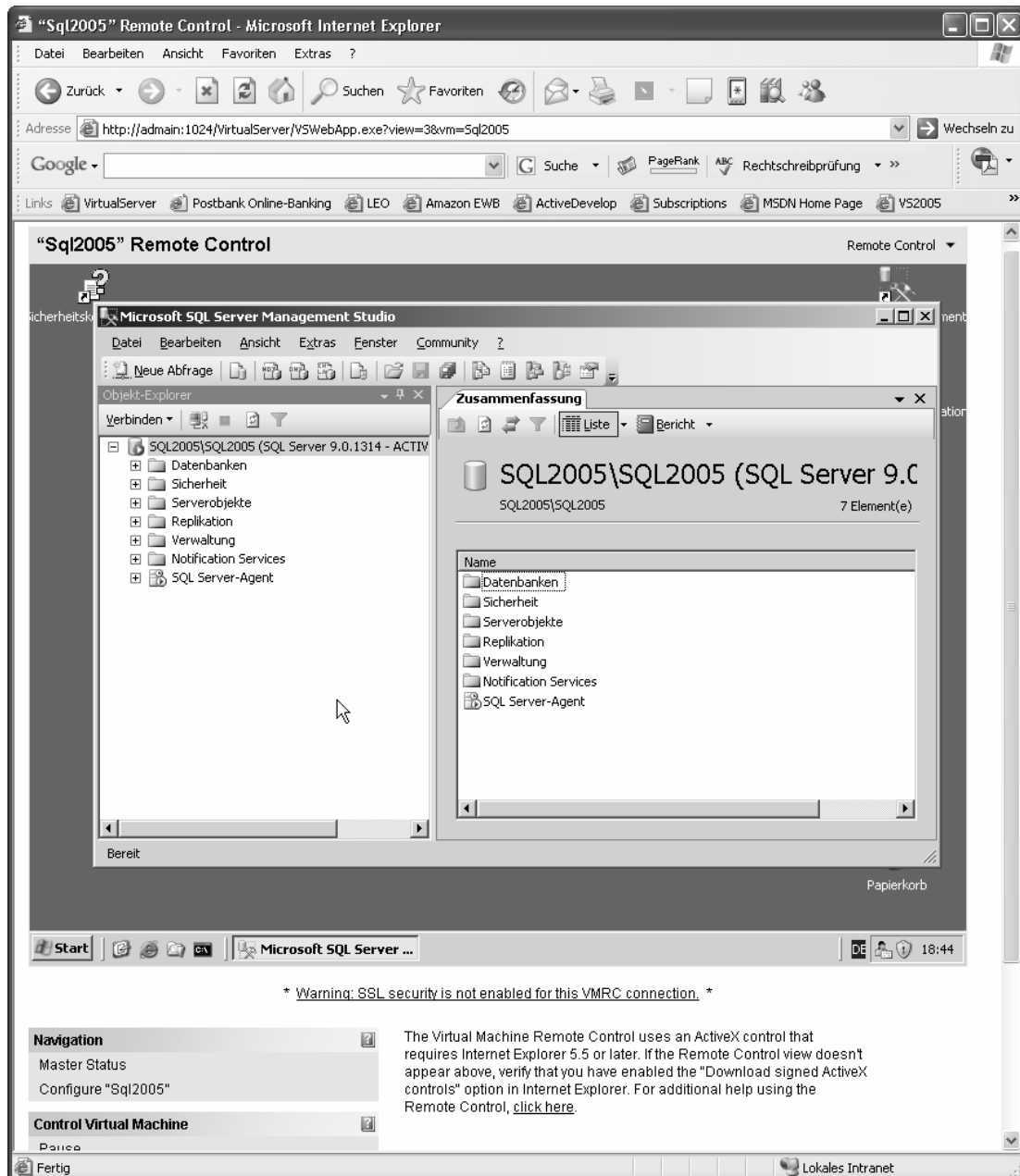


Abbildung 4.11: ... und dieser dient schließlich sogar dazu, einen virtuellen PC zu bedienen

Im Klartext heißt das: Statt einen Server beim Hardwareanbieter Ihres Vertrauens zu bestellen, auf dem beispielsweise ein SQL Server seine Dienste verrichtet und dann einen weiteren zu ordern, der als Internet-Anwendungsserver läuft, und noch einen weiteren, der alle Exchange-Server-Aufgaben

übernimmt, setzen Sie nur einen ausreichend groß dimensionierten ein, und lassen ihn als Gastgeber für weitere virtuelle Server werkeln.

Diese zusätzlichen Server laufen dann, anders als bei Virtual PC, nahezu unbemerkt als Dienst im Hintergrund – wenn Sie vor dem Desktop des eigentlichen Servers sitzen, werden Sie vergeblich nach Kommandozentralen oder Fenstern mit den virtuellen Servern suchen.

Stattdessen – und dieses Konzept finde ich persönlich sehr genial – administrieren Sie alle virtuellen Server über das hauseigene Intranet!

Wenn Sie mit einem virtuellen Server arbeiten möchten, klicken Sie ihn auf der durch Virtual Server zur Verfügung gestellten Intranetseite einfach an, – und anschließend haben Sie, wiederum im Internet-Explorer gekapselt, eine ähnliche Oberfläche, wie Sie sie vom Virtual PC kennen. Dazu kann der Virtual Server auch eine SCSI Schleife simulieren, wie es etwa für das Testen von Cluster unter Windows hilfreich sein kann, und den einzelnen laufenden Servern individuell RAM und Prozessorleistung zuteilen.

Hilfe zur Selbsthilfe

Wenn Sie sich schnell in eine neue Thematik der Frameworks einarbeiten müssen – und das gilt umso mehr wenn man Einsätze beim Kunden fährt –, lautet die oberste Prämisse: Intelligenz ist, wissen wo es steht!

Bei rund über 8.000 verschiedenen Klassen können Sie unmöglich, auch mit jahrelanger Erfahrung, die genaue Funktionsweise jeder dieser Klassen gelernt haben und direkt aus dem Stegreif anwenden. Was Sie allerdings lernen und perfektionieren können, ist, so schnell wie möglich an die gewünschten Informationen zu kommen.

Die Kombination der Hilfsmittel *Codeausschnittsbibliothek*, *Vervollständigungsliste von IntelliSense*, *dynamische Hilfe* und *Autokorrektur für automatisches Kompilieren* ist dabei eine Ressource für Recherchen, die Sie auch dann nutzen können, wenn Sie beim Kunden vor Ort am Projekt arbeiten, wo Ihnen nicht unbedingt immer ein Internetzugang zur Verfügung steht.

Bei bestimmten Problemstellungen kann Sie die Codeausschnittsbibliothek – die Sie im letzten Kapitel schon kennen gelernt haben – nicht nur mit Code versorgen; sie kann auch Ausgangspunkt für weitere Recherchen nach den richtigen Klassen und Funktionsweisen sein.

Ein Beispiel: Angenommen, ein Kunde kommt zu Ihnen, und bittet Sie, ähnlich wie im letzten Kapitel zu sehen, im Falle eines Fehlers eine E-Mail an eine bestimmte Adresse zu schicken. Leider kennen Sie sich in diesem Bereich noch gar nicht aus. Bis jetzt. Es gilt nun also, seine Professionalität unter Beweis zu stellen, und sich schnellstmöglich die Informationen zu beschaffen, die man für die Realisierung dieser Problemlösung benötigt.

Hier hilft das Suchen in der Codeausschnittsbibliothek erst einmal, um eine Verbindung zwischen dem Thema (E-Mail-Versand) und den zur Lösung benötigten Objekten herzustellen. Mit ein wenig Glück findet sich in der Codeausschnittsbibliothek etwas Entsprechendes:

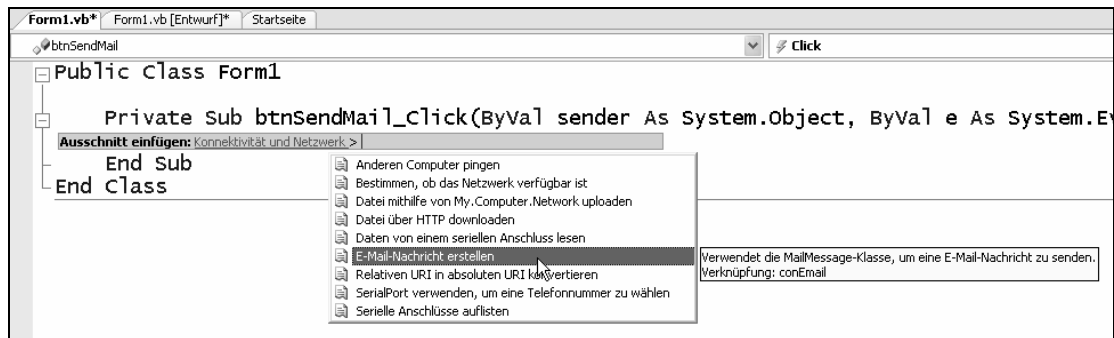


Abbildung 4.12: Über das Editor-Kontextmenü erreichen Sie zunächst die Codeausschnittbibliothek, in der Sie zunächst nach dem richtigen Codeschnipsel suchen, um ein Objekt für weitere Recherchen zu erhalten ...

Recht schnell haben Sie auf diese Weise einen ersten kleinen Codeblock erstellt, und die dort vorhandenen Objekte können Sie nun als Ausgangspunkt für weitere Recherchen verwenden.

Der eingefügt Code reicht Ihnen leider momentan noch nicht aus, um das Problem in den Griff zu bekommen, denn dieser Code

```
Imports System.Net.Mail
```

```
Public Class Form1
```

```
    Private Sub btnSendMail_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnSendMail.Click
```

```
        Dim message As New MailMessage("sender@address", "from@address", "Subject", "Message Text")
        Dim emailClient As New SmtpClient("Email Server Name")
        emailClient.Send(message)
```

```
    End Sub
```

```
End Class
```

ist zwar in der Lage, eine E-Mail zu versenden, aber leider nicht in der Lage, sich auch beim Mail-Server zuvor anzumelden. Doch alleine das Einfügen des Codeausschnittes hat Ihnen schon einige zusätzliche Hinweise gebracht, wo Sie mit weiteren Recherchen fortfahren können:

- Das Codeausschnittseinfügen hat unter anderem bewirkt, dass in der obersten Zeile eine Imports-Anweisung eingefügt wurde. Weitere Klassen, die sich mit der Thematik befassen, werden also mit großer Wahrscheinlichkeit auch im System.Net.Mail-Namespace zu finden sein.
- Eine Nachricht wird offensichtlich durch die MailMessage-Klasse gekapselt.
- Ein Smtp-Client zum Weiterleiten einer Mail, wird offensichtlich durch die SmtpClient-Klasse angesteuert.

Mit diesen Informationen lässt sich doch jetzt schon einiges anfangen. Rekapitulieren wir. Wir können zum jetzigen Zeitpunkt zwar eine Mail schicken, uns aber nicht am Mail-Server anmelden. Also müssen wir herausfinden, welche zusätzlichen Möglichkeiten die SmtpClient-Klasse bietet, und ob es dort nicht irgendwelche »Dinge« gibt, die uns bei der Problemlösung helfen können.



Abbildung 4.13: Die dynamische Hilfe hält kontextabhängige Querverweise in die eigentliche Hilfe parat

An diesem Punkt kommt – es geht ja um die schnelle Suche nach Infos – die dynamische Hilfe ins Spiel. Wenn Sie mit dem Cursor auf die Instanzierungsanweisung für das `SmtpClient`-Objekt `emailClient` fahren, dann stellt die dynamische Hilfe im gleichen Moment einige Querverweisschlüsse zur Verfügung, wie in Abbildung 4.13 zu sehen. Und dort sehen wir als Hilfetopic beispielsweise den Dokumentationsverweis zur `SmtpClient`-Klasse.

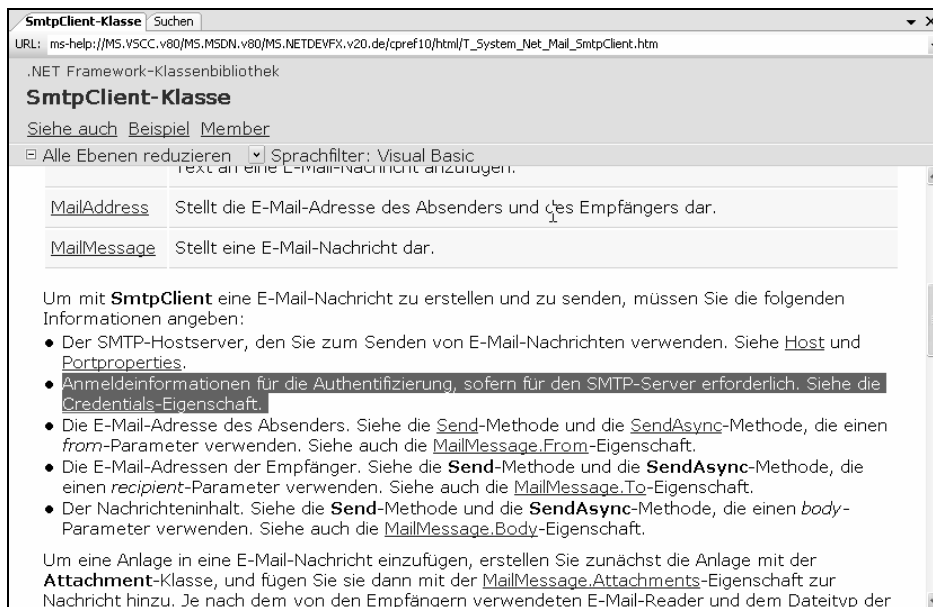


Abbildung 4.14: Ein genaues Lesen der Hilfetexte ist oft hilfreich, um schnell an erforderliche Infos zu kommen

Ein Klick auf den Verweis öffnet anschließend die eigentliche Hilfe. Und hier hilft ein genaues Lesen der Hilfetexte oft schon enorm weiter, zumindest um weitere Hinweise für die Lösung des Problems zu erhalten. In Abbildung 4.14 sieht man es exemplarisch. Es findet sich tatsächlich ein Hinweis auf »Anmeldeinformationen für die Authentifizierung«, und ein weiterer Klick auf den in diesem Zu-

sammenhang stehenden Link bringt uns noch einen Schritt in der Recherche-Kette weiter: Der Link zur Dokumentation der Credentials-Eigenschaft.

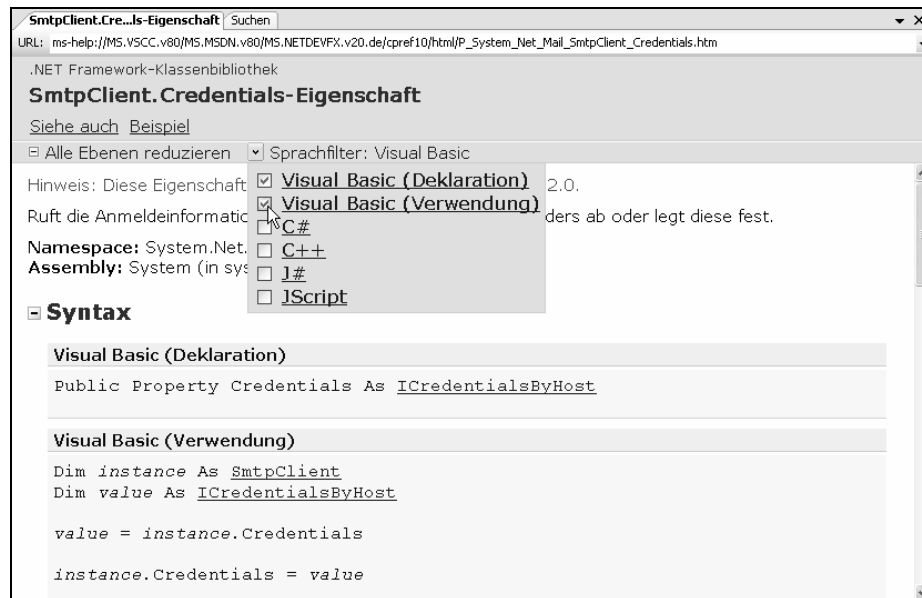


Abbildung 4.15: Sorgen Sie bei der Recherche für den »Blick aufs Wesentliche«

An diesem Punkt angelangt sind zwei Dinge zu bemerken: Wenn Sie zum ersten Mal mit der Hilfe von Visual Studio gearbeitet haben, werden Sie spätestens jetzt feststellen, dass Hilfetemen zu Klassen, Methoden, Eigenschaften oder Ereignissen immer nach dem gleichen Schema aufgebaut sind. So gibt es für Beispiele und Prototypenbeschreibungen die Möglichkeit, wie in Abbildung 4.15 zu sehen, über den Sprachfilter zu steuern, für welche der .NET-Sprachen die Beispiele dargestellt werden sollen. Beschränken Sie an dieser Stelle die Beispiele auf Visual Basic, um sich nicht durch zu viel »anderen« Beispielcode ablenken zu lassen.

Gleichzeitig ist es auch hilfreich, sich mit möglichst allen Techniken im objektorientiert arbeitenden Visual Basic auszukennen. So werden Sie später wissen, nachdem Sie den OOP-Teil dieses Buches durchgearbeitet haben, dass die so genannten Interfaces (Schnittstellen) Vorschriften für erst eigentlich verwendbare Klassen sind, bestimmte Funktionalitäten einzubinden. Sie werden dann auch wissen, dass Sie Interfaces bereits an ihrer Namenskennung identifizieren können – diese beginnen nämlich grundsätzlich mit einem »I«. Sie werden dann Ihr Wissen in einen Kontext setzen können, und schlussfolgern, dass, wenn die Credentials-Eigenschaft vom Typ `ICredentialsByHost` ist, es irgendwelche konkreten Klassen geben muss, die dieses Interface einbinden. Denn genau diese Klassen können diese Eigenschaft nämlich verarbeiten. Es gilt nun also abzuchecken, welche Klassen diese Schnittstelle einbinden, und ob sie für unsere Problemlösung in Frage kommen.

TIPP: Vielleicht sind Sie für den Moment noch durch die vielen, vielleicht für Sie neuen OOP-Elemente wie Schnittstellen und Klassen überfordert. In diesem Fall empfehle ich Ihnen, dieses Kapitel abermals zu lesen, wenn Sie sich mit der OOP-Programmierung im entsprechenden Teil dieses Buches vertraut gemacht haben.

Leider bietet die Hilfe bis heute keine Möglichkeit, eine Liste aller Klassen abzurufen, die eine bestimmte Schnittstelle einbinden. Aber es gibt ja immer noch die Suchen-Funktion der Hilfe, und die ist in Visual Studio 2005 gar nicht schlecht. Da für jede Klasse, die eine bestimmte Schnittstelle einbindet, diese Schnittstelle auch in der Hilfe beschrieben wird, ist die Wahrscheinlichkeit denkbar groß, auf diese Weise eine Klasse zu finden, mit der die Credentials – die Anmeldeinformationen – in der Credentials-Eigenschaft verpackt werden können. Und siehe da, die Eingabe von ICredentialsByHost als Suchbegriff in der Visual Studio-Hilfe liefert im Handumdrehen die folgenden Suchergebnisse:

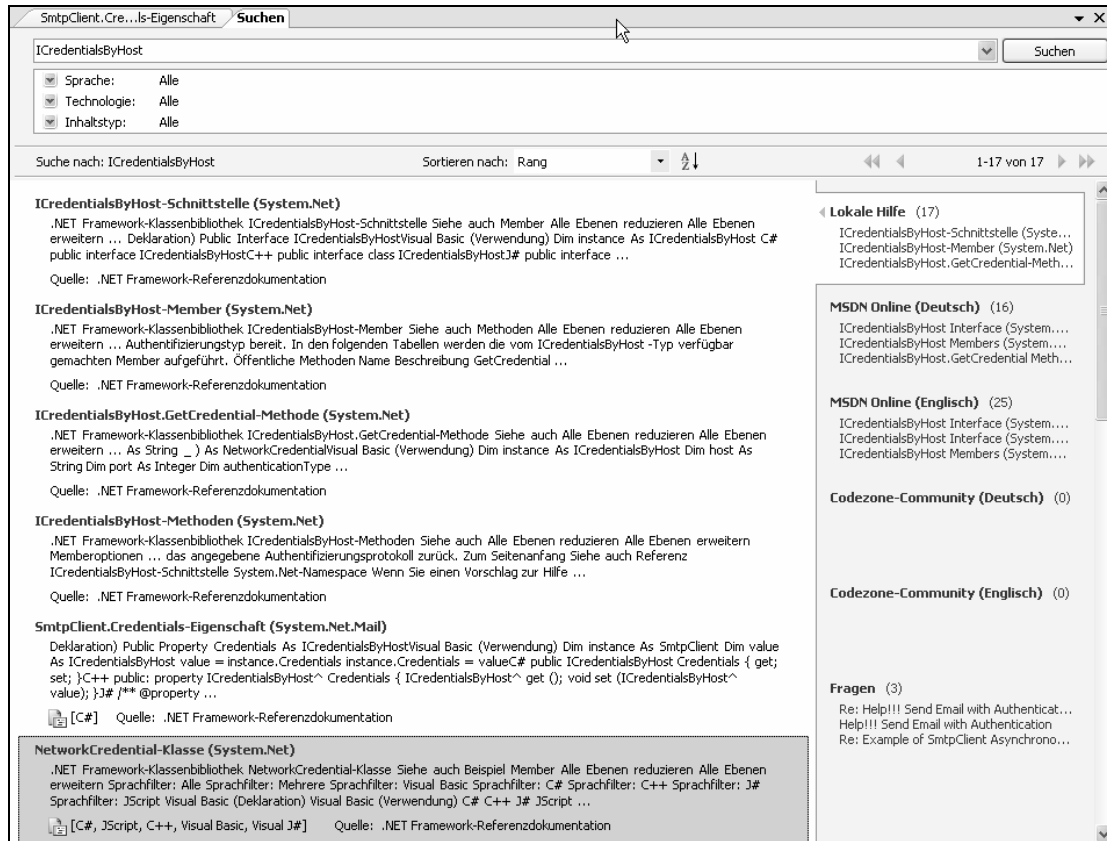


Abbildung 4.16: Erster Versuch, Volltreffer. Die *NetworkCredential*-Klasse sieht doch recht viel versprechend aus.

Ein Klick auf die *NetworkCredential*-Klasse in der Suchergebnisliste offenbart dann auch, wonach wir suchten – nach einer Klasse, die die Anmeldeinformationen aufnimmt, und die entsprechenden Schnittstellen einbindet, sodass eine Instanz dieser Klasse an die *SmtplibClient*-Instanz – oder besser: deren *Credentials*-Eigenschaft übergeben werden kann.

Mit diesem Wissen kann das Problem nun vergleichsweise schnell gelöst werden und könnte dann beispielsweise folgendermaßen aussehen:

```
Public Class Form1
```

```
    Private Sub btnSendMail_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
        Handles btnSendMail.Click  
        'Gesucht und gefunden! - Aber Achtung. Die Anmeldedaten werden im Klartext übertragen!  
        'Dieses Verfahren also nur bei Nicht-AD-Clients im Intranet anwenden!  
        Dim myCred As New NetworkCredential("k_loeffel", "passwort", "ActiveDevelop.local")  
  
        'Das wurde durch die Codeausschnittsbibliothek eingefügt  
        Dim message As New MailMessage("VomTestprogramm@loeffelmann.de", _  
            "klaus@loeffelmann.de", _  
            "Testmail", _  
            "Das Programm hat eine Testnachricht versendet!")  
        Dim emailClient As New SmtplibClient("192.168.0.1")  
        'Die Credentials übergeben wir nun!  
        emailClient.UseDefaultCredentials = False  
        emailClient.Credentials = myCred  
        'Und ab dafür!  
        emailClient.Send(message)  
    End Sub  
End Class
```

Erweitern Sie die Codeausschnittsbibliothek um eigene Codeausschnitte

Es gibt bei der Entwicklung von Projekten immer wiederkehrende Dinge, die oftmals in pure Fließbandarbeit ausarten. Was mich persönlich beispielsweise am meisten nervt, ist das Programmieren von Dateiauswahl-Dialogen. Der Benutzer muss auf eine Schaltfläche klicken, daraufhin öffnet sich ein Datei-Öffnen-Dialog, der durch die `OpenFileDialog`-Klasse gesteuert wird, der Benutzer wählt seine Datei aus, oder er bricht den Dialog ab.

BEGLEITDATEIEN: Ein kleines Projekt, das die generelle Vorgehensweise in VB2005 demonstriert, finden Sie unter `.\VB 2005 - Entwicklerbuch\B - IDE\04 - TippsUndTricks\Codeausschnitte` unter dem Projektmappennamen `Codeausschnitte.sln`.

Starten Sie dieses Programm, können Sie mit der Auslassungsschaltfläche (...) den Datei-Öffnen-Dialog ins Leben rufen, eine Textdatei auswählen und den Dialog mit *OK* bestätigen. Der Dateiname steht anschließend im Textfeld neben der Auslassungsschaltfläche:

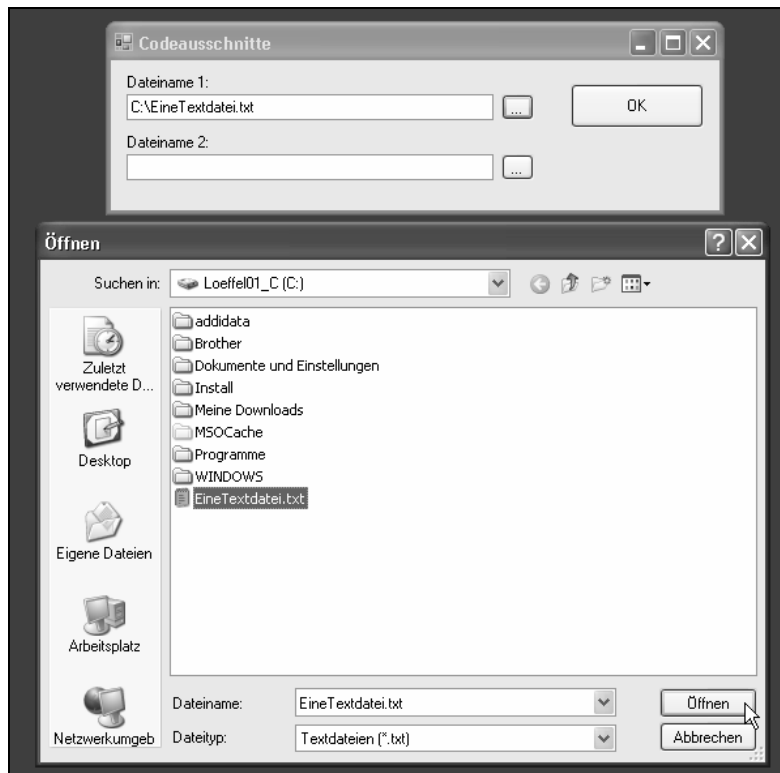


Abbildung 4.17: Ein Vorgang, den Sie in Ihrem Leben sicherlich schon zigmal programmieren mussten: das Zur-Verfügung-Stellen von Dateiauswahl-Dialogen

In Visual Basic 2005 bzw. dem .NET-Framework funktioniert das Aufrufen eines solchen Dialogs in einer Windows Forms-Anwendung auf folgende Weise:

```
Public Class Form1
```

```
    Private Sub btnDateiAuswählen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnDateiAuswählen.Click
```

```
        'Eine neue OpenFileDialog-Klasse instanzieren
        Dim dateiÖffnenDialog As New OpenFileDialog
```

```
        'Alles Weitere bezieht sich nun darauf, bis 'End With'
        With dateiÖffnenDialog
            .CheckFileExists = True ' Datei muss existieren
            .CheckPathExists = True ' der Pfad ebenfalls
            .DefaultExt = "*.txt" ' Standardendung ist *.TXT
```

```
        'Alle angezeigten Dateifilter werden folgendermaßen angegeben
        .Filter = "Textdateien (*.txt)|*.txt|Alle Dateien (*.*)|*.*"
```

```
        'Diese Enum-Variable nimmt das Dialogergebnis (OK, Abbrechen) entgegen
        Dim dialogErgebnis As DialogResult = .ShowDialog
```

```

        'Falls das Dialogergebnis 'Abbrechen' war,
        If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If
        txtDateiname1.Text = .FileName
    End With
End Sub
End Class

```

Auch wenn Sie einen solchen Dialog in VB2005 noch nicht programmiert haben – ich denke, der Code ist nicht sonderlich schwer zu verstehen. Was man auch gut nachvollziehen kann: Dieser Code kann, wenn man ihn öfters braucht, auch recht lästig zu tippen werden – so was immer wieder zu implementieren kann dann wirklich zur Fließbandarbeit ausarten.

Das Ziel sollte es deswegen sein, einen entsprechenden Coderumpf in der Codeausschnittsbibliothek zu hinterlegen. In diesem Fall können Sie ihn, wann immer Sie ihn benötigen, dort herausholen, an der entsprechenden Stelle einfügen und sich so einen Haufen Arbeit sparen. Also, auf geht's:

Erstellen einer Code Snippets-XML-Vorlage

Codeausschnitte nennen sich auf Englisch *Code Snippets* (eigentlich im Deutschen mit dem viel bekannteren Begriff »Codeschnipsel« zu übersetzen – aber eine Grundsatzdiskussion über den Gebrauch der Sprache in Microsoft-Technologien möchte ich Ihnen und mir an dieser Stelle lieber ersparen...). Und jedes der schon in Visual Studio vorhandenen Codeschnipsel befindet sich in einer im XML-Format abgelegten Datei, die die Endung *.snippet* trägt. Doch keine Angst: Auch wenn Sie sich noch nicht mit dem Thema XML beschäftigt haben – Sie werden keine Probleme haben, die entsprechenden XML-Rümpfe zu bauen, und sie auf Ihre Bedürfnisse anzupassen.

Für die ersten Experimente zur Erstellung der Snippet-XML-Vorlage finden Sie im gleichen Verzeichnis, in dem sich auch das Beispielprojekt befindet, eine Datei namens *Vorlage.snippet*. Um diese XML-Datei zunächst zu öffnen, verfahren Sie wie folgt:

- Wählen Sie aus dem Menü *Datei* den Menüpunkt *Öffnen* und *Datei*.
- Suchen Sie im Datei-Öffnen-Dialog im entsprechenden Projektverzeichnis nach der Datei *Vorlage.snippet*, und klicken Sie auf *OK*, um die Datei in den Editor zu laden.

Die Datei, die Sie anschließend sehen sollten, schaut folgendermaßen aus:

```

<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>

      <Title>
        <!-- An dieser Stelle den Namen des Codeausschnitts einfügen: -->
        MeinCodeausschnitt
      </Title>

```

```

</Header>
<Snippet>
  <References>
    <Reference>

      <!-- An dieser Stelle den Namen einer benötigten
      Assembly-Referenz einfügen: -->
      <Assembly>System.Windows.Forms.dll</Assembly>

    </Reference>
  </References>

  <!-- Hier folgt die eigentliche Codedefinition: -->
  <Code Language="VB">
    <![CDATA[
      ' Hier steht der einzufügende
      ' Visual Basic 2005 - Programmcode
    ]]>

  </Code>
</Snippet>
</CodeSnippet>
</CodeSnippets>

```

Sie erkennen im Listing drei in Fettschrift gesetzte Blöcke, die folgende Funktion haben:

- Der erste Block definiert den Namen des Code Snippets. Zwischen den XML-Tags `<Title>` und `</Title>` platzieren Sie den Namen, den das Code Snippet erhalten soll.
- Nun kann es sein, dass Sie im Codeblock, den Sie später in Ihr Projekt einfügen wollen, Referenzen auf erforderliche Assemblies benötigen. Wenn Sie mit dieser Aussage im Moment noch nichts anfangen können, empfehle ich die Lektüre des nächsten Kapitels, das sich gerade für VB6-Umsteiger eignet, um sich die Basics zu Assemblies und Assembly-Verweisen anzueignen. Da sich die `OpenFileDialog`-Klasse in der *Assembly System.Windows.Forms.dll* befindet, ergibt ein Verweis auf diese Assembly an dieser Stelle Sinn. Sollte es nämlich später in Ihrem Projekt noch keinen Verweis auf diese Assembly geben, wird beim Einfügen des Snippets der Verweis automatisch im Projekt eingerichtet, und Sie garantieren damit, dass alle verwendeten Objekte Ihres Snippets auch vom Basic-Compiler direkt gefunden werden können.
- Im dritten Block schließlich wird der eigentliche Code platziert, der beim Aufruf des Snippets eingefügt werden soll. Dieser Code muss sich in den eckigen Klammern der `CDATA`-Direktive befinden, so wie in der Vorlage der beiden Visual Basic-Kommentar-Zeilen zu sehen.

Im Grunde genommen, ist also nicht viel zu tun. In der Vorlage brauchen wir nur die entsprechenden Anpassungen vorzunehmen, den Code einzufügen und die Snippet-Vorlage unter einem neuen Namen zu speichern.

- Ändern Sie also den ersten Block folgendermaßen ab:

```
<Header>
<Title>
  <!-- An dieser Stelle den Namen des Codeausschnitts einfügen: -->
  Datei-Öffnen-Dialog
</Title>
</Header>
```

- Den zweiten Block belassen Sie, wie er ist – dort steht nämlich der für unsere Zwecke benötigte richtige Assembly-Verweis auf die *System.Windows.Forms.dll* bereits drin.
- Im dritten Block fügen wir nun den eigentlichen Code ein. Dazu klauen wir uns den Code einfach aus dem Projekt, kopieren ihn und fügen ihn an der richtigen Stelle in der XML-Snippet-Datei wieder ein, sodass sich folgendes Ergebnis ergibt:

```
<!-- Hier folgt die eigentliche Codedefinition: -->
<Code Language="VB">
  <![CDATA[
    Dim dateiÖffnenDialog As New OpenFileDialog
    With dateiÖffnenDialog
      .CheckFileExists = True
      .CheckPathExists = True
      .DefaultExt = "*.txt"
      .Filter = "Textdateien (*.txt)|*.txt|Alle Dateien (*.*)|*.*"
      Dim dialogErgebnis As DialogResult = .ShowDialog
      If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
        Exit Sub
      End If
    End With
  ]]>

</Code>
```

Und damit ist die erste einfache Code Snippet-Vorlage bereits fertig gestellt. Speichern Sie diese jetzt noch unter einen anderen Namen ab (*Datei | Vorlage.Snippet speichern unter...*) – beispielsweise unter *DateiÖffnenDialog.Snippet*.

Als nächstes müssen wir Visual Studio das neue Snippet noch beibringen. Wie das geschieht, zeigt der folgende Abschnitt.

Hinzufügen einer neuen Snippet-Vorlage zur Snippet-Bibliothek (Codeausschnittsbibliothek)

Für diese Zwecke kennt Visual Studio den so genannten Codeausschnitt-Manager, den Sie über das *Extras*-Menü öffnen können.

- Klicken Sie auf den entsprechenden Menüpunkt, erscheint ein Dialog, wie Sie ihn auch in Abbildung 4.18 sehen können.

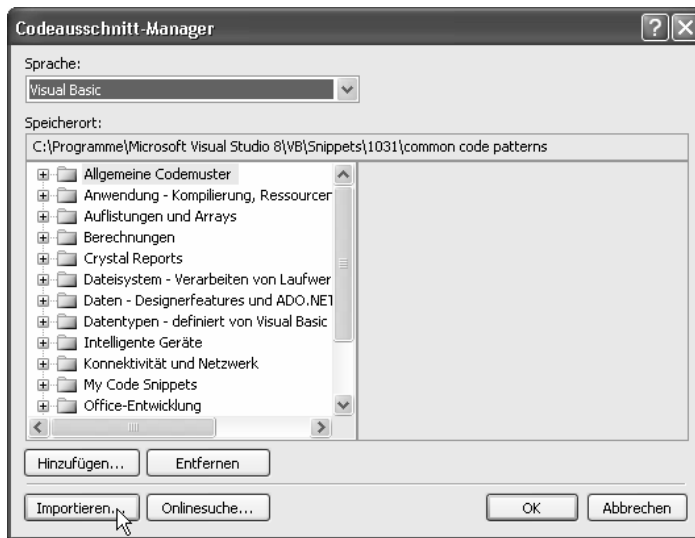


Abbildung 4.18: Mit dem Codeausschnitt-Manager können Sie vorhandene Codeausschnitte verwalten und neue Codeausschnittvorlagen hinzufügen

- Klicken Sie auf *Importieren*, um die Snippet-Vorlage auswählen zu können. Geben Sie dabei die gerade gespeicherte Snippet-Vorlage *DateiÖffnenDialog.Snippet* an.

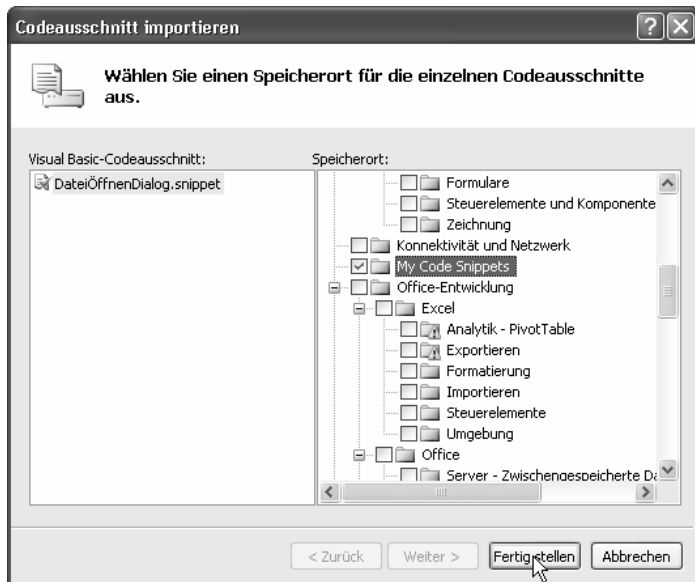


Abbildung 4.19: Bestimmen Sie, in welcher Rubrik der neue Codeausschnitt einsortiert werden soll

- Wählen Sie im Dialog, der anschließend erscheint, in der Liste *Speicherort*, in welcher Rubrik der neue Codeausschnitt einsortiert werden soll.
- Klicken Sie anschließend auf *Fertigstellen*.

Verwenden des neuen Codeausschnittes

Nachdem der neue Codeausschnitt in der Bibliothek einsortiert worden ist, können Sie als nächstes ausprobieren, ob er sich tatsächlich von dort aus abrufen und verwenden lässt. Und dazu verfahren Sie wie folgt:

- Doppelklicken Sie im Projektmappen-Explorer auf *Form1*, um diese im Designer darzustellen.
- Doppelklicken Sie im Formular auf die zweite Auslassungsschaltfläche (die zum Abrufen der 2. Datei dienen soll), um den Codeeditor zu öffnen und den Rumpf für die Ereignisbehandlungsroutine einfügen zu lassen.
- Wenn der Editor den Code dargestellt und den Cursor im Coderumpf platziert hat, klicken Sie auf die rechte Maustaste, um das Kontextmenü zu öffnen, und wählen anschließend den Menüpunkt *Ausschnitt einfügen*.

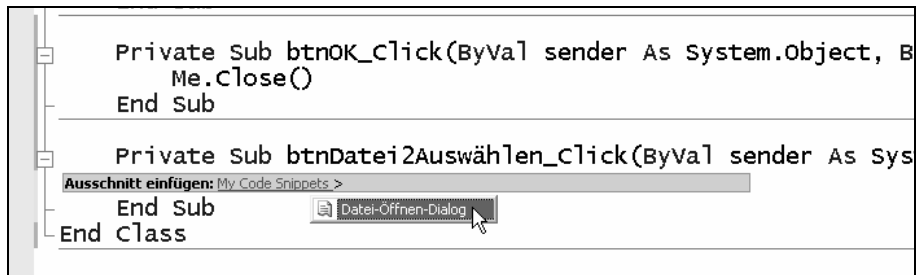


Abbildung 4.20: Der neue Codeausschnitt steht jetzt zum Einfügen in der Codeausschnittsbibliothek zur Verfügung

- Doppelklicken Sie in der Liste, die sich nun öffnet, nacheinander auf *My Code Snippets* und auf *Datei-Öffnen-Dialog*. Der gewünschte Codeblock wird nach dem zweiten Doppelklick im Editor eingefügt.

Parametrisieren von Codeausschnitten

Codeausschnitte in der Form, wie Sie sie gerade erstellt und der Bibliothek hinzugefügt haben, erleichtern die Arbeit schon ungemein. Aber das Konzept von Codeausschnitten in Visual Studio geht, was den Komfort anbelangt, noch weit über das hinaus, was wir bislang kennen gelernt haben. Betrachten wir uns noch mal zur Rekapitulation den Codeausschnitt, den wir gerade eingefügt haben:

```
Dim dateiÖffnenDialog As New OpenFileDialog
With dateiÖffnenDialog
    .CheckFileExists = True
    .CheckPathExists = True
    .DefaultExt = "*.txt"
    .Filter = "Textdateien (*.txt)|*.txt|Alle Dateien (*.*)|*.*"
    Dim dialogErgebnis As DialogResult = .ShowDialog
    If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
        Exit Sub
    End If
End With
```

Sie sehen, dass es hier einige Änderungsanforderungen gibt, die Sie zusätzlich erledigen müssen, um diesen Codeausschnitt auf einen jeweils neuen Kontext anzupassen – denn Sie werden schließlich nicht immer nur Textdateien, sondern auch mal Bilddateien öffnen wollen. Oder Sie möchten vielleicht, dass eben nicht auf das Vorhandensein von Pfad und Datei geprüft werden soll, wenn ein Anwender eine Datei auswählt.

Bequemer wäre es, ein vorhandenes Verfahren beim Bearbeiten eingefügter Codeausschnitte zu verwenden, das Sie gezielt innerhalb des eingefügten Ausschnittes zu den Stellen springen lässt, die geändert werden müssen. Es müsste also beispielsweise reichen, nachdem Sie den Codeausschnitt eingefügt haben, mit **Tabulator** direkt auf das erste True hinter `.CheckFileExists` zu gelangen, um es etwa in False zu ändern. Und in diesem Moment sollte sich der zweite hinter `.CheckPathExist` stehende Wert auch gleichzeitig in False ändern. Mit weiteren **Tabulator**-Tastendrücken gelangen Sie dann auf das erste hinter `.DefaultExt` stehende `*.txt`. Wenn Sie dieses ändern, beispielsweise in `*.bmp`, sollten sich auch gleichzeitig alle anderen `*.txt` in `*.bmp` ändern. Genau das ist möglich – aber dafür bedarf es ein wenig an Umgestaltung unserer ursprünglichen Codeausschnitt-Vorlagendatei.

Um eine solche Funktionalität zu implementieren, bedarf es zweier neuer XML-Elemente in der Vorlagendatei – literale Ersetzungen und Objektersetzungen.

Literale Ersetzungen in Codeausschnittvorlagen

Literale Ersetzungen sind definierte Platzhalter reinen Texts, die an verschiedenen Stellen innerhalb des Codeausschnittes die gleichen Inhalte darstellen sollen. Ist ein literaler Platzhalter definiert, und wird er innerhalb der Codeausschnittvorlage verwendet, dann wird dieser beim Einfügen des eigentlichen Codeausschnitts in den Code im Editor entsprechend markiert. Ein Druck auf **Tabulator** springt direkt in das entsprechende Platzhalter-Feld, und Sie können den Text wunschgemäß ändern. Wird die exakt gleich lautende literale Ersetzung an anderer Stelle in der Codevorlage abermals verwendet, bewirkt die erste Änderung eine Anpassung des Textes an allen entsprechenden anderen Stellen. Die folgende Abbildung soll dieses verdeutlichen:

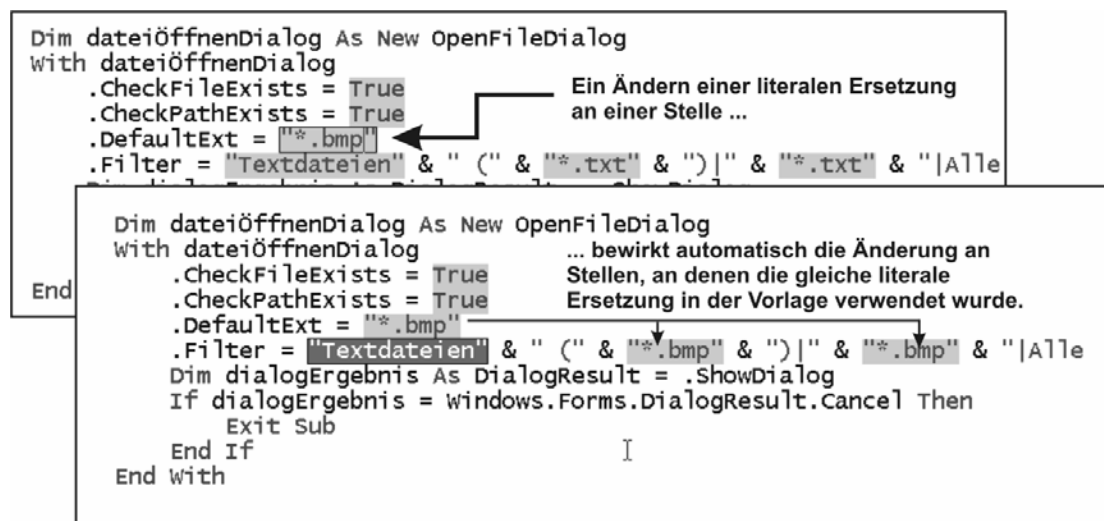


Abbildung 4.21: Das Prinzip von mehrfach eingesetzten literalen Ersetzungen

Für unsere Codeausschnitt-Vorlage bedeutet das, dass eine literale Ersetzung definiert werden muss, und diese an Stelle der eigentlichen Zeichenketten *.txt im Codetext verwendet wird. Die erste Stufe der Umgestaltung unserer Originalvorlage sieht damit aus, wie im anschließend gezeigten Listing:

TIPP: Falls Sie die Änderungen nicht alle eingeben möchten, ist das O.K. Die komplette XML-Datei befindet sich natürlich auch in den Begleitdateien, und Sie können sie später direkt Öffnen und der Ausschnittsbibliothek hinzufügen. Achten Sie aber darauf, die einzelnen Änderungsstufen an der XML-Datei wirklich gesehen und verstanden zu haben.

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <!-- An dieser Stelle den Namen des Codeausschnitts einfügen: -->
      <Title>
        Datei-Öffnen-Dialog (erweitert)
      </Title>
    </Header>
    <Snippet>

      <Declarations>
        <!-- An dieser Stelle folgen die Deklarationen für literale Ersetzungen -->
        <Literal>
          <ID>Standardfilter</ID>
          <ToolTip>Definiert die zu verwendende Standardfilterabkürzung (z.B. *.txt).</ToolTip>
          <Default>"*.txt"</Default>
        </Literal>
      </Declarations>

      <!-- An dieser Stelle den Namen einer benötigten
        Assembly-Referenz einfügen: -->
      <References>
        <Reference>
          <Assembly>System.Windows.Forms.dll</Assembly>
        </Reference>
      </References>

      <!-- Hier folgt die eigentliche Codedefinition: -->
      <Code Language="VB">
        <![CDATA[
          Dim dateiÖffnenDialog As New OpenFileDialog
          With dateiÖffnenDialog
            .CheckFileExists = True
            .CheckPathExists = True
            .DefaultExt = $Standardfilter$
            .Filter = "Textdateien (" & $Standardfilter$ & ")|" &
              $Standardfilter$ & "|Alle Dateien (*.*)|*.*"
          End With
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

```

        Dim dialogErgebnis As DialogResult = .ShowDialog
        If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If
    End With
]]>
</Code>
</Snippet>
</CodeSnippet>
</CodeSnippets>

```

An den in Fettschrift gehaltenen Stellen können Sie erkennen, wie das Zusammenspiel zwischen literalen Ersetzungen und deren Einbau in den eigentlichen Code funktioniert.

Zunächst einmal bedarf es eines Declaration-Blocks im XML-Code der Vorlage, in denen jede literale Ersetzung definiert wird. Jedes neue Literal, das Sie definieren möchten, leiten Sie anschließend, wie im Beispiellisting zu sehen, mit dem <Literal>-Tag ein. Anschließend folgt die Definition der literalen Ersetzung durch den <ID>-Tag, das dem Kind einen Namen gibt – in diesem Beispiel *Standardfilter*. Anstelle also später, im Code, direkt vordefinierten Text zu verwenden, setzen Sie später an den entsprechenden Stellen diesen Namen ein – und zwar in *\$*-Zeichen eingeklammert (also etwa *\$Standardfilter\$*).

Damit später beim ersten Einfügen des Ausschnittes in den Code an den entsprechenden Stellen überhaupt etwas erscheint, definieren Sie im gleichen Abschnitt auch einen Standardwert, der zum Einfügen kommt. Und dieser Standardwert ergibt sich aus den Angaben, die durch das <Default>-Tag definiert werden. Für das i-Tüpfelchen an Komfort sorgt schließlich noch eine einblendbare Tooltip-Beschreibung, die mit dem <Tooltip>-Tag festgelegt wird.

Ist die Definition dieser literalen Ersetzung vervollständigt, können Sie sie anschließend in die eigentliche Codevorlage wie beschrieben einbauen. In unserem Beispiel ist zunächst nur eine einzige Zeile davon betroffen:

```

        .Filter = "Textdateien (" & $Standardfilter$ & ")|" &
                $Standardfilter$ & "|Alle Dateien (*.*)|*.*"

```

Sie sehen, dass hier nun nicht mehr die festen Texte (**.txt*) direkt stehen, sondern diese durch die literalen Ersetzungen ausgetauscht wurden.

Auf diese Weise können Sie auch weitere literalen Ersetzungen in die Codeausschnittsvorlage einbauen – Sinn in unserem Beispiel macht es auch, das Wort *Textdateien* durch eine literale Ersetzung auszutauschen. Zwar kommt es im Ausschnitt nur ein einziges Mal vor; aber wie Sie gesehen haben, trägt nicht nur das Austauschen von Ersetzungen an mehreren Stellen gleichzeitig zum Komfort bei und spart Zeit. Auch das bloße Vorhandensein einer literalen Ersetzung an nur einer Stelle sorgt für Zeitersparnis und zusätzlichen Komfort, da sich eine entsprechende Stelle direkt mit der Tabulatortaste erreichen lässt. Unsere XML-Datei erfährt also einen weiteren »Umbau«:

```

<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
  xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>

```

```

<!-- An dieser Stelle den Namen des Codeausschnitts einfügen: -->
<Title>
  Datei-Öffnen-Dialog (erweitert)
</Title>

</Header>
<Snippet>

<Declarations>
  <!-- An dieser Stelle folgen die Deklarationen für literale Ersetzungen -->
  <Literal>
    <ID>Standardfilter</ID>
    <ToolTip>Definiert die zu verwendende Standardfilterabkürzung (z.B. *.txt).</ToolTip>
    <Default>"*.txt"</Default>
  </Literal>

  <Literal>
    <ID>Standardfiltername</ID>
    <ToolTip>Definiert den zu verwendende Standardfilternamen
      (z.B. "Textdateien" für *.txt).</ToolTip>
    <Default>"Textdateien"</Default>
  </Literal>
</Declarations>

<!-- An dieser Stelle den Namen einer benötigten
  Assembly-Referenz einfügen: -->
<References>
  <Reference>
    <Assembly>System.Windows.Forms.dll</Assembly>
  </Reference>
</References>

<!-- Hier folgt die eigentliche Codedefinition: -->
<Code Language="VB">
  <![CDATA[
    Dim dateiÖffnenDialog As New OpenFileDialog
    With dateiÖffnenDialog
      .CheckFileExists = True
      .CheckPathExists = True
      .DefaultExt = $Standardfilter$
      .Filter = $Standardfiltername$ & " (" & $Standardfilter$ & ")|" & _
        $Standardfilter$ & "|Alle Dateien (*.*)|*.*"
    End With
    Dim dialogErgebnis As DialogResult = .ShowDialog
    If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
      Exit Sub
    End If
  End With
]]>

</Code>
</Snippet>
</CodeSnippet>
</CodeSnippets>

```

Für den letzten Feinschliff in Sachen Ersetzungen fehlt nun noch die angekündigte Funktionalität, die dafür sorgt, dass auch die Überprüfung auf vorhandene Dateien bzw. Ordner in einem Rutsch ein- bzw. ausgeschaltet werden kann. Doch für diesen Zweck sind literale Ersetzungen nicht geeignet, denn: Hier müssen wir eine Objektvariable manipulieren – im konkreten Fall bedeutet das, eine boolesche Variable mit den entsprechenden Werten versehen. Und zu diesem Zweck kommen die so genannten *Objektersetzungen* bei Codeausschnittvorlagen zum Einsatz.

Objektersetzungen in Codeausschnittvorlagen

Das Prinzip von Objektersetzungen ist dem von literalen Ersetzungen sehr ähnlich. Objektersetzungen werden ebenfalls im Declarations-Teil der Vorlagen-XML-Datei definiert, nur mit einer leicht veränderten Syntax, wie der folgende Listingausschnitt unserer XML-Datei demonstriert:

```
.
.
.
</Header>
<Snippet>

  <Declarations>
    <!-- An dieser Stelle folgen die Deklarationen für literale Ersetzungen -->
    <Literal>
      <ID>Standardfilter</ID>
      <ToolTip>Definiert die zu verwendende Standardfilterabkürzung (z.B. *.txt).</ToolTip>
      <Default>"*.txt"</Default>
    </Literal>
    <Literal>
      <ID>Standardfiltername</ID>
      <ToolTip>Definiert den zu verwendende Standardfilternamen
        (z.B. "Textdateien" für *.txt).</ToolTip>
      <Default>"Textdateien"</Default>
    </Literal>

    <!-- An dieser Stelle folgen die Deklarationen für Objektersetzungen -->
    <Object>
      <ID>ÜberprüfeAufVorhandensein</ID>
      <Type>System.Boolean</Type>
      <ToolTip>Legt fest, ob auf Vorhandensein von Pfad und Datei geprüft werden soll.</ToolTip>
      <Default>True</Default>
    </Object>

  </Declarations>

  <!-- An dieser Stelle den Namen einer benötigten
    Assembly-Referenz einfügen: -->
.
.
.
```

Übrigens: XML-Puristen mögen mir in diesen Beispielen die Verwendung von deutschen Umlauten verzeihen – ich habe sie nur der Einfachheit halber verwendet.

Objektersetzungen werden, anders als literale Ersetzungen, mit dem <Object>-Tag eingeleitet. Aber auch Objektersetzungen bedürfen einer ID, die als Platzhalter im eigentlichen VB-Code eingesetzt werden kann. Zusätzlich zu literalen Ersetzungen müssen Sie aber ebenfalls bestimmen, um was für einen Objekttyp es sich bei der Objektersetzung handelt, und das geschieht mithilfe des <Type>-Tags, wie im Beispiellisting zu sehen. Wiederum genau so wie bei literalen Ersetzungen können Sie einen Standardwert mit <Default> und einen erklärenden Tooltip mit <Tooltip> definieren.

Eine auf diese Weise definierte Objektersetzung kommt dann im eigentlichen einzufügenden Code der Codeausschnittvorlage folgendermaßen zum Einsatz:

```

.
.
.
<!-- Hier folgt die eigentliche Codedefinition: -->
<Code Language="VB">
  <![CDATA[
    Dim dateiöffnenDialog As New OpenFileDialog
    With dateiöffnenDialog
      .CheckFileExists = $ÜberprüfeAufVorhandensein$
      .CheckPathExists = $ÜberprüfeAufVorhandensein$
      .DefaultExt = $Standardfilter$
      .Filter = $Standardfiltername$ & " (" & $Standardfilter$ & ")|" & $Standardfilter$ & _
        "|Alle Dateien (*.*)|*.*"
    Dim dialogErgebnis As DialogResult = .ShowDialog
    If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
      Exit Sub
    End If
  End With
]]>
.
.
.

```

Sie sehen: Die Anwendung von Objektersetzungen ist zu diesem Zeitpunkt dieselbe wie bei literalen Ersetzungen. Und im Übrigen: Unsere Codeausschnittvorlage ist nun fertig!

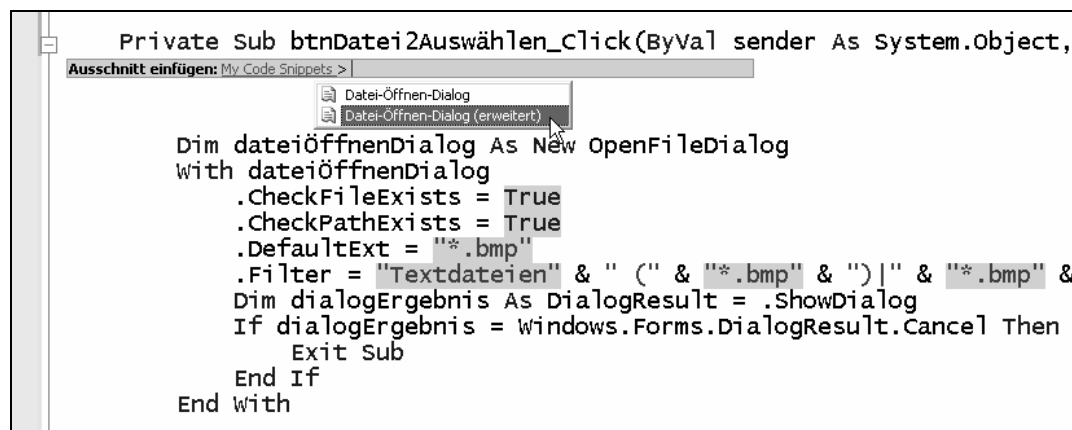


Abbildung 4.22: Die fertige Codeausschnittvorlage in Aktion!

Falls Sie die Änderungen bis zu diesem Zeitpunkt selbst durchgeführt haben, speichern Sie die XML-Vorlage am besten jetzt erst einmal ab – beispielsweise unter dem Namen *DateiÖffnenDialog-Ex.snippet*. Falls nicht, finden Sie die Vorlage unter dem gleichen Namen im Projektverzeichnis des Beispielprojektes.

Und dann wiederholen Sie das Spielchen aus Abschnitt »Hinzufügen einer neuen Snippet-Vorlage zur Snippet-Bibliothek (Codeausschnittsbibliothek)« ab Seite 131.

Das Einfügen des neuen Codeausschnittes und der eingefügte Ausschnitt selbst sollten anschließend so aussehen, wie es Abbildung 4.22 zeigt.