Markt+Technik

Davis Chapman

...in 21 Tagen

Visual C++.NET

Der optimale Weg zum Programmierprofi

Windows-Applikationen mit .NET

Mehr zu C++ und den MFC

.NET Framework SDK
das gesamte Buch im HTML-Format
Sourcecode aller Programmbeispiele
u.v.a.m

SAMS

Inhaltsverzeichnis

Tag 15 Eigene ActiveX-Steuerelemente

Tag 16 Funktionen für Webbrowser

Tag 17 Multitasking

Tag 18 Internet und Netzwerke

Tag 19 Verwalteter Code

Tag 20 ATL-Komponenten

Tag 21 Interaktion mit Visual Basic .NET und C# .NET-Komponenten

---

Einführung

Willkommen zu Visual C++ .NET. In den kommenden 21 Tagen lernen Sie, wie man die Funktionsmerkmale einsetzt, die Microsoft seiner C++-Entwicklungsumgebung spendiert hat, damit Sie fortschrittliche Anwendungen für das Betriebssystem Windows erstellen können. Darüber hinaus erhalten Sie eine Einführung in die Erstellung von Anwendungen für die neue Microsoft-Plattform .NET. Als die Entwickler von Microsoft mit der Entwicklung von Visual C++ anfingen, beschlossen sie, ihren Weltklasse-C++- Compiler zu nehmen und zu einer Entwicklungsumgebung und einem Werkzeugsatz zu verbinden, mit denen Entwickler Windows-Anwendungen mit bisher beispielloser Geschwindigkeit und Leichtigkeit erzeugen konnten. Seit dieser ersten Version hat Microsoft die Werkzeuge ständig verbessert, um die Anwendungsentwicklung noch einfacher zu gestalten. Sobald Microsoft neue Technologien für die Windows-Plattform eingeführt hatte, wurden diese Werkzeuge auch Bestandteil der Visual C++-Umgebung, um die neuesten Technologien sofort in Anwendungen einbinden zu können.

Wenn Sie heute erstmals mit C++ Bekanntschaft schließen, sollte Sie das nicht abschrecken. Dieses Buch ermöglicht es Ihnen gleichermaßen, die Programmiersprache C++ und das Erstellen von Anwendungen mit den Werkzeugen von Visual C++ zu erlernen. Ich kann zwar keine ausführliche Einführung in die Programmiersprache C++ bieten wie Bücher, die sich der Sprache selbst widmen, doch ich habe im gesamten Buch versucht, die Funktionsweise der Sprache zu erklären.

C++-Exkurs: Hilfe zur Programmierung

Wenn Sie das Buch durcharbeiten, stoßen Sie auf speziell gekennzeichnete Abschnitte, die den verschiedenen Aspekten der Programmiersprache C++ gewidmet sind, die Sie im Beispielcode finden. Wenn Sie bereits mit C++ vertraut sind und dieses Buch lesen, um das Werkzeug Visual C++ kennen zu lernen, können Sie diese Abschnitte überspringen.

MFC-Exkurs: Eine schnelle Betrachtung von Klassen

Sollten Sie bereits die früheren Versionen dieses Buchs kennen, fällt Ihnen sicher auf, dass es in großen Teilen neu geschrieben wurde. Das Ziel der neuen Version besteht darin, nicht nur eine Einführung in die verschiedenen Tools und Ressourcen zu geben, die Sie für das Erstellen von Anwendungen mit Visual C++ verwenden können, sondern auch ein tiefer gehendes Verständnis für die Möglichkeiten zu vermitteln, die sich Ihnen mit den behandelten Funktionsmerkmalen erschließen.

Ich habe sogar eigene Abschnitte eingefügt, die sich weiter gehend mit den zur Implementierung bestimmter Funktionalitätsbereiche verwendeten Klassen beschäftigen. Wie die speziellen Abschnitte über die Programmiersprache C++ können Sie auch diese Abschnitte über die Klassenbibliothek Microsoft Foundation Classes (MFC) problemlos überspringen und später zu ihnen zurückkehren, wenn Sie bereit sind, tiefer unter die Oberfläche von Visual C++ zu dringen. So wird Ihnen dieses Buch noch lange über die 21 Tage hinaus nützlich sein.

Wie dieses Buch organisiert ist

Das Buch gliedert sich in drei Wochen, die jeweils aus sieben in sich geschlossenen Kapiteln für die einzelnen Tage bestehen. Aber selbst, wenn das Buch in Wochen organisiert ist, gilt das nicht gleichermaßen für die Themen.

Am Ende der einzelnen Tage finden Sie ein paar Fragen, um Ihren Wissensstand zu überprüfen, und ein oder zwei Übungen, mit denen Sie das behandelte Thema vertiefen können. Falls Sie nicht alles auf Anhieb wissen, ist das auch nicht weiter schlimm. Die Antworten zu den Fragen und kurze Anleitungen zu den Übungen finden Sie im Anhang A, »Workshop-Auflösung«.

In der ersten Woche geht es um die Grundlagen beim Erstellen von Anwendungen mit Visual C++. Sie lernen, wie man die Editoren einsetzt, um Anwendungsfenster zu gestalten. Weiterhin lernen Sie die verschiedenen Steuerelemente kennen, die Ihnen als Windows-Anwendungsentwickler zur Verfügung stehen. Sie erfahren eine Menge über die Entwicklungsumgebung von Visual C++ und die Tools, die sie mitbringt.

Mit Beginn der zweiten Woche werden die Themen komplizierter, und Sie steigen tiefer in die Programmierung ein. Sie erstellen zwar Ihre Anwendungen mit den gleichen Werkzeugen, jedoch wird der Programmcode anspruchsvoller. In der zweiten Woche beschäftigen Sie sich bereits mit etwas komplizierteren Themen wie der Anzeige von Grafiken und dem Erstellen von SDI- und MDI-Anwendungen. Gegen Ende dieser Woche arbeiten Sie erstmals mit Datenbanken.

In der dritten Woche lernen Sie, ActiveX-Steuerelemente zu erstellen und zu verwenden. Sie erfahren, wie man Multitasking-Anwendungen realisiert, die mehrere Aufgaben gleichzeitig ausführen. Schließlich lernen Sie einige der Grundlagen der Verwendung von Visual C++ auf der neuen Microsoft-Plattform .NET und wie sie sich auf die Ansätze zur Gestaltung und Programmierung von Anwendungen auswirkt.

Nachdem Sie die dritte Woche abgeschlossen haben, können Sie sich in die Welt der Windows-Programmierung mit Visual C++ stürzen. Nunmehr verfügen Sie über die Fertigkeiten und das erforderliche Know-how, um die meisten der heute verfügbaren Windows-Anwendungen erstellen zu können.

Konventionen

Wenn Sie das Buch durchblättern, fallen Ihnen sicherlich eine Reihe von Konventionen auf, die Sie dabei unterstützen sollen, die behandelten Themen möglichst gut zu erfassen.

Der gesamte Quellcode ist in dieser Schrift gesetzt, wie es Listing 0.1 zeigt. Dazu gehören der vollständige Quellcode aus den Anwendungen, die Sie erstellen, und die Demonstrationsbeispiele, die den Einsatz der verschiedenen Funktionen zeigen. Syntax- Variablen, die Sie durch einem passenden Wert ersetzen müssen, sind in dieser Schrift gedruckt. Wenn Sie neuen Code in einer Funktion hinzufügen oder Änderungen an bereits bestehendem Code vornehmen, sind die hinzuzufügenden oder zu ändernden Code-Zeilen fett gedruckt.

Listing 0.1: Etwas Beispielcode

```
1: void main()
2: {
3:    // Wenn Sie Code in einem existierenden Codefragment hinzufügen
4:    // oder ändern, kennzeichne ich die Zeilen durch Fettdruck
5: }
```

Besondere Informationen erkennen Sie an diesen grafischen Elementen:

Dieses Symbol kennzeichnet Text, der den vorherigen Code erklärt.

Dieses Symbol kennzeichnet einen neuen Begriff, der in einem Absatz definiert und erklärt wird. Der neu definierte Begriff ist kursiv gedruckt.

Hinweise bringen eine tiefer gehende Erläuterung eines Themas oder erklären interessante oder wichtige Punkte. Sie bieten auch Informationen, die Ihnen bei der Vermeidung von Problemen helfen können oder die Sie bei der Verwendung der beschriebenen Funktionsmerkmale beachten sollten.

Tipps bieten Ratschläge oder schlagen einfachere oder alternative Vorgehensweisen vor.

Warnungen machen Sie auf mögliche Probleme oder Gefahren aufmerksam und wie Sie sie vermeiden oder beseitigen.

Genug gesagt! Sie haben dieses Buch nicht gekauft, um etwas darüber zu lesen, sondern um damit zu arbeiten und zu lernen, wie man mit Visual C++ Windows-Anwendungen erstellt. Also frisch ans Werk. Blättern Sie um und vertiefen Sie sich in die Welt der Visual C++-Programmierumgebung.

Feedback

Falls Sie Fragen oder Anregungen haben, positive oder negative Kritik loswerden wollen, sonstiges Wichtiges oder einfach Nettigkeiten, so freuen sich Verlag und Fachlektor der vorliegenden Ausgabe über eine E-Mail von Ihnen. Senden Sie diese doch bitte an Jochen.Ruhland@mut.de.

Über den Autor

Davis Chapman begann Computer zu programmieren, als er an seinem Master-Abschluss in musikalischer Komposition arbeitete. Während er Anwendungen für Computermusik schrieb, entdeckte er, dass ihm die Gestaltung und Entwicklung von Computersoftware Freude bereitete. Es dauerte nicht lange und ihm wurde klar, dass seine Chance aufs tägliche Brot viel größer war, wenn er bei seinem neu entdeckten Talent blieb und seinen schwer verdienten Status als »brotloser Künstler« zu Gunsten eines Teilzeithobbys aufgab. Seitdem hat sich Davis auf die Kunst der Softwaregestaltung und -entwicklung konzentriert, mit einem zentralen Schwerpunkt auf Client-/Server- und Web-/Internet- Technologien. Davis war der Hauptautor von Building Secure Applications with Visual Basic, Visual C++ 6 in 21 Tagen, Web Development with Visual Basic 5 und Building Internet Applications with Delphi 2. Davis leistete außerdem Beiträge als Autor zu MFC Programming with Visual C++ 6 Unleashed, Special Edition, Using Active Server Pages und Running a Perfect Web Site, Second Edition. Er lebt und arbeitet seit 12 Jahren als Berater in Dallas, Texas. Davis ist unter davis@chaperada.net erreichbar.

Über die Fachlektoren

Tony Davis ist der Mitbegründer und Präsident von MillenniSoft, Inc., einer Firma für Software-Entwicklung und -Beratung in Dallas, Texas. Er ist seit über 14 Jahren Softwarearchitekt, -designer und -entwickler für Business-Anwendungen und auf Windows- Client-/Server-Anwendungen auf Basis von C++ spezialisiert.

Erik Thompson arbeitet als leitender Software-Ingenieur, spezialisiert auf C++, COM, ATL und Middle Tier-Applikationen. Er arbeitet seit Mitte der 80er Jahre mit Computern. Wenn er nicht für seine Arbeit programmiert, ist er als Editor und Autor von Artikeln über neue Technologien und Interessensbereiche bei The Code Project (www.codeproject.com) tätig. Erik lebt in Redmond, Washington.

Fachlektor der deutschsprachigen Ausgabe

Jochen Ruhland studierte Mathematik und Informatik in Kassel und arbeitet bei einer Internet-Firma im Netzwerkbereich. Seit 20 Jahren ist er mit Computern, deren Verbindung und deren Programmierung befasst. Er ist Autor mehrerer Bücher über Betriebssysteme und Programmierung und lebt in München.

Woche 1 im Überblick

Willkommen in der Welt von Visual C++. In den nächsten drei Wochen lernen Sie, wie man verschiedene Anwendungen mit diesem extrem flexiblen und umfangreichen Programmierwerkzeug erstellt. Jeden Tag lernen Sie einen anderen Bereich der Funktionalität kennen und erfahren, wie man diese in eigenen Anwendungen realisiert. Zum besseren Verständnis sind die behandelten Themen mit einer Fülle von Beispielen unterlegt. Außerdem erstellen Sie immer eine Anwendung, um das Ganze in die Tat umzusetzen. Eine neue Technik lernt man am besten, wenn man selbst damit arbeitet. Learning by doing - das ist der Weg, den Sie mit diesem Buch einschlagen.

Im Verlauf der ersten Woche schließen Sie zunächst einmal Bekanntschaft mit den Grundlagen, die zum Erstellen einer Anwendung mit Visual C++ gehören. Am ersten Tag beschäftigen Sie sich deshalb mit der Entwicklungsumgebung von Visual C++ und erstellen damit eine einfache Anwendung.

Am zweiten Tag, »Fehlerbeseitigung in Ihrer Anwendung«, lernen Sie einige wertvolle Tools und Techniken für die Fehlerbeseitigung in den Anwendungen kennen, die Sie mit Visual C++ erstellen. Sie werden lernen, wie man spezielle Debug-Makros einfügt, die dabei helfen, mögliche Fehler in Ihrer Anwendung ans Licht zu bringen, bevor Sie die Anwendung an Benutzer weitergeben. Sie werden außerdem lernen, wie man den integrierten Debugger im Visual Studio Integrated Development Environment (IDE) verwendet und wie man mit ihm Probleme im Code findet und berichtigt.

> *Warum sollten Sie lernen, Fehler in Programmen zu beseitigen, bevor Sie ein Programm geschrieben haben? Weil Sie, indem Sie lernen, wie Sie Fehler beseitigen und Ihren Code schrittweise durchlaufen, in die Lage versetzt werden, Ihren Code in Aktion zu sehen und ein besseres Verständnis dafür erlangen, wie er funktioniert. Wenn Sie außerdem anfangen, Anwendungen mit mehr als einer oder zwei Code-Zeilen zu schreiben, schleichen sich leicht Fehler ein, und Sie müssen wissen, wie Sie diese finden und korrigieren.*

Am dritten Tag, »Steuerelemente«, lernen Sie mehr über die Besonderheiten beim Erstellen von Visual C++-Anwendungen. Es geht um Standardsteuerelemente, die man in nahezu allen Windows-Anwendungen antrifft. Sie erfahren, wie man diese Steuerelemente im Anwendungsfenster unterbringt und mit ihnen umgeht.

Der vierte Tag, »Maus und Tastatur«, zeigt, wie Ihre Anwendungen auf Maus- und Tastatur-Ereignisse

reagieren können. Sie ermitteln, wo sich der Mauszeiger im Fenster Ihrer Anwendung befindet. Weiterhin erfahren Sie, wie man herausfindet, welche Tasten der Benutzer auf der Tastatur betätigt und wie man auf derartige Benutzeraktionen reagieren kann.

Schwerpunktthema des fünften Tages sind die »Timer« (Zeitgeber) in einer Visual C++- Anwendung. Dabei lassen Sie mehrere Timer gleichzeitig laufen und es wird gezeigt, wie man die einzelnen Timer auseinander halten kann.

Am sechsten Tag, »Dialogboxen«, bauen Sie zusätzliche Fenster in Ihre Anwendung ein und erfahren, wie man mithilfe dieser Fenster Informationen vom Benutzer einholt. Vordefinierte Dialogfelder erlauben es, dem Benutzer einfache Entscheidungsfragen zu stellen, und Sie erzeugen auch eigene Dialogfelder, um detailliertere Angaben abzurufen.

In der Lektion 7 erstellen Sie »Menüs«, fügen sie in Ihre Anwendung ein und lernen, wie man Funktionen in der Anwendung über die hinzugefügten Menüs aufrufen kann.

Tag 1

Erste Schritte mit Visual C++

In den nächsten drei Wochen lernen Sie, wie man die verschiedenartigsten Anwendungen mit Visual C++ von Microsoft erstellt. Sie vollziehen alle Schritte selbst nach und gewinnen auf diese Weise eine gehörige Portion Programmiererfahrung. Deshalb frisch ans Werk!

Als eine der ersten Unterscheidungen müssen Sie verstehen, dass die Programmiersprache Visual C++ unabhängig von der Integrierten Entwicklungsumgebung (Integrated Development Environment - IDE) Visual Studio ist. Wann immer Werkzeuge und Editoren erwähnt werden, bezieht sich das auf die Visual Studio-IDE. Wenn Sie nur Visual C++ gekauft haben und nicht das ganze Visual Studio-Paket, arbeiten Sie trotzdem in der Visual Studio-IDE. Das ist die Entwicklungsumgebung und der Werkzeugsatz, die von allen in Visual Studio enthaltenen Programmiersprachen einschließlich Visual C++ verwendet werden. Kurz, in den nächsten 21 Tagen arbeiten Sie mit den Visual Studio-Werkzeugen und -Editoren, aber der Programmcode, den die Visual Studio-Werkzeuge erzeugen und den Sie selbst eingeben, ist die Programmiersprache Visual C++. Es ist wichtig, sich der Unterscheidung zwischen beidem bewusst zu sein, da Sie andere Werkzeuge für die Arbeit mit der Programmiersprache Visual C++ verwenden, andererseits jedoch auch andere Programmiersprachen mit den Visual Studio-Werkzeugen programmieren können.

Die erste Lektion konzentriert sich auf die Entwicklungsumgebung Visual Studio und einige Werkzeuge, die für den Aufbau von Anwendungen zur Verfügung stehen. Auch wenn Visual C++ mehr Werkzeuge bietet, als Sie bei der Entwicklung einer Anwendung wahrscheinlich jemals brauchen (und die Sie auch nicht an einem einzigen Tag beherrschen lernen), beschränken wir uns auf die hauptsächlichen Werkzeuge, die das ganze Buch hindurch zur Anwendung kommen und die Sie auch in nahezu jeder Anwendung, die Sie mit Visual C++ erstellen, einsetzen.

Diese Lektion beschäftigt sich mit folgenden Themen:

Die Hauptbereiche der Entwicklungsumgebung von Visual C++:

- Der Anwendungs-Assistent und wie man ihn einsetzt, um die Infrastruktur für Anwendungen zu erstellen
- Der Dialog-Editor und wie man ihn einsetzt, um Dialogfenster zu zeichnen (so wie man Fenster mit Visual Basic, PowerBuilder oder Delphi erstellt)
- Die Eigenschaftenansicht (Properties Pane) und wie man damit arbeitet, um die Anwendungsfenster mit Funktionalität auszustatten

1.1 Die Entwicklungsumgebung von Visual C++

Bevor wir die kurze Rundreise durch die Entwicklungsumgebung von Visual C++ beginnen, sollten Sie Visual C++ auf Ihrem Computer starten, damit Sie unmittelbar nachvollziehen können, wie die einzelnen Bereiche angeordnet sind und wie man diese Anordnung an die eigenen Anforderungen anpasst.

Nach dem Start von Visual Studio, der visuellen Entwicklungsumgebung von Microsoft, erscheint ein Fenster wie in Abbildung 1.1. Jeder Bereich erfüllt in der Umgebung von Visual Studio einen bestimmten Zweck. Man

kann diese Bereiche neu anordnen, um die Entwicklungsumgebung Visual Studio an die eigenen Erfordernisse anzupassen.

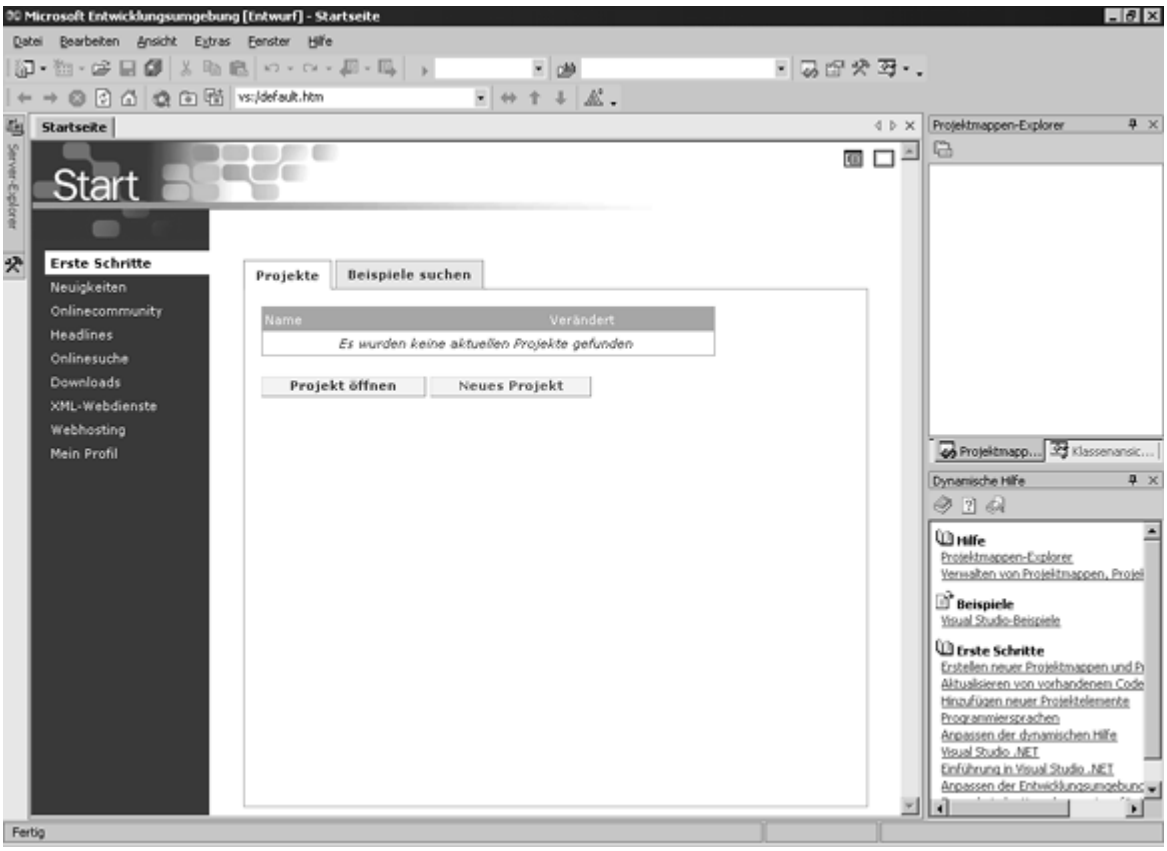


Abbildung 1.1: Der Eröffnungsbildschirm von Visual C++

Der Browser-Startbildschirm erzeugt, wie auch andere Funktionen innerhalb der Visual Studio-Umgebung, Scripte auf Ihrem Computer und führt sie aus. Wenn Sie ein Virenschutzprogramm auf dem gleichen Rechner wie Visual Studio .NET oder Visual C++ .NET installiert haben, können bei der Ausführung dieser Scripte Warnungen vor potenziellen Viren angezeigt werden.

Der Projektmappen-Explorer

Beim ersten Start von Visual C++ sieht es so aus, dass der große Bereich auf der rechten Seite der IDE zwar eine Menge Platz beansprucht, aber wenig Informationen bietet. Dieser so genannte Projektmappen-Explorer bildet den Ausgangspunkt, um zu den verschiedenen Teilen der Entwicklungsprojekte zu navigieren. Der Projektmappen- Explorer entspricht dem Project Explorer in früheren Versionen von Visual Basic, Visual InterDev oder Visual J++. Diese Ansicht gestattet es, die Teile einer Anwendung in drei verschiedenen Modi zu betrachten:

- In der Klassenansicht können Sie auf C++-Klassenebene durch den Quellcode navigieren und ihn bearbeiten.
- Die Ressourcenansicht erlaubt es, die verschiedenen Ressourcen in der Anwendung aufzusuchen und zu bearbeiten. Dazu gehören die Entwürfe von Dialogfenstern, Symbolen und Menüs. Diese Ansicht ist nur verfügbar, wenn ein Projekt geöffnet ist.
- Die Projektmappen-Explorer-Ansicht als Projektmappen-Explorer beschriftet, bietet eine Übersicht über die Dateien, aus denen eine Anwendung besteht. Man kann die Dateien anzeigen und in ihnen navigieren. Sie werden später sehen, dass Sie mehrere Projekte gleichzeitig geöffnet haben können. Dann können Sie innerhalb des Projektmappen-Explorers zwischen den Projekten navigieren.

Denken Sie daran, dass die anfängliche Sammlung von Ansichten in diesem Bereich nur die Standard-Anordnung ist. Sie können die ursprünglichen drei Ansichten in jeden beliebigen Bereich der IDE ziehen und der Gruppierung selbst neue Ansichten hinzufügen.

Der Ausgabebereich

Der Ausgabebereich ist möglicherweise beim ersten Start von Visual C++ noch nicht sichtbar. Nachdem Sie Ihre erste Anwendung kompiliert haben, erscheint der Ausgabebereich am unteren Rand der Visual Studio-Umgebung und bleibt geöffnet, bis man ihn tatsächlich schließt. Im Ausgabebereich stellt die Visual Studio-IDE alle Informationen bereit, die für Sie relevant sind. Hier können Sie den Fortschritt beim Kompilieren verfolgen und Warnungen und Fehlermeldungen einsehen. Auch der Debugger von Visual C++ zeigt im Ausgabebereich alle Variablen mit ihren aktuellen Werten an, wenn man den Code schrittweise abarbeitet. Nachdem Sie den Ausgabebereich oder eine der ursprünglich in diesem Bereich gruppierten Ansichten geschlossen haben, wird er automatisch geöffnet, wenn Visual C++ eine neue Meldung für Sie hat.

Der Editorbereich

Der Editorbereich ist im Grunde der gesamte Bereich der Entwicklungsumgebung, der nicht anderweitig von Ansichten, Menüs oder Symbolleisten eingenommen wird. Hier erledigen Sie alle Bearbeitungsaufgaben beim Einsatz von Visual C++, hier erscheinen die Fenster des Quellcode-Editors, wenn Sie den C++-Quellcode bearbeiten, und hier wird auch der Dialog-Designer angezeigt, wenn Sie ein Dialogfeld entwerfen. Der Editorbereich dient auch dem Symbol-Editor zur Anzeige, wenn man Symbole für den Einsatz in den Anwendungen gestaltet.

Menüleisten

Beim ersten Start von Visual C++ erscheinen unmittelbar unterhalb der Menüleiste zwei Symbolleisten. Weitere sind verfügbar. Sie können sie dem eigenen Arbeitsstil entsprechend anpassen und neu erzeugen. Zu Beginn sind folgende Symbolleisten zu sehen:

- Die Standardsymbolleiste enthält die meisten der Standardwerkzeuge für das Öffnen und Speichern von Dateien, Ausschneiden, Kopieren, Einfügen und verschiedene allgemein gebräuchliche Befehle.
- Die andere Symbolleiste enthält passende Funktionen für Ihre Aktivitäten im Editorbereich. Wenn Sie beispielsweise Visual C++ zum ersten Mal öffnen, sehen Sie eine Symbolleiste, mit der Sie durch die Dokumentation von Visual C++ navigieren können. Wenn Sie C++-Code bearbeiten, sehen Sie eine Symbolleiste, die häufig beim Editieren von Code gebrauchte Funktionen enthält. Wenn Sie ein Dialogfenster erstellen, enthält die Symbolleiste Funktionen für Größen- und Ausrichtungskontrollen.

Die Umgebung von Visual Studio neu arrangieren

Im Visual Studio gibt es zwei einfache Wege, um die Entwicklungsumgebung neu zu arrangieren. Erstens kann man mit der rechten Maustaste auf den Bereich der Symbolleisten klicken. Diese Aktion öffnet das in Abbildung 1.2 dargestellte Kontextmenü, über das man verschiedene Symbolleisten und Fensterbereiche ein- und ausschalten kann.

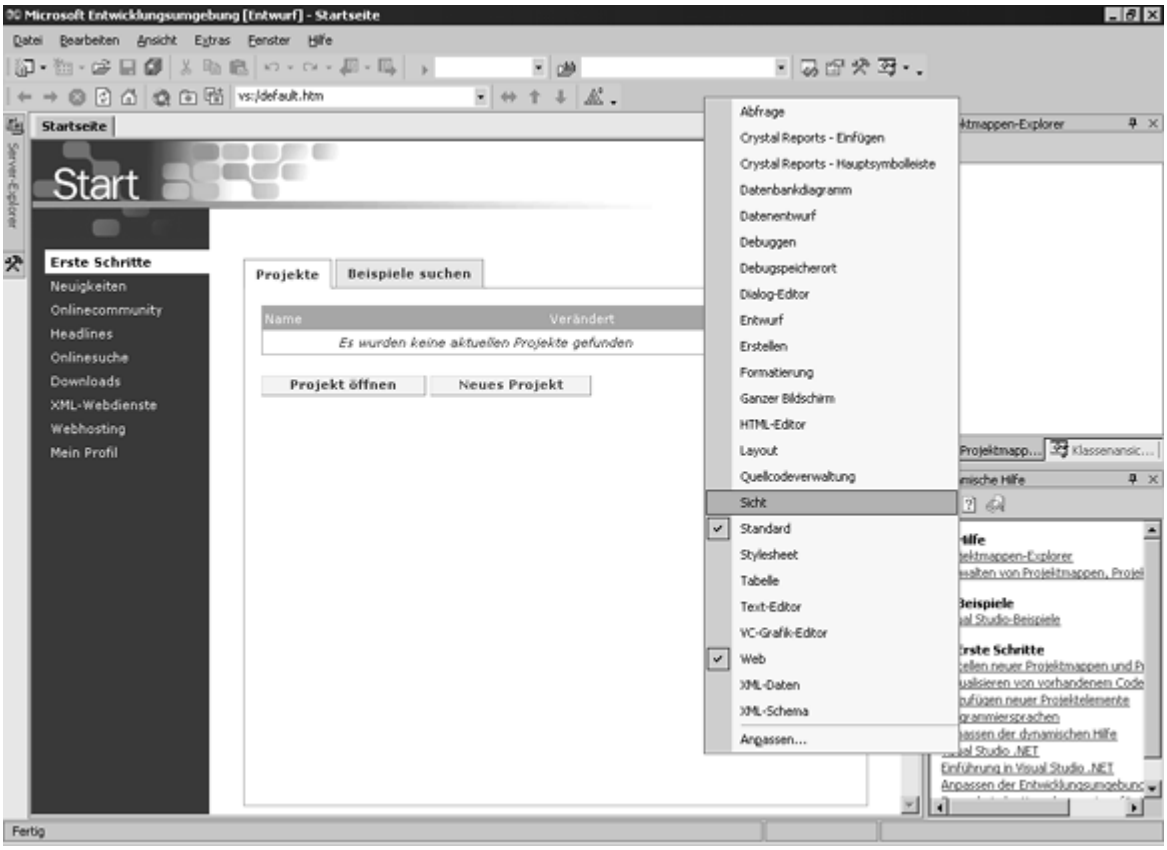


Abbildung 1.2: Kontextmenü zum Ein- und Ausschalten von Symbolleisten

Ein anderer Weg, um die Entwicklungsumgebung in einfacher Weise neu anzuordnen, besteht darin, die Balken aus horizontalen Linien am linken Ende der Symbolleisten oder Fensterbereiche mit der Maus zu ziehen. Die Symbolleisten kann man von der Position, an der sie momentan verankert sind, wegziehen und unverankerte Symbolleisten erzeugen. Man kann diese (und die Fensterbereiche) an jeden Rand von Visual Studio ziehen, um sie an der neuen Position anzudocken. Selbst wenn die Symbolleisten verankert sind, kann man sie mithilfe der Docking-Balken nach links oder rechts an den gewünschten Platz verschieben (oder nach oben oder unten, wenn sie links oder rechts verankert sind).

Die Ansichten zu verschieben ist etwas schwieriger. In der neuen Visual Studio-IDE können Sie die einzelnen Registerkarten von einer Ansicht trennen und zu eigenen Ansichten umwandeln oder sie in neuen Kombinationen zusammenfügen. Wenn Sie zwei oder mehr Registerkarten gruppiert haben, besitzt diese Gruppe eine Reihe von Registern, mit denen Sie die anzuzeigende Ansicht auswählen können.

In den Arbeits-, Eigenschaften- und Ausgabebereichen können Sie das gesamte Fenster mithilfe der Titelleiste innerhalb der Developer Studio-Umgebung verschieben. Diese Fenster sind im Developer Studio verankert.

1.2 Das erste Projekt

Als Einstieg in Visual C++ erstellen Sie eine einfache Anwendung mit zwei Schaltflächen, wie es Abbildung 1.3 zeigt. Die erste präsentiert dem Benutzer eine einfache Begrüßungsmeldung (siehe Abbildung 1.4), während die zweite Schaltfläche die Anwendung schließt. Um diese Anwendung zu erstellen, sind folgende Dinge zu realisieren:

1. Einen neuen Arbeitsbereich anlegen.

2. Mit dem Anwendungs-Assistent das Anwendungsgerüst erstellen.
3. Das vom Anwendungs-Assistent automatisch erstellte Dialogfeld neu gestalten, um der Anwendung das gewünschte Aussehen zu geben.
4. Den C++-Code hinzufügen, um die Begrüßung für den Benutzer anzuzeigen.
5. Ein neues Symbol für die Anwendung erzeugen.

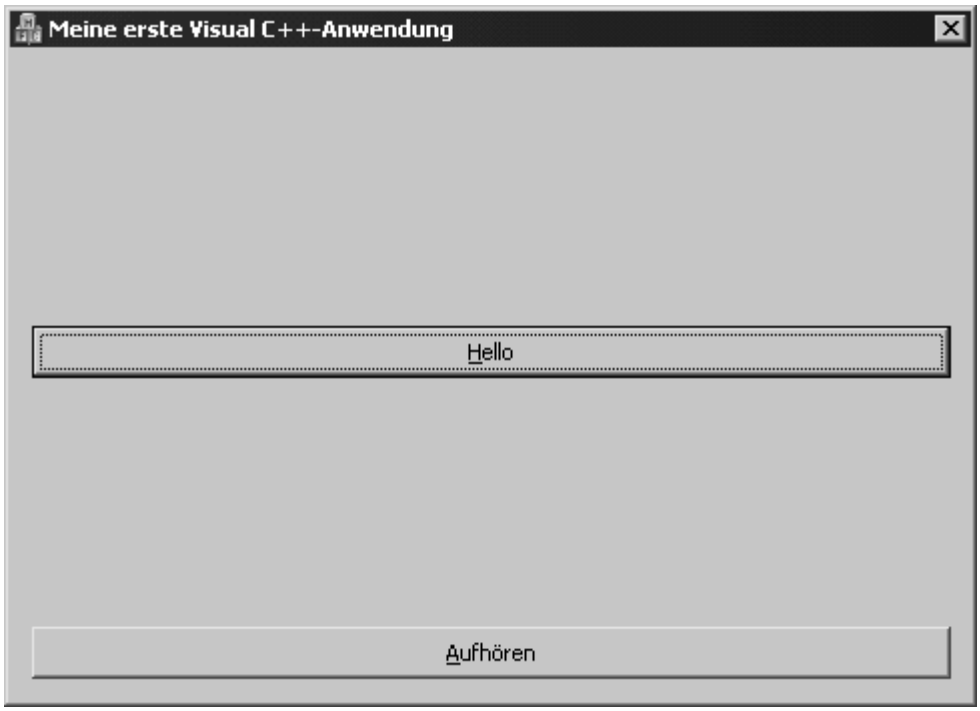


Abbildung 1.3: Ihre erste Anwendung mit Visual C++

Abbildung 1.4: Wenn der Benutzer auf die erste Schaltfläche klickt, erscheint eine einfache Begrüßung.

Den Projekt-Arbeitsbereich anlegen

Für jedes zu entwickelnde Anwendungsprojekt ist in Visual C++ ein eigener Projekt- Arbeitsbereich erforderlich. Zum Arbeitsbereich gehören die Verzeichnisse, in denen der Quellcode der Anwendung untergebracht ist, sowie die Verzeichnisse, in denen die verschiedenen Konfigurationsdateien für das Erstellen gespeichert sind. Mit den folgenden Schritten erzeugt man einen neuen Projekt-Arbeitsbereich:

1. Klicken Sie in der VS (Visual Studio) .NET-Startseite auf Neues Projekt (vergewissern Sie sich, dass links auf der Seite die Registerkarte Erste Schritte ausgewählt ist). Der in Abbildung 1.5 gezeigte Assistent für Neues Projekt wird angezeigt.

Wenn Sie die Startseite geschlossen haben, wählen Sie Ansicht /Webbrowser / Browser anzeigen aus dem Menü, um die VS.NET-Startseite wieder zu öffnen.

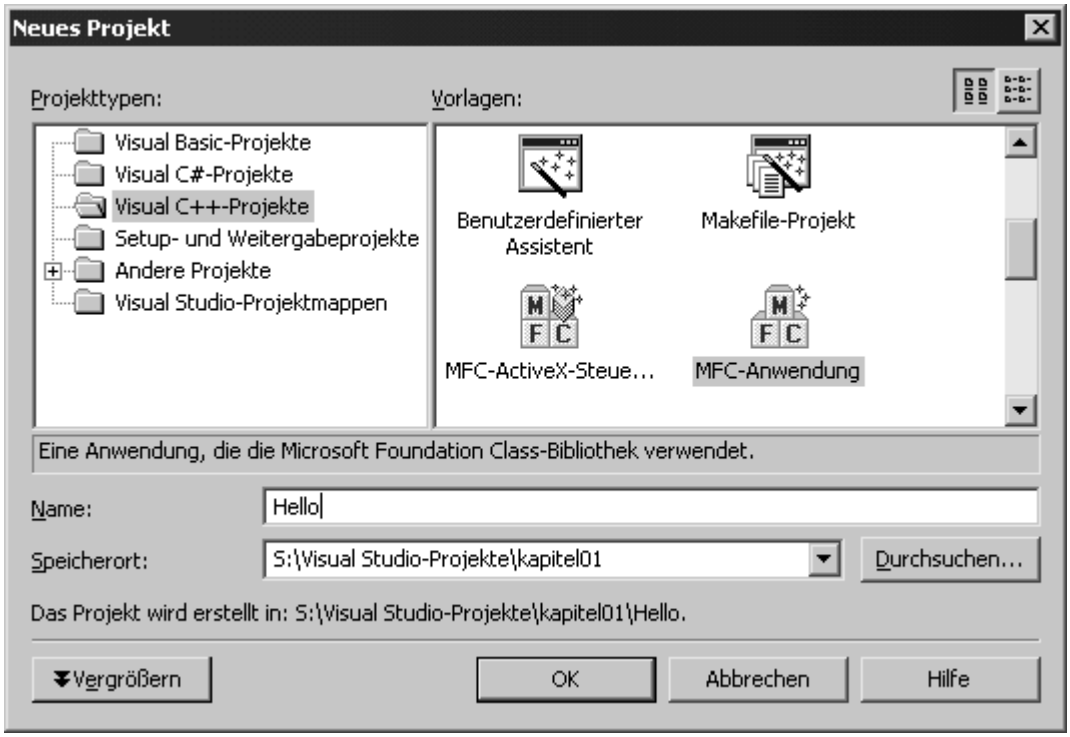


Abbildung 1.5: Der Assistent für die Auswahl neuer Projekte

2. Wählen Sie Visual C++-Projekte aus dem Baum Projekttypen aus (welche Projetktypen sie genau sehen, hängt von dem von Ihnen erworbenen Programmpaket und den gewählten Installationsoptionen ab). Wählen Sie MFC-Anwendung auf der rechten Seite des Dialogfelds Neues Projekt im Vorlagen-Bereich (Sie werden nach unten scrollen müssen, um die Vorlage MFC-Anwendung zu finden).
3. Geben Sie Hello im Feld Name ein. Das ist der Name des Projekts.
4. Klicken Sie auf OK. Der Assistent für neue Projekte tut zwei Dinge: Er weist ein Projektverzeichnis zu (im Feld Speicherort angegeben; das Verzeichnis wird jedoch erst dann tatsächlich angelegt, wenn der Anwendungs-Assistent beendet ist) und startet den Anwendungs-Assistenten.

Den Anwendungsrahmen mit dem Anwendungs-Assistenten erstellen

Der MFC Anwendungs-Assistent stellt Ihnen eine Reihe von Fragen über den Typ der Anwendung, die Sie erstellen möchten und über Merkmale und Funktionalität, die Sie dazu brauchen. Anhand dieser Angaben erzeugt der Assistent ein Anwendungsgerüst, das Sie sofort kompilieren und ausführen können. Dieses Gerüst bietet die erforderliche Infrastruktur, auf der Sie Ihre Anwendung aufbauen. Führen Sie die folgenden Schritte aus, damit Sie die Arbeitsweise kennen lernen:

1. Wählen Sie auf der linken Seite des Assistenten Anwendungstyp aus. Sie erhalten ein paar Optionen für den Typ der zu erstellenden Anwendung. Geben Sie auf der rechten Seite an, dass Sie eine auf Dialogfeldern basierende Anwendung erstellen möchten (siehe Abbildung 1.6).

Abbildung 1.6: Den Anwendungstyp festlegen

2. Wählen Sie auf der linken Seite BenutzeroberflächenFeatures (Merkmale für die Benutzerschnittstelle) aus. In diesem Bereich des Anwendung-Assistenten können Sie das Aussehen des Hauptfensters Ihrer Anwendung festlegen. Löschen Sie im Feld unten den Projektnamen (Hello) und geben Sie den Titel ein, der in der Titelleiste des Hauptfensters Ihrer Anwendung erscheinen soll. Für dieses Beispiel geben Sie Meine erste Visual C++-Anwendung ein (siehe Abbildung 1.7).

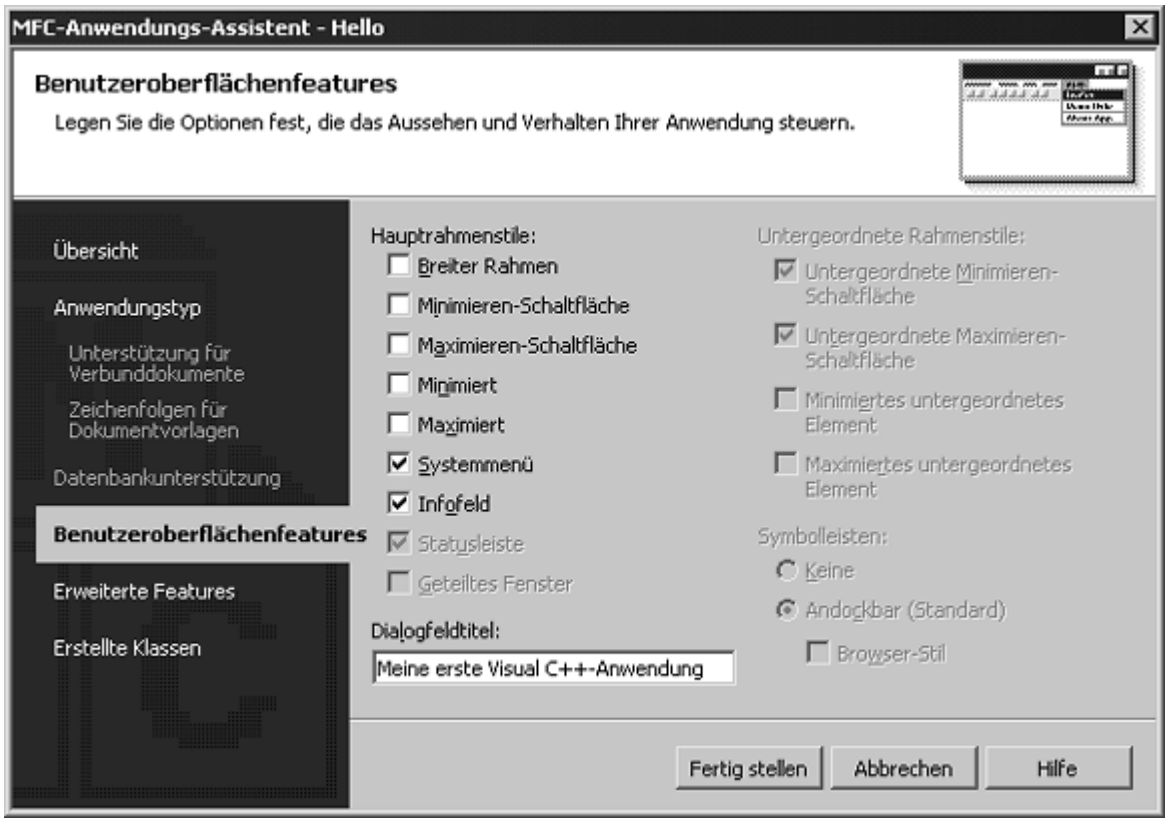


Abbildung 1.7: Den Titel der Anwendung festlegen

3. Klicken Sie auf Fertig stellen, damit der MFC Anwendungs-Assistent Ihren Anwendungsrahmen erstellt.
4. Nachdem der Anwendungs-Assistent das Projektgerüst erstellt hat, gelangen Sie in die Umgebung von Visual Studio zurück. Der Arbeitsbereich zeigt nun eine Baumansicht der Klassen in Ihrem Projektgerüst, wie es aus Abbildung 1.8 hervorgeht. Es erscheint auch das Hauptfenster (ein Dialogfeld) im Editorbereich von Visual Studio.

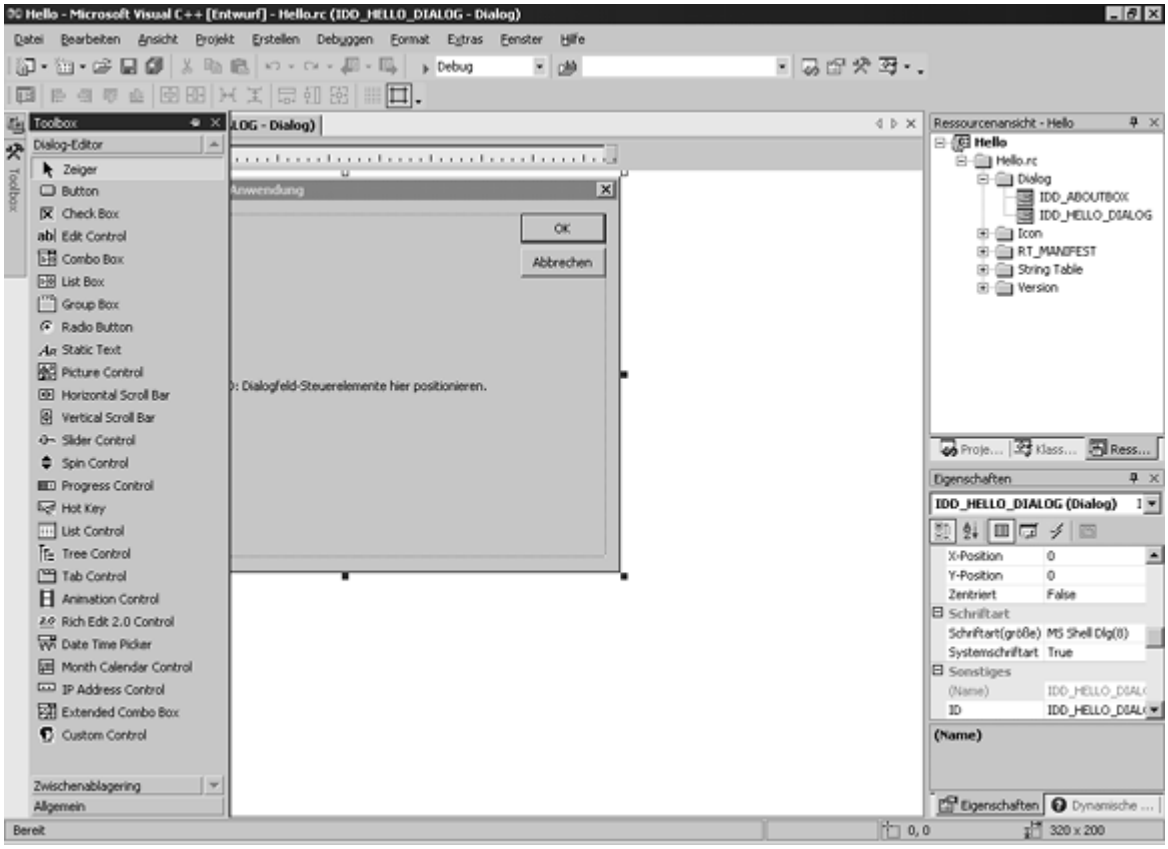


Abbildung 1.8: Der Arbeitsbereich mit einer Baumansicht der Projektklassen

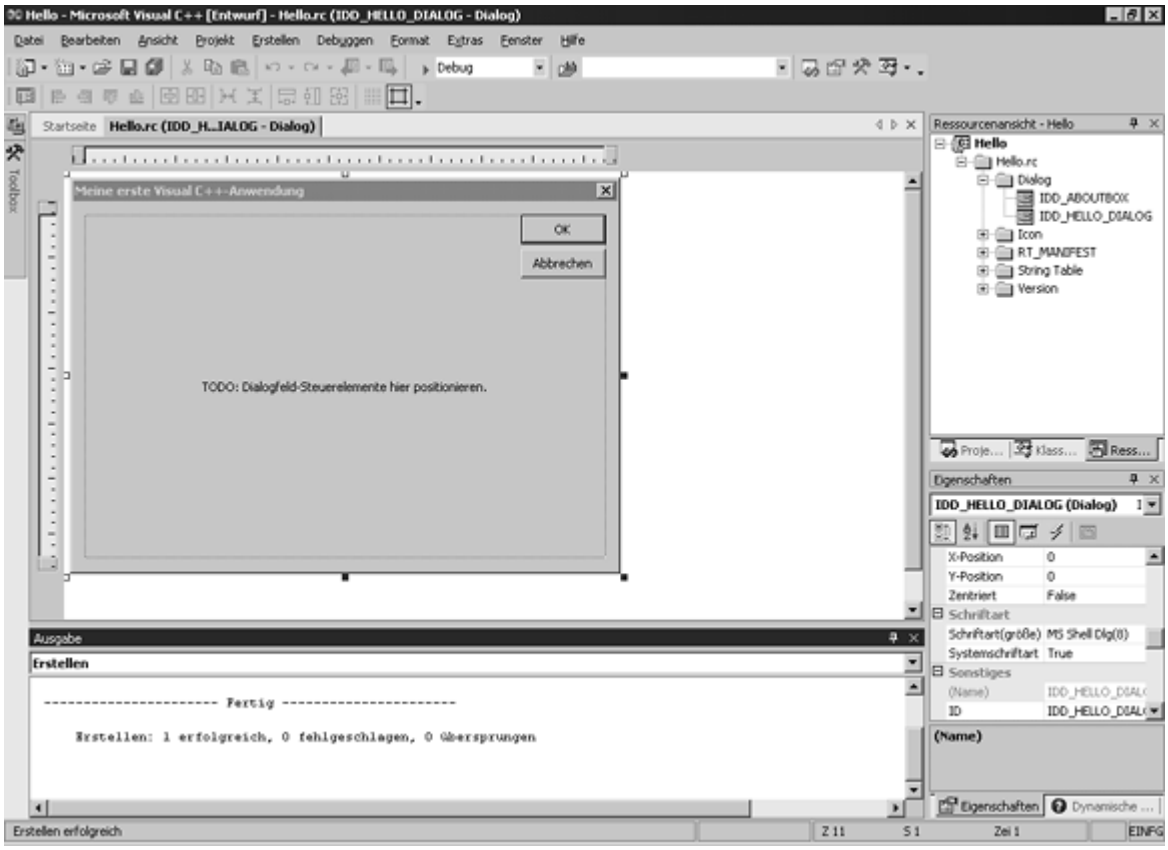


Abbildung 1.9: Der Ausgabebereich zeigt alle Compiler-Fehler an.

5. Wählen Sie Erstellen / Projektmappe erstellen, um die Anwendung zu kompilieren.
6. Wenn der Visual C++-Compiler die Anwendung erstellt, erscheinen Fortschrittsanzeigen und andere Compiler-Meldungen im Ausgabebereich. Nachdem die Anwendung erstellt ist, sollte der Ausgabebereich eine Meldung enthalten, dass keine Fehler oder Warnungen vorhanden sind (siehe Abbildung 1.9).
7. Wählen Sie Debuggen / Starten, um Ihre Anwendung zu starten.
8. Die Anwendung präsentiert ein Dialogfeld mit einer TODO-Meldung und den beiden Schaltflächen OK und Abbrechen, wie es Abbildung 1.10 zeigt. Um die Anwendung zu schließen, können Sie auf eine der beiden Schaltflächen klicken.

Abbildung 1.10: Die ausgeführte Anwendung ohne Änderungen des Projektgerüsts

MFC-Exkurs: Wie funktionieren Windows-Anwendungen?

Dieser Abschnitt heute und an den folgenden Tagen erforscht die Microsoft Foundation Classes (MFC) und wie Sie mit ihrer Hilfe schnell und einfach Windows-Anwendungen erstellen. Diese Informationen müssen Sie nicht unbedingt kennen, um durch das Buch zu kommen, also können sie übersprungen werden. Sie werden viel über MFC lernen und wie man damit Anwendungen erstellt, ohne diese Abschnitte zu lesen. Wenn Sie allerdings verstehen möchten, wie die Dinge unter der Oberfläche funktionieren, finden Sie diese Informationen hier.

Die Microsoft Foundation Classes sind eine Reihe von C++-Klassen, die einen großen Teil der an der Interaktion mit den Windows-Betriebssystemen beteiligten Programmierung abstrahieren. Wenn Sie schon Windows-Anwendungen mit anderen visuellen Programmierwerkzeugen wie Visual Basic oder Delphi erstellt haben, haben Sie mit Komponenten gearbeitet, die Ihnen vorgepackte Funktionalität boten. Ganz ähnlich arbeitet MFC. Wie in Visual Basic haben Sie eine Befehlsschaltfläche, die Sie auf einem Fenster platzieren können. Das Fenster wiederum besitzt verschiedene Eigenschaften, die Sie einstellen können, um sein Aussehen und Verhalten zu bestimmen. MFC besitzt eine Klasse - in diesem Fall CButton -, in der die Funktionalität einer Befehlsschaltfläche verkapselt ist.

Wenn Sie schon einmal versucht haben herauszufinden, wie Windows-Programme ohne einen Anwendungsrahmen wie MFC in C/C++ geschrieben sind, wissen Sie, dass alle Windows-Programme mit einer einzelnen Funktion namens WinMain beginnen. Sie ist der Startpunkt der Anwendung und erzeugt ihr Hauptfenster. Danach startet die Funktion eine Schleife zum Empfang von Ereignismeldungen, die sie dann zur Verarbeitung an die entsprechenden Funktionen zur Ereignisbehandlung in einer Kontrolle oder in einem Fenster weiterleitet (siehe Abbildung 1.11). Wenn keine passende Funktion zur Behandlung der Meldung existiert, wird sie an eine vom Betriebssystem kontrollierte Standard-Funktion weitergeleitet.

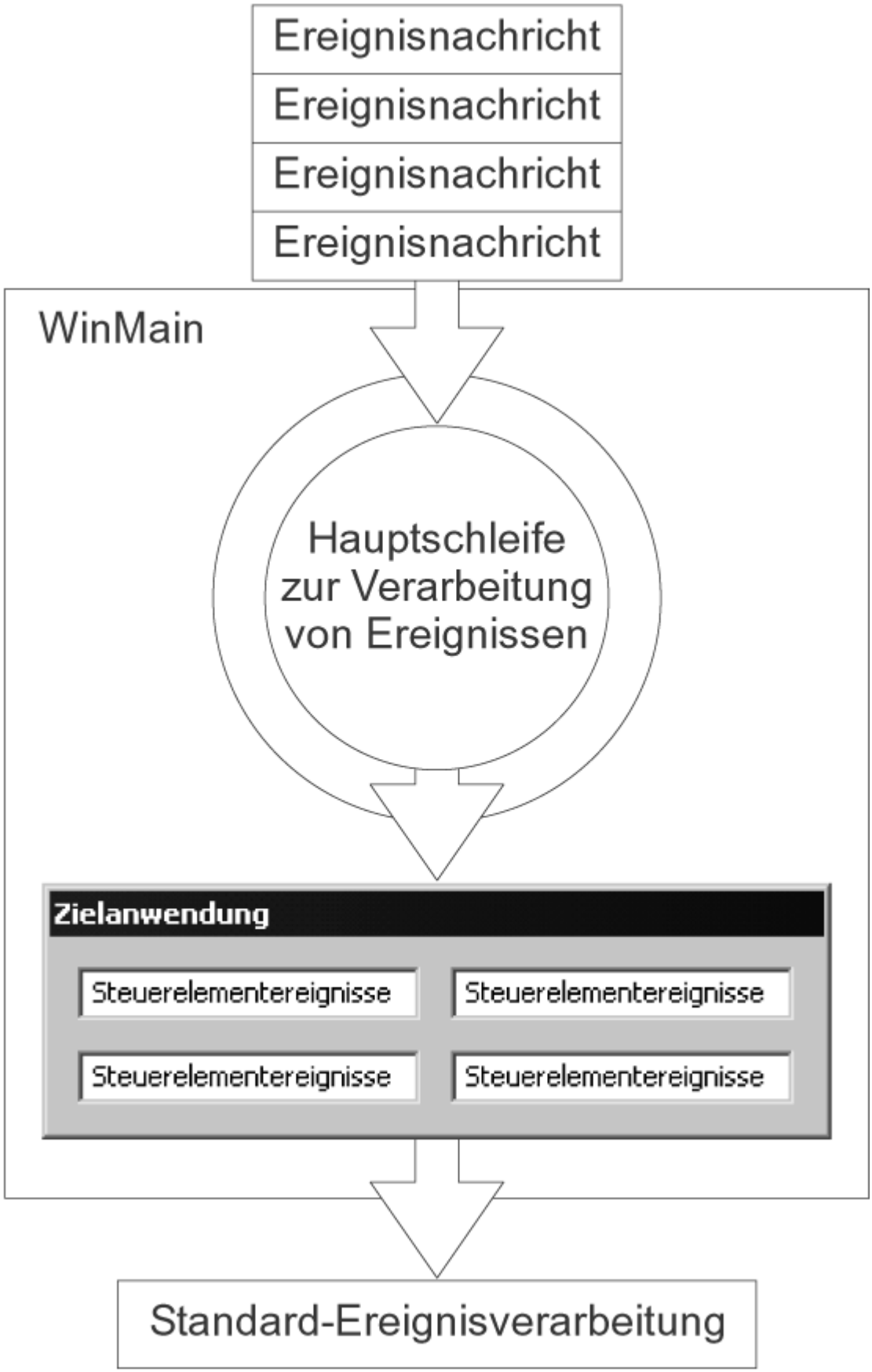


Abbildung 1.11: Der grobe Verarbeitungsweg von Anwendungen in Windows

MFC nimmt Ihnen den Großteil der Erstellung von Windows-Anwendungen ab. Wenn Sie weit genug bohren, finden Sie die Funktion WinMain tief im MFC-Quellcode vergraben. Was Sie sehen, wenn Sie die Klassenansicht in der Workshop-Ansicht betrachten, sind drei Klassen: CAboutDlg, CHelloApp und

CHelloDlg. Die zweite dieser Klassen, CHelloApp, ist aus der Klasse CWinApp abgeleitet. Diese Klasse verkapselt einen Großteil der WinMain-Funktionalität und nimmt damit einiges an Plackerei bei der Programmierung von Windows-Anwendungen von Ihren Schultern. Die Klasse CWinApp enthält auch einen Großteil der allgemeinen Anwendungsfunktionalität, die weder zu einer Kontrolle noch zu einem anderen spezialisierteren Objekt gehört, wie zum Beispiel die Funktionalität zum Lesen und Schreiben von Konfigurationsinformationen in der Registry-Datenbank und die Funktionalität für die Befehlszeilenverarbeitung.

Die anderen beiden Klassen, CAboutDlg und CHelloDlg, sind beide aus der Klasse CDialog abgeleitet, bei der es sich um eine spezialisierte Ableitung der eher als Vielzweck-Klasse zu sehenden CWnd handelt. Die Klasse CWnd verkapselt einen Großteil der Fenster- Funktionalität, während sich die Klasse CDialog auf Dialogfenster konzentriert. In den nächsten Tagen werden Sie viel mehr über die Klasse CDialog und ihren Vorfahren CWnd erfahren.

Wenn man eine Klasse als Ableitung einer anderen Klasse bezeichnet, bedeutet das, dass die erste Klasse die Eigenschaften und Funktionalitäten der zweiten ererbt. Man sagt auch, die zweite Klasse ist der Vorfahre der ersten. Die tatsächliche Bedeutung ist, dass die ableitende Klasse (diejenige, die erbt) die Vorfahr-Klasse als Teil ihrer selbst besitzt. Ohne dass der abgeleiteten Klasse Funktionalität hinzugefügt wird, besitzt sie schon die gesamte Funktionalität der Vorfahr- Klasse. Wegen der ganzen von der Basis-/Vorfahr-Klasse bereitgestellten Funktionalität kann die abgeleitete Klasse sich auf die Aspekte konzentrieren, in denen sie sich von ihrem Vorfahren unterscheidet. So wird die zur Bereitstellung der Funktionalität der abgeleiteten Klasse notwendige Code-Menge stark reduziert.

1.3 Das Anwendungsfenster gestalten

Nachdem Sie nun über ein lauffähiges Anwendungsgerüst verfügen, richten Sie Ihr Augenmerk auf das Fensterlayout der Anwendung. Das Hauptfenster kann zwar bereits im Editorbereich zur weiteren Bearbeitung zur Verfügung stehen, dennoch müssen Sie zum Dialogfeld im Arbeitsbereich navigieren, um das Fenster in zukünftigen Entwicklungsschritten leichter aufzufinden. Um das Layout für das Anwendungsdialogfeld zu gestalten, führen Sie die folgenden Schritte aus:

1. Aktivieren Sie im Arbeitsbereich die Registerkarte Ressourcenansicht, wie es Abbildung 1.12 zeigt.

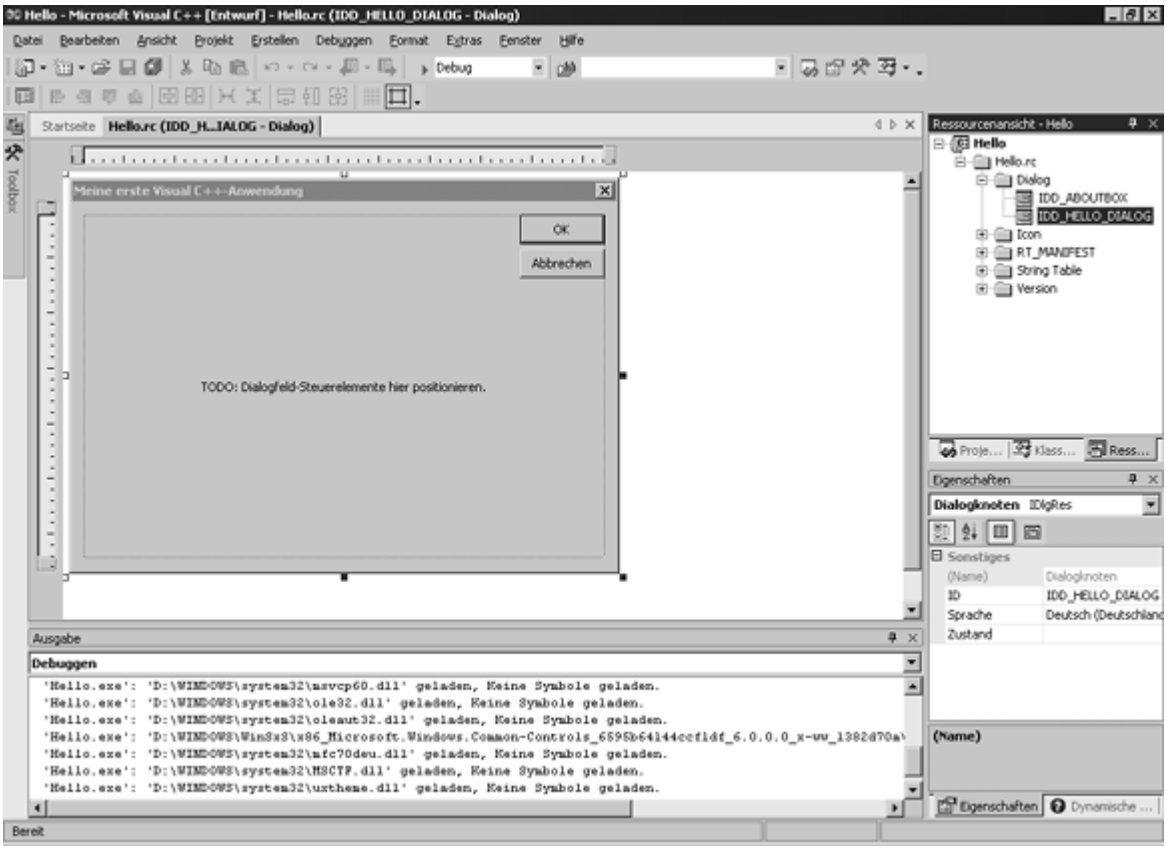


Abbildung 1.12: Die Registerkarte Ressourcenansicht im Arbeitsbereich

2. Erweitern Sie die Baumansicht der Ressourcen, um die verfügbaren Dialogfelder anzuzeigen. Jetzt können Sie auf das Dialogfeld IDD_HELLO_DIALOG doppelklicken, um das Fenster im Editorbereich von Visual Studio zu öffnen.
3. Klicken Sie auf den im Dialogfeld angezeigten Text und löschen Sie ihn mit der (Entf)-Taste.
4. Markieren Sie die Schaltfläche Abbrechen, ziehen Sie sie an den unteren Rand des Dialogfelds und verändern Sie die Größe der Schaltfläche, sodass sie die gesamte Breite des Layoutbereichs des Fensters einnimmt (siehe Abbildung 1.13).

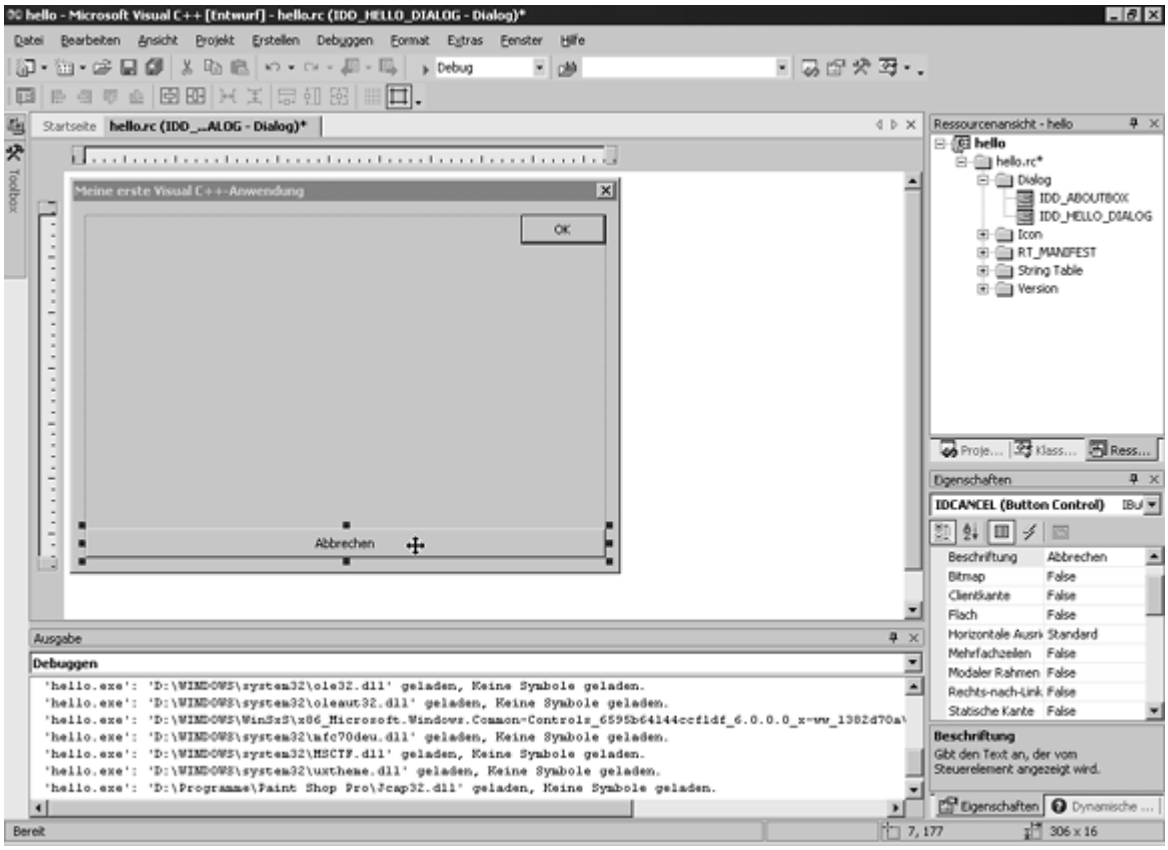


Abbildung 1.13: Die Schaltfläche Abbrechen platzieren

5. Ändern Sie in der Eigenschaftenansicht (rechts direkt unter der Projektmappen- Explorer-Ansicht) den Wert im Feld Beschriftung auf &Aufhören.
6. Verschieben Sie die Schaltfläche OK in die Mitte des Fensters und ändern Sie die Größe der Schaltfläche entsprechend Abbildung 1.14.

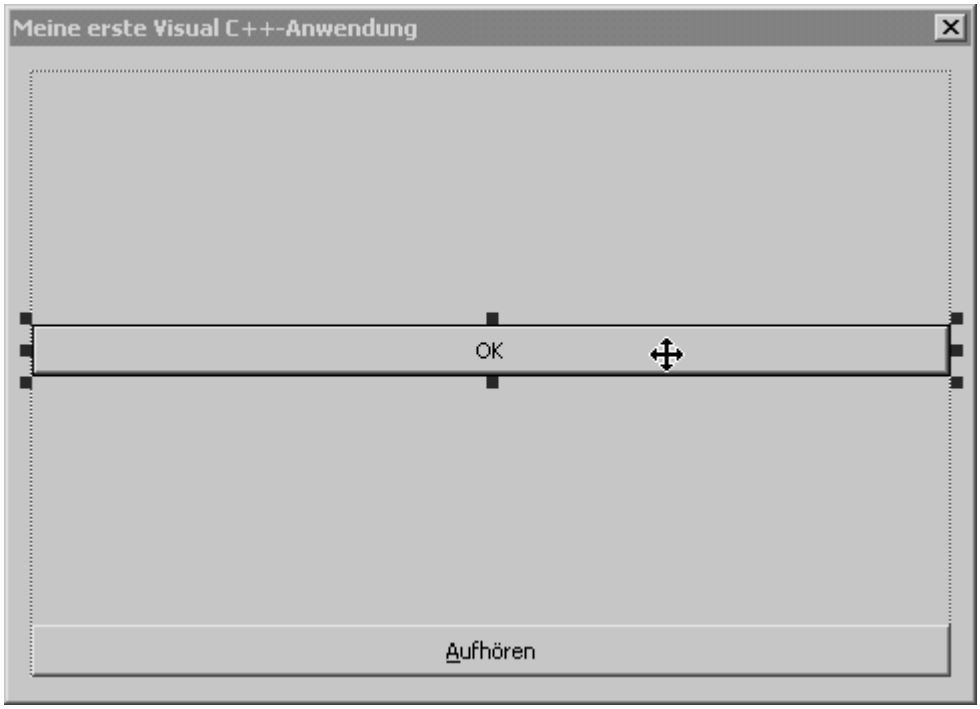


Abbildung 1.14: Die Schaltfläche OK platzieren

7. Ändern Sie im Eigenschaftenbereich die ID in IDHELLO und den Titel in &Hello.

8. Wenn Sie nun die Anwendung kompilieren und ausführen, entspricht sie dem gerade vorgenommenen Entwurf, wie es Abbildung 1.15 zeigt.

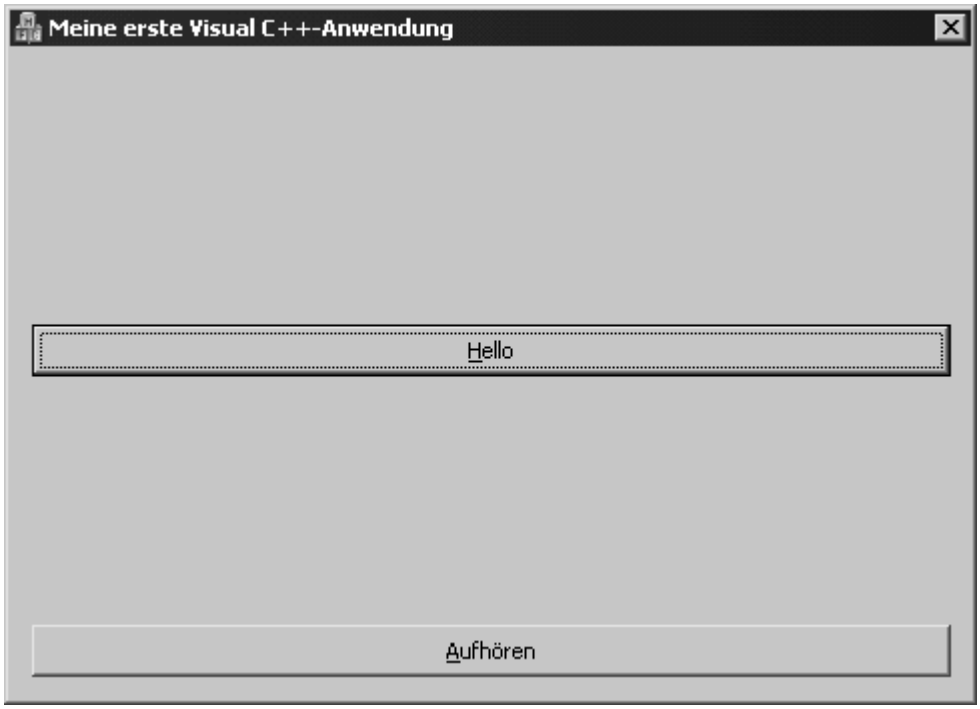


Abbildung 1.15: Die neu gestaltete Anwendung ausführen

Für die Neulinge in der Anwendungsprogrammierung für Windows: Das kaufmännische Und (&) im Titel gibt an, dass es sich bei dem darauf folgenden Buchstaben um das Tastenkürzel für diese Schaltfläche, dieses Menü oder diese Kontrolle handelt. In älteren Windows-Versionen wurde das A in Aufhören unterstrichen dargestellt (unter bestimmten Umständen, beispielsweise bei gedrückter (Alt)-Taste, wird das A immer noch unterstrichen dargestellt, dies hängt auch von den gewählten Darstellungseigenschaften des Systems ab). Das bietet dem Benutzer eine Abkürzung zum Auslösen dieser Schaltfläche, indem er einfach die (Alt)-Taste zusammen mit dem Zeichen direkt hinter dem kaufmännischen Und drückt. Die Schaltfläche (oder das Menü oder die Kontrolle) wird auf diese Weise ausgelöst. Damit hat der Benutzer eine Alternative zur Verwendung der Maus.

Wenn Sie die Anwendung ausprobieren, stellen Sie fest, dass Klicken auf die Schaltfläche Aufhören die Anwendung wie in der ursprünglichen Version schließt. Allerdings bewirkt die Schaltfläche Hello nun überhaupt nichts mehr, da Sie die ID der Schaltfläche geändert haben. MFC-Anwendungen enthalten im Quellcode eine Reihe von Makros, die auf der Basis der ID und der Ereignisnachricht eines Steuerelements festlegen, welche Funktionen in der Anwendung aufzurufen sind. Da Sie die ID der Schaltfläche Hello geändert haben, wissen diese Makros nicht mehr, welche Funktion beim Anklicken der Schaltfläche aufzurufen ist.

1.4 Code in die Anwendung aufnehmen

Über die Eigenschaftenansicht von Visual C++ können Sie das Dialogfeld mit Code verbinden. Mit der

Eigenschaftenansicht erstellen Sie die Tabelle der Windows- Nachrichten, welche die Anwendung empfangen kann, einschließlich der zu verarbeitenden Funktionen. Auf diese Angaben greifen die MFC-Makros zurück, um die Funktionalität mit Windows-Steuerelementen zu verbinden. Die Funktionalität für Ihre erste Beispielanwendung weisen Sie in folgenden Schritten zu:

1. Um der Schaltfläche Hello eine Funktion zuzuordnen, markieren Sie die Schaltfläche Hello und wählen Sie dann Steuerelementereignisse (das Blitzzeichen) in der Eigenschaftenansicht (siehe Abbildung 1.16).

Abbildung 1.16: Steuerelementereignisse in der Eigenschaftenansicht auswählen

2. Wählen Sie aus der Nachrichtenliste BN_CLICKED. Der Wertebereich neben der Ereignis-ID sollte zu einer Drop-Down-Liste werden. Klicken Sie auf den Pfeil der Liste. Im Drop-Down-Teil der Liste erscheint ein Vorschlag für eine Funktion namens OnBnClickedHello (siehe Abbildung 1.17). Wählen Sie diesen Funktionsnamen aus dem Drop-Down-Bereich. Die Funktion OnBnClickedHello wird im Editorbereich geöffnet.

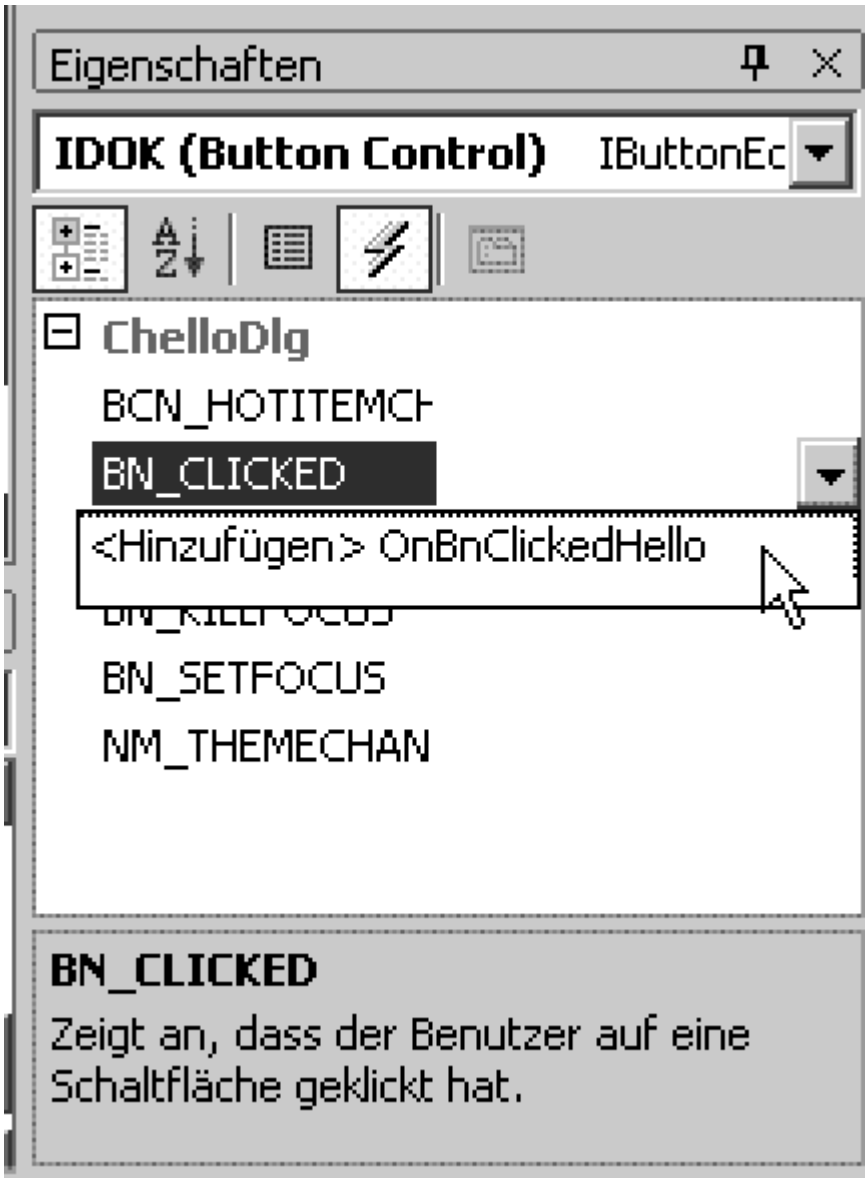


Abbildung 1.17: Die vorgeschlagene Ereignisfunktion

3. Tragen Sie den fett gedruckten Code aus Listing 1.1 unmittelbar unter der TODO- Kommentarzeile ein, wie es Abbildung 1.18 verdeutlicht. Vergewissern Sie sich, dass Sie den Code genau wie gezeigt eingeben.

Listing 1.1: HELLODLG.CPP - die Funktion OnBnClickedHello

```
1: void CHelloDlg::OnBnClickedHello(void)
2: {
3:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:    // Benachrichtigung ein.
5:    // Benutzer begrüßen
6:   MessageBox("Hallo. Dies ist meine erste Visual C++ Anwendung!");
7: }
```

4. Wenn Sie die Anwendung kompilieren und ausführen, sollte nach Anklicken der Schaltfläche Hello die in Abbildung 1.19 gezeigte Nachricht erscheinen.

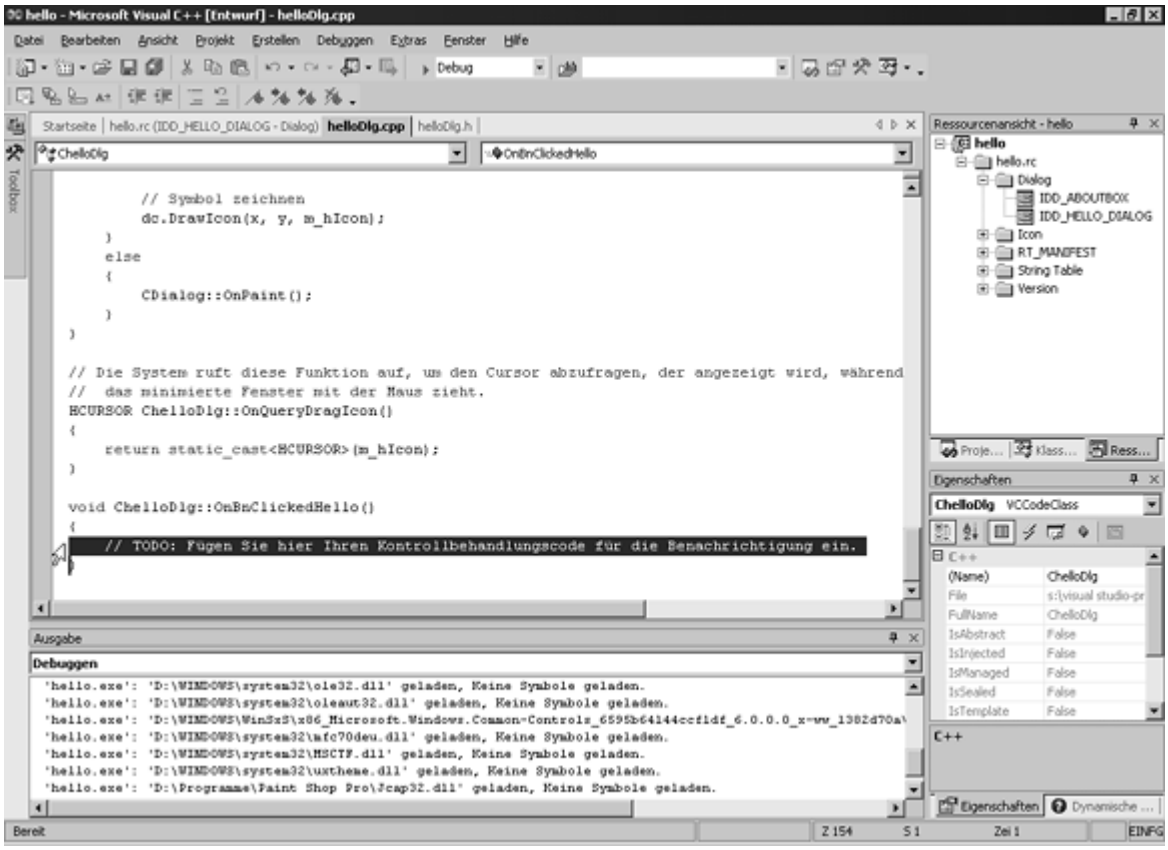


Abbildung 1.18: In der Quellcode-Ansicht fügen Sie Listing 1.1 ein.

Beachten Sie, dass der Editor beginnt, Ihnen Funktionsdefinitionen zu zeigen, während Sie Code eingeben. Diese Funktion nennt sich IntelliSense. Sie versucht zu erraten, welche Funktion oder Objekt-Methode/-Eigenschaft Sie eingeben, und ermöglicht Ihnen, die Funktion, Methode oder Eigenschaft aus einer Liste verfügbarer Funktionen etc. zu wählen. Das kann helfen, Tippfehler zu verhindern. Wenn Sie den Funktionsnamen eingegeben und die Klammer geöffnet haben, werden die Funktionsparameter aufgelistet, wobei derjenige hervorgehoben ist, den Sie gerade eingeben. IntelliSense ist als Programmierhilfe gedacht, damit Sie wissen, welche Funktionen und Eigenschaften verfügbar sind und welche Parameter eine Funktion benötigt, die Sie aufrufen. Diese Funktionalität wurde erstmals in Visual Basic 5 eingeführt und dann einige Jahre später zu Visual C++ 6 hinzugefügt.

Abbildung 1.19: Die Anwendung begrüßt Sie nun.

Wenn Sie beim Versuch, Ihre Anwendung zu kompilieren, Fehler erhalten, vergewissern Sie sich noch einmal, dass Sie den Code exakt wie in Listing 1.1 eingegeben haben. C++ unterscheidet zwischen Groß- und Kleinschreibung, sodass schon ein Fehler verursacht wird, wenn man den Funktionsnamen nicht mit den richtigen Großbuchstaben an den richtigen Stellen geschrieben hat.

C++-Exkurs: Funktionsstruktur und -syntax

Wenn Sie die Programmiersprache C++ bereits kennen und dieses Buch lesen, um etwas über das Visual C++-Werkzeug zu lernen, können Sie diesen Abschnitt überspringen. Wann immer Sie diesen Abschnitt heute und an den folgenden Tagen sehen, ist er der Programmiersprache C++, ihrer Syntax und ihrer Struktur gewidmet.

Der Code in diesem Buch ist mit Zeilennummern am Anfang jeder Zeile abgedruckt. Geben Sie diese bitte nicht mit ein.

Wenn Sie den Code in Listing 1.1 ansehen, sollten Sie mehrere Aspekte beachten, da Sie diese in dem C++-Code, den Sie schreiben, sehen und verwenden werden. Wir beginnen mit der ersten Zeile des Listings:

```
void CHelloDlg::OnBnClickedHello(void)
```

Diese Zeile ist ein Teil des von Visual C++ Developer Studio erstellten Funktionsrahmens. Es handelt sich dabei um den Funktionsnamen und die Aufrufsyntax. An einer anderen Stelle im Code, gewöhnlich in einer Header-Datei (erkennbar an der Dateinamenserweiterung ».h«), befindet sich eine Deklaration dieser Funktion, die Sie an einem der folgenden Tage sehen werden. Dieses Code-Listing ist die Implementierung der Funktion.

Das erste Wort in der Zeile, void, ist der Datentyp des Rückgabewerts. Wenn Sie das Wort void hier oder an Stelle der Funktionsargumente verwendet sehen (innerhalb der Klammern am Ende der Zeile), bedeutet das, dass es keinen Rückgabewert gibt bzw. dass keine Parameter an die Funktion übergeben werden.

Das zweite Wort in der Zeile, CHelloDlg, ist die Klasse, von der diese Funktion ein Mitglied ist. Sie wird gefolgt von zwei Doppelpunkten (::) und dem Funktionsnamen. Zwischen diesen Elementen sind Leerzeichen erlaubt, jedoch nicht zwischen den beiden Doppelpunkten.

Dem Funktionsnamen folgt, in Klammern eingeschlossen, die Liste der an die Funktion zu übergebenden Argumente. Die Klammern müssen immer verwendet werden, selbst wenn die Funktion keine Parameter übernimmt. Das ist anders als bei Sprachen wie den frühen Versionen von Visual Basic, in denen die Parameter optional sind, wenn die Funktion keine Parameter übernimmt oder wenn es sich bei ihr um eine Subroutine ohne Rückgabewert handelt.

C++-Regel

Beim Aufruf von Funktionen in C++ müssen die Klammern nach dem Funktionsnamen immer angegeben werden, selbst wenn der Funktion keine Parameter übergeben werden.

In der nächsten Zeile des Listings markiert eine öffnende geschweifte Klammer ({) den Beginn des Funktionskörpers. Dieser wird von einer schließenden Klammer (}) gefolgt, die sein Ende markiert. Die

Klammern schließen in C++ Code-Abschnitte ein, genau wie in Java und ähnlich der Verwendung von BEGIN und END in Visual Basic und Delphi.

C++-Regel

Funktionskörper werden immer von einem geschweiften Klammernpaar ({}) eingeschlossen.

Die nächste Code-Zeile, bei der es sich um die letzte vom Developer Studio erzeugte handelt, beginnt mit zwei Schrägstrichen (//). Die Verwendung zweier Schrägstriche ohne etwas dazwischen (keine Leerzeichen, Tabulator-Sprünge etc.) zeigt an, dass alles Folgende in dieser Zeile ein Kommentar ist und vom Compiler ignoriert werden soll (sofern die Schrägstriche nicht Teil eines Text-Strings sind). Das ist eine von zwei Möglichkeiten, Ihren Code zu kommentieren. Die zweite Methode eignet sich besser für mehrere Kommentarzeilen. Sie besteht aus einem Schrägstrich gefolgt von einem Stern (/ *), um den Beginn des Kommentars zu markieren, und dem Gegenstück (*/), um das Ende des Kommentars festzulegen.

Zu guter Letzt gibt es noch einiges Wissenswerte zu der eigentlichen Code-Zeile, die Sie hinzugefügt haben:

```
MessageBox("Hallo. Dies ist meine erste Visual C++ Anwendung!");
```

Erstens unterscheidet C++ zwischen Groß- und Kleinschreibung. Funktionsnamen und Variablen müssen exakt so geschrieben werden, wie sie deklariert wurden, d.h. die folgenden drei Funktionsnamen werden vom Compiler unterschiedlich behandelt:

```
MessageBox
messageBox
messagebox
```

Hätten Sie den Code nicht wie gezeigt eingegeben, sondern die Schreibweise der Funktion MessageBox geändert, hätten Sie beim Versuch, die Anwendung zu kompilieren, eine Fehlermeldung erhalten mit dem Hinweis, dass es sich bei der Funktion messageBox um einen »undeclared identifier«, einen nicht deklarierten Bezeichner, handelt.

C++-Regel

Die Schreibweise zählt! C++ unterscheidet zwischen Groß- und Kleinschreibung.

Auf den Funktionsnamen folgt in Klammern eingeschlossen das Funktionsargument und zum Abschluss ein Semikolon. In C++ wird jede Zeile mit einem Semikolon abgeschlossen. Ausnahmen dieser Regel sind Flusskontroll-Statements wie if, while und for, denen entweder mehrere Code-Zeilen in einem Klammernpaar folgen können, oder eine einzelne Zeile, die von einem Semikolon abgeschlossen ist. Hätten Sie am Ende dieser Zeile kein Semikolon gesetzt, dann hätten Sie eine Fehlermeldung erhalten, dass ein ; vor der schließenden Klammer fehlt (dieser Fehler zeigt immer auf die Zeile nach derjenigen, in der das Semikolon eigentlich fehlt). Kommentare werden nicht als Code betrachtet, müssen also nicht mit einem Semikolon abgeschlossen werden. Wenn Sie allerdings einen Kommentar zusammen mit Code in eine Zeile setzen, müssen Sie die Code-Zeile vor dem Kommentaranfang mit einem Semikolon beenden.

Definition

Jede Code-Zeile, die eine auszuführende Aktion enthält, muss mit einem Semikolon (;) abgeschlossen werden.

Wenn Sie die beiden oben erwähnten Fehler gemacht haben - die Funktion MessageBox falsch geschrieben und die Zeile nicht mit einem Semikolon abgeschlossen -, haben Sie beim Versuch, Ihren Code zu kompilieren, zwei Fehlermeldungen erhalten (siehe Abbildung 1.20). Wenn Sie auf eine der Meldungen doppelklicken, erscheint eine Markierung neben der Code-Zeile, in der der Fehler entdeckt wurde (siehe Abbildung 1.21).

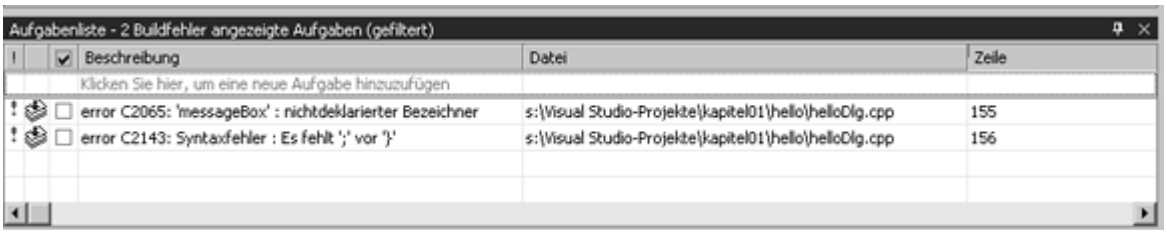


Abbildung 1.20: Fehlermeldungen zeigen, welche Fehler im Code korrigiert werden müssen.

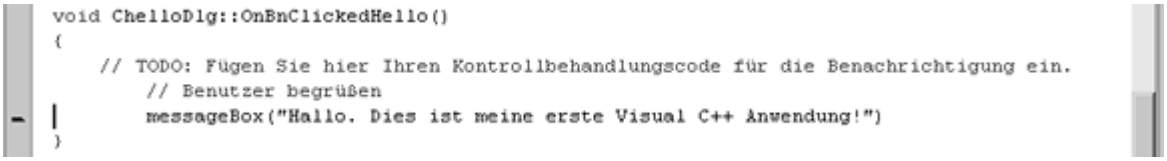


Abbildung 1.21: Ein Doppelklick auf eine Fehlermeldung bringt Sie zu den Code-Zeilen, in denen der Fehler entdeckt wurde.

1.5 Der letzte Schliff

Die Funktionalität der Anwendung ist nun vollständig realisiert. Mit ein paar netten Details können Sie dem Projekt den letzten Schliff verleihen:

- Das Symbol des Dialogfelds erstellen
- Die Schaltflächen Maximieren und Minimieren hinzufügen

Das Symbol für das Dialogfeld

Wenn Sie sich das Symbol in der oberen linken Ecke des Anwendungsfensters ansehen, stellen Sie drei Kästchen mit den Buchstaben M, F und C fest. Was hat aber MFC mit Ihrer Anwendung zu tun? MFC steht für Microsoft Foundation Classes. Aus technischer Sicht handelt es sich um die C++-Klassenbibliothek, mit der Ihre Anwendung aufgebaut ist. Aber möchten Sie das jedem Benutzer mitteilen, der Ihre Anwendung zu Gesicht bekommt? Höchstwahrscheinlich nicht. Daher sollten Sie das Anwendungssymbol bearbeiten, damit es Ihre Anwendung mit einem anderen Bild repräsentiert. Gehen wir ans Werk.

1. In der Baumansicht der Ressourcen im Arbeitsbereich erweitern Sie den Zweig Icon und markieren das Symbol IDR_MAINFRAME, wie es Abbildung 1.22 zeigt. Daraufhin erscheint das Anwendungssymbol im Editorbereich der IDE.

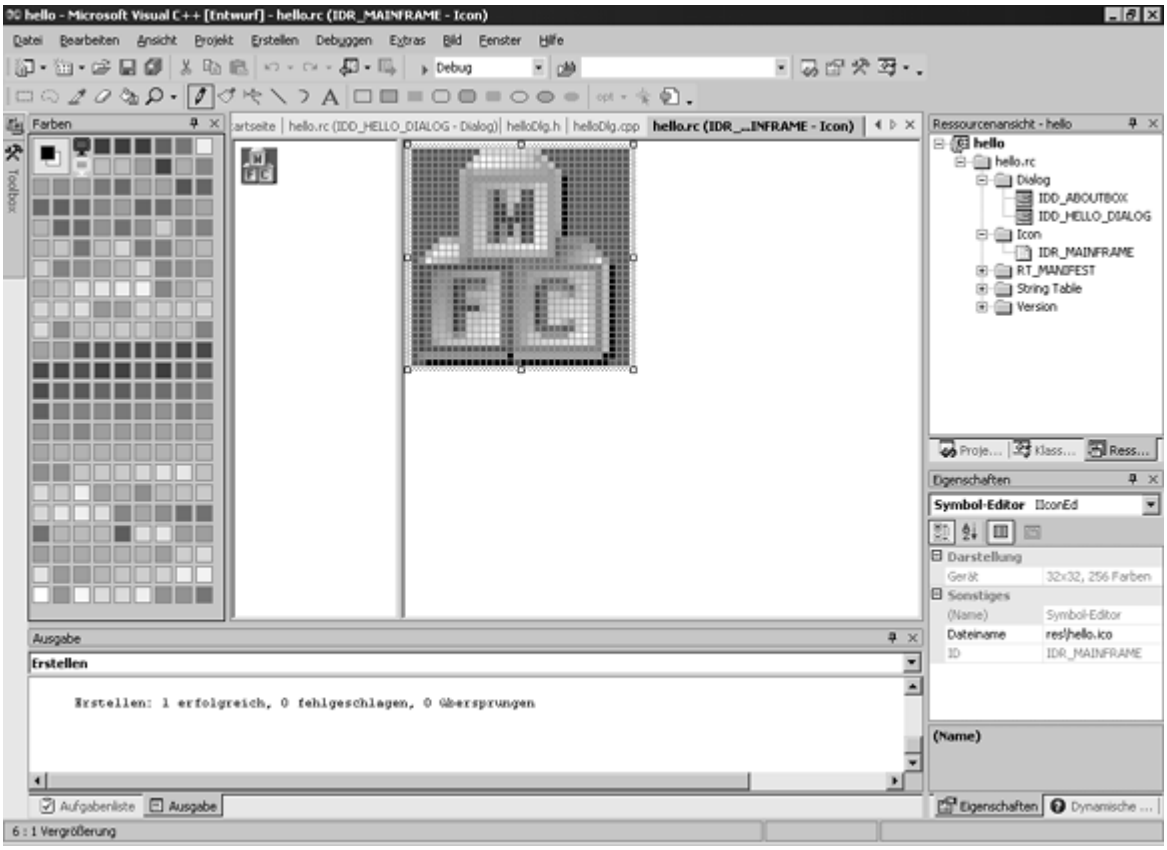


Abbildung 1.22: Das MFC-Standardsymbol

2. Mit den zur Verfügung stehenden Zeichenwerkzeugen gestalten Sie das Symbol um, sodass ein Bild entsteht, mit dem Sie Ihre Anwendung präsentieren wollen (siehe Abbildung 1.24). Sofern das Farbenfenster nicht geöffnet ist, können Sie es öffnen, indem Sie in den weißen Bereich um das Symbol herum rechtsklicken (siehe Abbildung 1.23). Das jeweils dargestellte Farbfenster hängt vom Betriebssystem und der Bildschirmeinstellung ab.

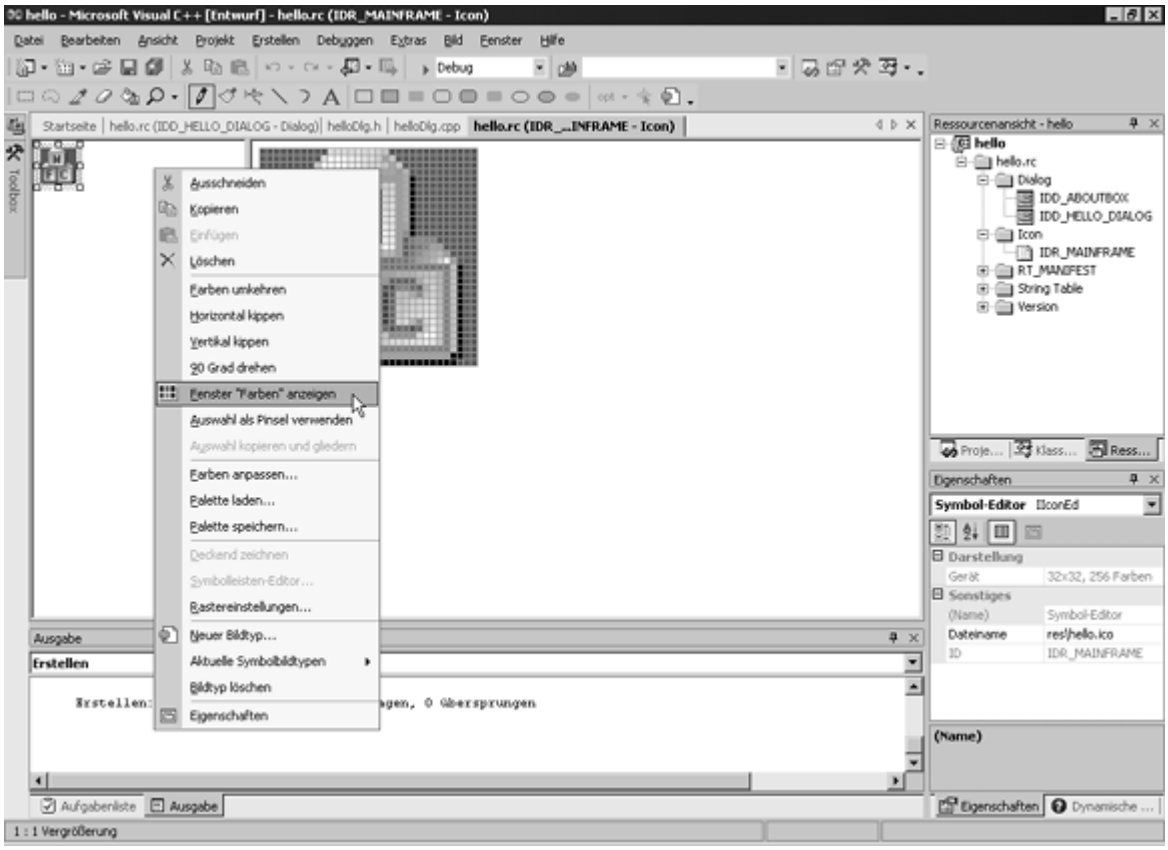


Abbildung 1.23: Das Farbenfenster öffnen

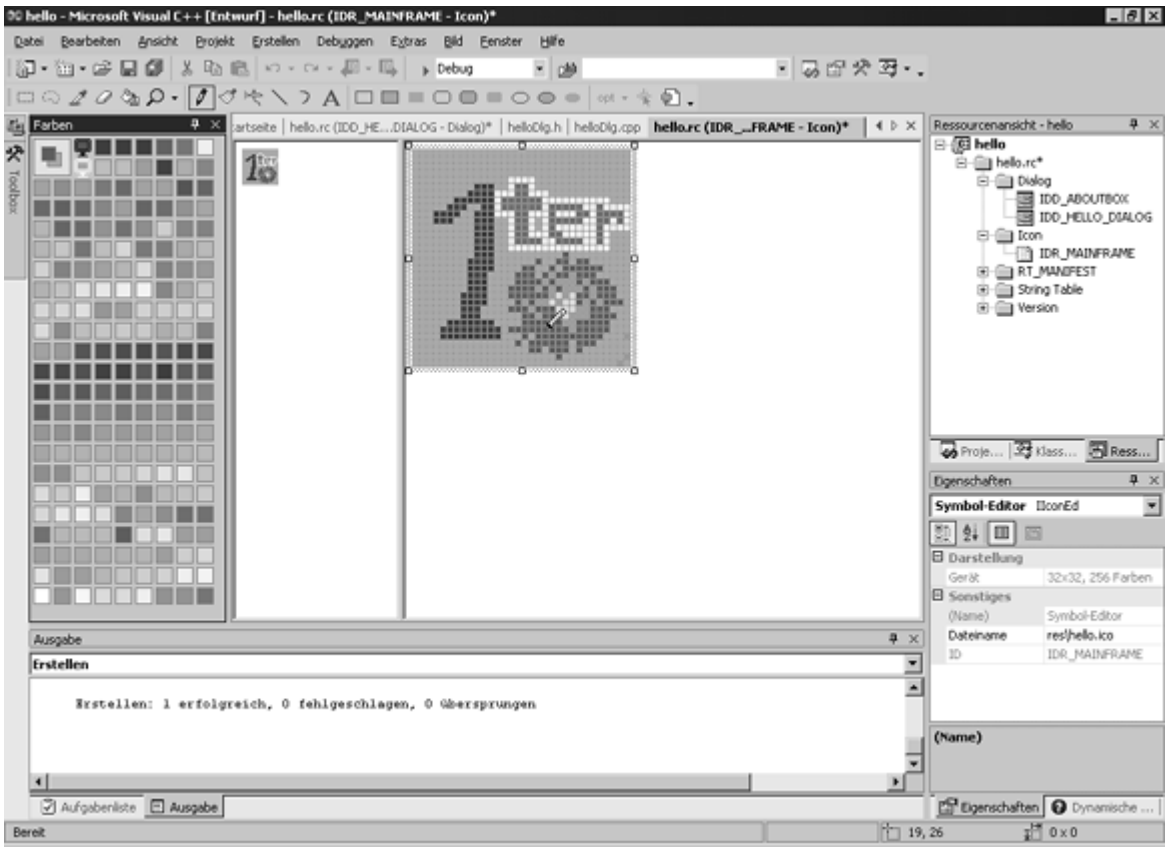


Abbildung 1.24: Ein eigenes Symbol für die Anwendung

Jedes in Developer Studio erstellte Symbol besteht eigentlich aus mehreren Symbolen. Das Symbol, das zum Ändern angezeigt wird, ist 32 mal 32 Pixel groß. Es erscheint gewöhnlich auf dem Desktop und im Info-über-Fenster. Es gibt auch noch ein 16 mal 16 großes Symbol, das gewöhnlich in der oberen linken Ecke des Anwendungsmenüs und in der Taskleiste angezeigt wird. Weitere Symbole besitzen die gleichen Größen, aber eine andere Anzahl von Farben. Sie sind für verschiedene Situationen bestimmt, in denen Symbole mit einer anderen Anzahl von Farben angezeigt werden können. Wahrscheinlich müssen Sie Ihr benutzerdefiniertes Symbol mehrmals in verschiedenen Größen und mit unterschiedlicher Farbenanzahl duplizieren. Um die anderen Symbole zur Bearbeitung zu öffnen, wählen Sie Bild / Aktuelle Symbolbildtypen aus dem Menü.

3. Wenn Sie die Anwendung kompilieren und ausführen, sehen Sie Ihr Symbol in der oberen linken Ecke des Anwendungsfensters. Klicken Sie auf das Symbol und wählen Sie den Befehl Info über Hello aus dem Dropdown-Menü.
4. Im Info-Dialog, den Visual C++ automatisch erstellt hat, ist eine große Version Ihres Symbols in all seiner Schönheit zu sehen (siehe Abbildung 1.25).

Abbildung 1.25: Das Info-Fenster in der Anwendung

Die Schaltflächen Maximieren und Minimieren hinzufügen

Im Dialog-Editor, wo Sie das Anwendungsfenster gestaltet haben, können Sie auch die Schaltflächen Minimieren und Maximieren in die Titelleiste des Anwendungsfensters mit folgenden Schritten einfügen:

1. Markieren Sie das Dialogfenster selbst, als wollten Sie es in der Größe verändern.

2. Suchen Sie in der Eigenschaftenansicht die Eigenschaften Miniimieren-Feld und Maximieren-Feld. Setzen Sie beide Werte auf TRUE. Wenn Sie Minimieren und Maximieren aktiviert haben, können Sie Ihre Anwendung kompilieren und starten. Die Schaltflächen Minimieren und Maximieren erscheinen in der Titelleiste, wie es in Abbildung 1.26 zu sehen ist.

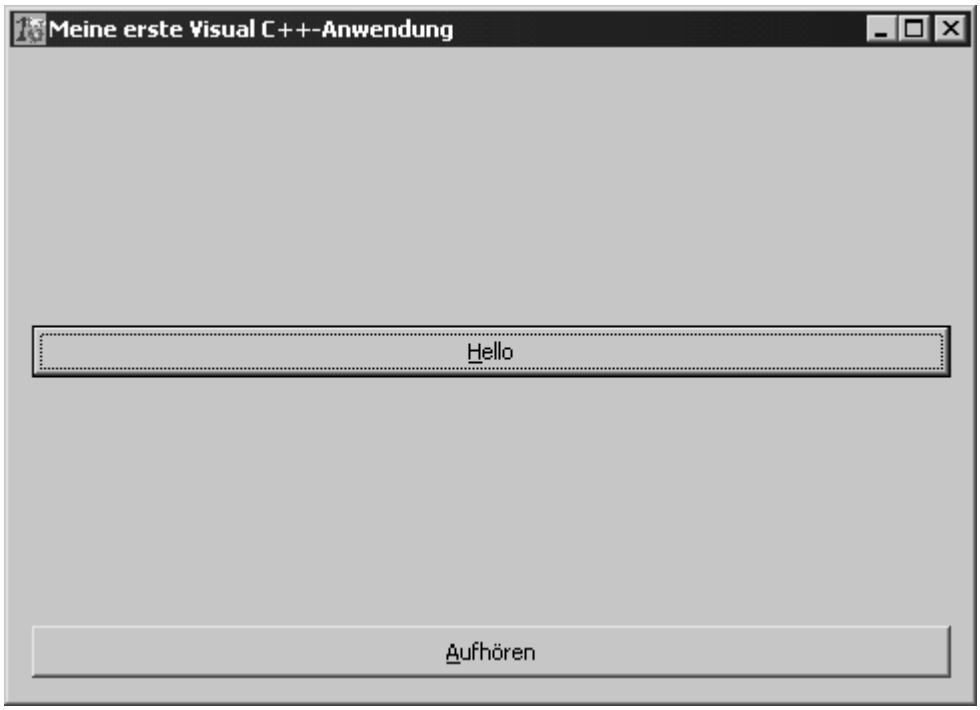


Abbildung 1.26: Das Anwendungsfenster mit den Schaltflächen Minimieren und Maximieren

1.6 Zusammenfassung

Heute haben Sie einen ersten Eindruck erhalten, wie man Anwendungen mithilfe von Visual C++ erstellt. Sie haben die verschiedenen Bereiche von Visual Studio für Visual C++ sowie den Zweck dieser Bereiche kennen gelernt. Weiterhin wurde gezeigt, wie man die Umgebung von Visual Studio an die eigenen Arbeitsgewohnheiten anpasst. Mit den Assistenten von Visual C++ haben Sie ein Anwendungsgerüst erstellt und dann die visuellen Komponenten, die Sie im Anwendungsfenster positioniert haben, mit Funktionalität ausgestattet.

1.7 Workshop

Das Quiz am Ende jedes Tages soll Ihnen dabei helfen, die neu erworbenen Kenntnisse zu den behandelten Themen zu festigen. Die Übungen geben Ihnen die Möglichkeit, praktische Erfahrungen mit dem gelernten Stoff zu sammeln. Die Quiz-Antworten finden Sie zusammen mit möglichen Lösungen der Übungen im Anhang A.

Fragen und Antworten

Frage:
Wie kann man statt des Anwendungsnamens einen anderen Titel im Meldungsfeld anzeigen?

Antwort:
Per Vorgabe verwendet das Meldungsfeld den Anwendungsnamen als Fenstertitel. Einen eigenen Titel kann man angeben, indem man einen zweiten String im Aufruf der Funktion MessageBox übergibt. Der erste String ist immer die anzuzeigende Nachricht, während der zweite String als Fenstertitel dient. Beispielsweise könnte die Funktion OnHello folgendermaßen aussehen:

```
//Benutzer begrüßen
 MessageBox("Hallo. Dies ist meine erste Visual C++ Anwendung!",
            "Meine erste Anwendung");
```

Frage:
Kann ich den Text im Infofenster ändern, um den Namen meiner Firma und weiterführende Copyright-Informationen anzugeben?

Antwort:
Ja. Das Infofenster befindet sich im Ordner Dialog auf der Registerkarte Ressourcenansicht des Arbeitsbereichs. Wenn Sie auf das Dialogfeld IDD_ABOUTBOX doppelklicken, wird das Info-Dialogfeld im Dialog-Designer geöffnet. Sie können nun das Dialogfeld in jeder gewünschten Weise gestalten.

Quiz

1. Wie ändert man den Titel (die Beschriftung) einer Schaltfläche?

2. Welche Aufgaben erledigt man mit dem MFC Anwendungs-Assistenten?
3. Wie weist man dem Klickereignis einer Schaltfläche Funktionalität zu?

Übung

Nehmen Sie eine zweite Schaltfläche in das Infofenster Ihrer Anwendung auf. Lassen Sie die Schaltfläche eine andere Meldung als die im ersten Fenster anzeigen.

Tag 2

Fehlerbeseitigung in Ihrer Anwendung

Niemand ist perfekt. Diese Regel gilt für alle Programmierer. Niemand schreibt gleich beim ersten Mal perfekten Code. Wenn ein Programmierer behauptet, perfekten Code zu schreiben, ist das entweder gelogen oder er ist ein Außerirdischer. Also müssen Sie wissen, wie Sie die mit Visual C++ mitgelieferten Tools verwenden, um Ihren Code zu debuggen. So können Sie Irrtümer in Ihrem Code finden, bevor Sie die Anwendung an einen Benutzer weitergeben.

Heute werden Sie einige der Werkzeuge für die Fehlerbeseitigung kennen lernen, die Sie im Kampf gegen Bugs verwenden können. Manche davon sind passive Tools; sie warten darauf, dass sie Probleme finden und Ihnen das mitteilen. Andere sind aktiv; mit ihnen können Sie in Ihrer Anwendung graben, während diese läuft, und beobachten, wie Ihr Code Schritt für Schritt abgearbeitet wird.

An diesem Tag beschäftigen Sie sich mit folgenden Themen:

- Der Unterschied zwischen Debug- und Release-Builds und warum man beide verwenden sollte
- Die Makros, die Sie in Ihren Code einfügen können, um einfach darauf zu warten, dass Probleme gefunden werden
- Der Debugger von Developer Studio, mit dem Sie Ihren Code Schritt für Schritt ausführen und die Verarbeitung verschiedener Situationen verfolgen können

2.1 Was ist Fehlerbeseitigung?

Ganz am Anfang bestand der Speicher von Computern aus elektrischen Relais, nicht aus Silikon-Chips. Als auf diesen Rechnern einmal laufende Programme anfingen, zu versagen, untersuchten Techniker die Computer, um das Problem zu finden. Sie fanden eine Motte, die sich in einem der Speicher-Relais verfangen hatte. Ein wirkliches Insekt (engl. Bug) verhinderte, dass die Programme richtig funktionierten. Daher stammt der Ausdruck Bug. Dieser wird heute zur Bezeichnung aller möglichen Fehler in Computer- Software verwendet, ob durch den Fehler nun das gesamte Programm oder Betriebssystem abstürzt oder ob nur falsche Ergebnisse geliefert werden.

Damit sie fehlerfrei wird, durchläuft Software meist mehrere Phasen der Fehlerbeseitigung, in denen jeweils unterschiedliche Ansätze verwendet werden. Dabei handelt es sich um folgende Phasen:

- Erster Test und Fehlerbeseitigung
- Komponententest
- Integrationstest
- Build- und Regressions-Test
- Alpha- und Beta-Test
- Release Build

Erster Test und Fehlerbeseitigung

Jeder Entwickler führt diese Phase der Fehlerbeseitigung aus, wenn eine Funktion oder ein Code-Abschnitt fertiggestellt wird. Der Entwickler erstellt einen einfachen Anwendungsrumpf, der den neu entwickelten Code aufruft, und führt ihn mit jedem möglichen Wert und jeder möglichen Kombination von Bedingungen aus, mit denen der Code aufgerufen werden kann. In dieser Testphase ist der Entwickler dafür verantwortlich, dass jede Code-Zeile unter allen möglichen Bedingungen durchlaufen wird, um sicherzustellen, dass die korrekte Funktionalität gewährleistet wird und keine unvorhergesehenen Ergebnisse entstehen. Hier sollten die meisten logischen Fehler erkannt und eliminiert werden.

Komponententest

Der Komponententest ist die nächste Phase im Prozess der Fehlerbeseitigung. Wenn jeder Entwickler alle einzelnen Module und Komponenten fertig gestellt und getestet hat, werden diese in einer funktionalen Einheit zusammengefügt. Die Idee dahinter ist, dass diese Einheit als »Blackbox« mit bekannten Eingaben und vorhersagbaren Ausgaben verwendet wird, wobei die interne Implementierung und Funktionalität nach außen hin nicht bekannt oder sichtbar ist. Diese Einheiten werden mit allen möglichen Variationen der Eingabe getestet, um zu überprüfen, ob sie korrekt arbeiten und die korrekten Ergebnisse ausgeben. Während dieser Testphase sollte sich der Entwickler in erster Linie mit den Eingaben und Ausgaben der Einheit befassen und nicht mit der internen Logik - falls nicht die Ergebnisse erkennen lassen, dass es ein Problem in der internen Logik gibt.

Integrationstest

Beim Integrationstest werden zwei oder mehrere Einheiten zusammengefügt und als Ganzes getestet. Diese Testphase soll sicherstellen, dass die Einheiten zusammenarbeiten und Probleme in der Interaktion erkennen. In dieser Phase wird ein großer Teil der Testaktivität von einem weiteren Teammitglied übernommen, das alle gefundenen Probleme über dokumentierte Fehlerberichte an den Entwickler zurückgibt. Diese Berichte beschreiben die Art des Problems und enthalten für den Entwickler so viele Informationen wie möglich, damit er das Problem reproduzieren kann. Alle diese Fehlerberichte werden protokolliert und im Auge behalten, damit die Teammitglieder alle Probleme und deren Zustände zu jedem Zeitpunkt verfolgen können.

Build- und Regressions-Test

Ein Build einer Anwendung entsteht, wenn die gesamte Anwendung kompiliert und als Ganzes gelinkt (zusammengefügt) wird. Hier treten alle Probleme in Bezug auf Funktionsdefinitionen zutage, die gewöhnlich das erfolgreiche Erstellen der gesamten Anwendung verhindern. Dieser Test wird durchgeführt, um sicherzustellen, dass die ganze Anwendung kompiliert und ausgeführt werden kann. Ein Teammitglied erhält die Verantwortung für diesen Test. Manche Firmen wie zum Beispiel Microsoft führen täglich einen Build-Test für ihre Entwicklungen durch. Verhindert der Code eines Entwicklers die Erstellung, erhält dieser Entwickler die Build-Verantwortung - das heißt er ist jeden Tag für den Build-Test verantwortlich, bis der Code eines anderen Entwicklers die Erstellung verhindert.

Ist eine Anwendung erfolgreich erstellt, durchläuft sie eine Reihe von Regressions-Tests. Dabei handelt es sich um vollständig automatisierte und über Nacht ausgeführte Tests. Die Tests unterziehen die Anwendung einer Reihe von Aktionen, um die als funktionsfähig bekannte Funktionalität zu testen. Der Zweck dieser Tests ist, sicherzustellen, dass keine Fehler in Code eingeführt wurden, der schon als korrekt funktionierend bekannt war. In einer Anwendung gefundene Fehler werden dokumentiert und protokolliert; der Bericht wird dem entsprechenden Entwickler zusammen mit dem Testablauf übergeben, sodass er die Fehler reproduzieren kann. Für diese Art von Tests existieren separate Programme, die (vorwiegend) in der kommerziellen Anwendungsentwicklung verwendet werden.

Alpha- und Beta-Test

Die nächsten beiden Testphasen geben die Anwendung in die Hände von Benutzern. Diese Benutzer werden gebeten, die Anwendung täglich zu verwenden und alle Probleme, auf die sie treffen, zu berichten. In der ersten dieser beiden Phasen, dem Alpha-Test, sind die Benutzer Mitarbeiter der entwickelnden Firma. Während der zweiten Phase, dem Beta-Test, wird die Anwendung an außenstehende Benutzer weitergegeben. Dabei erhöht sich auch stufenweise die Anzahl der Benutzer, die die Anwendung testen. Während der Alpha-Phase testen

möglicherweise nur eine Handvoll Benutzer die Anwendung, während sich die Anzahl der Benutzer in der Beta-Phase immer weiter erhöht, bis einige Hunderte, wenn nicht gar Tausende Benutzer die Anwendung testen.

Wenn während dieser Testphasen Probleme gefunden werden, erhalten interne Tester die Problemberichte und versuchen, die Probleme zu reproduzieren. Dabei dokumentieren sie die dafür notwendigen Schritte. Danach wird der Problembericht an die Entwicklungsleitung gesendet, die die Gewichtigkeit jedes einzelnen Problems einstuft und den Problemen Prioritäten zuordnet. Als Nächstes werden die Problemberichte den einzelnen Entwicklern zur Berichtigung übergeben und sämtliche Testreihen von vorne begonnen.

Release-Build

Wenn feststeht, dass eine Anwendung bereit zur Veröffentlichung ist, wird ein Release-Build erstellt. Im Release-Build wird der gesamte Debug-Code aus der Anwendung entfernt. Der Visual C++-Compiler macht das automatisch. Im normalen Modus von Visual C++, dem Debug-Modus, fügt der Compiler alle möglichen Zusatzinformationen in den kompilierten Code ein, die angeben, welche Code-Zeile gerade ausgeführt wird. Diese Informationen werden zwar nur vom Debugger gelesen und nicht ausgeführt, doch sie vergrößern die Anwendung deutlich und verlangsamen die Ausführung ein wenig (was bei der heutigen Generation schneller Prozessoren allerdings kaum zu bemerken ist). Beim Release-Build wird auch der Code entfernt, der die Fehlerbeseitigung unterstützt - mehr hierüber erfahren Sie im nächsten Abschnitt - und damit eine bedeutende Menge von Code, der nicht mehr kompiliert werden muss.

Ist ein Release-Build erstellt, durchläuft es Regressions-Tests, um sicherzustellen, dass bei der Erstellung nicht versehentlich Programmlogik entfernt wurde. Wenn das der Fall ist, wird das betreffende Modul identifiziert und der für den problematischen Code verantwortliche Entwickler wird auf das Problem aufmerksam gemacht. Um in Visual C++ zwischen Debug- und Release-Modus hin- und herzuschalten, wählen Sie Konfigurations-Manager aus dem Menü Erstellen und aktivieren in der Dropdown- Liste im Dialog des Konfigurations-Managers den gewünschten Modus (siehe Abbildung 2.1).

Wenn ein Release-Build erstellt wird, wird mit diesen Makros eine große Menge von Debug-Code entfernt. Außerdem wird der Maschinencode optimiert, um die Ausführung der Anwendung zu beschleunigen und die Größe der endgültigen Programmdatei zu verringern. All dies kann kleine - und weniger kleine - Fehler in die Anwendung bringen. Da sich diese Arten von Fehlern manchmal in Release-Builds einschleichen, führen die Firmen noch viele Tests mit den Release-Builds von Anwendungen aus.

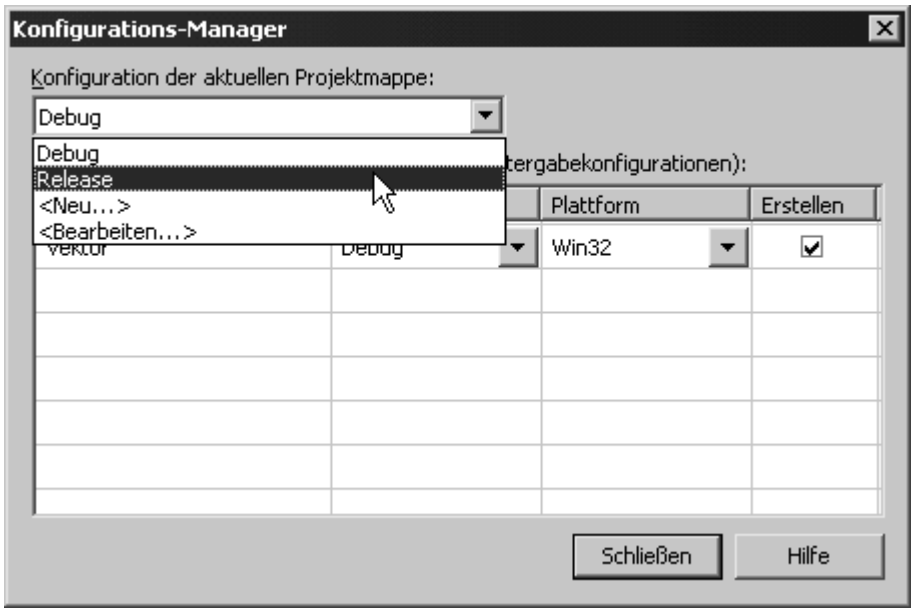


Abbildung 2.1: Zwischen Debug- und Release-Modus umschalten

2.2 Code zur Unterstützung der Fehlerbeseitigung

Sie können in Ihrem Code vier Makros großzügig verwenden, um Annahmen zu bestätigen und Alarm zu schlagen, wenn in Ihrer Anwendung Probleme auftreten. Diese Makros werden nur in Debug-Builds Ihrer Anwendung ausgeführt. Wenn Sie Release- Builds erstellen, werden die Makros entfernt.

Ein Makro ist in C++ (genau wie in der Vorgängersprache C) eine mit der Präcompiler-Direktive #define erstellte Pseudo-Funktion. Vor dem Kompilieren wird das Makro durch den in der Makrodefinition verwendeten Code ersetzt, um die Funktionalität des Makros zu definieren. Ein Vorteil der Verwendung von Makros ist, dass die Entwickler von MFC eine Implementierung eines bestimmten Makros für Debug-Builds und eine andere für Release-Builds definieren können.

C++-Exkurs: Präcompiler-Direktiven

In den Programmiersprachen C und C++ werden alle Quellcode-Dateien einem Präcompiler übergeben, bevor der Code kompiliert wird. Der Präcompiler verarbeitet einige bestimmte Direktiven, wobei der zu kompilierende Code deutlich erweitert wird. Alle Präcompiler-Direktiven lassen sich leicht erkennen, da sie mit einem Doppelkreuz (#) beginnen, das als erstes Zeichen in der Zeile vor jedem anderen Zeichen, ausgenommen Whitespace-Zeichen, steht. In Tabelle 2.1 sind die wichtigsten Präcompiler-Direktiven aufgelistet.

Direktive	Beschreibung
`#include datei_name`	Diese Direktive weist den Präcompiler an, den Inhalt der angegebenen Datei an dieser Stelle einzufügen. Steht der Dateiname in spitzen Klammern (<>), werden die vorkonfigurierten Verzeichnisse nach der angegebenen Datei durchsucht. Werden Anführungszeichen verwendet (""), wird die Datei in dem mit dem Dateinamen angegebenen Pfad gesucht. Dabei kann es sich um einen (vom Projekt-Verzeichnis aus) relativen oder um einen

	absoluten Pfad handeln (einschließlich des Laufwerkbuchstabens).
`#define KONSTANTE wert`	Definiert einen konstanten Wert. Die Konstante wird gewöhnlich in Großbuchstaben deklariert. Der Präcompiler setzt vor dem Kompilieren an jeder Stelle, an der die Konstante auftaucht, den Wert ein.
`#define KONSTANTE(par1) definition`	Definiert ein Makro. Der Name des Makros wird gewöhnlich in Großbuchstaben deklariert. Der Präcompiler setzt vor dem Kompilieren an jeder Stelle, an der das Makro auftaucht, die Definition des Makros ein. Wenn der Präcompiler das Makro verarbeitet, durchläuft er die Definition und setzt an jeder Stelle, an der die Parameter auftauchen, die bei jeder Verwendung des Makros angegebenen Variablen oder Werte ein.
`#ifdef KONSTANTE`	Weist den Präcompiler an, den folgenden Code-Abschnitt nur dann einzufügen, wenn die angegebene Konstante bereits definiert wurde.
`#ifndef KONSTANTE`	Weist den Präcompiler an, den folgenden Code-Abschnitt nur dann einzufügen, wenn die angegebene Konstante nicht definiert ist.
`#else`	Wird mit den letzten beiden zusammen verwendet und markiert den nächsten Abschnitt, der einzufügen ist, wenn die davor stehende #ifdef oder #ifndef-Bedingung nicht zutrifft.
`#endif`	Markiert das Ende der Abschnitte, die auf Grund von Bedingungen in den zu kompilierenden Code eingefügt werden oder nicht. Die Direktive #else kann optional verwendet werden, die Direktive #endif muss immer den Abschluss einer #ifdef oder #ifndef-Direktive bilden.

Tabelle 2.1: Präcompiler-Direktiven in C/C++

Direktive	**Beschreibung**
`#pragma option`	Legt compiler- oder maschinenspezifische Optionen fest. Die meisten C/C++-Compiler stellen viele solcher Optionen bereit, die mit der #pragma-Direktive verwendet werden können. Um festzustellen, welche #pragma-Optionen ein bestimmter Compiler bietet, sehen Sie in der Dokumentation des Compilers nach (wir werden auf unserem Weg durch das Buch von Zeit zu Zeit einen Blick auf einige der in Visual C++ verfügbaren #pragma-Optionen werfen).

Tabelle 2.1: Präcompiler-Direktiven in C/C++ (Forts.)

Beginnen Sie langsam, wenn Sie mit Präcompiler-Direktiven arbeiten. Sie sollten anfangs nur die #include-Direktive verwenden, um benötigte Header-Dateien einzufügen, und die #define-Direktive zur Definition von Konstanten. Lassen Sie die anderen Direktiven weg, bis Sie ein bisschen Erfahrung gesammelt haben. Ihre Verwendung kann zu verwirrenden und unerwarteten Ergebnissen führen, wenn Sie das Wann und Wie ihrer Anwendung nicht gründlich durchdenken.

Annahmen überprüfen

Sie können in Ihrem Code vier Makros verwenden, um die während der Entwicklung und Erstellung Ihrer Anwendungen getroffenen Annahmen zu überprüfen. Das erste Makro mit dem Namen ASSERT besitzt eine Reihe von Abwandlungen. Es hat folgende Syntax:

```
ASSERT(bAnnahme);
```

Der diesem Makro übergebene Parameter ist ein beliebiger Boolescher Ausdruck, der immer TRUE ergeben sollte. Wenn Sie beispielsweise eine Funktion mit einem Parameter a haben, der immer einen nicht-negativen Wert besitzt, können Sie dies wie folgt mit einem ASSERT testen:

```
ASSERT(a >= 0);
```

Wird beim Aufruf dieser Funktion jemals ein negativer Wert als Parameter a übergeben, zeigt ASSERT eine Warnung wie die in Abbildung 2.2 an und weist Sie auf das bestimmte fehlgeschlagene ASSERT hin. Dabei erfahren Sie, in welcher Quelldatei und in welcher Zeile des Quelltextes der Fehler aufgetreten ist. Sie haben die Möglichkeit zu entscheiden, ob die Applikation sofort beendet wird, Sie in den Debugger wechseln möchten oder das Ergebnis der Prüfung ignoriert werden kann.

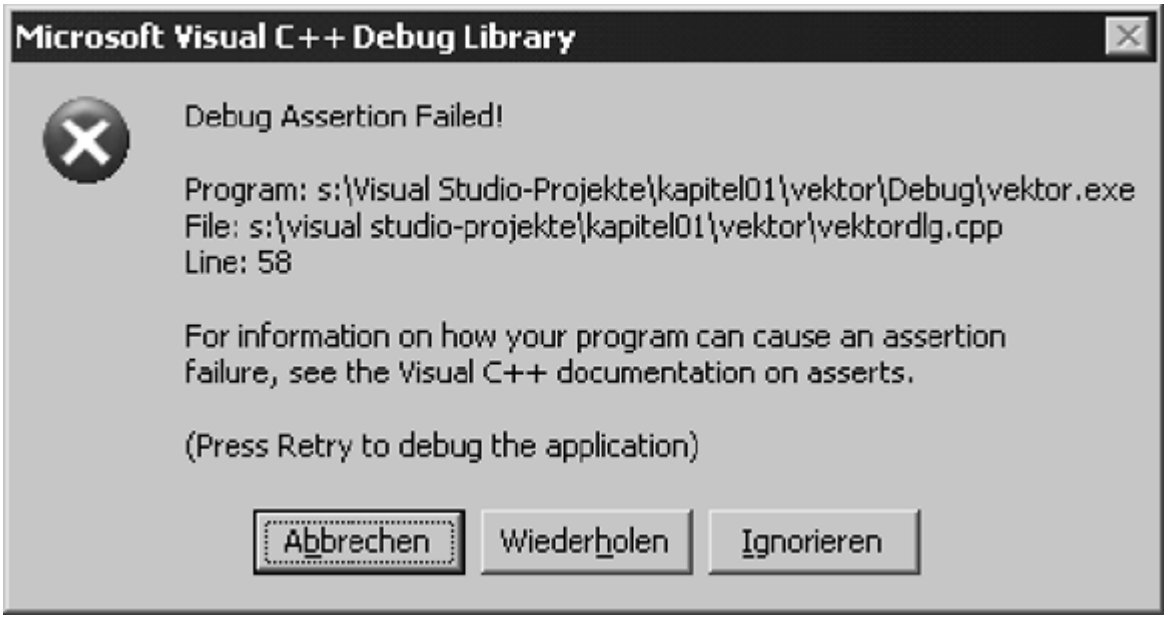


Abbildung 2.2: Ein ASSERT macht Sie auf ein Problem in Ihrer Anwendung aufmerksam.

Platzieren Sie niemals notwendige Funktionalität in dem an das ASSERT-Makro übergebenen Parameter.

Denken Sie bei der Verwendung von ASSERT daran, dass ASSERT an allen Stellen aus der Anwendung entfernt wird, wenn Sie ein Release-Build erstellen. Wenn Sie in dem an das Makro übergebenen Parameter Programmlogik verwendet haben, wird auch diese aus der Anwendung entfernt.

Mit zwei Abarten von ASSERT können Sie Klassen und Objekte in Ihrem Code testen. Die erste, ASSERT_VALID, wird wie folgt verwendet:

```
ASSERT_VALID(pObjekt);
```

Mit dieser Version werden Objekte in Ihrer Anwendung getestet, um festzustellen, dass sie gültig sind und es keine Probleme mit ihrem internen Zustand gibt. Der an dieses Makro übergebene Parameter ist ein instanziiertes C++-Objekt.

Das Makro ASSERT_VALID funktioniert nicht bei allen C++-Objekten. Das Objekt muss die Basisklasse CObject ererben und die Member-Funktion AssertValid überschrieben haben. Die meisten Standard-MFC-Klassen entsprechen diesen Kriterien. Um Ihre eigenen Klassen zu testen, müssen Sie sich bei der Entwicklung neuer Klassen dieser Anforderungen bewusst sein.

Die zweite Abart von ASSERT ist das Makro ASSERT_KINDOF, das wie folgt verwendet wird:

```
ASSERT_KINDOF(klassenname, pObjekt);
```

Dieses Makro prüft, ob ein Objekt eine bestimmte Klasse ist oder sich von dieser Klasse ableitet. Der erste Parameter ist der Name der Klasse, von der der zweite Parameter eine Instanz sein muss.

Diese Funktion kann nur zur Überprüfung der Zugehörigkeit eines Objektes zu einer Klasse verwendet werden, die anderen bestimmten Kriterien entspricht; sie muss beispielsweise ein Abkömmling der Klasse CObject sein und in ihrer Klassendefinition muss eines von zwei anderen Makros verwendet werden. Bei diesen beiden Makros handelt es sich um DECLARE_DYNAMIC und DECLARE_SERIAL. Das ist uns allerdings noch etwas zu hoch, wenn Sie die Programmiersprache C++ noch nicht kennen. Sie sollten diesen Hinweis noch einmal lesen, wenn Sie in Tag 12 »Dateizugriff« ein besseres Verständnis von Serialisierung, C++ und der MFC-Klassenbibliothek erlangt haben.

Wie das Makro ASSERT werden auch die beiden Abarten vollständig aus dem Release-Build von Anwendungen entfernt. Verwenden Sie diese also nicht an Stelle notwendiger Logik. Man sollte die Makros nur verwenden, um Annahmen im Code zu überprüfen.

Eine letzte Abart des ASSERT-Makros unterscheidet sich von den anderen, ist aber letztlich dasselbe. Dabei handelt es sich um das Makro VERIFY. Es wird genau wie das ASSERT-Makro verwendet:

```
VERIFY(bAnnahme)
```

Ein entscheidender Unterschied zwischen den Makros VERIFY und ASSERT ist, dass die als Parameter an VERIFY übergebene Logik bei der Erstellung eines Release-Builds in der Anwendung bleibt. Das heißt Sie können im Parameter des VERIFY-Makros Programmlogik unterbringen und diese Logik bleibt erhalten, wenn Sie ein Release-Build erstellen. Wie das ASSERT-Makro testet VERIFY allerdings nur in Debug-Builds Ihrer Anwendung den Ausdruck und alarmiert Sie, wenn dieser FALSE ist.

Fluss und Ausführung folgen

Manchmal wollen Sie den Weg der Ausführung wissen, den Ihre Anwendung nimmt, oder bestimmte Dinge, die während der Ausführung vorgehen, wollen aber die Anwendung nicht Zeile für Zeile abarbeiten. Eigentlich wollen Sie ein Protokoll, in dem steht, wann die Anwendung bestimmte Punkte im Code erreicht oder passiert und in welchem Zustand sich die Variablen befinden. Das ist zwar einfach (den Code hinzufügen, der Meldungen in einer Datei protokolliert), doch es ist mühselig, diesen Code wieder zu entfernen, wenn Sie die Anwendung weitergeben.

Eine einfache Möglichkeit, diese Protokollierung zu erreichen, ohne sich die Mühe machen zu müssen, alle Protokollmeldungen wieder zu entfernen, ist die Verwendung des Makros TRACE. Das TRACE-Makro sendet eine ihm übergebene Meldung (einen String) an ein beliebiges Debug-Ausgabefenster, das Sie auf Ihrem Rechner gleichzeitig mit Ihrer Anwendung laufen lassen (siehe Abbildung 2.3). Das Makro hat folgende Syntax:

```
TRACE("Dies ist meine Trace-Meldung.\n");
```

```
double fGeld;
int iNr = 99;
TRACE("Dies ist meine Trace-Meldung!\n" );
TRACE("Diese Zahl: %5d ist fünf Ziffern lang.", iNr);
TRACE("\n");
fGeld = 123.456;
TRACE("Ich habe %3.2f Euro
TRACE("\n");
fGeld = 1.234;
TRACE("Ich habe %-3.2f Euro
TRACE("\n");
fGeld = 123.456;
TRACE("Ich habe %+3.2f Euro
TRACE("\n");
fGeld = -123.456;
TRACE("Ich habe %+3.2f Euro
TRACE("\n");
```

```
Ausgabe
Debuggen
'vektor.exe': 'D:\WINDOWS\system32\oleaut32.dl
'vektor.exe': 'D:\WINDOWS\WinSxS\x86_Microsoft
'vektor.exe': 'D:\WINDOWS\system32\mfc70deu.dl
'vektor.exe': 'D:\WINDOWS\system32\MSCTF.dll'
'vektor.exe': 'D:\WINDOWS\system32\uxtheme.dll
Dies ist meine Trace-Meldung!
Diese Zahl:    99 ist fünf Ziffern lang.
Ich habe 123.46 Euro in meiner Geldbörse.
Ich habe 1.23 Euro in meiner Geldbörse.
Ich habe +123.46 Euro auf dem Konto.
```

Abbildung 2.3: Trace-Meldungen aus einer Debug-Sitzung

Das TRACE-Makro funktioniert wie die Funktion printf in C, also können Sie ihm genau wie bei printf Variablen übergeben, die dynamisch in die Meldung eingefügt werden.

Der Ausgabe-String des TRACE-Makros ist auf 512 Zeichen begrenzt. Ist der an das TRACE-Makro übergebene formatierte String zu lang (einschließlich des abschließenden '\0'-Zeichens), löst es einen ASSERT aus.

Wie das ASSERT-Makro wird auch das TRACE-Makro automatisch vom Compiler aus Release- Builds entfernt.

C++-Exkurs: Strings formatieren

Die Programmiersprache C war der Vorgänger von C++. C++ wurde als objektorientierte Version von C entwickelt. Sie können mit jedem C++-Compiler Anwendungen in Standard-C schreiben, da ein Teil des C++-Standards die vollständige Unterstützung von C ist.

In der Programmiersprache C gibt es eine Familie von Funktionen für die Formatierung und die Ausgabe von Strings. Die wichtigste Funktion, printf, wird verwendet, um Strings zu formatieren und an die Standard-Ausgabe auszugeben (für Anwendungen im Zeichenmodus, die in einem DOS-Fenster auf einem Windows-System laufen). Von dieser Funktion gibt es noch zwei Varianten. Die erste davon, fprintf, ermöglicht es, die String-Meldung an eine Datei oder ein Ausgabegerät zu senden. Dann gibt es noch die Funktion sprintf, die den String formatiert und in einem Zeichen-Array platziert. Von Zeit zu Zeit sieht man diese Funktionen noch

in C++-Code verwendet.

Die Funktion printf übernimmt einen String als ersten Parameter, der von einer beliebigen Anzahl von Variablen gefolgt sein kann, die in den String eingefügt werden. Für jede Variable muss es einen Platzhalter im String geben. Diese Platzhalter werden durch das Prozentzeichen (%), gefolgt von einem Zeichen, das den Datentyp des an dieser Stelle einzufügenden Wertes angibt, festgelegt. In Tabelle 2.2 sind diese Zeichen und die ihnen entsprechenden Datentypen aufgelistet.

Platzhalter	Datentyp
%c	Ein Zeichen mit der Länge eines Bytes
%C	Ein langes oder UNICODE-Zeichen
%d oder %i	Dezimaler Integer-Datentyp, (Integer oder Long)
%u	Integer-Datentyp ohne Vorzeichen (unsigned)
%f	Fließkomma-Datentyp (Double oder Float)
%s	Durch ein Nullzeichen ('\0') terminierte Zeichenkette (String)

Tabelle 2.2: Datentypen-Platzhalter für printf

Platzhalter	Datentyp
%S	Langer oder UNICODE-String
%x	Hexadezimaler Wert (klein geschrieben)
%X	Hexadezimaler Wert (groß geschrieben)
%o	Oktaler Wert
%%	Prozent-Zeichen. Da das Prozent-Zeichen verwendet wird, um anzugeben, dass es sich beim nächsten Zeichen um einen Platzhalter für einen Variablenwert handelt, müssen Sie zwei Prozent-Zeichen verwenden, um ein Prozent-Zeichen in Ihren String einzufügen.

Tabelle 2.2: Datentypen-Platzhalter für printf (Forts.)

Wenn Sie eine Zahlenvariable so formatieren wollen, dass sie mindestens eine bestimmte Anzahl von Ziffern lang ist, können Sie die Anzahl der Ziffern zwischen dem Prozent- und dem Platzhalter-Zeichen einfügen:

```
printf("Diese Zahl: %5d ist fünf Ziffern lang.", iNr);
```

Wenn INr im obigen Beispiel den Wert 99 haben würde, so erhielte man folgende Ausgabe:

```
Diese Zahl:    99 ist fünf Ziffern lang.
```

Für Fließkomma-Zahlen können Sie auch die Genauigkeit der Nachkommastellen festlegen, indem Sie nach der Ziffer, welche die Länge angibt, einen Dezimalpunkt und die Genauigkeit einfügen:

```
printf("Ich habe %7.2f Euro in meiner Geldbörse.", fGeld);
```

Wenn der Wert von fGeld 123.456 war, sieht die Ausgabe des obigen Funktionsaufrufs so aus, da die Ausgabe auf zwei Nachkommastellen gerundet wird (zwischen »habe« und »123.46« stehen zwei Leerzeichen weil »123.46« nur 6 Zeichen umfasst):

```
Ich habe  123.46 Euro in meiner Geldbörse.
```

Wenn Sie Zahlen formatieren, werden diese normalerweise rechtsbündig im Ausgabefeld ausgerichtet. Wenn also fGeld den Wert 1.234 hätte, so daß nicht alle Stellen genutzt werden, würde die Ausgabe so aussehen:

```
Ich habe    1.23 Euro in meiner Geldbörse.
```

Wenn die Zahl linksbündig ausgerichtet sein soll, fügen Sie im Formatierungsausdruck ein Minuszeichen (-) ein:

```
printf("Ich habe %-7.2f Euro in meiner Geldbörse.", fGeld);
```

Die Ausgabe wird dann so aussehen:

```
Ich habe 1.23    Euro in meiner Geldbörse.
```

Hierbei werden die zum Auffüllen des Ausgabefeldes benötigten Leerzeichen hinter die Zahl geschrieben. Wenn Sie ein Pluszeichen (+) direkt hinter das Prozent-Zeichen setzen, wird die Zahl immer mit einem Vorzeichen ausgegeben, ansonsten wird nur ein Minuszeichen (-) ausgegeben, wenn die Zahl einen negativen Wert hat:

```
printf("Ich habe %+7.2f Euro auf dem Konto.", fGeld);
```

Das bewirkt, dass bei positiven Zahlen ein Pluszeichen ausgegeben wird:

```
Ich habe +123.46 Euro auf dem Konto.
```

Hat allerdings fGeld den Wert -123.456, so ergibt sich folgende Ausgabe:

```
Ich habe -123.46 Euro auf dem Konto.
```

Sie können auch nicht druckbare Zeichen in die Strings einfügen, indem Sie einen Backslash ('\'), gefolgt von den Zeichen, welche das nicht druckbare Zeichen repräsentiert, verwenden. So können Sie Tabulator-Sprünge, Zeilenumbrüche usw. einfügen. In Tabelle 2.3 sind die wichtigsten nicht druckbaren Zeichen aufgelistet.

Platzhalter	Beschreibung
\n	Zeilenumbruch. Ein String wird auf mehrere Zeilen verteilt.
\t	Tabulator-Sprung
\r	Zeilenrücklauf - Rückkehr zum Textanfang, jedoch keine neue Zeile
\\	Backslash-Zeichen. Da der Backslash angibt, dass das folgende Zeichen im String ein Platzhalter für ein nicht druckbares Zeichen ist, müssen Sie zwei Backslash-Zeichen verwenden, um einen einzelnen Backslash in den String einzufügen.

Tabelle 2.3: Platzhalter für nicht druckbare Zeichen in printf

2.3 Die Debug-Tools von Visual Studio

Für die detaillierte Überprüfung Ihres Codes während seiner Verwendung bietet Visual C++ einen Satz von Debug-Tools, mit deren Hilfe Sie auf jeder passenden Ebene Ihren Code untersuchen können. Sie können Ihren Code Schritt für Schritt durchlaufen und beobachten, was in seinem Inneren vor sich geht, oder Sie können beobachten, wie von außen Ereignismeldungen an Ihre Anwendung übergeben werden. Welche Bedürfnisse Sie in Bezug auf die Fehlerbeseitigung auch haben - die Chancen stehen gut, dass das benötigte Tool von Visual C++ mitgebracht wird.

Der Debugger von Visual Studio

Das Werkzeug, das Sie hauptsächlich für die Fehlerbeseitigung in Ihren Anwendungen verwenden werden, ist der in Visual Studio eingebaute integrierte Debugger. Mit ihm können Sie Ihren Code Schritt für Schritt durchlaufen und untersuchen, wie alle Variablen und Datenstrukturen verändert werden, während Ihre

Anwendung läuft.

Haltepunkte setzen

Bevor Sie mit der Fehlerbeseitigung in Ihrer Anwendung beginnen, müssen Sie entscheiden, wo Sie anfangen. Gewöhnlich starten Sie nicht mit der ersten Code-Zeile und durchlaufen die gesamte Anwendung (da wäre eine Menge Code zu durchlaufen). Wahrscheinlich wollen Sie nur einen ausgewählten Teil des Codes durchlaufen. Dazu müssen Sie in Ihrem Code einen Haltepunkt setzen.

Ein Haltepunkt (Breakpoint) ist eine Stelle in einer Anwendung, an der die Ausführung angehalten wird, wenn die Anwendung in einem Debugger läuft. Mit anderen Worten, die Anwendung stoppt in der Code-Zeile mit dem Haltepunkt. So können Sie Ihre Anwendung ganz normal laufen lassen; der Haltepunkt hält sie an der Stelle an, ab der Sie Zeile für Zeile weitergehen möchten.

Von Zeit zu Zeit hat man einen Haltepunkt auf einer ungültigen Code-Zeile, beispielsweise in einem Kommentar oder in der falschen Zeile einer Anweisung, die sich über mehrere Zeilen erstreckt. In diesem Fall erhalten Sie beim Start der Anwendung im Debugger eine Meldung, die Sie über die Situation informiert. Die Haltepunkte werden automatisch in die nächste gültige Code-Zeile verschoben. Außerdem geht die Anwendung bei der allerersten Zeile ausführbaren Codes in den schrittweisen Modus, tief innerhalb von MFC.

Sie können Haltepunkte mit der Taste (F9) setzen und entfernen oder Sie können Neuer Haltepunkt aus dem Menü Debuggen wählen (das Menü Debuggen ist nur aktiv, wenn ein Projekt geöffnet ist). Wenn Sie das Menü verwenden, wird ein Dialog für die Eigenschaften des Haltepunktes geöffnet (siehe Abbildung 2.4), in dem Sie einstellen können, unter welchen Bedingungen der Haltepunkt die Ausführung des Codes anhalten soll und unter welchen nicht.

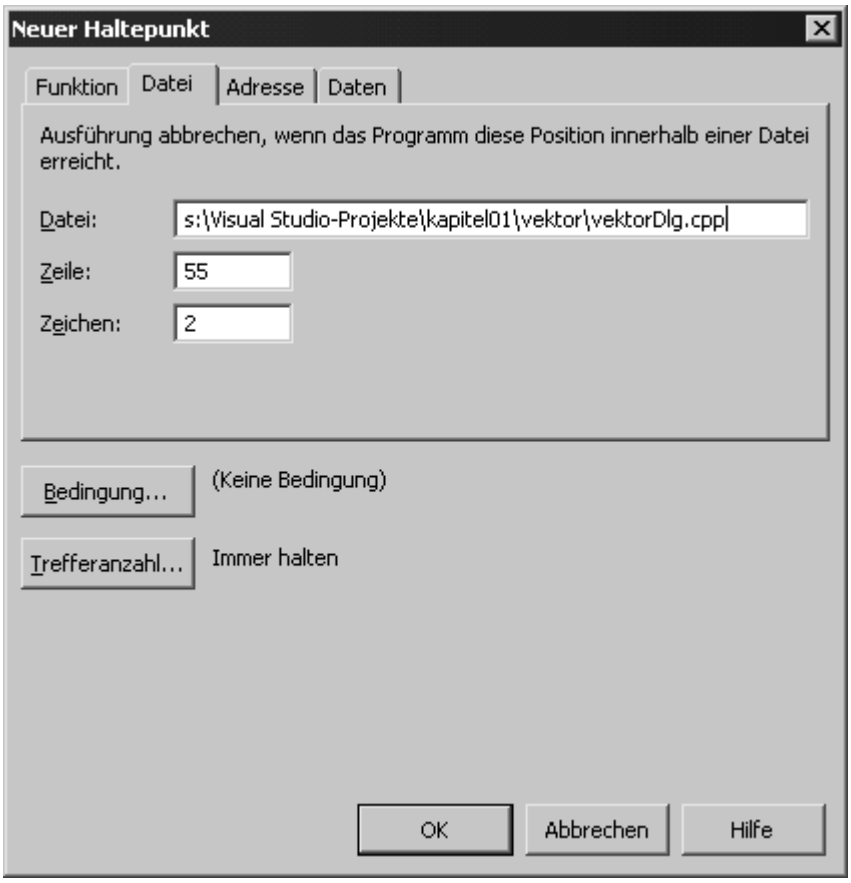


Abbildung 2.4: Der Eigenschaften-Dialog für einen neuen Haltepunkt

Wenn Sie einen Haltepunkt über das Menü Debuggen hinzufügen, ist es einfacher, die Registerkarte Datei zu verwenden, da so der Haltepunkt in einer bestimmten Code-Zeile in die Quellcode-Datei eingefügt wird. In der Registerkarte Funktion müssen Sie die Funktion und die Zahl der Zeile in der Funktion angeben, in der Sie den Haltepunkt platzieren wollen.

Sie können die Haltepunkte-Ansicht öffnen, indem Sie im Menü Debuggen aus dem Untermenü Fenster den Punkt Haltepunkte auswählen. In der Haltepunkte-Ansicht können Sie einen wählen, dessen Eigenschaften Sie ändern möchten, neue hinzufügen, Haltepunkte deaktivieren oder löschen oder in die Code-Zeile eines bestimmten Haltepunktes springen (siehe Abbildung 2.5). In Tabelle 2.4 sind die Schaltflächen der Werkzeugleiste in der Haltepunkte-Ansicht beschrieben.

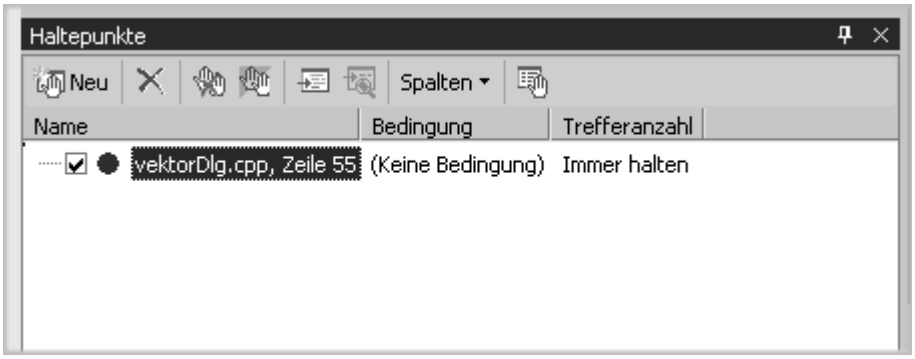


Abbildung 2.5: Die Haltepunkte-Ansicht

Schaltfläche	Beschreibung
Neuer Haltepunkt 	Fügt einen Haltepunkt ein. Sie müssen die Funktion oder Datei sowie die Zeilennummer in der Funktion oder Datei angeben.
Haltepunkt löschen 	Löscht den ausgewählten Haltepunkt aus dem Projekt
Alle Breakpoints löschen 	Löscht alle derzeit im Projekt gesetzten Breakpoints
Aktivieren/Deaktivieren 	Aktiviert/deaktiviert alle Breakpoints im Projekt. So können Sie die Breakpoints deaktivieren, damit die Anwendung nicht anhält, ohne sie löschen zu müssen.
Zum Quellcode 	Zeigt den ausgewählten Haltepunkt im Quellcode des Projekts an
Zur Disassemblierung 	Zeigt den ausgewählten Haltepunkt im Assembler-Code des Projekts an, der dem ausgeführten Maschinencode entspricht
Spalten 	Zeigt eine Liste von Datenspalten an, die in der Breakpoints-Ansicht angezeigt werden können
Eigenschaften 	Öffnet den Eigenschaften-Dialog für den ausgewählten Haltepunkt, sodass Sie die Bedingungen sehen können, unter denen der Haltepunkt ausgelöst wird

Tabelle 2.4: Schaltflächen auf der Haltepunkt-Werkzeugleiste

Schrittweise im Code navigieren

Nachdem Sie Ihre Anwendung mit einem Haltepunkt angehalten haben, müssen Sie immer um eine Zeile (oder mehrere Zeilen) weitergehen können. Dafür gibt es mehrere Funktionen. Diese Funktionen finden sich alle im Menü Debuggen sowie auf der Debug-Werkzeugleiste. In Tabelle sind die Schaltflächen beschrieben.

Schaltfläche	Beschreibung
Nächste Anweisung anzeigen 	Markiert die Code-Zeile, die als Nächstes ausgeführt wird
Einzelschritt 	Handelt es sich bei der aktuellen Code-Zeile um eine Funktion, betritt der Debugger diese. Andernfalls springt er zur nächsten Code-Zeile in der aktuellen Funktion.
Prozedurschritt 	Springt in die nächste Code-Zeile der aktuellen Funktion. Diese Methode betritt keine Funktionen, sie führt die Funktion nur aus und springt mit ihren Ergebnissen in die nächste Zeile.
Ausführen bis Rücksprung 	Verlässt die aktuelle Funktion und springt an die Stelle ihres Aufrufs

Tabelle 2.5: Schaltflächen in der Werkzeugleiste zum Navigieren im Code

Lokale und aktuelle Variablen untersuchen

Wenn Sie Ihre Debug-Sitzung begonnen haben, wird Ihnen ein neues Fenster unter dem Code-Editierbereich der Visual Studio-Umgebung auffallen. Dabei handelt es sich um den Variablen-Bereich, in dem Sie Änderungen der Werte von Variablen und Objekten verfolgen können, während Sie Ihre Anwendung durchlaufen. Standardmäßig wird die Lokal-Ansicht gezeigt, in der alle in der aktuellen Funktionen definierten Variablen angezeigt werden. Oft sind jedoch auch andere Variablen von Interesse, die nicht in der aktuellen Funktion definiert werden. Hier zeigt die Auto-Ansicht die in der aktuellen Code-Zeile verwendeten Variablen zusammen mit den in der vorhergehenden Zeile verwendeten oder geänderten Variablen an (siehe Abbildung 2.6).

Wahrscheinlich werden Sie beide Methode abwechselnd verwenden, um die Variablen in Ihrer Anwendung anzuzeigen. In diesem Modus können Sie alle derzeit definierten und verwendeten Variablen sehen. So flexibel dieser Modus auch ist, so ist er dennoch nicht die einzige Methode, mit der Sie Ihre Variablen betrachten werden.

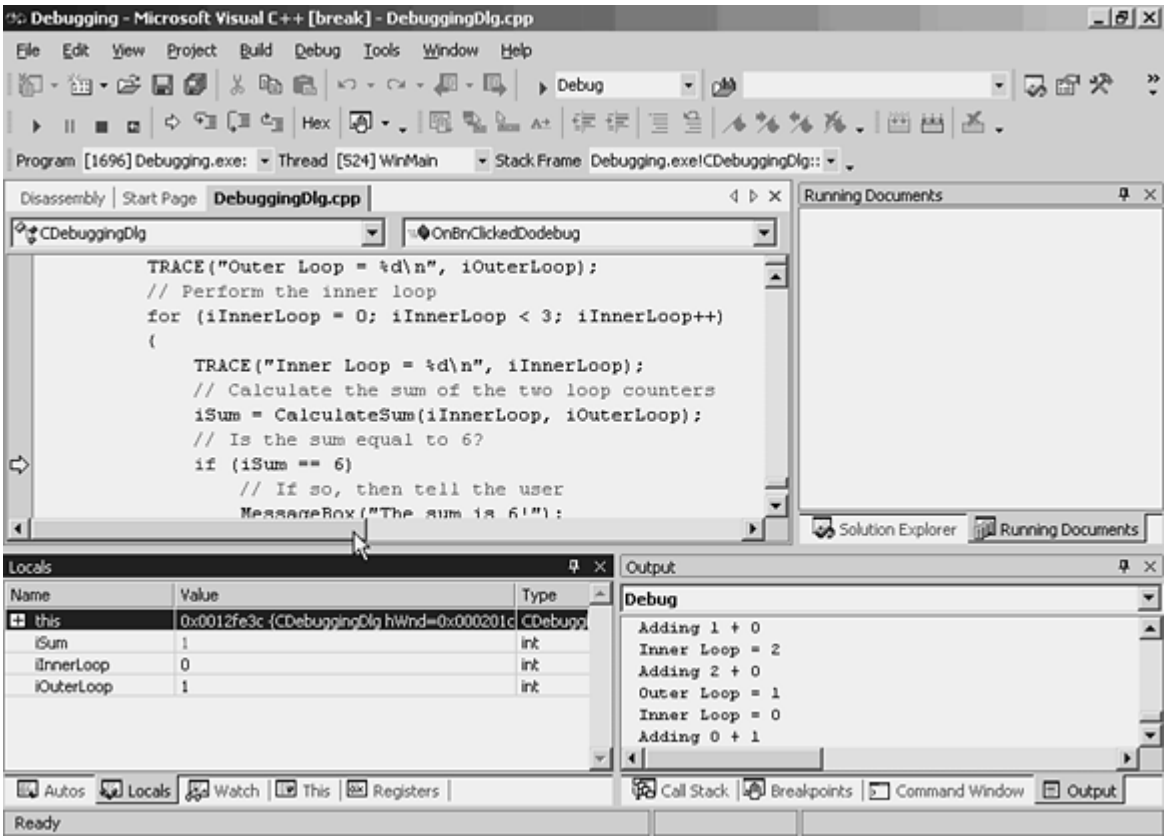


Abbildung 2.6: Die lokalen Variablen sind im Lokal-Fenster sichtbar, die aktuell verwendeten im Auto-Fenster.

Bestimmte Variablen beobachten

Manchmal müssen Sie eine bestimmte Variable oder ein bestimmtes Objekt beobachten, während Sie Ihren Code durchlaufen. Alle anderen Daten in Ihrer Anwendung interessieren Sie nicht, es geht Ihnen nur um diese bestimmte Variable (oder diesen Variablensatz). In diesen Fällen müssen Sie den Wert der Variable auch dann sehen, wenn sie in der aktuellen Code-Zeile nicht verwendet wird.

Die erste Möglichkeit, den aktuellen Wert einer bestimmten Variablen zu überprüfen, ist, den Mauszeiger über die Variable zu bewegen und ihn ein paar Sekunden dort zu lassen. Der Debugger von Visual Studio zeigt den aktuellen Wert der Variablen in Form eines Popup-Texts an - wie die Quickinfo bei einer Schaltfläche in der Werkzeugleiste, die den Namen oder die Funktion der Schaltfläche anzeigt, wenn Sie den Mauszeiger darüber bewegen (siehe Abbildung 2.7).

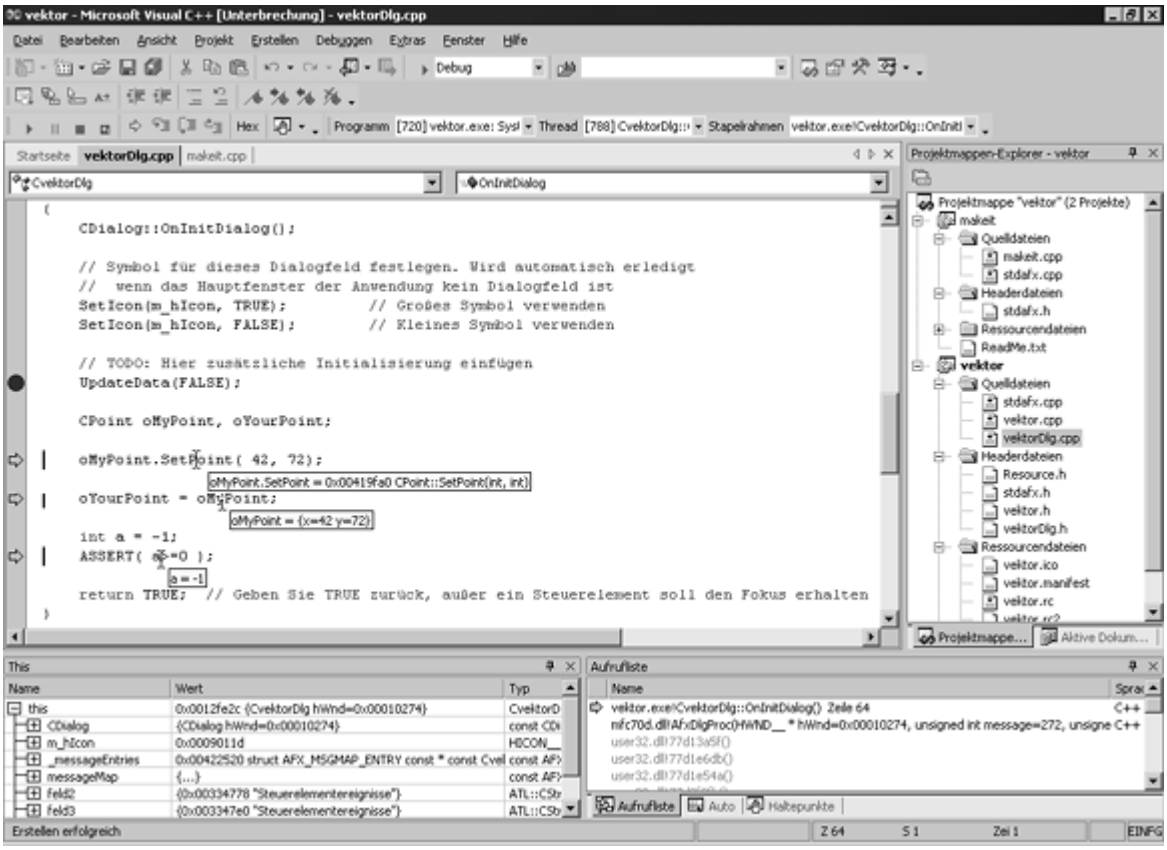


Abbildung 2.7: Verwenden Sie Quick Watch, um den aktuellen Wert einer Variablen, eines Objektes oder einer Funktion zu überprüfen.

Die andere Möglichkeit ist die Überwachungs-Ansicht. Wenn Sie Fenster / Überwachen / Überwachen 1 aus dem Menü Debuggen wählen, finden Sie eine einfache Tabelle, in der Sie in der ersten Spalte Variablennamen eingeben. In der zweiten Spalte werden die aktuellen Werte der Variablen angezeigt, in der dritten der Datentyp (siehe Abbildung 2.8).

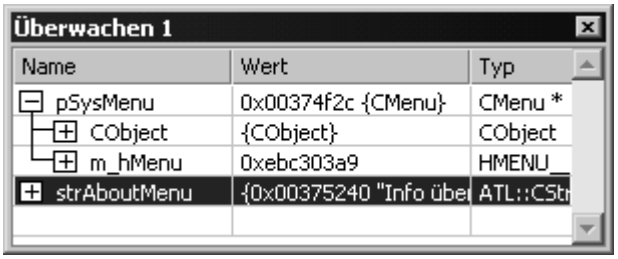


Abbildung 2.8: In der Überwachungs-Ansicht können Sie bestimmte Variablen und Objekte beobachten.

Die Variable this untersuchen

Die Karteikarte ganz rechts im Variablenfenster bietet eine Baumansicht der Variablen this. Sie können durch das This-Fenster navigieren, um jeden Aspekt des aktuellen Objekts sowie seinen Zustand und Wert zu finden (siehe Abbildung 2.9). Hierbei können Sie durch Auswählen der Plus-Symbole einzelne Objekte detaillierter untersuchen.

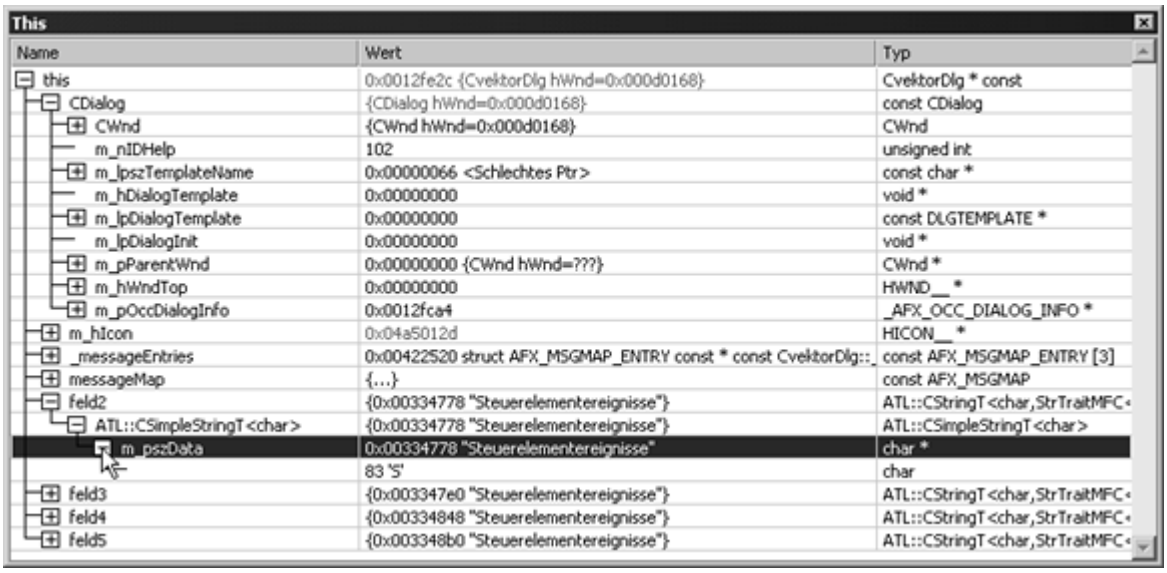


Abbildung 2.9: Im This-Fenster können Sie alle Aspekte des aktuellen Objekts untersuchen.

C++-Exkurs: Was ist this?

In C++ wird das aktuell vom Code ausgeführte Objekt immer als this bezeichnet. Es handelt sich dabei im Grunde um eine bewegliche Variable, die immer verwendet werden kann, um sich auf die Klasse zu beziehen, in der Sie Code schreiben. Wenn Sie beispielsweise Funktionalität zur Klasse für ein Fenster hinzugefügt haben und das Fenster an ein anderes Objekt übergeben müssen, können Sie this als entsprechenden Parameter für die Funktion verwenden, die Sie im anderen Objekt aufrufen. (Tatsächlich übergeben Sie nicht das Fenster, sondern einen Zeiger auf das Fenster, aber das ist ein Thema für einen späteren Zeitpunkt in diesem Buch.)

Die Aufrufliste überprüfen

Manchmal befinden Sie sich inmitten Ihres Codes, tief im Herzen Ihrer Anwendung, und müssen herausfinden, auf welchem Weg Sie dahin gekommen sind. Diese Information finden Sie in der Ansicht Aufrufliste (Call Stack). Die Aufruflisten-Ansicht befindet sich in der Visual Studio-Umgebung rechts unten (siehe Abbildung 2.10). Von hier aus können Sie die Reihe der Funktionsaufrufe betrachten und an jeder Stelle im Stapel klicken, um diese im Code zu sehen und den Zustand der Variablen zum Zeitpunkt des Funktionsaufrufs zu untersuchen.

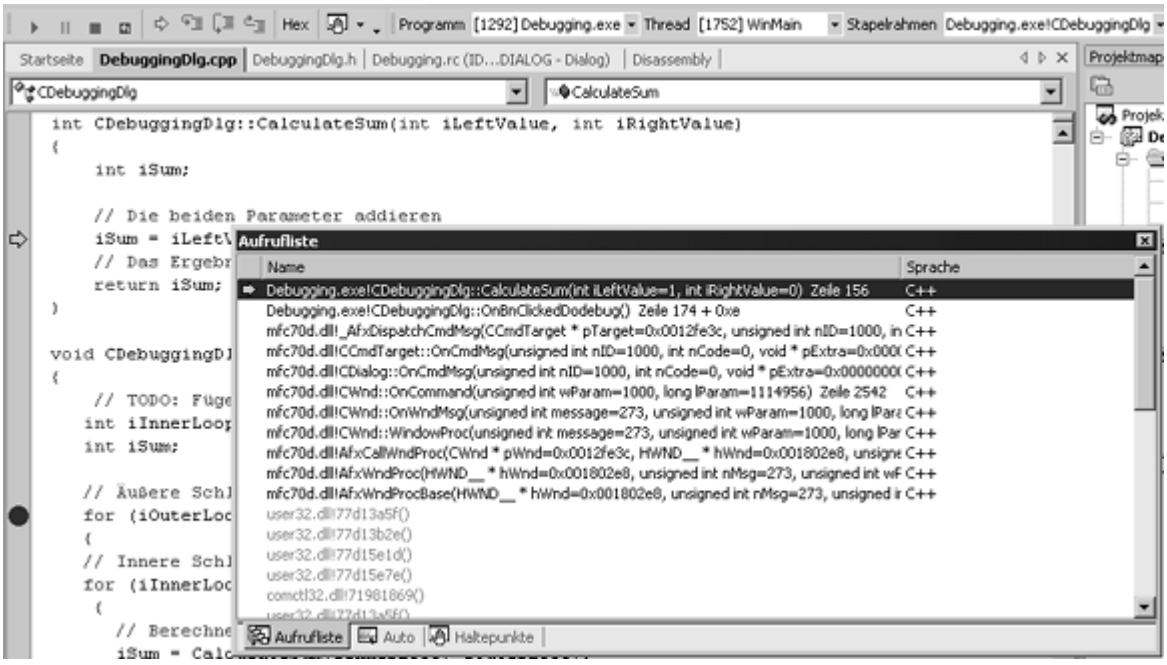


Abbildung 2.10: In der Aufruflisten-Ansicht können Sie sehen, wie Sie dahin gekommen sind, wo Sie gerade sind.

Befinden sich im Aufrufstapel aufgelistete Funktionsaufrufe in einem kompilierten Objekt, dessen Code nicht verfügbar ist, können Sie nicht im Quellcode an diese Stelle springen. Stattdessen finden Sie sich möglicherweise mitten in einem Knäuel Assembler-Code wieder, da der Debugger Sie im Maschinencode zu der Stelle des Funktionsaufrufs bringt.

Eine Situation, in der ich den Aufrufstapel verwende, ist, wenn ich einen ASSERT-Fehler in einer der MFC-Klassen erhalte. In diesem Fall verwende ich den Aufrufstapel, um zu sehen, wo die Anwendung meinen Code verlassen hat, und um festzustellen, was mein Code an den MFC-Code übergeben hat, der den Fehler erzeugt hat.

Spy++

An dieser Stelle lernen Sie ein anderes Debug-Werkzeug kennen. Dieses Tool ist unabhängig von der Visual Studio-Umgebung, wird aber mit ihr zusammen ausgeliefert. Dieses Tool mit dem Namen Spy++ findet sich im Menü Extras. Es erlaubt Ihnen, alle an eine bestimmte Anwendung gesendeten Ereignismeldungen zu sehen und zu protokollieren. Mit dieser Anwendung können Sie auch alle derzeit auf dem System existierenden Fenster und die Beziehungen zwischen diesen Fenstern sehen.

Wenn Sie Spy++ zum ersten Mal starten, zeigt es eine Baumansicht aller derzeit existierenden Fenster auf Ihrem System an (siehe Abbildung 2.11). Sie können auch die aktuellen Prozesse oder Threads betrachten, um das gesuchte Fenster zu finden (siehe Abbildung 2.12).

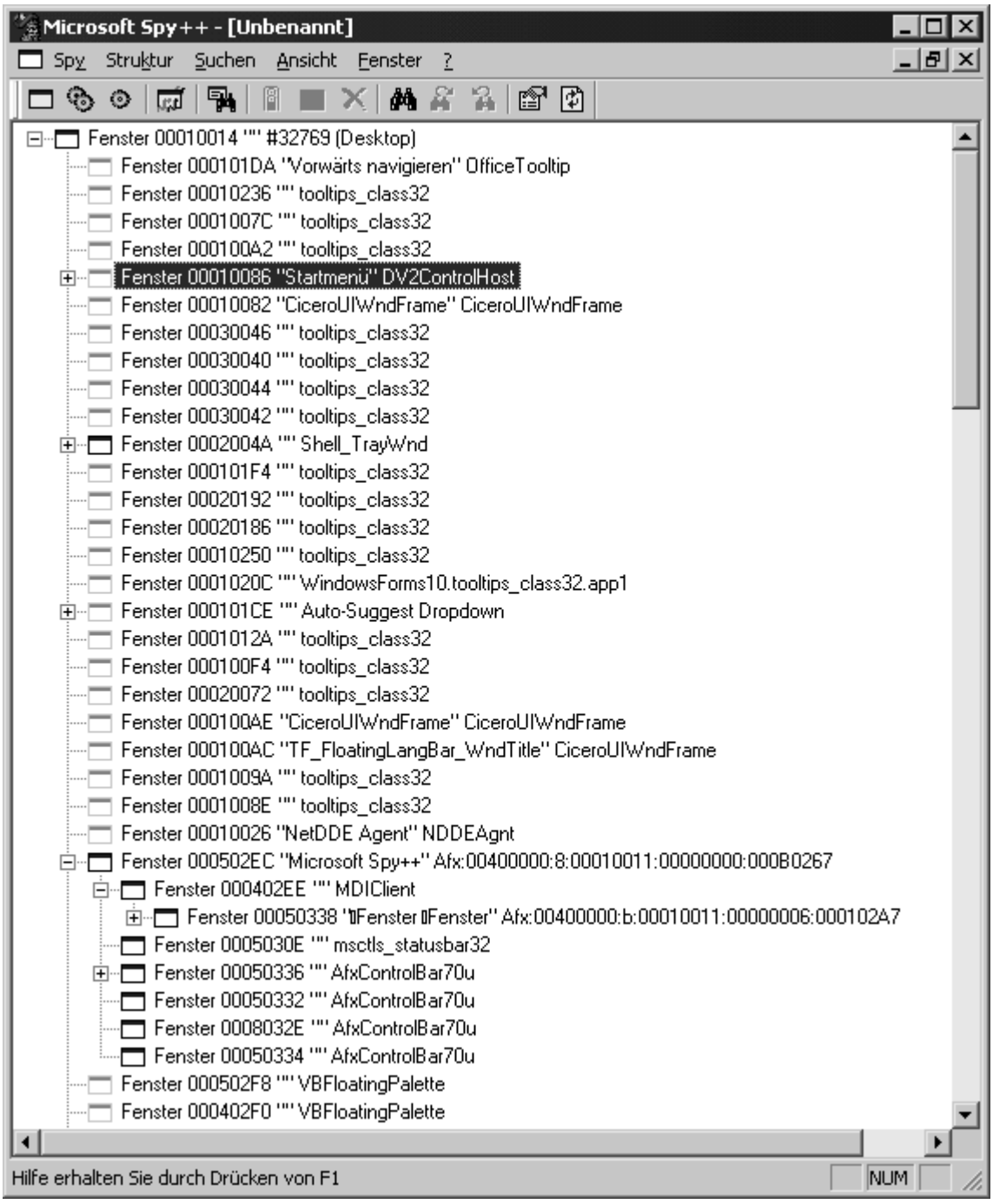


Abbildung 2.11: Spy++ zeigt die existierenden auf dem System laufenden Fenster.

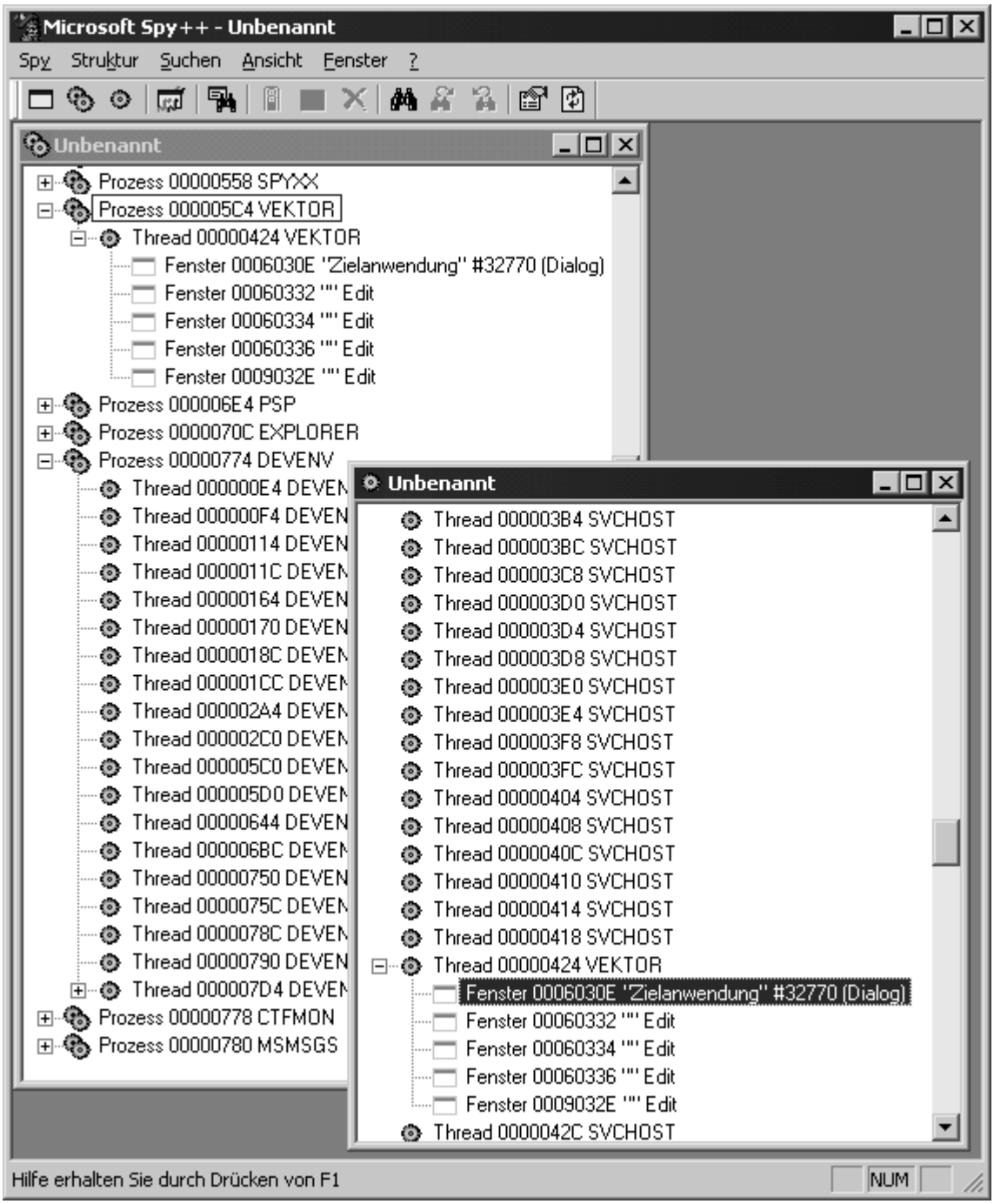


Abbildung 2.12: Über Prozesse oder Threads lassen sich verschiedene Fenster lokalisieren.

Wenn Sie Spy++ zum ersten Mal starten, starren Sie möglicherweise auf all die aufgelisteten Fenster und denken, dass Sie gar nicht so viele Fenster auf Ihrem Rechner offen haben. Denken Sie daran, dass all das als Fenster betrachtet wird, was Meldungen empfangen kann. Alle Kontrollen, die Sie in einem Dialogfenster platzieren, sind ebenfalls »Fenster«. Und dann sind da noch viele offene Fenster, die Sie nicht sehen, die aber trotzdem Meldungen vom Betriebssystem erhalten, und die werden alle in Spy++ angezeigt.

Nachdem Sie das Fenster gefunden haben, für das Sie die Ereignismeldungen sehen wollen, markieren Sie es und wählen dann Meldungen protokollieren aus dem Menü Spy, um den Dialog in Abbildung 2.13

anzuzeigen. In diesem Dialog können Sie angeben, wie die Meldungen abgefangen werden sollen. Möchten Sie alle Meldungen für die übergeordnete Anwendung oder das übergeordnete Fenster? Auf den anderen Registerkarten können Sie sogar angeben, welche Ereignismeldungen Sie sehen wollen und welche nicht. Wenn Sie eingestellt haben, wie Sie die Meldungen sehen wollen, klicken Sie auf OK und Spy++ beginnt damit, die von dem oder den angegebenen Fenster(n) erhaltenen Meldungen zu protokollieren (siehe Abbildung 2.14).

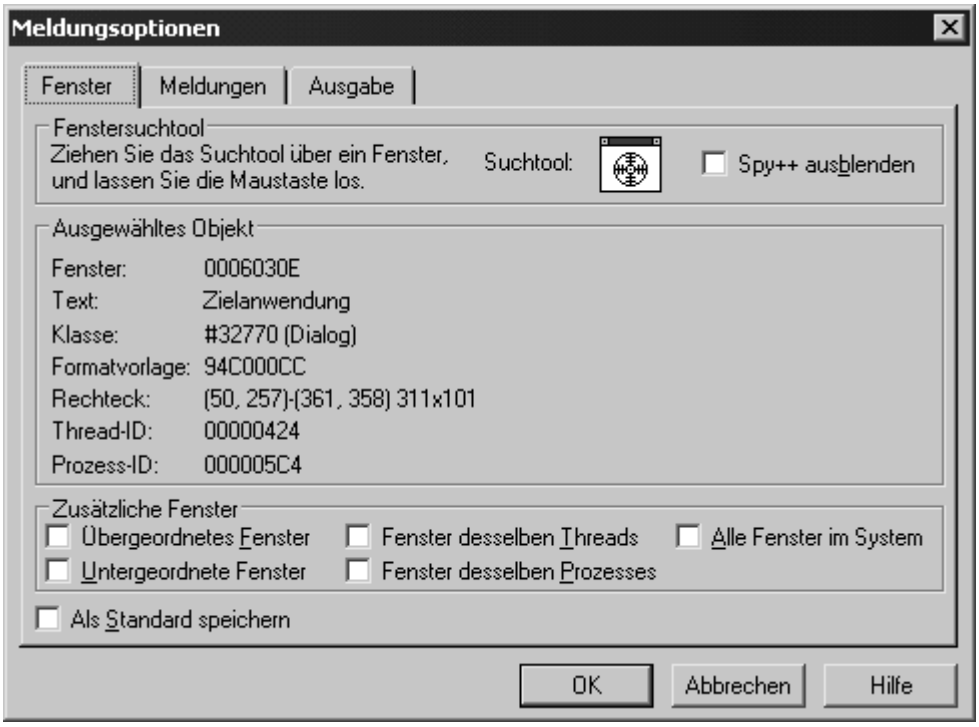


Abbildung 2.13: Konfiguration, nach der Ereignismeldungen empfangen werden sollen

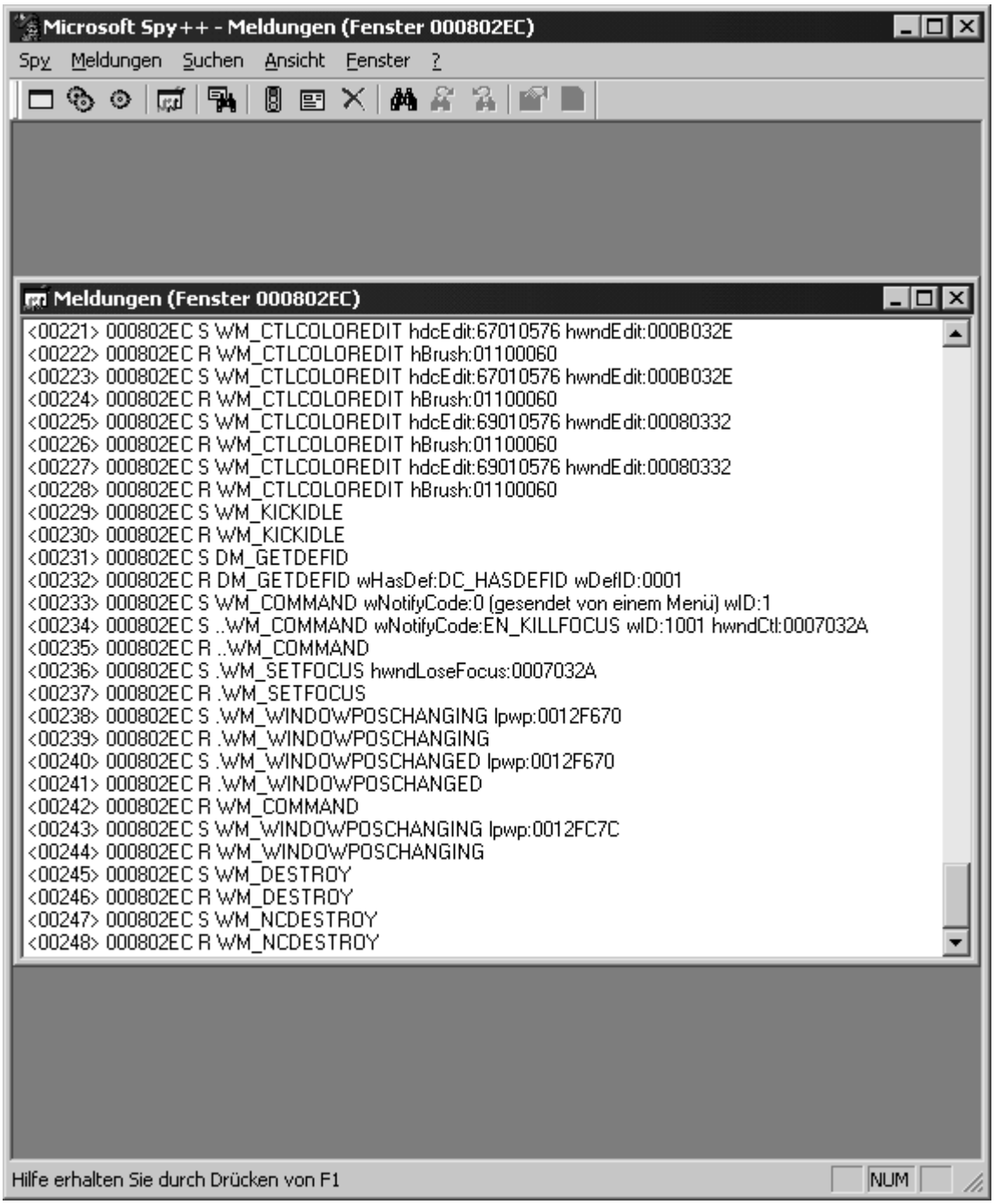


Abbildung 2.14: Die vom angegebenen Anwendungsfenster empfangenen Meldungen

An laufende Prozesse anhängen

Ein letztes nützliches Tool in der Visual Studio-Umgebung ermöglicht, den Debugger von Visual Studio an eine laufende Anwendung anzuhängen. Natürlich werden Sie eine Anwendung, in der Sie Fehler beseitigen wollen, meistens im Debugger starten und dieses Tool nicht benötigen. Es gibt jedoch Situationen, in denen es sehr hilfreich sein kann.

Angenommen, Sie haben eine Anwendung, die in einer Endlosschleife festzuhängen scheint. Wenn Sie sie im Debugger starten, können Sie ihre Ausführung einfach abbrechen und herauszufinden versuchen, wo das Problem liegt. Wenn nicht, müssen Sie den Debugger irgendwie an Ihre Anwendung anhängen.

Eine Endlosschleife ist eine Situation, in der eine Schleife in Ihrem Code niemals beendet wird. Es kann sich um eine Schleife mit Bedingung handeln, in der die Ausstiegsbedingung nie ausgelöst wird, oder um eine for-Schleife, deren Zähler nicht erhöht wird oder um etwas Ähnliches. Kurz, es ist eine Stelle in Ihrem Code, an der die Anwendung festhängt. Gewöhnlich können Sie in diesem Fall nur noch die Anwendung abschießen und dabei alles verlieren, was sie bisher schon verarbeitet hatte.

Wenn Sie das Menü Debuggen öffnen, finden Sie einen Menüeintrag namens Prozesse. Wenn Sie diesen Eintrag auswählen, erhalten Sie den in Abbildung 2.15 gezeigten Dialog. Dieser Dialog enthält eine Liste aller auf dem Computer laufenden Prozesse. In diesem Dialog können Sie die gewünschte Anwendung auswählen und auf Anfügen klicken. Nun wird der Dialog in Abbildung 2.16 angezeigt, in dem Sie auswählen können, welchen Debugger Sie für die Anwendung verwenden möchten. Für Ihre Visual C++-Anwendungen verwenden Sie die Option Native. Wenn Sie die Anwendung als verwaltete C++-Anwendung erstellt haben (dazu kommen wir in einigen Tagen), sollten Sie die Option Common Language Runtime verwenden. Klicken Sie auf OK; Ihre Anwendung ist nun im Debugger und alle vorhandenen Breakpoints werden ausgelöst. Wenn Sie keine Breakpoints haben, können Sie auf Unterbrechen klicken und an der Stelle, an der sich der Code gerade befindet, einen Haltepunkt erzwingen.

Das Menü Debuggen ist nur aktiv, wenn ein Projekt geöffnet ist. Um dem hier beschriebenen Ablauf zu folgen, müssen Sie entweder ein neues Projekt erstellen oder ein existierendes öffnen.

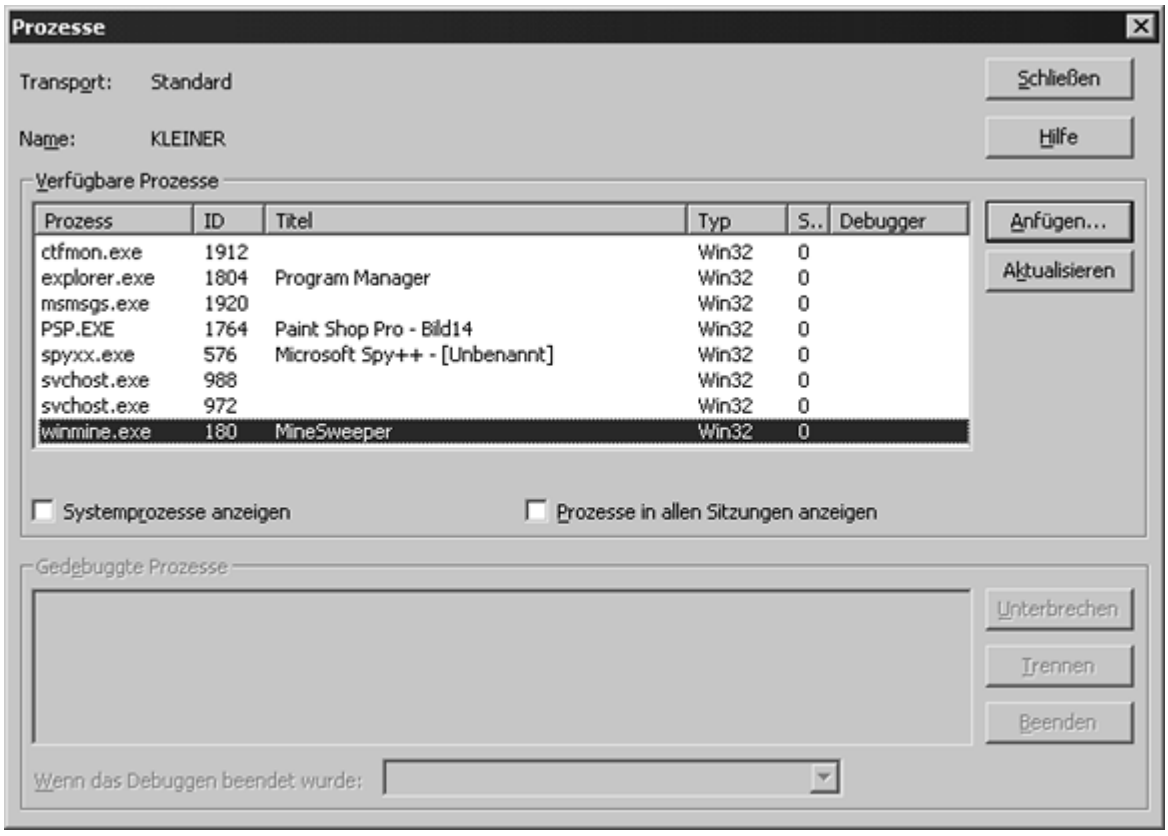


Abbildung 2.15: Sie können eine laufende Anwendung auswählen, um den Debugger an

sie anzuhängen.

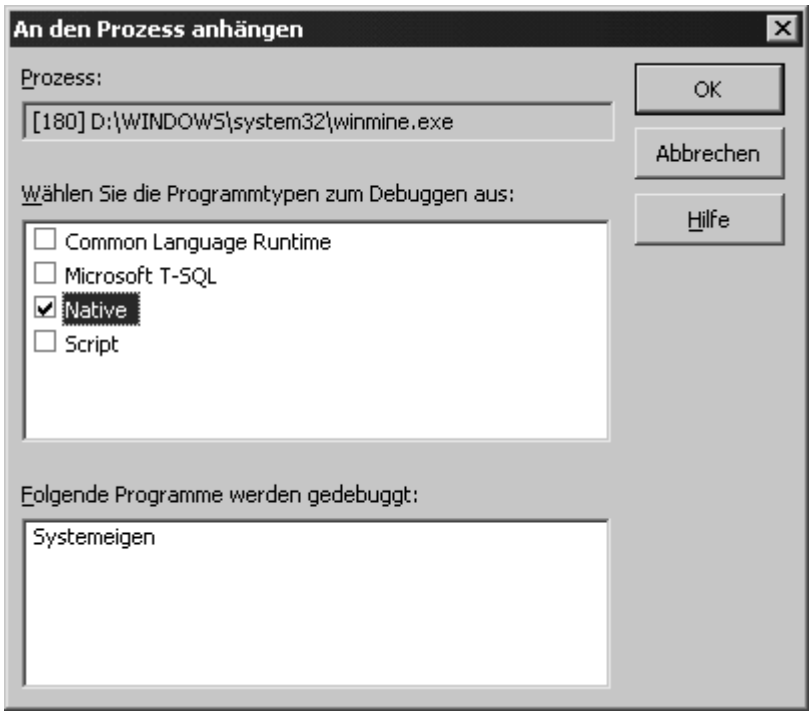


Abbildung 2.16: Sie können wählen, welchen Debugger Sie verwenden möchten.

2.4 Eine fehlerhafte Anwendung erstellen

Nun, da Sie die grundlegenden Funktionen der in Visual Studio enthaltenen Debug-Tools kennen, werden Sie eine fehlerhafte Anwendung erstellen und den Debugger von Visual Studio verwenden, um Probleme zu finden und zu korrigieren. Wenn Sie bereits C++ kennen und Erfahrung mit der Verwendung von Debuggern haben, können Sie den Rest dieses Kapitels überspringen.

Die Anwendung, die Sie erstellen werden, ist der gestrigen sehr ähnlich. Sie hat zwei Schaltflächen, von denen eine die Anwendung schließt und die andere ein paar Schleifen und Berechnungen ausführt. Das Hauptfenster sieht wie das in Abbildung 2.17 aus.

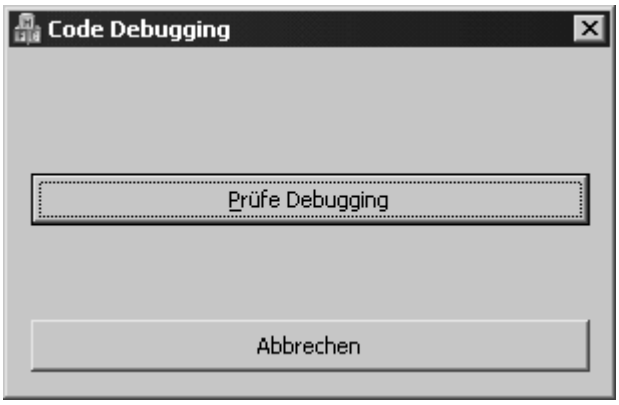


Abbildung 2.17: Die fertige Anwendung

Wenn die obere Schaltfläche angeklickt wird, führt die Anwendung zwei verschachtelte Schleifen aus und ruft eine Funktion auf, welche die Zähler beider Schleifen addiert. Wenn die beiden Zähler zusammen die größtmögliche Summe erreichen, wird eine Meldung angezeigt, um den Benutzer davon zu informieren (siehe Abbildung 2.18).

Abbildung 2.18: Die Anzeige der Meldung

Um diese Anwendung zu erstellen, müssen Sie folgende Schritte ausführen:

1. Erzeugen Sie mit dem MFC Anwendungs-Assistent einen Anwendungsrahmen.

2. Gestalten Sie den Hauptdialog neu, sodass die beiden Schaltflächen wie oben angezeigt werden.
3. Fügen Sie die Funktion ein, um die beiden Zahlen zu addieren.
4. Fügen Sie die Ereignis-Funktion für das Klickereignis auf der oberen Schaltfläche ein.
5. Fügen Sie Code in die Klickereignis-Funktion ein, der zwei verschachtelte Schleifen ausführt und in jeder Schleife die Addierfunktion aufruft.

Den Anwendungsrahmen erzeugen

Zuerst müssen Sie die Anwendungsrahmen erzeugen und das Hauptdialogfeld neu zeichnen. Dazu folgen Sie denselben Schritten wie gestern, mit ein paar kleinen Änderungen.

1. Erstellen Sie wie gestern in den Schritten 1-4 des Abschnitts »Den Projekt- Arbeitsbereich anlegen« ein neues Projekt namens Debugging.

2. Verwenden Sie wie gestern in den Schritten 1-4 des Abschnitts »Den Anwendungsrahmen mit dem MFC Anwendungs-Assistent erstellen« den MFC Anwendungs-Assistent, um den Anwendungsrahmen zu erstellen. Geben Sie in Schritt 2 als Titel für das Hauptfenster der Anwendung Code Debugging ein.
3. Gestalten Sie wie gestern in den Schritten 1-7 des Abschnitts »Das Anwendungsfenster gestalten« Ihr Dialogfenster neu.
4. Ändern Sie für die OK-Schaltfläche die ID-Property auf ID_DODEBUG und die Eigenschaft Beschriftung auf &Prüfe Debugging. Ihr Hauptdialog sollte jetzt wie der in Abbildung 2.19 aussehen.

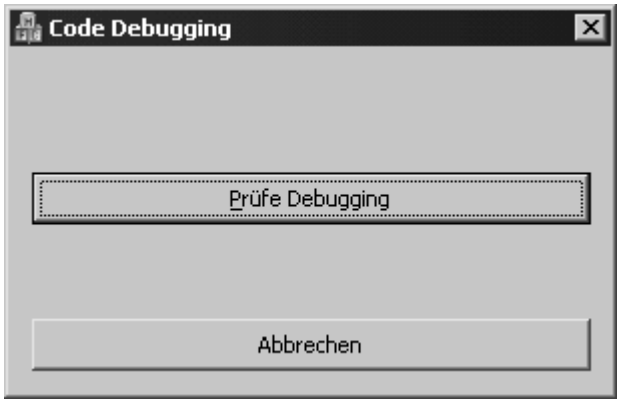


Abbildung 2.19: Der umgestaltete Hauptdialog

Eine Funktion hinzufügen

Als Nächstes müssen Sie die Funktion Hinzufügen, welche die beiden Zähler addiert und das Ergebnis

zurückgibt. Um diese Funktion hinzuzufügen, führen Sie folgende Schritte aus:

1. Wählen Sie die Klassenansicht (Class View).

2. Rechtsklicken Sie auf die Klasse CDebuggingDlg (möglicherweise müssen Sie die Baumansicht erweitern, um die Klasse CDebuggingDlg zu sehen).
3. Wählen Sie aus dem Kontext-Menü Hinzufügen / Funktion hinzufügen (siehe Abbildung 2.20) aus.

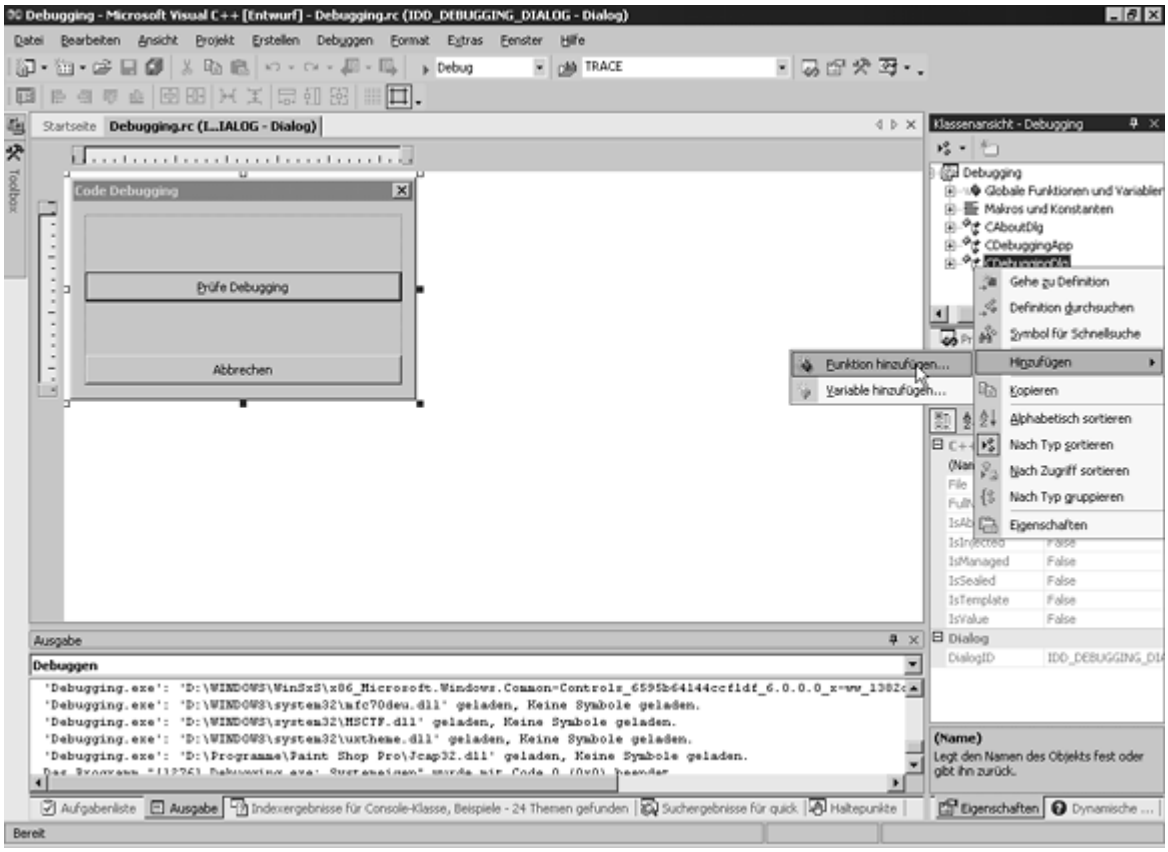


Abbildung 2.20: Eine Funktion zur Dialog-Klasse hinzufügen

4. Geben Sie im Assistent zum Hinzufügen von Funktionen int als Rückgabetyp und CalculateSum als Funktionsname an.
5. Geben Sie int als Parameter-Typ und iLeftValue als Parameter-Name ein und klicken Sie auf Hinzufügen. Hierdurch wird iLeftValue in der Parameter-Liste platziert. Fügen Sie auf die gleiche Art einen zweiten Parameter namens iRightValue ein. Der Assistent sollte jetzt wie in Abbildung 2.21 aussehen.

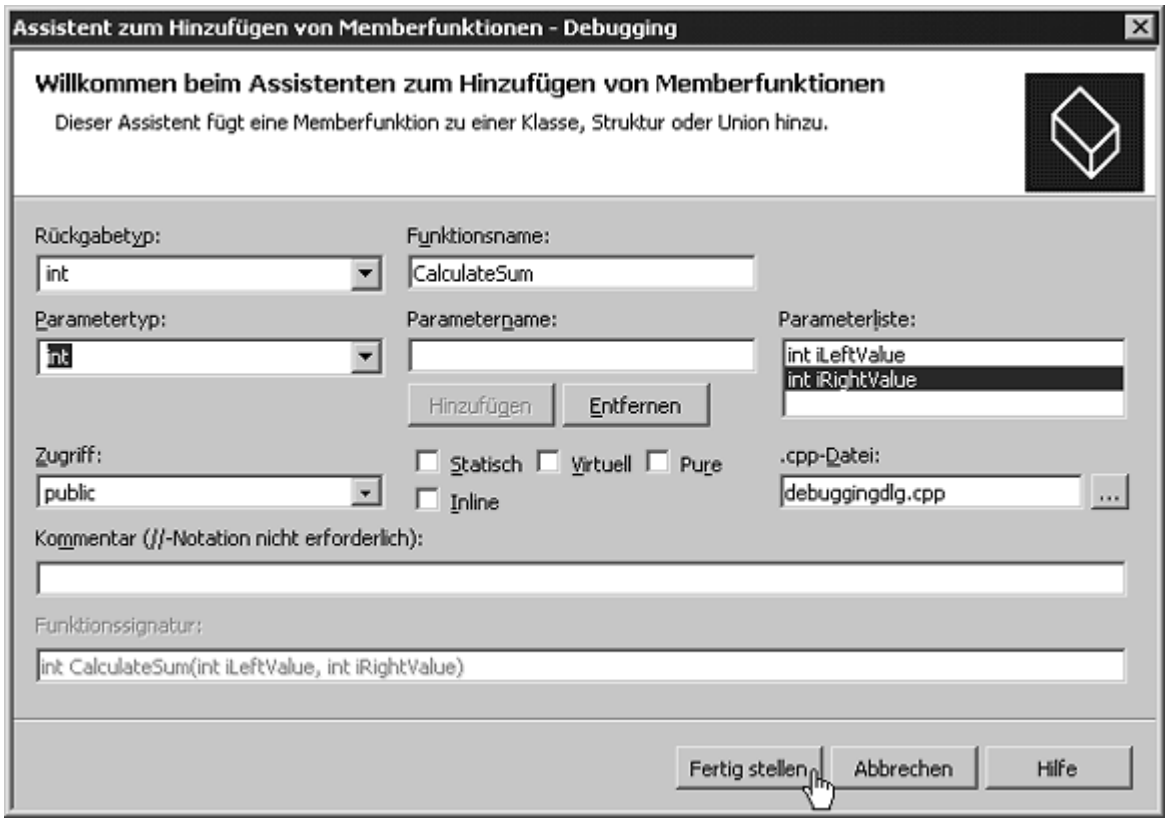


Abbildung 2.21: Der Dialog-Klasse eine Funktion hinzufügen

6. Klicken Sie auf Fertig stellen, um die neue Funktion hinzuzufügen.
7. Fügen Sie den fett gedruckten Code aus Listing 2.1 in die Funktion CalculateSum ein.

Listing 2.1: DEBUGGINGDLG.CPP: die Funktion CalculateSum

```
1: int CDebuggingDlg::CalculateSum(int iLeftValue, int iRightValue)
2: {
3:    int iSum;
4:
5:    // Die beiden Parameter addieren
6:    iSum = iLeftValue - iRightValue;
7:    // Das Ergebnis zurückgeben
8:    return iSum;
9: }
```

Ja, der obige Code ist FALSCH! Sie brauchen falschen Code in der Anwendung, um Fehler zum Beseitigen zu haben. Um diesen Code fehlerhaft zu machen, haben Sie angegeben, den zweiten Wert vom ersten zu subtrahieren, anstatt die beiden zu addieren.

C++-Exkurs: Operatoren in C/C++

C++ besitzt die üblichen Operatoren für die Addition und Subtraktion von Werten. Andere Operatoren entsprechen nicht denen anderer Programmiersprachen. In Tabelle 2.6 sind die gebräuchlichsten Operatoren aufgelistet:

Operator	Beschreibung
+	Addiert zwei Zahlen
-	Subtrahiert die zweite Zahl von der ersten
*	Multipliziert zwei Zahlen
/	Dividiert die erste Zahl durch die zweite
%	Führt eine Modulo-Division aus (gibt den Rest einer Division zurück)

Tabelle 2.6: Operatoren in C/C++

Zusätzlich gibt es Zuweisungsoperatoren. Diese Operatoren weisen einer Variablen links des Operators einen Wert zu. In Tabelle 2.7 sind die gebräuchlichsten Zuweisungsoperatoren aufgelistet:

Operator	Beschreibung
=	Gleich. Einfache Zuweisung des Wertes auf der rechten zur Variablen auf der linken Seite
+=	Addiert den Wert auf der rechten Seite zum aktuellen Wert der Variablen auf der linken Seite und weist das Ergebnis der Variablen auf der linken Seite zu
-=	Subtrahiert den Wert auf der rechten Seite vom aktuellen Wert der Variablen auf der linken Seite und weist das Ergebnis der Variablen auf der linken Seite zu
*=	Multipliziert den Wert auf der rechten Seite mit dem aktuellen Wert der Variablen auf der linken Seite und weist das Ergebnis der Variablen auf der linken Seite zu
/=	Dividiert den aktuellen Wert der Variablen auf der linken Seite durch den Wert auf der rechten Seite und weist das Ergebnis der Variablen auf der linken Seite zu

Tabelle 2.7: Zuweisungsoperatoren in C/C++

Nun, da Sie wissen, wie man grundlegende mathematische Operationen in C++ ausführt, werfen Sie einen Blick auf die Struktur der Funktion, die Sie gerade hinzugefügt haben. Beachten Sie als Erstes, dass die einzige nicht als Parameter in die Funktion übergebene Variable in der ersten Zeile der Funktion deklariert wird. In C++ müssen alle Variablen vor ihrer ersten Verwendung deklariert werden. Das könnte direkt vor ihrer Verwendung geschehen, sogar in der Zeile, in der sie verwendet werden:

```
int iSum = iLeftValue - iRightValue;
```

Sie deklarieren Variablen wie folgt: als Erstes in der Zeile kommt der Datentyp, in diesem Falle int (kurz für Integer). Dann folgt ein Leerzeichen oder Tabulator-Sprung (oder so viele Leerzeichen Sie wollen), gefolgt vom Namen der Variablen. Wenn Sie Variablen deklarieren, können Sie mehrere Variablen desselben Typs durch Kommata getrennt in einer Zeile deklarieren:

```
int iVar1, iVar2, iVar3;
```

C++-Regel

Alle Variablen müssen vor ihrer ersten Verwendung deklariert werden.

In Tabelle 2.8 sind die in C++ eingebauten grundlegenden Datentypen aufgelistet:

Datentyp	Beschreibung
int	Integer (je nach Implementation 16 oder 32 Bit)
long	Long Integer (immer 32 Bit)

float	Fließkommazahl mit einfachen Genauigkeit
double	Double (große Fließkommazahl mit höherer Genauigkeit)
char	Zeichen
bool	Ein eingebauter Boolescher Datentyp (ab Visual C++ 5); Werte können true oder false sein

Tabelle 2.8: Die grundlegenden Datentypen in C/C++

Der Datentyp bool ist nicht der gleiche wie der Datentyp BOOL, den man häufig verwendet sieht. BOOL ist als Integer definiert, sein Wert kann TRUE oder FALSE sein. Diese beiden Booleschen Datentypen sind nicht austauschbar, Sie müssen aufpassen, welche Werte Sie mit welchem von ihnen verwenden.

Diese Liste ist zwar nicht vollständig, doch das sind die wichtigsten in die Sprache eingebauten Datentypen. Die meisten anderen Datentypen bauen auf diesen Typen auf.

Die zweite Code-Zeile in der Funktion ist eine einfache mathematische Operation, die das Ergebnis der zuvor deklarierten Variablen zuweist. In der letzten Zeile wird das Ergebnis mit dem Schlüsselwort return an die aufrufende Funktion zurückgegeben. Mit dem Schlüsselwort return werden alle Funktionsergebnisse an die aufrufende Funktion zurückgegeben. Es wird immer wie folgt verwendet:

```
return einwert;
```

Wenn der Funktionstyp void ist, können Sie das Schlüsselwort return ohne Argument verwenden, da die Funktion keinen Rückgabewert haben sollte.

C++-Regel

Das Schlüsselwort return wird immer verwendet, um den Ergebniswert an die aufrufende Funktion zurückzugeben. In jeder Funktion muss mindestens einmal der Befehl return verwendet werden. Zwar können nach einem return noch weitere Anweisungen stehen, diese werden jedoch nicht ausgeführt.

Die Hauptschleife hinzufügen

Nun kommt der letzte Schritt in der Vorbereitung der Fehlerbeseitigung in Ihrer Anwendung. Sie müssen an die obere Schaltfläche in Ihrem Dialog eine Ereignis- Funktion anhängen, indem Sie den Schleifencode hinzufügen. Dazu führen Sie folgende Schritte aus:

1. Fügen Sie der oberen Schaltfläche wie gestern in den Schritten 1 und 2 im Abschnitt »Code in die Anwendung aufnehmen« beschrieben eine Behandlungsroutine hinzu.

2. Wählen Sie wie im gestrigen Schritt 3 die Funktion OnBnClickedDoDebug aus der Klasse CDebuggingDlg aus.
3. Fügen Sie dem Funktionsrahmen den fett gedruckten Code in Listing 2.2 hinzu.

Listing 2.2: DEBUGGINGDLG.CPP: die Funktion OnClickedDodebug

```
1: void CDebuggingDlg::OnBnClickedDodebug(void)
2: {
3:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:    // Benachrichtigung ein.
```

```
 5:    int iInnerLoop, iOuterLoop;
 6:    int iSum;
 7:
 8:    // Äußere Schleife ausführen
 9:    for (iOuterLoop = 0; iOuterLoop < 5; iOuterLoop++)
10:    {
11:    // Innere Schleife ausführen
12:    for (iInnerLoop = 0; iInnerLoop < 3; iInnerLoop++)
13:     {
14:       // Berechne die Summe der beiden Schleifenzähler
15:       iSum = CalculateSum(iInnerLoop, iOuterLoop);
16:       // Ist die Summe gleich 6?
17:       if (iSum = 6)
18:         // Wenn ja, teile es dem Benutzer mit
19:         MessageBox("Die Summe ist 6!");
20:     }
21:   }
22: }
```

In der obigen Funktion ist wieder ein Fehler. Wegen des heutigen Themas »Fehlerbeseitigung« ist das beabsichtigt. Der Fehler findet sich im Vergleich in Zeile 15, ob iSum gleich 6 ist.

C++-Exkurs: Flusskontrolle

Die Funktion in Listing 2.2 verwendet zwei neue Flusskontroll-Statements. Wenn Sie schon eine der höheren Programmiersprachen verwendet haben, konnten Sie wahrscheinlich erkennen, um was es sich handelt, sind jedoch nicht mit ihrer Verwendung in C++ vertraut. Es handelt sich um die for-Schleife (in anderen Programmiersprachen oft als for...next-Schleife bezeichnet) und um die if-Bedingung (in anderen Programmiersprachen oft als if...then bezeichnet).

Für die for-Schleife haben Sie nur das Statement for, das von einer einzelnen Code-Zeile gefolgt sein kann (wenn die gesamte Funktionalität der Schleife in einer Zeile ausgedrückt werden kann) oder von mehreren in Klammern eingeschlossenen Code-Zeilen (Code-Block genannt):

```
for (...)
   Funktionalität in einer Zeile;
```

oder

```
for (...)
{
   Funktionalität in mehreren Zeilen;
}
```

Die Schleife selbst wird in der Klammer definiert, die auf das for-Statement folgt. Sie ist in drei durch Semikola voneinander getrennte Statements unterteilt. Das erste Statement ist die Initialisierung der Schleife, hier wird der Schleifenzähler auf seinen Anfangswert gesetzt. Das zweite Statement ist die Bedingung für die Fortführung der Schleife. Das dritte Statement gibt an, wie der Zähler bei jedem Schleifendurchgang erhöht wird. Jeder dieser drei Teile kann auch entfallen, beispielsweise wenn die Initialisierung an anderer Stelle im

Programm erfolgt. Wenn Sie beispielsweise diese Schleife definierten:

```
for (i = 0; i < 10; i++)
```

wird sich das wie folgt lesen:

1. Beginne mit i gleich 0.
2. Wiederhole die Schleife solange i kleiner als 10 ist.
3. Nach Abarbeitung der Befehle in der Schleife erhöhe i um 1.

Was Ihnen auch neu sein könnte, ist die Inkrementierung des Schleifenzählers. In den Programmiersprachen C und C++ gibt es zwei Operatoren, um einen Wert um 1 zu inkrementieren (erhöhen) oder zu dekrementieren (verringern). Um zu inkrementieren, verwenden Sie zwei Pluszeichen (++) entweder vor oder hinter der Variablen (es ist wichtig, an welche Stelle Sie den Operator setzen, aber dazu kommen wir im nächsten Absatz). Zum Dekrementieren verwenden Sie zwei Minuszeichen (--).

Ob Sie den Operator vor oder hinter der Variablen platzieren, hängt davon ab, wie Sie die Variable an dieser Stelle verwenden. Befindet sich der Operator vor der Variable, wird die Variable vor ihrer Verwendung inkrementiert (oder dekrementiert). Befindet sich der Operator hinter der Variablen, wird sie nach ihrer Verwendung inkrementiert (oder dekrementiert). Um zu sehen, wie das funktioniert, lassen Sie uns dieses Code-Beispiel betrachten:

```
int a, b, c, d;
a = 1;
b = ++a;
c = 1;
d = c++;
```

Im obigen Code wird die Variable a erhöht, bevor ihr Wert der Variablen b zugewiesen wird. Der Wert von b ist danach 2. Dagegen wird Variable c erst erhöht, nachdem ihr Wert der Variablen d zugewiesen wurde, sodass d den Wert 1 hat.

Ein weiterer Punkt, der Ihre Neugier geweckt haben könnte, ist der Zähler, der in beiden Schleifen bei 0 beginnt. Das ist eine akzeptierte Praxis in der C++-Programmierung im Gegensatz zu Pascal oder Basic. In den Programmiersprachen C und C++ fängt alles bei 0 an zu zählen, nicht bei 1. Die erste Position in einem Array ist Position 0, nicht Position 1.

Das andere Flusskontrolle-Konstrukt, das Sie verwendet haben, ist das if-Konstrukt. Die Syntax dafür ist das Schlüsselwort if, gefolgt von der in Klammern eingeschlossenen Bedingung. Wird mit dieser Bedingung nur eine Code-Zeile ausgeführt, kann sie direkt auf das if-Statement folgen. Sind es mehrere Code-Zeilen, müssen diese in Klammern eingeschlossen werden:

```
if (eine Bedingung)
  Funktionalität in einer Zeile;
else
  Funktionalität in einer Zeile;
```

oder

```
if (eine Bedingung)
{
  Funktionalität in mehreren Zeilen;
}
```

Nach dem Befehl können mittels des Schlüsselwortes else ein oder mehrere Befehle angegeben werden, die ausgeführt werden, wenn die Bedingung nicht zutrifft.

```
if (eine Bedingung)
  Funktionalität in einer Zeile, wenn die Bedingung zutrifft;
```

oder

```
if (eine Bedingung)
{
  Funktionalität in mehreren Zeilen;
  wenn die Bedingung zutrifft;
}
else
{
  Funktionalität in mehreren Zeilen;
  wenn die Bedingung nicht zutrifft;
}
```

Anders als bei anderen Programmiersprachen muss die Bedingung immer in Klammern gesetzt werden. Es gibt einige grundlegende Operatoren, die Sie an dieser Stelle häufig verwenden werden. Sie sind in Tabelle 2.9 aufgelistet. Generell kann als Bedingung jeder beliebige Ausdruck stehen. Ergibt der Ausdruck einen von 0 verschiedenen Wert, so gilt die Bedingung als erfüllt.

Bedingung	Beschreibung
==	Ist gleich (beachten Sie, dass zwei Gleichheitszeichen verwendet werden, nicht nur eines)
!=	Ungleich
<	Kleiner als
<=	Kleiner oder gleich
>	Größer als
>=	Größer oder gleich

Tabelle 2.9: Die grundlegenden Vergleichsoperatoren in C/C++

2.5 Fehlerbeseitigung in der Anwendung

An diesem Punkt sind Sie bereit, mit der Fehlerbeseitigung in Ihrer Anwendung zu beginnen. Sie werden Ihre Anwendung kompilieren und dann in der äußeren Schleife einen Haltepunkt platzieren. Danach starten Sie Ihre Anwendung und lösen die Schleifenberechnungen aus, sodass Sie Ihren Code schrittweise ausführen können. Dabei werden Sie die Fehler korrigieren, die Sie während des Debuggens im Code finden. Folgen Sie also diesen Schritten:

1. Wählen Sie Erstellen / Projektmappe erstellen, um Ihre Anwendung zu kompilieren.

2. Platzieren Sie den Texteingabecursor auf dem ersten for-Statement und drücken Sie (F9).
3. Wählen Sie Debuggen / Starten, um Ihre Anwendung zu starten.

Wenn Sie Debuggen / Starten ohne Debuggen wählen, starten Sie die Anwendung nicht im Debugger. Sie müssen die Anwendung schließen und noch einmal mit dem Befehl Starten aus dem Menü Debuggen starten.

4. Klicken Sie auf Prüfe Debugging im Hauptdialog Ihrer Anwendung. Ihre Anwendung sollte an dem Haltepunkt anhalten, den Sie im Code platziert haben.

Fehlerbeseitigung in den Schleifen

Auf Grund der Art und Weise, in der die beiden Zähler addiert werden, tritt das Problem in der Funktion CalculateSum beim ersten Durchgang durch die äußere Schleife nicht auf. Daher konzentrieren Sie sich zuerst auf das Problem in der Funktion OnClickedDebug.

Wenn Sie sich mit der Schaltfläche Prozedurschritt schrittweise durch den Code bewegen (Abbildung 2.22), stellen Sie fest, dass die if-Bedingung beim allerersten Schleifendurchgang ausgelöst wird (wenn Sie den Code genau wie in Listing 2.2 eingegeben haben). Darüber hinaus springt der Wert der Variablen iSum bei der Auswertung der Bedingung von 0 auf 6. Wenn Sie die if-Bedingung betrachten, sehen Sie Folgendes:

```
if (iSum = 6)
```

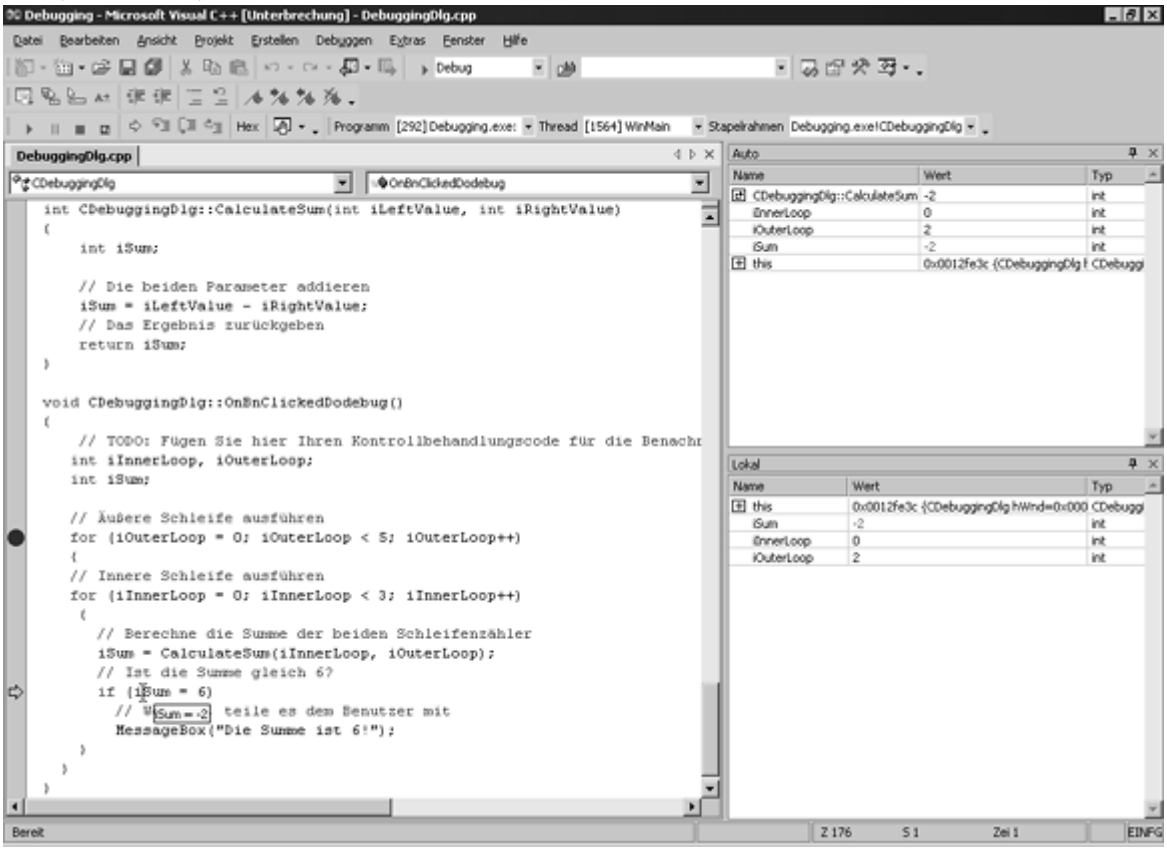


Abbildung 2.22: Durchlaufen Sie Ihre Anwendung mit der Funktion Prozedurschritt.

Das ist eines der einfachsten versehentlich zu programmierenden Probleme und dabei eines der am schwierigsten zu erkennenden, da der Code richtig »aussieht«. Wenn Sie jedoch die Bedingungen in C/C++ durchdenken, erinnern Sie sich daran, dass der Bedingungsoperator für »ist gleich« aus zwei Gleichheitszeichen besteht, nicht nur aus einem.

Da Sie hier nur ein Zeichen haben, weisen Sie der Variablen iSum den Wert 6 zu. Da der neue Wert der Variablen iSum 6 ist, wird die Bedingung als TRUE ausgewertet und die Meldung angezeigt.

Um diesen Fehler zu beheben, fügen Sie ein zweites Gleichheitszeichen in die Bedingung ein, ohne Leerzeichen zwischen den beiden Gleichheitszeichen. Die Zeile sollte dann so aussehen:

```
if (iSum == 6)
```

Wenn Sie in Ihrem Code weitergehen, sehen Sie, dass der Compiler von Visual C++ einen Moment braucht, um die Anwendung neu zu kompilieren und zu linken. Diese Fähigkeit gibt es bei Visual C++ erst seit der vorhergehenden Version. Davor musste man die Anwendung immer abbrechen, die Änderungen durchführen, neu kompilieren und die Debug-Sitzung neu starten. Jetzt können Sie einfach Ihre Korrekturen einfügen und weitermachen.

1. Ändern Sie das if-Statement, indem Sie ein zweites Gleichheitszeichen einfügen.

2. Springen Sie zur nächsten Code-Zeile. Beachten Sie, dass die Anwendung neu kompiliert wird, um

Ihre Änderungen zu aktivieren. Eventuell bekommen Sie an dieser Stelle eine Mitteilung von Visual C++ .NET, das Sie fragt, ob die Anwendung neu kompiliert werden soll.

Nicht jede Änderung kann automatisch neu kompiliert werden, ohne die Debug-Sitzung zu beenden. Bei manchen Änderungen muss man immer noch die Debug-Sitzung abbrechen um neu zu kompilieren, sodass man die Sitzung danach wieder von vorne starten muss.

C++-Exkurs

In den Programmiersprachen C und C++ ist das Boolesche FALSE immer 0, weil es in C/ C++ keinen integrierten Booleschen Datentyp gibt. Da 0 FALSE ist, wird jeder Wert ungleich 0 als TRUE ausgewertet. In C/C++ ist es üblich, einen Booleschen Datentyp zu definieren, in dem FALSE als 0 und TRUE als 1 definiert ist.

C++-Regel

Beim Datentyp BOOL ist 0 immer FALSE. Jeder andere Wert ist TRUE. Beim Datentyp bool werden true und false durch ein einzelnes Bit repräsentiert, wobei false 0 ist und true 1.

Fehlerbeseitigung in der Berechnung

Nachdem Sie noch einige Male die Funktion OnClickedDebug durchlaufen haben (zumindest so oft, bis Sie im zweiten Durchlauf der äußeren Schleife sind), sehen Sie, dass die Funktion CalculateSum seltsame Werte zurückgibt. Es sieht so aus, als würde diese Funktion den Zähler der äußeren Schleife von dem der inneren Schleife subtrahieren. Das kann nicht stimmen, sie sollten addiert werden! Sie müssen auf die Schaltfläche Einzelschritt wechseln, um in die Funktion CalculateSum zu springen und zu sehen, was in dieser Funktion vor sich geht (siehe Abbildung 2.23).

In der Funktion CalculateSum werden Sie feststellen, dass die wichtigste Code-Zeile so aussieht:

```
iSum = iLeftValue - iRightValue;
```

Es scheint, als ob diese Funktion tatsächlich die Zähler voneinander subtrahiert statt sie zu addieren. Sie korrigieren das, sodass die Zeile wie folgt aussieht:

```
iSum = iLeftValue + iRightValue;
```

Wenn Sie weitergehen, finden Sie keine weiteren Probleme, die korrigiert werden müssen, also beschließen Sie, die Funktion mit der Schaltfläche Ausführen bis Rücksprung zu verlassen.

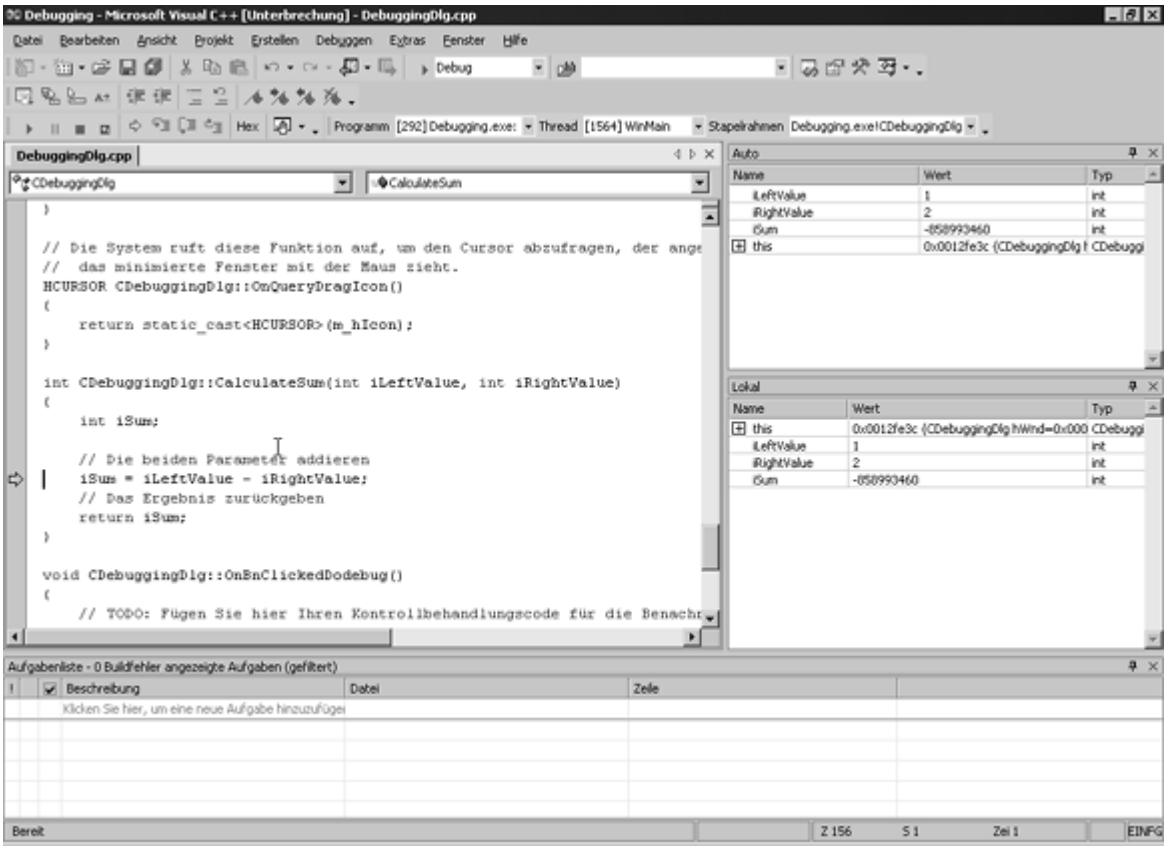


Abbildung 2.23: In die Funktion CalculateSum mit der Funktion Einzelschritt springen

2.6 Zusammenfassung

Heute haben Sie eine wertvolle Fähigkeit erlernt, die Sie bei allen Anwendungen verwenden werden, die Sie mit Visual C++ erstellen. Grob geschätzt werden Sie ein Drittel bis die Hälfte der Zeit der Entwicklung und Programmierung einer Anwendung für die Fehlerbeseitigung benötigen. Das ist eine durchschnittliche Schätzung. Die Anwendung braucht möglicherweise viel weniger oder viel mehr Zeit für die Fehlerbeseitigung. Das ist auch eine Fähigkeit, die sich durch Übung verbessert.

2.7 Workshop

Fragen und Antworten

Frage:
Warum muss ich ein Release-Build meiner Anwendung erstellen? Warum kann ich nicht einfach das Debug-Build ausliefern?

Antwort:
Es gibt zwei wichtige Gründe, warum Sie anstatt eines Debug- ein Release-Build Ihrer Anwendung erstellen und weitergeben. Erstens ist das Release-Build kleiner und läuft schneller. Bei kurzen Routinen ist es zwar schwierig, den Unterschied zu bemerken, doch wenn Ihre Anwendung längere Verarbeitungen ausführt, kann der Unterschied in der Geschwindigkeit eine merkliche Verbesserung bedeuten.

Antwort:
Der zweite Hauptgrund dafür, ein Release-Build auszuliefern, ist der Schutz Ihres »geistigen Eigentums«. Wenn Sie ein Debug-Build Ihrer Anwendung ausliefern, kann sie jeder in einem Debugger ausführen und sich seinen Weg durch die Anwendung suchen. Wenn Sie Ihren Code nicht mit Ihrer Anwendung weitergeben, sieht der Benutzer nur den Assembler-Code Ihrer Anwendung, doch für viele fortgeschrittene Programmierer reicht das aus, um den ursprünglichen Quellcode wieder herzustellen.

Es gibt spezialisierte Debugger, mit deren Hilfe man Ihr Release-Build durchlaufen und den Assembler-Code sehen kann. Diese spezialisierten Debugger werden jedoch nicht mit Entwicklungswerkzeugen für C++ mitgeliefert, sondern müssen separat gekauft werden.

Frage:
Wie funktioniert der Debugger? Woher weiß er, wo sich in meinem Code die Ausführung der Anwendung befindet?

Antwort:
In einem Debug-Build fügt der Compiler zusätzliche Informationen ein, die nicht ausgeführt werden, wenn die Anwendung läuft. Diese Informationen sind mit dem ausführbaren Code vermischt. Sie geben an, welche Quellcode-Datei und welche Code-Zeile welchem Satz von Maschinenbefehlen entspricht. Es sind auch Informationen über jede Variable im Code enthalten und über deren Speicherort, während die Anwendung läuft. Diese Informationen verwendet der Debugger, um den richtigen Code anzuzeigen, während Sie Ihre Anwendung durchlaufen.

Frage:
Ich möchte nicht, dass mein Haltepunkt jedes Mal ausgelöst wird, wenn meine Anwendung ihn erreicht. Kann man erreichen, dass er nur ausgelöst wird, wenn bestimmte Werte erreicht sind?

Antwort:
Wenn Sie die Haltepunkte-Ansicht öffnen, können Sie die Eigenschaften jedes Ihrer Haltepunkte sehen. Im Eigenschaftendialog für Haltepunkte können Sie Bedingungen an den Haltepunkt vergeben, sodass beispielsweise nur angehalten wird, wenn eine bestimmte Variable einen bestimmten Wert hat.

Quiz

1. Welche drei wichtigsten Navigationsbefehle werden beim Debuggen verwendet?

2. Was ist der Unterschied zwischen den Makros ASSERT und VERIFY?
3. Warum sollte man das Werkzeug Spy++ verwenden?

Übung

Fügen Sie ASSERT-Makros ein, um die an die Funktion CalculateSum übergebenen Parameter zu überprüfen. Sie müssen sicherstellen, dass die übergebenen Parameter größer oder gleich 0 sind. Fügen Sie außerdem ein ASSERT-Makro ein, um sicherzustellen, dass der Wert von iSum vor seiner Rückgabe nicht negativ ist.

Kapitel Inhalt Index SAMS Top Kapitel

Tag 3

Steuerelemente

In nahezu jeder Windows-Anwendung begegnet man Schaltflächen, Kontrollkästchen, Textfeldern und DropDown-Listenfeldern. Man bezeichnet diese Komponenten als Steuerelemente und viele dieser Steuerelemente sind in das Betriebssystem selbst integriert. In Visual C++ lassen sich diese allgemeinen Steuerelemente einsetzen, indem man sie einfach per Drag&Drop in einem Dialogfeld platziert.

Am heutigen Tag lernen Sie, ...

- welche grundlegenden Steuerelemente in Visual C++ zur Verfügung stehen,
- wie man Variablen deklariert und sie mit einem Steuerelement verbindet,
- wie man die Werte von Steuerelementen und Variablen synchronisiert,
- wie man die Reihenfolge festlegt, in der die Benutzer durch die Steuerelemente im Anwendungsfenster navigieren,
- wie man Aktionen mit Steuerelementen auslöst,
- wie man (bei laufender Anwendung) das Aussehen von Steuerelementen manipuliert und verändert.

3.1 Standardsteuerelemente von Windows

Zum Betriebssystem Windows gehören verschiedene Standardsteuerelemente wie Schieberegler, Strukturansicht, Listenelement oder Statusanzeige. In der heutigen Lektion geht es zunächst einmal um ein halbes Dutzend Steuerelemente, die sich in fast jeder Windows-Anwendung finden:

- Statisches Textfeld (Static Text)
- Eingabefeld für Text (Edit Control)
- Schaltfläche (Button)
- Kontrollkästchen (Check Box)
- Optionsfeld (Radio Button)
- Kombinationsfeld, auch als DropDown-Listenfeld bezeichnet (Combo Box, Drop Down List)

Diese und andere Steuerelemente stehen zur sofortigen Nutzung in Visual C++- Anwendungen bereit. Sie sind in der Toolbox-Ansicht im Dialogeditor von Developer Studio untergebracht (siehe Abbildung 3.1).

Abbildung 3.1: Die in der Toolbox-Ansicht verfügbaren Standardsteuerelemente

Statischer Text

Das Steuerelement Text verwendet man, um Text für den Benutzer anzuzeigen. Der Benutzer kann diesen Text weder ändern noch in anderer Form mit dem Steuerelement interagieren. Das Steuerelement ist als Nur-Lesen-Element vorgesehen. Allerdings kann man im Code der laufenden Anwendung den durch das Steuerelement angezeigten Text ohne weiteres ändern.

Die wichtigsten Eigenschaften des Steuerelements Text sind in Tabelle 3.1 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Der Standardwert ist immer IDC_STATIC. Wenn Sie das Steuerelement Text kontrollieren wollen, um sein Erscheinungsbild oder seinen Text während der Laufzeit zu ändern, müssen Sie die Eigenschaft ID ändern.
Beschriftung	Legt den im Steuerelement Text angezeigten Text fest
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, während die Anwendung läuft

Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Wenn das Steuerelement deaktiviert ist, erscheint es grau hinterlegt. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Tabstopp	Gibt an, ob Benutzer das Steuerelement durch Navigieren mit der Tabulatortaste erreichen können. Man könnte annehmen, dass diese Eigenschaft beim Steuerelement Text nichts zu suchen hat. Wenn Sie dem Beschriftungstext allerdings Kurztasten zuweisen, wird diese Eigenschaft äußerst wichtig, da der Fokus dem nächsten Steuerelement nach dem Text zugewiesen wird.

Tabelle 3.1: Wichtige Eigenschaften des Steuerelements Text

MFC-Exkurs: Die Klasse CStatic

Für jedes Steuerelement, das in einer Windows-Anwendung benutzt werden kann, existiert eine entsprechende MFC-Klasse. Diese Klasse verkapselt die gesamte Steuerfunktionalität und bietet damit eine einfache Möglichkeit, mit dem Steuerelement zu interagieren. Das Steuerelement Text ist in der Klasse CStatic verkapselt.

Die Klasse CStatic ist ein Abkömmling der Klasse CWnd, der Basis-Klasse für alle visuellen Komponenten in einer MFC-Anwendung. Die Klasse CStatic wird gewöhnlich für die Anzeige von Text verwendet, mit dem der Benutzer nicht interagieren kann.

Eingabefelder

In einem Eingabefeld kann der (spätere) Anwender Text eingeben oder ändern. Das Steuerelement gehört zu den Hauptwerkzeugen, die dem Benutzer die Möglichkeit bieten, einer Anwendung bestimmte erforderliche Informationen bereitzustellen. Es ist in der Lage, beliebigen Text bis zu einer bestimmten Länge aufzunehmen, der sich auslesen und nach Bedarf weiterverarbeiten lässt. Das Eingabefeld akzeptiert ausschließlich reinen Text, Formatierungen stehen dem Benutzer nicht zur Verfügung. Wichtige Eigenschaften von Eingabefeldern sind in Tabelle 3.2 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Eingabefeld identifizieren und mit ihm arbeiten zu können.
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist
Text ausrichten (Align Text)	Gibt an, ob der Text im Steuerelement am rechten oder linken Rand ausgerichtet oder zentriert ist
Mehrfachzeile (Multiline)	Gibt an, ob sich der Text im Eingabefeld über mehrere Zeilen erstrecken kann
Zahl	Beschränkt den Text, den ein Eingabefeld annimmt, auf Zahlen
Kennwort	Verbirgt den Text im Eingabefeld und zeigt stattdessen Sterne ('*')
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.2: Wichtige Eigenschaften von Eingabefeldern

MFC-Hinweis: Die Klasse CEdit

Das Steuerelement Eingabefeld ist in der Klasse CEdit verkapselt. Die Klasse CEdit ist ein Abkömmling der Klasse CWnd und erweitert diese durch verschiedene Methoden für die Bearbeitung von und die Interaktion

mit Text im Steuerelement. Die Klasse CEdit enthält sogar Methoden für die Interaktion mit der Zwischenablage, darunter Copy (Kopieren), Cut (Ausschneiden), Paste (Einfügen) und Undo (Rückgängig), also die Methoden, die keine Parameter übernehmen.

Schaltflächen

Über eine Schaltfläche löst der Benutzer eine bestimmte Aktion aus. Die Beschriftung oder Caption - der Titel - der Schaltfläche sollte einen Hinweis auf die Aktion liefern, die beim Klicken auf die Schaltfläche ausgeführt wird. Eine Schaltfläche kann auch Bilder enthalten, die man - allein oder zusammen mit einer Textbeschreibung - benutzt, um den Zweck der Schaltfläche zu vermitteln. Wichtige Eigenschaften des Steuerelements Schaltfläche sind in Tabelle 3.3 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um die Schaltfläche identifizieren und mit ihr interagieren zu können.
Beschriftung	Gibt den auf der Schaltfläche angezeigten Text an
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Text auf der Schaltfläche erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Standardschaltfläche (Default Button)	Gibt an, dass dieses Steuerelement ausgelöst werden soll, wenn der Benutzer die Eingabetaste betätigt
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.3: Wichtige Eigenschaften des Steuerelements Schaltfläche

MFC-Hinweis: Die Klasse CButton

Die Klasse CButton ist die MFC-Klasse, die das Steuerelement Schaltfläche verkapselt. Diese Klasse ist ein Abkömmling der Klasse CWnd und verkapselt nicht nur die Schaltfläche, sondern auch das Kontrollkästchen und das Optionsfeld. Sie wird von der Klasse CBitmapButton ererbt, mit der Sie ein Bild auf der Schaltfläche darstellen können.

Sie können den Zustand einer Schaltfläche mit den Methoden GetState und SetState abfragen und ändern. Sie können das Erscheinungsbild und die Verhaltensweise der Schaltfläche mit den Methoden GetButtonStyle und SetButtonStyle kontrollieren.

Kontrollkästchen

Als Kontrollkästchen bezeichnet man die kleinen quadratischen Elemente, die der Benutzer durch Anklicken ein- bzw. ausschaltet und damit einen bestimmten Wert setzt bzw. zurücksetzt. Grundsätzlich handelt es sich um An-/Ausschalter, gelegentlich mit einem dritten Zwischenzustand. In Tabelle 3.4 sind wichtige Eigenschaften des Steuerelements Kontrollkästchen aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Kontrollkästchen identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.

Drei-Status (Tri-state)	Gibt an, dass das Kontrollkästchen drei Zustände anstatt der gewöhnlichen zwei annehmen kann. Der dritte Zustand ist Deaktiviert, das heißt der Wert des Steuerelements ist weder TRUE noch FALSE.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.4: Wichtige Eigenschaften des Steuerelements Kontrollkästchen

Optionsfelder

Ein Optionsfeld wird als Kreis dargestellt. Klickt der Benutzer ein Optionsfeld an, erscheint ein Punkt im Kreis. Das Optionsfeld ähnelt dem Kontrollkästchen, wird aber in einer Gruppe verwendet, in der nur ein Optionsfeld eingeschaltet sein kann. Normalerweise setzt man Optionsfelder in Gruppen mit mindestens zwei Optionen ein, wobei die Optionsfelder von einem Gruppenfeld umgeben sind. Das Gruppenfeld gewährleistet die visuelle Unabhängigkeit der einzelnen Optionsfeldgruppen und weist den Benutzer darauf hin, dass in jeder Gruppe jeweils nur ein Optionsfeld eingeschaltet sein kann. In Tabelle 3.5 sind wichtige Eigenschaften der Optionsfelder aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Optionsfeld identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird
Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Gruppe (Group)	Gibt an, dass ein Steuerelement das erste in einer Gruppe von Steuerelementen ist. Diese Eigenschaft ist für die meisten Steuerelemente verfügbar, für Optionsfelder ist sie jedoch besonders wichtig. Diese Eigenschaft sollte nur beim ersten Optionsfeld in einer Gruppe von Optionsfeldern auf TRUE gesetzt werden.
Auto	Veranlasst das Optionsfeld, automatisch seinen Status zu ändern, wenn es angeklickt wird. Standardmäßig ist diese Eigenschaft auf TRUE gesetzt.
LeftText	Veranlasst die Beschriftung, links vom Optionsfeld zu erscheinen. Normalerweise befindet sie sich rechts davon.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.5: Wichtige Eigenschaften des Steuerelements Optionsfeld

Kombinationsfelder

Ein Kombinationsfeld (oder DropDown-Listenfeld) besteht aus einem Eingabefeld, dem eine Liste mit verfügbaren Werten zugeordnet ist. Mit einem Kombinationsfeld kann man eine Liste von Auswahlfeldern bereitstellen, wobei der Benutzer einen Wert aus der Liste auswählen kann. Manchmal hat der Benutzer die Möglichkeit, selbst einen Wert einzutippen, wenn die Liste keinen passenden Wert enthält. Wichtige Eigenschaften für Kombinationsfelder sind in Tabelle 3.6 aufgelistet.

Eigenschaft	Beschreibung
ID	Identifiziert das Steuerelement. Sie müssen diese Eigenschaft ändern, um das Kombinationsfeld identifizieren und mit ihm interagieren zu können.
Beschriftung	Gibt den Text an, der im Steuerelement angezeigt wird

Sichtbar (Visible)	Gibt an, ob das Steuerelement sichtbar ist, wenn die Anwendung läuft
Deaktiviert (Disabled)	Gibt an, dass das Steuerelement deaktiviert ist. Der Beschriftungstext erscheint als Umriss oder Gravierung in der Oberfläche des Dialogfensters.
Sortieren (Sortierenieren)	Kontrolliert, ob die Einträge in der DropDown-Liste sortiert sind
Typ (Type)	Gibt an, welche Art Kombinationsfeld anzuzeigen ist: »Einfach«, »DropDown« (der Standard) oder »DropDown-Listenfeld«. »Einfach« zeigt die Liste immer an. im »DropDown«-Stil wird die Liste nur angezeigt, wenn das Steuerelement den Fokus hat, entweder weil der Benutzer es mit der Tabulatortaste angesprungen oder weil er auf den Pfeil geklickt hat, um die Liste zu öffnen. Bei »DropDown-Listenfeld« wird die aktuelle Auswahl in einem Text-Steuerelement angezeigt; ansonsten entspricht dieser Typ dem Typ »DropDown«.
Tabstopp	Gibt an, ob Benutzer das Steuerelement beim Navigieren mit der Tabulatortaste erreichen können

Tabelle 3.6: Wichtige Eigenschaften von Kombinationsfeldern

MFC-Hinweis: Die Klasse CComboBox

Die Klasse CComboBox verkapselt das Steuerelement Kombinationsfeld. Sie ist ein Abkömmling der Klasse CWnd und wird von der Klasse CComboBoxEx ererbt. Die Klasse CComboBoxEx stellt Funktionalität bereit, um Bilder in die Auswahl der DropDown-Liste aufzunehmen.

Die Klasse CComboBox stellt Funktionalität für die Interaktion mit der DropDown-Liste bereit. Sie bietet Funktionen, um die aktuelle Auswahl mit den Methoden GetCurSel und SetCurSel sowie SelectString und FindString abzufragen und zu setzen. Sie ermöglicht Ihnen auch, mit den Methoden AddString, InsertString und DeleteString Elemente in die Liste einzufügen und daraus zu entfernen. Wenn Sie den Inhalt der Liste zurücksetzen und neu beginnen müssen, können Sie die Methode ResetContent verwenden.

3.2 Steuerelemente in ein Fenster aufnehmen

Die heute zu erstellende Anwendung soll eine Reihe von Steuerelementen in einem Dialogfeld enthalten, wie es aus Abbildung 3.2 hervorgeht. Diese Steuerelemente haben verschiedene Aufgaben.

- Am oberen Rand des Fensters befindet sich ein Eingabefeld, in das der Benutzer eine Nachricht eintippen kann, die in einem Meldungsfeld erscheint, wenn er auf die Schaltfläche neben dem Feld klickt.
- Unterhalb dieses Eingabefelds sind zwei Schaltflächen angeordnet, die entweder das Eingabefeld mit einer Standardmeldung füllen oder einen vorhandenen Eintrag löschen.
- Unter diesen Schaltflächen befindet sich ein Kombinationsfeld mit einer Liste von Windows-Standardanwendungen. Wählt der Benutzer eines dieser Programme aus und klickt auf die Schaltfläche neben der DropDown-Liste, startet das ausgewählte Programm.
- Unterhalb des Kombinationsfelds sind zwei Gruppen von Kontrollkästchen angeordnet. Diese wirken auf die Steuerelemente im oberen Teil des Dialogfelds: die Steuerelemente für die Anzeige einer Benutzermeldung und die Steuerelemente für die Ausführung eines anderen Programms.
- Die Kontrollkästchen auf der linken Seite aktivieren und deaktivieren die einzelnen Gruppen von Steuerelementen.
- Mit den rechten Kontrollkästchen lassen sich die Gruppen der Steuerelemente anzeigen und ausblenden.
- Im unteren Teil des Dialogfelds befindet sich eine Schaltfläche, die das Schließen der Anwendung bewirkt.

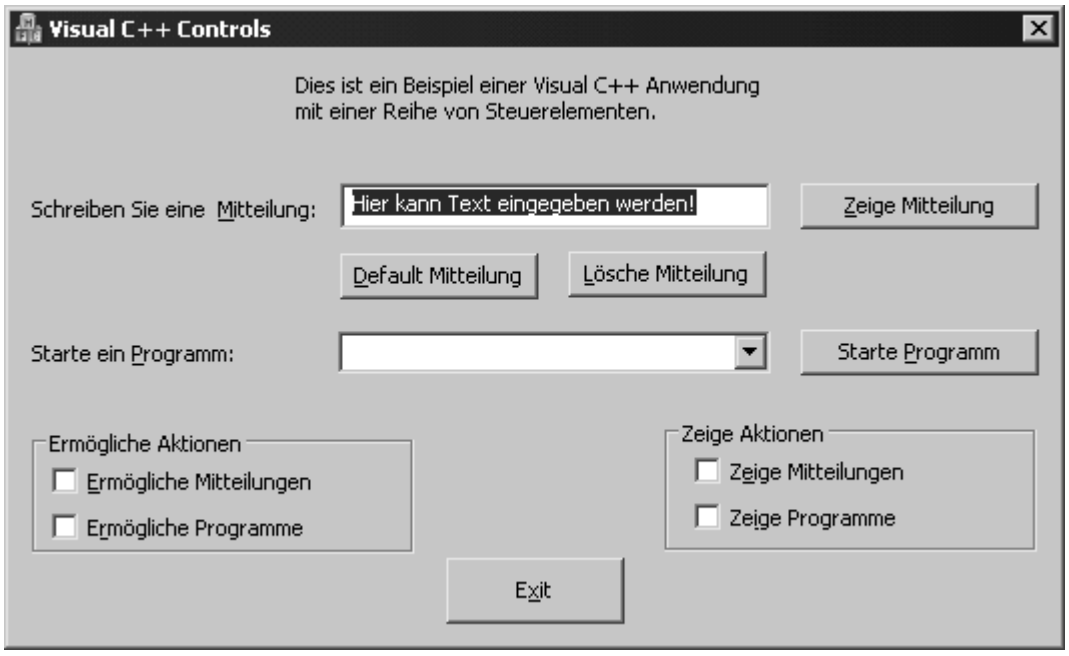


Abbildung 3.2: Die heutige Anwendung verwendet mehrere Standardsteuerelemente.

Anwendungsgerüst und Layout des Dialogfelds erstellen

Mit den gestern erworbenen Kenntnissen können Sie jetzt ein neues Anwendungsgerüst erstellen und das Layout des Anwendungsdialogfelds entwerfen. Führen Sie dazu die folgenden Schritte aus:

1. Erstellen Sie ein neues MFC-Application-Visual C++-Projekt und nennen Sie es Controls.

2. Verwenden Sie im MFC Anwendungs-Assistent die gleichen Einstellungen wie an den letzten beiden Tagen. Legen Sie den Titel des Dialogfelds mit Visual C++ Controls fest.
3. Nachdem Sie das Anwendungsgerüst erstellt haben, gestalten Sie das Hauptdialogfeld wie in der weiter oben gezeigten Abbildung 3.2.
4. Konfigurieren Sie die Eigenschaften der Steuerelemente gemäß Tabelle 3.7.

Steuerelement	Eigenschaft	Einstellung
Text	Beschriftung	Dies ist ein Beispiel einer Visual C++ Anwendung mit einer Reihe von Steuerelementen.
Text	ID Beschriftung	IDC_STATICMSG Schreiben Sie eine &Mitteilung:
Text	ID Beschriftung	IDC_STATICPGM Starte ein &Programm:
Eingabefeld	ID	IDC_MSG
Schaltfläche	ID Beschriftung	IDC_SHWMSG &Zeige Mitteilung
Schaltfläche	ID Beschriftung	IDC_DFLTMSG &Default Mitteilung
Schaltfläche	ID Beschriftung	IDC_CLRMSG &Lösche Mitteilung
Schaltfläche	ID Beschriftung	IDC_RUNPGM Starte &Programm
Schaltfläche	ID Beschriftung	IDC_EXIT E&xit
Kombinationsfeld	ID	IDC_PROGTORUN

Gruppenfeld	Beschriftung	Ermögliche Aktionen
Gruppenfeld	Beschriftung	Zeige Aktionen
Kontrollkästchen	ID Beschriftung	IDC_CKENBLMSG &Ermögliche Mitteilungen
Kontrollkästchen	ID Beschriftung	IDC_CKENBLPGM E&rmögliche Programme
Kontrollkästchen	ID Beschriftung	IDC_CKSHWMSG Z&eige Mitteilungen
Kontrollkästchen	ID Beschriftung	IDC_CKSHWPGM Ze&ige Programme

Tabelle 3.7: Eigenschaftseinstellungen für die Steuerelemente im Anwendungsdialogfeld

5. Nachdem Sie alle genannten Steuerelemente im Dialogfeld platziert und deren Eigenschaften konfiguriert haben, markieren Sie das Kombinationsfeld noch einmal und suchen Sie im Eigenschaftenbereich die Eigenschaft Daten. Geben Sie die folgenden Werte getrennt durch Semikola ein (siehe Abbildung 3.3).

- Notepad
- Paint
- Solitaire

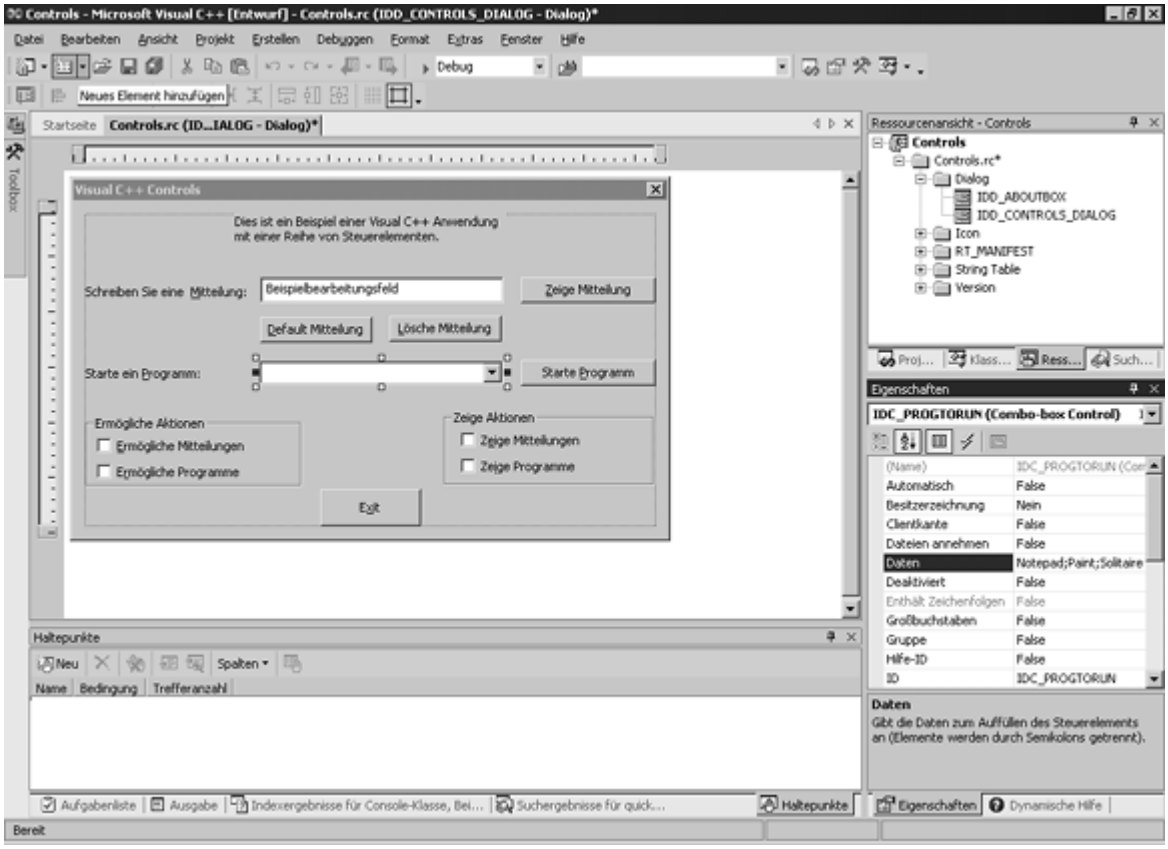


Abbildung 3.3: Die Einträge in der DropDown-Liste des Kombinationsfelds legen Sie in der Eigenschaftenansicht fest.

Wenn man ein Kombinationsfeld in das Fenster aufnimmt, ist darauf zu achten, dass man klickt und den Bereich für das Steuerelement so groß zieht, wie die DropDown-Liste sein soll. Nachdem man das Steuerelement im Fenster gezeichnet hat, kann man die Größe genauso ändern, wie es normalerweise üblich ist. Um die Ausdehnung der Liste nach unten festzulegen, klickt man auf den Pfeil - genauso, als würde man bei laufender Anwendung die DropDown-Liste aktivieren.

Die Tabulator-Reihenfolge von Steuerelementen festlegen

Nach der Anordnung der Steuerelemente im Fenster müssen Sie noch sicherstellen, dass der Benutzer bei der Navigation mithilfe der (ÿ)-Taste die Steuerelemente in der von Ihnen gewünschten Reihenfolge anspricht. Die Tabulator-Reihenfolge legen Sie mit den nachstehenden Schritten fest:

1. Markieren Sie im Bearbeitungsbereich von Visual Studio entweder das Dialogfeld oder eines der Steuerelemente im Fenster.

2. Wählen Sie Format / Tabulator-Reihenfolge. Daraufhin erscheinen im Fenster neben den Steuerelementen Nummern. Diese kennzeichnen die Reihenfolge, in der die Navigation durch das Dialogfeld verläuft (siehe Abbildung 3.4).

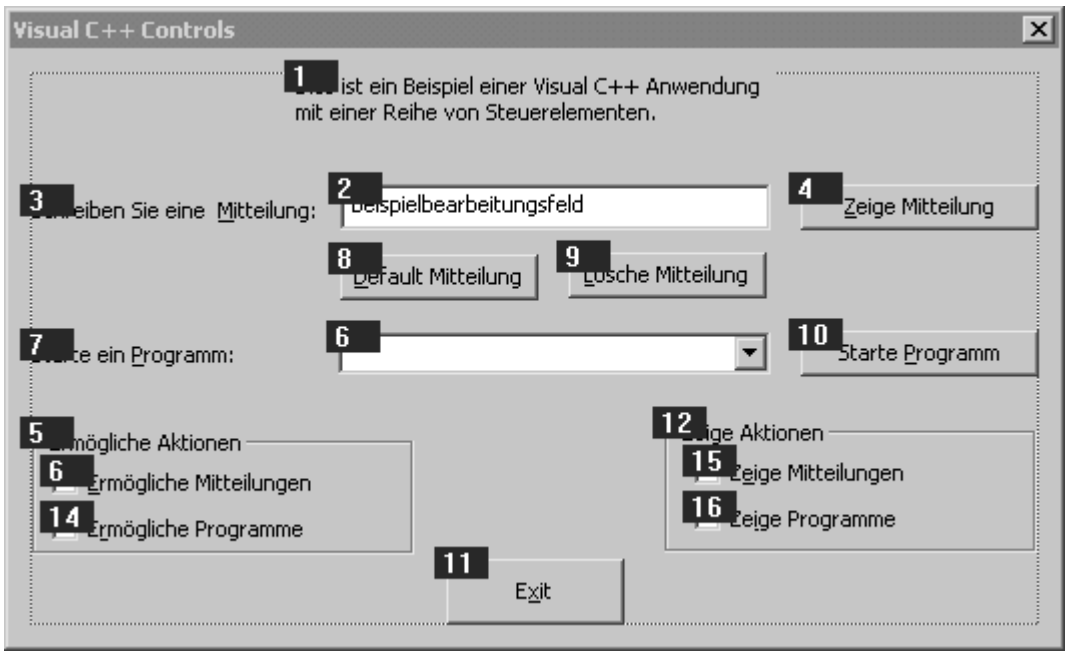


Abbildung 3.4: Das Einschalten der Tabulator-Reihenfolge zeigt die Reihenfolge der Navigation durch das Dialogfeld an.

Klicken Sie mit der Maus die Nummernfelder in der Reihenfolge an, in der der Benutzer durch das Dialogfeld navigieren soll. Die Steuerelemente nummerieren sich automatisch neu, wenn Sie sie nacheinander auswählen.

3. Nachdem Sie die Tabulator-Reihenfolge festgelegt haben, wählen Sie Format / Tabulator-Reihenfolge erneut, um zum Dialog-Editor zurückzukehren.

Statischer Text, der mit einer Zugriffstaste versehen ist, sollte in der Tabulator-Reihenfolge unmittelbar vor dem zugehörigen Steuerelement stehen. Da der Benutzer nicht mit statischem Text interagieren kann, geht der Fokus bei Wahl der Zugriffstaste direkt zum nächsten Steuerelement laut Tabulator-Reihenfolge.

Eine Zugriffstaste ist durch das unterstrichene Zeichen - den so genannten mnemonischen Code - in der Beschriftung einer Schaltfläche, eines Kontrollkästchens, eines Menüs oder eines anderen Steuerelements gekennzeichnet. Der Benutzer kann den unterstrichenen Buchstaben in Kombination mit der (Alt)-Taste drücken, um direkt zu diesem Steuerelement zu gelangen oder das angeklickte Ereignis auf dem Steuerelement auszulösen. Um eine Zugriffstaste festzulegen, schreibt man bei Eingabe des Titels ein kaufmännisches Und-Zeichen (&) unmittelbar vor das betreffende Zeichen. Achten Sie darauf, dass Sie für mehrere Zugriffstasten nicht dieselben Zeichen in demselben Fenster oder derselben Menügruppe vorsehen, da es den Benutzer nur verwirrt, wenn er eine Zugriffstaste drückt und nicht die erwartete Reaktion eintritt.

Bevor Sie sich mit dem Code der Anwendung beschäftigen, sollten Sie abschließend die Zugriffstasten auf Konflikte prüfen. Führen Sie dazu die folgenden Schritte aus:

1. Markieren Sie im Dialog-Editor das Dialogfeld oder eines der Steuerelemente. Klicken Sie mit der rechten Maustaste und wählen Sie Mnemonik überprüfen.

2. Wenn keine Konflikte bei Ihren mnemonischen Codes aufgetreten sind, zeigt Visual C++ ein entsprechendes Meldungsfeld an (siehe Abbildung 3.5).

Abbildung 3.5: Eine Prüfung der Zugriffstasten zeigt, ob Konflikte vorliegen.

3. Falls Konflikte vorhanden sind, meldet das Dialogfeld den betreffenden Buchstaben und gibt Ihnen die Möglichkeit, die Steuerelemente mit den gegensätzlichen Einträgen auswählen zu lassen (siehe Abbildung 3.6).

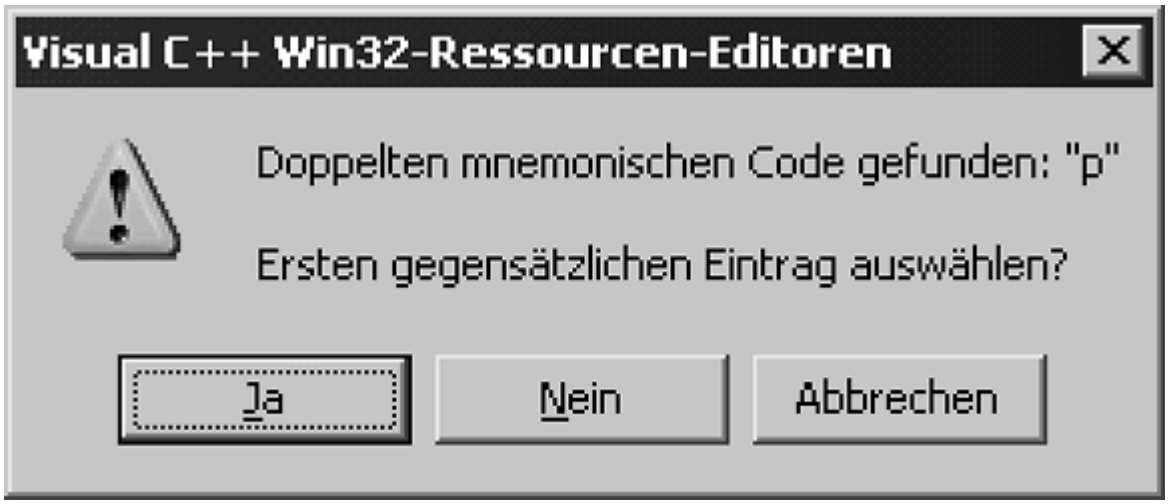


Abbildung 3.6: Doppelte mnemonische Codes lassen sich automatisch auswählen.

4. In den Beschreibungen von Tabelle 3.7 waren zwei doppelte mnemonische Codes angegeben, wählen Sie passenden andere Buchstabenkombinationen.

3.3 Variablen mit Steuerelementen verbinden

Wenn Sie bereits mit Visual Basic oder PowerBuilder programmiert haben, ahnen Sie sicherlich schon, dass es nun an der Zeit ist, etwas Code zu verfassen. Bei Visual C++- Anwendungen, die mit MFC erstellt werden, läuft dieser Prozess allerdings nicht genauso ab. Bevor Sie mit der Codierung beginnen können, müssen Sie allen Steuerelementen, denen ein Wert zugewiesen ist, Variablen zuordnen - allen, außer dem statischem Text und den Schaltflächen. Auf die Variablen greifen Sie zurück, wenn Sie den Code für die Anwendung schreiben. Die Werte, die der Benutzer in die Bildschirmsteuerelemente eingibt, übernimmt das Programm zur Weiterverarbeitung in diese Variablen. Analog dazu werden alle Werte, die der Code Ihrer Anwendung in diese Variablen stellt, in den Steuerelementen des Fensters aktualisiert, damit sie der Benutzer sieht.

Wie deklariert man nun diese Variablen und verbindet sie mit den Steuerelementen, die man im Fenster platziert hat? Führen Sie die folgenden Schritte aus:

1. Wählen Sie das Steuerelement aus, das Sie mit einer Variablen verbinden möchten.

2. Rechtsklicken Sie mit der Maus auf dem Steuerelement und wählen Sie aus dem erscheinenden Kontext-Menü Variable hinzufügen.
3. Wählen Sie die ID von einem der Steuerelemente, mit dem Sie eine Variable verbinden möchten, wie beispielsweise IDC_MSG. Die ID des gewählten Steuerelements sollte im Kombinationsfeld Steuerelement-ID bereits hervorgehoben sein.
4. Geben Sie Value (Wert) als Kategorie an.
5. Wählen Sie aus dem Kombinationsfeld Variablentyp den Datentyp der Variablen aus, hier wird bereits vom Assistenten ein sinnvoller Vorschlag gemacht.
6. Geben Sie im Eingabefeld Variablenname einen Namen ein.
7. Wenn Sie möchten geben Sie im Eingabefeld Kommentar am unteren Rand des Dialogs (siehe Abbildung 3.7) einen Kommentar ein, der die Variable und ihre Verwendung beschreibt. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.

Wiederholen Sie die Schritte 1-7 für alle anderen Steuerelemente, für die Sie Variablen zuordnen müssen. In der heutigen Anwendung betrifft das die Variablen gemäß Tabelle 3.8.

Steuerelement	Variablenname	Kategorie	Typ	Zugriff
IDC_MSG	m_strMessage	Value (Wert)	CString	public
IDC_PROGTORUN	m_strProgToRun	Value	CString	public
IDC_CKENBLMSG	m_bEnableMsg	Value	BOOL	public
IDC_CKENBLPGM	m_bEnablePgm	Value	BOOL	public
IDC_CKSHWMSG	m_bShowMsg	Value	BOOL	public
IDC_CKSHWPGM	m_bShowPgm	Value	BOOL	public

Tabelle 3.8: Variablen für die Steuerelemente der Anwendung

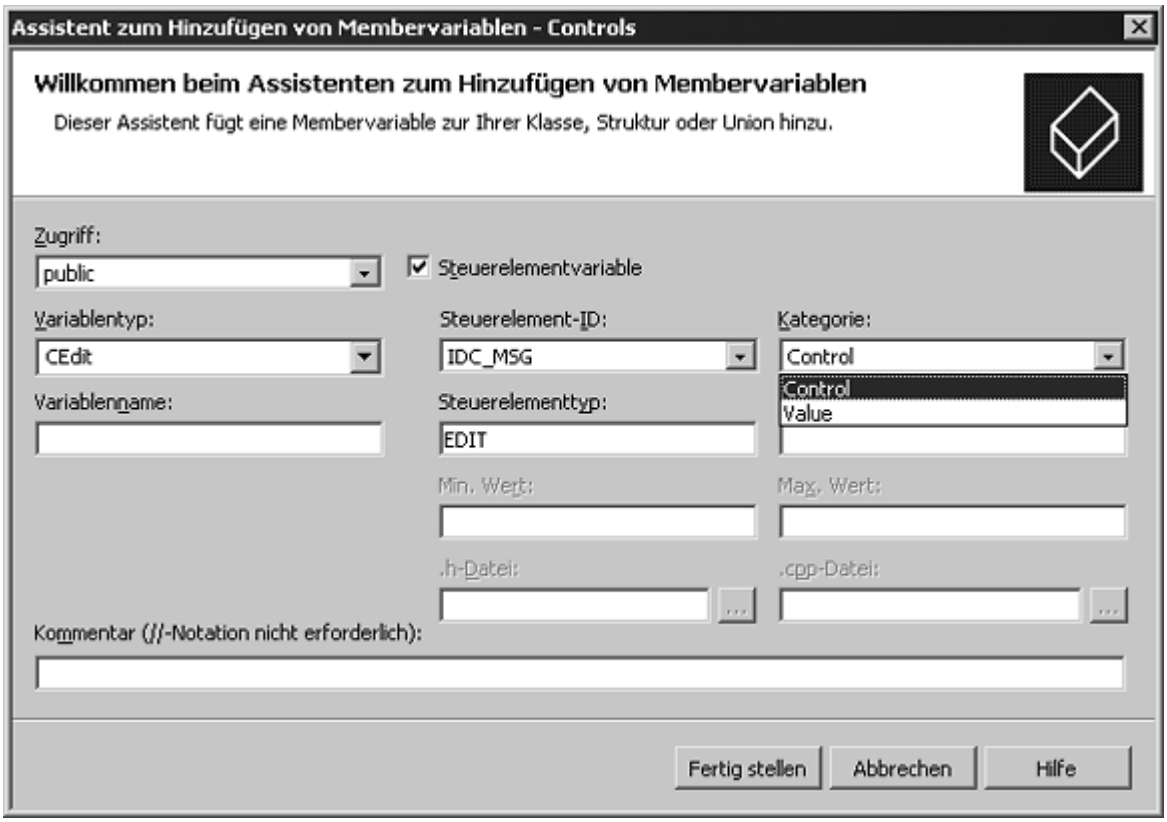


Abbildung 3.7: Einem Steuerelement eine Variable hinzufügen

Wenn Sie einen Datentyp benötigen, der sich im Kombinationsfeld mit den Datentypen nicht findet, können Sie im Eingabebereich des Kombinationsfelds Ihren eigenen Datentyp angeben.

Alle hier verwendeten Variablen beginnen mit dem Präfix m_, da es sich um Member-Variablen (oder Elementvariablen) einer Klasse handelt. Diese Konvention hat sich bei der Vergabe von Namen für Elemente von MFC-Klassen eingebürgert. Nach dem m_ verwendet man eine Variante der Ungarischen Notation, bei der die nächsten Buchstaben den Variablentyp beschreiben. Im obigen Beispiel bedeutet b einen Booleschen Wert, während str eine Variable als String (Zeichenfolge) kennzeichnet. Diese Namenskonvention finden Sie im vorliegenden Buch und auch in anderen Büchern, die sich dem Thema Programmierung mit Visual C++ und MFC widmen. Durch diese Schreibweise lässt sich der Code für andere Programmierer leichter erfassen, was umgekehrt auch auf Sie zutrifft.

3.4 Steuerelemente mit Funktionalität ausstatten

Bevor Sie den Code für alle Steuerelemente in Ihrem Anwendungsfenster schreiben, ist zunächst etwas Code erforderlich, um die Variablen zu initialisieren, d.h. Startwerte für die meisten Variablen festzulegen. Führen Sie dazu folgende Schritte aus:

1. Erweitern Sie in der Klassen-Ansicht die Klasse CControlDlgs und wählen Sie die Funktion OnInitDialog aus der Liste der Member-Funktionen.

2. Doppelklicken Sie auf den Knoten von OnInitDialog, um zum Quellcode der Funktion zu gelangen.

Gehen Sie zur Markierung TODO, die die Stelle kennzeichnet, wo Sie mit der Codeeingabe beginnen. Geben Sie hier den Code aus Listing 3.1 ein.

Listing 3.1: In die Funktion OnInitDialog geben Sie den Initialisierungscode ein.

```
 1: BOOL CControlsDlg::OnInitDialog()
 2: {
 3:    CDialog::OnInitDialog();
 4: ...
 5:   // Symbol für dieses Dialogfeld festlegen. Wird automatisch
 6:   // erledigt
 7:   //  wenn das Hauptfenster der Anwendung kein Dialogfeld ist
 8:   SetIcon(m_hIcon, TRUE);        // Großes Symbol verwenden
 9:   SetIcon(m_hIcon, FALSE);       // Kleines Symbol verwenden
10:
11:   // TODO: Hier zusätzliche Initialisierung einfügen
12:   // Eine Standardmeldung in das Nachrichteneingabefeld setzen
13:   m_strMessage = "Schreiben Sie hier eine Mitteilung";
14:
15:   // Alle Kontrollkästchen auf aktiviert setzen
16:   m_bShowMsg = TRUE;
17:   m_bShowPgm = TRUE;
18:   m_bEnableMsg = TRUE;
19:   m_bEnablePgm = TRUE;
20:   // Den Dialog mit den Werten aktualisieren
21:   UpdateData(FALSE);
22:   return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
24:                // soll den Fokus erhalten
25: }
```

Listing 3.1 zeigt nur einen Ausschnitt der Funktion OnInitDialog. Die Listings im gesamten Buch konzentrieren sich nur auf den Code, der hinzuzufügen oder zu modifizieren ist, und zeigen der Übersichtlichkeit wegen nicht den gesamten Code aller Funktionen. (Damit bleibt auch der Umfang des Buches in einem vernünftigen Rahmen.) Um Ihre Kenntnisse zu MFC und Visual C++ zu erweitern, sollten Sie sich auch den Code ansehen, der in den Listings im Buch ausgelassen wurde. Auf der Buch-CD finden Sie natürlich den vollständigen Code. Versuchen Sie zu verstehen, was der Code bewirkt.

Wenn Sie bereits in C oder C++ programmiert haben, ist Ihnen sicherlich aufgefallen, dass der Wert der Variablen m_strMessage in einer C-untypischen Art und Weise festgelegt wird. Das Ganze erinnert mehr an die Festlegung einer Stringvariablen in Visual Basic oder PowerBuilder. Das hängt damit zusammen, dass diese Variable vom Typ CString ist. Die Klasse CString erlaubt es, in einer Visual C++-Anwendung mit Strings zu arbeiten, genau wie man mit Strings in einer der anderen Programmiersprachen umgeht. Da es sich jedoch um die Programmiersprache C++ handelt, müssen Sie dennoch ein Semikolon am Ende eines jeden Statements schreiben.

Der Initialisierungscode ist nicht weiter kompliziert. Das Programm stellt zuerst in das Eingabefeld eine anfängliche Nachricht, die Sie dem Benutzer anzeigen wollen, und setzt dann alle Kontrollkästchen in den

eingeschalteten Zustand. Die letzte Zeile des hinzugefügten Codes verdient etwas mehr Beachtung.

Die Funktion UpdateData bildet den Schlüssel für die Arbeit mit Steuerelementvariablen in Visual C++. Die Funktion übernimmt die Daten aus den Variablen und aktualisiert mit den Werten die Steuerelemente auf dem Bildschirm. Umgekehrt übernimmt die Funktion die Daten aus den Steuerelementen und füllt die zugeordneten Variablen mit allen vom Benutzer geänderten Werten. Die Richtung der Datenübertragung steuert man mit dem an die Funktion UpdateData übergebenen Argument.

Ist das Argument auf FALSE gesetzt, werden die Werte in den Variablen an die Steuerelemente im Fenster übertragen. Übergibt man TRUE als Argument, erhalten die Variablen die aktuellen Werte der Steuerelemente im Fenster. Welcher Wert an die Funktion zu übergeben ist, hängt also davon ab, in welcher Richtung die Aktualisierung stattfinden soll.

Nachdem Sie eine oder mehrere Variablen in Ihrem Code aktualisiert haben, müssen Sie UpdateData aufrufen und FALSE als Argument übergeben. Wenn Sie die Variablen lesen müssen, um deren aktuellen Wert zu erhalten, ist UpdateData mit dem Argument TRUE aufzurufen, bevor Sie irgendeinen Wert aus den Variablen verarbeiten. Ein Gefühl für diese Vorgehensweise werden Sie entwickeln, wenn Sie mehr Code in Ihre Anwendung aufnehmen.

Achten Sie sorgfältig darauf, wo Sie die Funktion UpdateData aufrufen und welchen Wert Sie ihr übergeben. Wenn Sie beispielsweise die Werte all Ihrer Variablen initialisieren und dann UpdateData(TRUE) aufrufen, löschen Sie die Werte der Variablen aus und ersetzen sie durch die Werte der Steuerelemente. Wenn andererseits ein Benutzer den Wert in einem der Steuerelemente ändert und dann UpdateData(FALSE) aufgerufen wird, ändert sich der Wert des Steuerelements in den Wert der Variablen und die Eingabe des Benutzers wird gelöscht.

C++-Exkurs: Der Scope- Resolution-Operator

Wenn Sie den Codeausschnitt in Listing 3.1 betrachten, sehen Sie folgende Code-Zeile:

```
CDialog::OnInitDialog();
```

Basierend auf dem, was Sie bisher gesehen haben, sieht das wie die erste Zeile des Funktionslistings aus, wo die Klasse gefolgt von zwei Doppelpunkten und dem Funktionsnamen angegeben ist:

```
BOOL CControlsDlg::OnInitDialog()
```

In der ersten Zeile des Listings steht der Funktionsname mit dem Rückgabetyp und der Klasse, deren Mitglied die Funktion ist. Die dritte Zeile ruft dieselbe Funktion in der Basisklasse auf. So rufen Sie in C++ Basisklassen-Funktionen auf, indem Sie den Namen der Basisklasse gefolgt von zwei Doppelpunkten und dem Namen der Funktion angeben und alle benötigten Parameter übergeben.

Die beiden Doppelpunkte ohne Zwischenraum werden als Scope-Resolution-Operator bezeichnet und verwendet, um den Geltungsbereich (Scope) der Funktion anzugeben. In den beiden obigen Situationen gibt er an, dass die aufzurufende Funktion ein Mitglied der Klasse vor den beiden Doppelpunkten ist. Wenn Sie eine Funktion mit zwei Doppelpunkten vor dem Funktionsaufruf und ohne Klassennamen davor haben, gibt der Scope-Resolution-Operator an, dass es sich um eine globale Funktion handelt und nicht um ein Mitglied einer Klasse.

Sie können den Scope-Resolution-Operator auch verwenden, um den Geltungsbereich von Variablen zu definieren. Diese Verwendung findet sich allerdings viel seltener als die mit Funktionen.

Die Anwendung schließen

Als Erstes sollten Sie sich darum kümmern, dass der Benutzer die Anwendung schließen kann. Da Sie die Schaltflächen OK und Abbrechen gelöscht und eine neue Schaltfläche für das Schließen des Anwendungsfensters hinzugefügt haben, müssen Sie Code in die Funktion aufnehmen, die von der Schaltfläche Exit ausgelöst wird, um das Fenster zu schließen. Führen Sie dazu die folgenden Schritte aus:

1. Verwenden Sie die Registerkarte über dem Bearbeitungsbereich, um das Dialogfenster zu öffnen, und wählen Sie die Schaltfläche Exit aus. Fügen Sie mithilfe der Eigenschaftenansicht eine Funktion auf der BN_CLICKED-Nachricht für button ein, wie Sie es an den vergangenen beiden Tagen gelernt haben. Wenn Sie den Dialog geschlossen oder das Projekt geschlossen und neu geöffnet haben, verwenden Sie die Ressourcen-Ansicht, um den Dialog wieder zu öffnen.

2. Geben Sie den fett gedruckten Code aus Listing 3.2 ein.

Listing 3.2: Die Funktion OnBnClickedExit

```
1: void CControlsDlg::OnBnClickedExit()
2: {
3:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:    // Benachrichtigung ein.
5:    // Das Programm verlassen
6:    OnOK();
7: }
```

Ein einziger Funktionsaufruf in der Funktion OnBnClickedExit schließt das Fenster und beendet die Anwendung. Woher kommt diese OnOK-Funktion und warum mussten Sie sie nicht in der gestrigen Anwendung aufrufen? Zwei Funktionen, OnOK und OnCancel, sind in der Basisklasse CDialog definiert, von der Ihre Klasse CControlsDlg abgeleitet ist. In der Klasse CDialog hat die Nachrichtenzuordnungstabelle bereits die Objekt-IDs der Schaltflächen OK und Abbrechen mit den Funktionen OnOK bzw. OnCancel verbunden, sodass Schaltflächen mit diesen IDs automatisch die entsprechenden Funktionen aufrufen. Wenn Sie die Objekt-ID der Schaltfläche Exit mit IDOK festgelegt hätten, müssten Sie der Schaltfläche keinerlei Code zuordnen, solange Sie nicht die grundlegende Funktionalität von OnOK überschrieben.

Sie fragen sich vielleicht, warum Sie in diesem Fall nicht durch diesen Aufruf angeben mussten, dass sich die Funktion OnOK in der Basisklasse CDialog befindet:

```
CDialog::OnOK();
```

Sie mussten das nicht angeben, weil die Funktion in Ihrer Klasse nicht überschrieben wird. Wenn Sie eine Funktion aus einer Basisklasse aufrufen, die nicht in einer abgeleiteten Klasse überschrieben wurde, müssen Sie den Scope-Resolution-Operator nicht verwenden, um die Klasse anzugeben, in der die Funktion enthalten ist.

Die Nachricht des Benutzers anzeigen

Es sollte ein Leichtes sein, die vom Benutzer in das Eingabefeld eingegebene Nachricht anzuzeigen, da dieser Vorgang den gestern beschriebenen Abläufen ähnelt. Sie können der Schaltfläche Zeige Mitteilung eine Funktion zuordnen und die Funktion MessageBox aufrufen, wie es aus Listing 3.3 hervorgeht.

Listing 3.3: Die Funktion OnBnClickedShwmsg zeigt die Benutzernachricht an.

```
void CControlsDlg::OnBnClickedShwmsg()
{
  // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
  // Benachrichtigung ein.
  // Die Nachricht des Benutzers anzeigen
```

```
  MessageBox(m_strMessage);
}
```

Wenn Sie die Anwendung in dieser Phase kompilieren und ausführen, tritt ein Problem mit diesem Code zutage. Es erscheint der String, mit dem Sie die Variable m_strMessage in der Funktion OnInitDialog initialisiert haben, und nicht die Nachricht, die der Benutzer in das Eingabefeld eingibt. Das ist darauf zurückzuführen, dass Sie die Variable noch nicht mit dem Inhalt des Steuerelements im Fenster aktualisiert haben. Es ist UpdateData mit dem Argument TRUE aufzurufen, um die Werte der Steuerelemente zu lesen und die Variablen damit zu initialisieren, bevor Sie die Funktion MessageBox aufrufen. Ändern Sie die Funktion OnBnClickedShwmsg gemäß Listing 3.4 ab.

Listing 3.4: Die aktualisierte Version der Funktion OnBnClickedShwMsg

```
 1: void CControlsDlg::OnBnClickedShwmsg()
 2: {
 3:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:    // Benachrichtigung ein.
 5:    // Nachrichtenvariable mit der Benutzereingabe aktualisieren
 6:    UpdateData(TRUE);
 7:
 8:    // Die Nachricht des Benutzers anzeigen
 9:    MessageBox(m_strMessage);
10: }
```

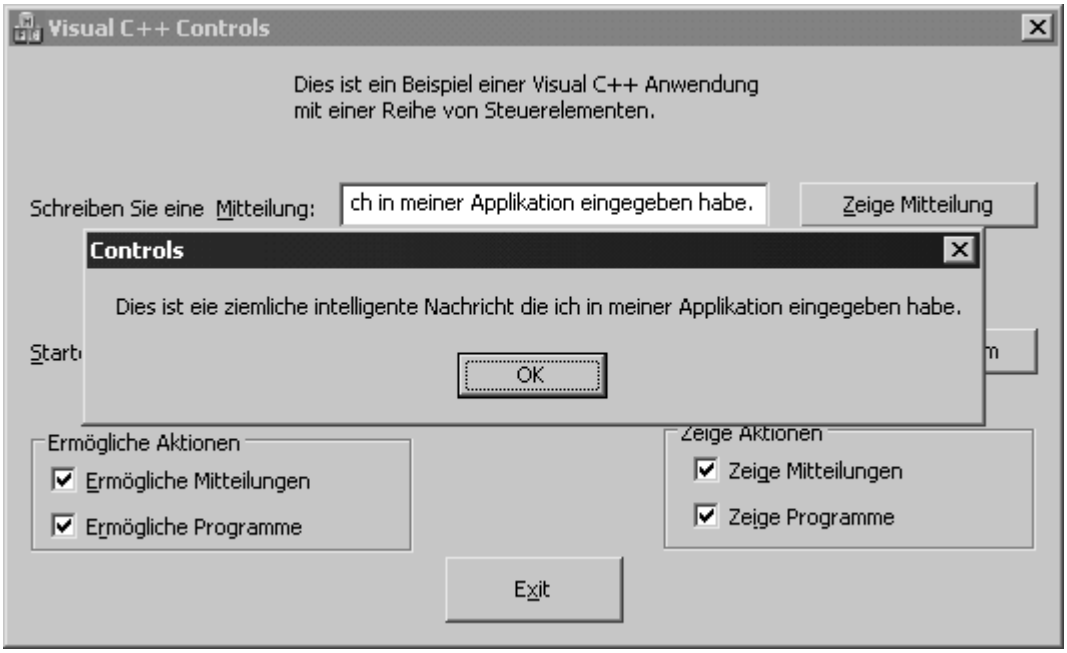


Abbildung 3.8: Die in das Eingabefeld eingetippte Nachricht wird dem Benutzer angezeigt.

Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, sollte die in das Eingabefeld eingegebene Meldung angezeigt werden, wie es Abbildung 3.8 verdeutlicht.

Die Nachricht des Benutzers löschen

Falls der Benutzer ein leeres Eingabefeld vorfinden möchte, bevor er eine Nachricht eintippt, können Sie der Schaltfläche Lösche Mitteilung eine Funktion zuordnen, um den Inhalt des Eingabefeldes zu leeren. Die Funktion fügen Sie in der gewohnten Weise über den Class Assistent hinzu. Die Funktionalität realisieren Sie ganz einfach, indem Sie die Variable m_strMessage auf einen leeren String setzen und dann die Steuerelemente im Fenster aktualisieren, um diesen Wert wiederzugeben. Der entsprechende Code ist in Listing 3.5 zu sehen.

Listing 3.5: Die Funktion OnBnClickedClrmsg

```
 1: void CControlsDlg::OnBnClickedClrmsg()
 2: {
```

```
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Nachricht löschen
 6:   m_strMessage = "";
 7:
 8:  // Bildschirm aktualisieren
 9:   UpdateData(FALSE);
10: }
```

Die Nachrichtensteuerelemente deaktivieren und ausblenden

In Bezug auf die Nachrichtensteuerelemente ist als Letztes noch die Funktionalität für die Kontrollkästchen Ermögliche Mitteilungen und Zeige Mitteilungen zu implementieren. Das erste dieser Kontrollkästchen aktiviert oder deaktiviert die Steuerelemente, die sich auf die Anzeige der Benutzernachricht beziehen. Wenn das Kontrollkästchen eingeschaltet ist, sind alle Steuerelemente aktiviert. Weist das Kontrollkästchen den ausgeschalteten Zustand auf, sind die betreffenden Steuerelemente deaktiviert. Analog dient das zweite Kontrollkästchen dazu, dieselbe Gruppe der Steuerelemente anzuzeigen bzw. auszublenden. Listing 3.6 zeigt den Code für beide Funktionen.

Listing 3.6: Die Funktionen für die Kontrollkästchen Ermögliche Mitteilungen und Zeige Mittteilungen

```
 1: void CControlsDlg::OnBnClickedCkenblmsg()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Aktuelle Werte vom Bildschirm holen
 6:   UpdateData(TRUE);
 7:
 8:   // Kontrollkästchen 'Ermögliche Mitteilungen' eingeschaltet?
 9:   if (m_bEnableMsg == TRUE)
10:   {
11:     // Ja, dann alle Steuerelemente aktivieren,
12:     // die für die Anzeige der Nachricht relevant sind.
13:     GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
14:     GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
15:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
16:     GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
17:     GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
18:   }
19:   else
20:   {
21:     // Nein, dann alle Steuerelement deaktivieren,
22:     // die für die Anzeige der Nachricht relevant sind.
23:     GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
24:     GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
25:     GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
26:     GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
27:     GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
28:   }
29: }
30:
31: void CControlsDlg::OnBnClickedCkshwmsg()
32: {
33:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
34:   // Benachrichtigung ein.
35:   // Aktuelle Werte vom Bildschirm holen
36:   UpdateData(TRUE);
37:   // Kontrollkästchen 'Zeige Mittteilungen' aktiviert?
38:   if (m_bShowMsg == TRUE)
39:   {
40:     // Ja, dann alle Steuerelemente anzeigen, die
41:     // für Anzeige der Nachricht relevant sind.
42:     GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
```

```
43:      GetDlgItem(IDC_SHWMSG)->ShowWindow(TRUE);
44:      GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
45:      GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
46:      GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
47:    }
48:    else
49:    {
50:      // Nein, dann alle Steuerelemente ausblenden, die
51:      // für Anzeige der Nachricht relevant sind.
52:      GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
53:      GetDlgItem(IDC_SHWMSG)->ShowWindow(FALSE);
54:      GetDlgItem(IDC_DFLTMSG)->ShowWindow(FALSE);
55:      GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
56:      GetDlgItem(IDC_STATICMSG)->ShowWindow(FALSE);
57:    }
58: }
```

Den ersten Teil dieser Funktionen sollten Sie mittlerweile verstehen. Als Erstes werden die Variablen mit den aktuellen Werten der Steuerelemente im Fenster aktualisiert. Es folgt ein Test der Booleschen Variablen, die mit dem jeweiligen Kontrollkästchen verbunden ist. Enthält die Variable den Wert TRUE, soll das Programm das Steuerelement aktivieren oder anzeigen. Hat die Variable den Wert FALSE, ist das Steuerelement zu deaktivieren bzw. auszublenden.

Von jetzt an ist der Code nicht mehr ganz so durchsichtig. Die erste Funktion, GetDlgItem, erhält als Parameter die ID des zu ändernden Steuerelements. Die Funktion liefert einen Zeiger auf das Steuerelement zurück. Dadurch lässt sich ein Zeiger auf beliebige Steuerelemente des Fensters bei laufender Anwendung abrufen. Der nächste Teil jedes Befehls ruft eine Member-Funktion des Steuerelementobjekts auf. Die zweite Funktion ist eine Member-Funktion des Steuerelements, für das von der ersten Funktion ein Zeiger zurückgegeben wurde.

Die zweiten Funktionen in diesen Aufrufen, EnableWindow und ShowWindow, sehen eher wie Funktionen für Fenster, aber nicht für Steuerelemente, aus. Natürlich sind sie für Fenster vorgesehen. Zufällig sind Steuerelemente aber auch Elemente der Klasse CWnd, die eine Basisklasse der Klasse CDialog ist, von der Sie Ihre Klasse CControlsDlg abgeleitet haben. In Windows sind nun mal alle Steuerelemente selbst Fenster, die völlig unabhängig von dem Fenster sind, in dem sie sich befinden. Damit kann man Steuerelemente als Fenster behandeln und ihre Fensterfunktionen aufrufen. In der Tat sind alle Steuerelementklassen von der Klasse CWnd abgeleitet, was ihr wahres Gesicht als Fenster offenbart.

Abbildung 3.9: Die Steuerelemente für Benutzermitteilungen lassen sich nun deaktivieren.

Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, können Sie die Kontrollkästchen Ermögliche

Mitteilungen und Zeige Mitteilungen ausprobieren. Die Funktionsweise sollte Abbildung 3.9 entsprechen.

C++-Exkurs: Boolesche Ausdrücke

In Listing 3.6 hätten Sie die if-Statements auch etwas anders schreiben können. Da die Booleschen Variablen, die Sie überprüfen, von ihrer Natur her entweder TRUE oder FALSE sind, müssen Sie ihren Wert nicht wirklich mit TRUE vergleichen. Stattdessen können Sie das if-Statement einfach so schreiben:

```
if (m_bEnableMsg)
```

Nach dieser Logik können Sie das Gleiche mit verschiedenen Variablen tun, bei denen Sie nur feststellen müssen, ob diese Variablen einen Wert besitzen (also beispielsweise nicht 0 sind). Das wird oft bei der Überprüfung von Zeigern verwendet, um sicherzustellen, dass sie auf etwas zeigen. Dabei wird immer angenommen, dass die fragliche Variable mit NULL (als 0 definiert) initialisiert wurde. Wenn Sie beispielsweise überprüfen möchten, ob einer Variable ein Wert zugewiesen wurde, könnten Sie das so tun:

```
if (iVar != 0)
```

Sie könnten jedoch auch Folgendes verwenden:

```
if (iVar)
```

In Umkehrung dieser Logik können Sie auch überprüfen, ob eine Variable gleich 0 ist, indem Sie wie folgt den Ausdruck negieren:

```
if (!iVar)
```

In diesem Fall bedeutet das Ausrufezeichen (!) nicht. Genau wie != »nicht gleich« bedeutet, liest sich !iVar als »nicht iVar« oder »iVar ist gleich null«.

C++-Exkurs: Objektzeiger

Der nächste Teil des obigen Code-Listings besteht in erster Linie aus einer Wiederholung ein und derselben Zeile. Diese Zeile sieht so aus:

```
GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
```

Der ersten Funktion, GetDlgItem, wird die ID eines Steuerelements im Dialog übergeben. Die Funktion gibt einen Zeiger auf das angegebene Steuerelement zurück. Wenn eine Funktion einen Zeiger zurückgibt, können Sie diesen in einer Variable speichern oder wie in diesem Beispiel den zurückgegebenen Zeiger verwenden, um eine Member-Funktion des Objekts aufzurufen, auf das der Zeiger verweist.

> *Ein Zeiger (auch Pointer genannt) ist ein Verweis auf ein Objekt oder eine Variable. Er besteht aus der Speicheradresse, an der sich das Objekt oder die Variable derzeit befindet. Wenn Sie einen Zeiger verwenden, anstatt der CPU das Objekt zu übergeben, teilen Sie der CPU mit, wo sie das Objekt findet. Dadurch sind Zeiger wesentlich kleiner und somit schneller verarbeitbar als umfangreiche Datenstrukturen.*

Wenn Sie eine Variable von einer bestimmten Klasse deklarieren, können Sie mithilfe der Punkt-Notation die Methoden dieser Klasse aufrufen und auf ihre Eigenschaften und Variablen zugreifen:

```
CMyObject obj;
obj.Method1();
```

Wenn Sie mit einem Zeiger arbeiten, ändert sich die Syntax an zwei Stellen. Zuerst wird ein Zeiger mit einem Stern (*) vor dem Variablennamen deklariert:

```
CMyObject *obj;
```

oder

```
CMyObject* obj;
```

Die erste Deklaration sagt aus, dass die Variable obj ein Zeiger auf eine Klasse CMyObject ist. Diese Zeile könnte auch noch eine weitere Variable enthalten, die eine Instanz der Klasse CMyObject ist:

```
CMyObject *obj, objInstance;
```

Die zweite Deklaration sagt aus, dass es sich bei allen in dieser Zeile deklarierten Variablen um Zeiger auf Instanzen der Klasse CMyObject handelt.

Wenn Sie einen Zeiger verwenden, um auf Member-Methoden, Eigenschaften oder Variablen eines Objekts zuzugreifen, verwenden Sie immer die ->-Notation:

```
obj->Method1();
```

Wenn Sie eine Instanz eines Objekts oder einer Variablen haben und einen Zeiger initialisieren wollen, der auf diese Instanz verweist, verwenden Sie das kaufmännische Und (&), um die Adresse des Objekts zu erhalten:

```
CMyObject objInstance, *pObj;
pObj = &objInstance;
```

Ein kaufmännisches Und vor einer Variablen wird als »Adresse von« gelesen, also setzen Sie im obigen Code den Wert von pObj, einem Zeiger, auf die Adresse von objInstance, einer Instanz der Klasse CMyObject.

Wenn wir zu unserem Beispielcode zurückkommen,. Kann die sich wiederholende Code- Zeile auch so geschrieben werden:

```
Cwnd *pWnd;

pWnd = getDlgItem(IDC_MSG);
pWnd->EnableWindow(TRUE);
```

Sie können diese Funktionalität jedoch auf eine einzelne Zeile verkürzen, indem Sie die Möglichkeit nutzen, Funktionalität in einer Code-Zeile zu vereinigen und wie in Listing 3.6 den von der Funktion GetDlgItem zurückgegebenen Zeiger verwenden, um auf die Member-Funktion EnableWindow des angegebenen Steuerelements zuzugreifen:

```
GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
```

Wenn man diese Fähigkeit von C++, Funktionalität zu kombinieren, noch weiter treibt, kann die Größe der beiden Funktionen in Listing 3.6 verkürzt werden, indem man das if- Statement herausnimmt und die Boolesche Variable an die Funktionen EnableWindow und ShowWindow übergibt, anstatt die Parameterwerte hart einzuprogrammieren. So könnten Sie diesen Code nehmen:

```
// Kontrollkästchen 'Enable Message Action' eingeschaltet?
if (m_bEnableMsg == TRUE)
{
  // Ja, dann alle Steuerelemente aktivieren,
  // die für Anzeige der Nachricht relevant sind.
  GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
```

```
  GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
}
else
{
  // Nein, dann alle Steuerelemente deaktivieren,
  // die für Anzeige der Nachricht relevant sind.
  GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
}
```

und auf Folgendes reduzieren:

```
// Alle für die Anzeige der Benutzernachricht relevanten
// Steuerelemente aktivieren oder deaktivieren
GetDlgItem(IDC_MSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_SHWMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_DFLTMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_CLRMSG)->EnableWindow(m_bEnableMsg);
GetDlgItem(IDC_STATICMSG)->EnableWindow(m_bEnableMsg);
```

MFC-Exkurs: Member-Methoden von CWnd

In der Klasse CWnd, der Basisklasse für alle Benutzerschnittstellenobjekte in MFC, wurden in Listing 3.6 drei Member-Methoden verwendet:

- GetDlgItem
- EnableWindow
- ShowWindow

Die erste dieser Methoden, GetDlgItem, kann verwendet werden, um einen Zeiger auf ein Kindfenster abzurufen. Der zurückgegebene Zeiger ist ein CWnd-Zeiger. Dieser Methode wird ein Parameter übergeben: die ID des Kindfensters. Gewöhnlich wird diese Funktion verwendet, um einen Zeiger auf ein Steuerelement oder einen Dialog abzurufen, doch man kann mit ihr einen Zeiger auf jedes beliebige Kindfenster bekommen.

Die zweite Methode, EnableWindow, schaltet die Möglichkeit, dass der Benutzer mit dem Fenster, für das sie aufgerufen wurde, interagieren kann, ein oder aus. Sie übernimmt einen Booleschen Wert als einzigen Parameter. Ist der Parameter TRUE, so ist das Fenster aktiviert und der Benutzer kann damit interagieren. Ist der Parameter FALSE, so ist das Fenster oder Steuerelement deaktiviert und ignoriert Benutzeraktionen.

Die dritte Methode, ShowWindow, ähnelt der Methode EnableWindow. Sie verbirgt oder zeigt das Fenster oder Steuerelement. Sie übernimmt einen Booleschen Wert als Parameter, der ihr mitteilt, ob das Fenster oder Steuerelement sichtbar sein soll oder nicht.

Eine andere Anwendung starten

Als letzte größere Aufgabe müssen wir noch die Funktionalität für die Steuerelemente implementieren, die ein anderes Programm starten. Oberhalb von Abbildung 3.3 haben Sie die Namen von drei Windows-Anwendungen in das Kombinationsfeld aufgenommen. Wenn Sie die Anwendung starten, erscheinen diese Namen in der DropDown-Liste. Man kann einen Eintrag auswählen und im Wertbereich des Kombinationsfelds erscheint der Name der jeweiligen Anwendung. Damit die auch eine echte Funktion ausführt, müssen Sie lediglich für die Schaltfläche Starte Programm Code hinzufügen, um tatsächlich den Wert aus dem Kombinationsfeld zu ermitteln und das passende Programm zu starten. Nachdem Sie das Funktionsgerüst für die Schaltfläche Starte Programm erstellt haben, fügen Sie den Code aus Listing 3.7 in die Funktion ein.

Listing 3.7: Die Funktion OnBnClickedRunpgm startet andere Windows-Anwendungen.

```
 1: void CControlsDlg::OnBnClickedRunpgm()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   //  Benachrichtigung ein.
 5:   // Aktuelle Werte vom Bildschirm holen
 6:   UpdateData(TRUE);
 7:
 8:   // Lokale Variable zur Aufnahme des Programmnamens deklarieren
 9:   CString strPgmName;
10:
11:   // Programmname in die lokale Variable kopieren
12:   strPgmName = m_strProgToRun;
13:
14:   // Programmname in Großbuchstaben umwandeln
15:   strPgmName.MakeUpper();
16:
17:   // Programm Paint gewählt?
18:   if (strPgmName == "PAINT")
19:     // Ja, Paint starten
20:     WinExec("mspaint.exe ", SW_SHOW);
21:
22:   // Programm Notepad (Editor) gewählt?
23:   if (strPgmName == "NOTEPAD")
24:     // Ja, Notepad starten
25:     WinExec("notepad.exe ", SW_SHOW);
26:
27:   // Programm Solitaire gewählt?
28:   if (strPgmName == "SOLITAIRE")
29:     // Ja, Solitaire starten
30:     WinExec("sol.exe ", SW_SHOW);
31: }
```

Wie zu erwarten, findet in dieser Funktion zunächst der Aufruf von UpdateData statt, um die Variablen mit den Werten der Steuerelemente im Fenster zu aktualisieren. Der nächste Teil erscheint allerdings ein wenig eigentümlich. Es wird eine neue CString- Variable deklariert und in diese der Wert des Kombinationsfeldes kopiert. Ist es wirklich notwendig, wenn der Wert bereits in einer CString-Variablen steht?

Das hängt davon ab, wie sich Ihre Anwendung verhalten soll. Die nächste Code-Zeile enthält einen Aufruf der CString-Funktion MakeUpper, die den String in Großbuchstaben konvertiert. Wenn man die CString-Variable verwendet, die mit dem Kombinationsfeld verbunden ist, wird beim nächsten Aufruf von UpdateData mit dem Argument FALSE der Wert im Kombinationsfeld ebenfalls in Großbuchstaben angezeigt. Unter diesen Umständen ist das wahrscheinlich ein ungünstiger Zeitpunkt und das Verhalten entspricht nicht den Vorstellungen. Deshalb kommt in der Funktion ein zusätzlicher CString zum Einsatz.

An die Umwandlung des Strings in Großbuchstaben schließt sich eine Folge von if- Anweisungen an, die den String mit den Namen der verschiedenen Programme vergleichen. Bei einer gefundenen Übereinstimmung ruft der Code die Funktion WinExec auf, um die betreffende Anwendung zu starten. Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, können Sie eine der Anwendungen aus der DropDown-Liste auswählen und sie durch Klicken auf die Schaltfläche Starte Programm öffnen.

In C und C++ ist der Unterschied zwischen einem einfachen Gleichheitszeichen (=) und einem doppelten Gleichheitszeichen (==) zu beachten. Das einfache Gleichheitszeichen führt eine Zuweisung des Wertes auf der rechten Seite des Gleichheitszeichens an die Variable auf der linken Seite durch. Steht auf der linken Seite des Gleichheitszeichens eine Konstante, wird das Programm nicht kompiliert und Sie erhalten die Fehlermeldung, dass man keinen Wert der rechten Seite an eine Konstante auf der linken Seite zuweisen kann. Das doppelte Gleichheitszeichen ist für Vergleiche vorgesehen. Achten Sie darauf, das doppelte Gleichheitszeichen zu verwenden, wenn Sie zwei Werte miteinander vergleichen möchten.

Wenn Sie in diesem Fall versehentlich das einfache Gleichheitszeichen verwenden, ändern Sie den Wert der Variablen auf der linken Seite. Hier liegt eine der größten Quellen für logische Fehler in C/C++-Programmen.

WinExec ist eine veraltete Windows-Funktion. Man sollte statt dessen die Funktion CreateProcess verwenden. Allerdings weist CreateProcess eine Reihe von Argumenten auf, die in dieser frühen Phase der Programmierung mit Visual C++ nicht so leicht zu verstehen sind. Die Funktion WinExec ist weiterhin verfügbar und als Makro implementiert, das die Funktion CreateProcess aufruft. Damit können Sie auf die wesentlich einfachere Funktion WinExec zurückgreifen, um eine andere Anwendung zu starten, wobei gesichert ist, dass Windows die eigentlich erwartete Funktion benutzt.

Die API-Funktion ShellExecute lässt sich ebenfalls für den Start einer anderen Anwendung einsetzen. Diese Funktion war ursprünglich vorgesehen, um Dateien zu öffnen oder zu drucken, man kann sie aber auch zum Starten anderer Programme nutzen.

3.5 Zusammenfassung

Heute haben Sie gelernt, wie man Standardsteuerelemente von Windows in einer Visual C++-Anwendung einsetzt. Es wurde gezeigt, wie man Variablen deklariert, sie mit den Steuerelementen verbindet und wie man die Werte der Variablen mit den Steuerelementen synchronisiert. Weiterhin haben Sie gelernt, wie man Steuerelemente manipuliert, indem man die Steuerelementobjekte mittels ihrer Objekt-ID abruft, und wie man das Steuerelement manipuliert, indem man es als Fenster behandelt. Als Nächstes wurde auf die Tabulator-Reihenfolge der Steuerelemente in einer Anwendung eingegangen. Damit legt man die Richtung und Reihenfolge fest, in welcher der Benutzer durch eine Windows-Anwendung navigiert. Schließlich haben Sie gelernt, wie man die Funktionalität einer Anwendung mit den Steuerelementen im Anwendungsfenster verbindet, wobei die verschiedenen Aktionen ausgelöst werden, wenn der Benutzer mit den jeweiligen Steuerelementen interagiert. Als Bonus wurde gezeigt, wie man andere Windows-Anwendungen aus der eigenen Anwendung heraus aufrufen kann.

3.6 Workshop

Fragen und Antworten

Frage:
Nachdem ich die Objekt-IDs der Steuerelemente im Fenster festgelegt habe, weisen drei Steuerelemente dieselbe ID, IDC_STATIC, auf. Bei diesen Steuerelementen handelt es sich um den Text im oberen Teil des Fensters und um die beiden Gruppenfelder. Die zwei anderen statischen Textsteuerelemente hatten zuerst die gleiche ID, bis ich sie geändert habe. Wieso haben diese Steuerelemente die gleiche ID und warum musste ich die IDs der beiden statischen Textsteuerelemente an dieser Stelle ändern?

Antwort:
Alle Steuerelemente, die normalerweise nicht für eine Benutzerinteraktion vorgesehen sind, beispielsweise statischer Text und Gruppenfelder, erhalten per Vorgabe die gleiche Objekt-ID. Das funktioniert, solange Ihre Anwendung keine Aktionen auf diesen Steuerelementen ausführen muss. Wenn Sie eine Interaktion mit einem dieser Steuerelemente beabsichtigen, wie es für den statischen Text mit den Aufforderungen für das Eingabefeld und das Kombinationsfeld geschehen ist, müssen Sie den betreffenden Steuerelementen eine eindeutige ID zuweisen. In diesem Fall war die eindeutige ID erforderlich, damit Sie das Steuerelementobjekt abrufen können, um das Steuerelement zu aktivieren/zu deaktivieren bzw. anzuzeigen/auszublenden. Ebenfalls müssen Sie eine eindeutige ID zuweisen, wenn Sie eine Variable mit einem Steuerelement verbinden wollen, um die Beschriftung auf dem Steuerelement dynamisch ändern zu können.

Antwort:
Die Anwendung verhält sich in unvorhergesehener Weise, wenn Sie eines der statischen Steuerelemente, welche die gleiche ID haben, verändern. Als Faustregel gilt, dass man statischen Steuerelementen die

gleiche ID zuweisen kann, wenn man die Steuerelemente überhaupt nicht ändert. Wird eine Interaktion mit den Steuerelementen erforderlich, müssen Sie jedem Steuerelement eine eindeutige Objekt-ID zuordnen.

Frage:
Gibt es eine andere Möglichkeit, die Steuerelemente zu manipulieren, als die Steuerelementobjekte unter Verwendung ihrer Objekt-IDs abzurufen?

Antwort:
Im Assistent zum Hinzufügen von Membervariablen können Sie Variablen für Ihre Steuerelemente deklarieren, indem Sie als Variablenkategorie Control festlegen. Damit erhält man grundsätzlich ein Objekt, das die MFC-Klasse des Steuerelements darstellt, und man kann dann direkt das Steuerelement ändern und damit interagieren. Für das Steuerelement lassen sich dann alle Funktionen der Klasse CWnd aufrufen, wie Sie es beim Aktivieren/Deaktivieren bzw. Anzeigen/ Ausblenden der Steuerelemente in Ihrer Anwendung vorgenommen haben. Sie können auch die Klassenmethoden der Steuerelemente aufrufen. Das bietet Ihnen die Möglichkeit, im Code spezielle Aufgaben für den jeweiligen Steuerelementtyp zu realisieren. Wenn Sie zum Beispiel dem Kombinationsfeld eine weitere Variable zuordnen und festlegen, dass es sich um eine Variable der Kategorie Control handelt, können Sie über diese Variable die Elemente in der DropDown-Liste des Steuerelements hinzufügen.

Quiz

1. Warum muss man die Tabulator-Reihenfolge der Steuerelemente im Anwendungsfenster festlegen?

2. Wie kann man eine Zugriffstaste einbinden, die den Benutzer zum Eingabefeld oder Kombinationsfeld bringt?
3. Warum muss man den statischen Textfeldern vor dem Eingabefeld und vor den Kombinationsfeldern eindeutige Objekt-IDs zuweisen?
4. Warum muss man die Funktion UpdateData aufrufen, bevor man den Wert eines Steuerelements überprüft?

Übungen

1. Fügen Sie für die Schaltfläche Default Mitteilung Code hinzu, um den Text im Eingabefeld auf »Schreiben Sie hier eine Mitteilung« zurückzusetzen.

2. Nehmen Sie Code auf, um die Steuerelemente zur Auswahl und zum Start einer anderen Anwendung zu aktivieren oder zu deaktivieren sowie anzuzeigen oder auszublenden.
3. Erweitern Sie den Code in der Funktion OnBnClickedRunpgm (in Listing 3.7), damit der Benutzer den Namen des auszuführenden Programms selbst eingeben kann.

Tag 4

Maus und Tastatur

Je nach Art der zu erstellenden Anwendung muss man sich Gedanken darüber machen, welche Aktionen der Benutzer mit der Maus unternimmt - wann und wo er mit der Maus klickt, welche Taste er dabei betätigt und wann er die Taste loslässt. Außerdem muss man darüber informiert sein, was der Benutzer bei gedrückter Maustaste unternimmt.

Weiterhin ist es gegebenenfalls erforderlich, Tastaturereignisse zu verarbeiten. Wie bei der Maus braucht man die Angaben, wann der Benutzer eine Taste drückt, wie lange er sie gedrückt hält und wann er sie wieder loslässt.

In der heutigen Lektion lernen Sie, ...

- welche Mausereignisse verfügbar sind und wie man bestimmt, welche Ereignisse für eine Anwendung relevant sind,
- wie man Mausereignisse in einer Visual C++-Anwendung auffangen und darauf reagieren kann,
- welche Tastaturereignisse verfügbar sind und welche Aktionen diese Ereignisse auslösen,
- wie man Tastaturereignisse auffängt und eine Aktion in Abhängigkeit von der gedrückten Taste unternimmt.

4.1 Mausereignisse

Wie Sie gestern erfahren haben, bietet der Klassen-Assistent für die meisten Steuerelemente nur eine ausgewählte Anzahl von Ereignissen. Bei Mausereignissen beschränkt sich das hauptsächlich auf Klicken und Doppelklicken. Ein kurzer Blick auf die Maus macht aber deutlich, dass zum Auffangen von Mausereignissen mehr gehören muss, als das Erkennen der beiden genannten Aktionen. Wie steht es zum Beispiel mit der rechten Maustaste? Wie lässt sich herausfinden, ob sie gedrückt ist? Und was passiert in Zeichenprogrammen? Wie können diese Programme der Spur folgen, die man mit dem Mauszeiger vorgibt?

Wenn Sie das Dialogfeld markieren und sich die Option Meldungen (Nachrichten) in der Eigenschaftenansicht ansehen und durch die verfügbaren Nachrichten scrollen, finden Sie eine Reihe von Ereignissen, die sich auf die Maus beziehen. Mit diesen Ereignissen kann man alle Aufgaben erledigen, die in einer Anwendung anfallen. Tabelle 4.1 bringt eine Übersicht dieser Ereignisse.

Nachricht	Beschreibung
WM_LBUTTONDOWN	Die linke Maustaste wurde gedrückt.
WM_LBUTTONUP	Die linke Maustaste wurde losgelassen.
WM_LBUTTONDBLCLK	Mit der linken Maustaste wurde ein Doppelklick ausgeführt.
WM_RBUTTONDOWN	Die rechte Maustaste wurde gedrückt.
WM_RBUTTONUP	Die rechte Maustaste wurde losgelassen.
WM_RBUTTONDBLCLK	Mit der rechten Maustaste wurde ein Doppelklick ausgeführt.
WM_MBUTTONDOWN	Die mittlere Maustaste (einer 3-Tasten-Maus) wurde gedrückt.
WM_MBUTTONUP	Die mittlere Maustaste (einer 3-Tasten-Maus) wurde losgelassen.
WM_MBUTTONDBLCLICK	Mit der mittleren Maustaste (einer 3-Tasten-Maus) wurde ein Doppelklick ausgeführt.
WM_XBUTTONDOWN	Eine Erweiterungstaste einer Microsoft Intellimouse wurde gedrückt.
WM_XBUTTONUP	Eine Erweiterungstaste einer Microsoft Intellimouse wurde losgelassen.

WM_XBUTTONDBCLICK	Mit einer Erweiterungstaste einer Microsoft Intellimouse wurde ein Doppelklick ausgeführt.
WM_MOUSEMOVE	Die Maus wird über den Fensterbereich der Anwendung verschoben.
WM_MOUSEWHEEL	Das Mausrad wird bewegt.

Tabelle 4.1: Nachrichten für Mausereignisse

Mit der Maus zeichnen

Heute erstellen Sie ein einfaches Zeichenprogramm, mit dem der Benutzer einfache Figuren in einem Dialogfeld zeichnen kann. Das Programm verwendet einige der verfügbaren Mausereignisse. Im Wesentlichen stützt sich die Anwendung auf die Nachricht WM_MOUSEMOVE, die signalisiert, dass die Maus verschoben wird. Dabei erfahren Sie, wie man innerhalb der Behandlungsfunktion ermittelt, ob die linke Maustaste gedrückt oder losgelassen wurde. Weiterhin lernen Sie, wie man die Position der Maus im Fenster bestimmt. Das klingt alles ganz einfach. Führen Sie also die folgenden Schritte aus:

1. Legen Sie ein neues MFC-Anwendung Visual C++-Projekt an und nennen Sie das Projekt Mouse.

2. Legen Sie im MFC Anwendungs-Assistent für das Projekt fest, dass Sie eine dialogfeldbasierte Anwendung erstellen.
3. Übernehmen Sie die Standardeinstellungen des MFC Anwendungs-Assistent. Geben Sie als Titel der Anwendung Mouse and Keyboard an.
4. Nachdem das Anwendungsgerüst erstellt wurde, entfernen Sie alle Steuerelemente aus dem Dialogfeld. Damit steht die gesamte Fensterfläche des Dialogfelds zum Zeichnen zur Verfügung. Dieser Schritt ist auch erforderlich, damit die Anwendung alle Tastaturereignisse auffangen kann.

Wenn in einem Dialogfeld irgendwelche Steuerelemente vorhanden sind, gelangen alle Tastaturereignisse zum Steuerelement, das momentan den Eingabefokus hat. (Ein Steuerelement, das den Eingabefokus hat, ist hervorgehoben oder enthält den Cursor zur Eingabe.) Damit sich Tastaturereignisse in einem Dialogfeld auffangen lassen, muss man alle Steuerelemente aus dem Dialogfeld löschen.

5. Markieren Sie das Dialogfenster und wählen Sie dann in der Eigenschaftenansicht den Modus Meldungen (siehe Abbildung 4.1). Wählen Sie aus der Liste der Nachrichten WM_MOUSEMOVE und fügen Sie eine Funktion ein, indem Sie im Kombinationsfeld OnMouseMove wählen.

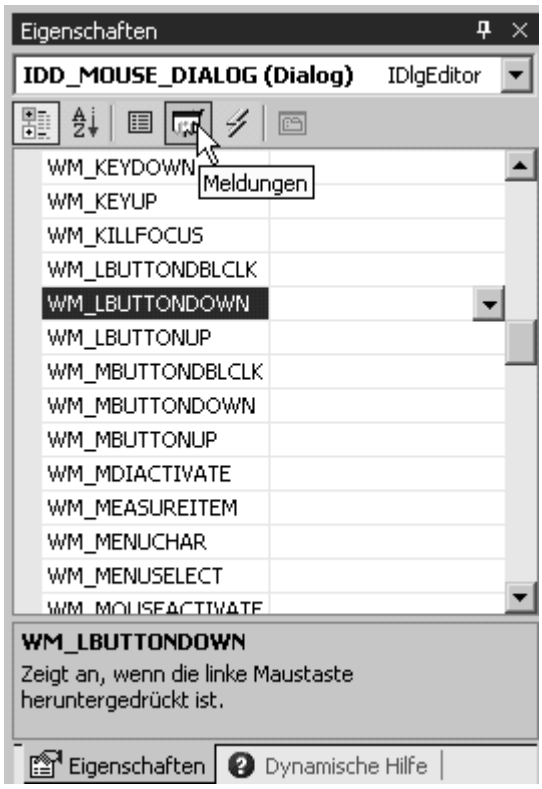


Abbildung 4.1: Auswahl des Modus Meldungen in der Eigenschaftenansicht

6. Fügen Sie den fett gedruckten Code aus Listing 4.1 ein.

Listing 4.1: Die Funktion OnMouseMove

```
 1: void CMouseDlg::OnMouseMove(UINT nFlags, CPoint point)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:   // und/oder benutzen Sie den Standard.
 5:
 6:   // Linke Maustaste gedrückt?
 7:   if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
 8:   {
 9:     // Gerätekontext holen
10:     CClientDC dc(this);
11:
12:     // Pixel zeichnen
13:     dc.SetPixel(point.x, point.y, RGB(0, 0, 0));
14:   }
15:
16:   CDialog::OnMouseMove(nFlags, point);
17: }
```

Sehen Sie sich die Funktionsdefinition am Beginn des Listings an. An die Funktion werden zwei Argumente übergeben. Das erste ist eine Gruppe von Flags, über die man ermitteln kann, ob und welche Maustaste gedrückt ist. Das geschieht in der ersten Code- Zeile mit der if-Anweisung:

```
if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
```

Der erste Teil der auszuwertenden Bedingung filtert das Flag heraus, das kennzeichnet, ob die linke Maustaste gedrückt ist. In der zweiten Hälfte werden die gefilterten Flags mit dem Flag verglichen, das für das Drücken der linken Maustaste steht. Bei einer Übereinstimmung ist die linke Maustaste gedrückt.

Sie können den Ausdruck vereinfachen, indem Sie ihn auf die &-Operation beschränken. Wenn Sie die Variable nFlags über & mit dem Flag MK_LBUTTON verknüpfen, erhalten Sie einen positiven Wert, wenn das Flag gesetzt ist, oder 0, wenn es nicht gesetzt ist. Das kann im if-Statement als Boolescher Wert ausgewertet werden:

```
if (nFlags & MK_LBUTTON)
```

Das zweite Argument an diese Funktion gibt die Position der Maus an. Dieses Argument enthält die Bildschirmkoordinaten der aktuellen Mausposition. Mithilfe dieser Angaben lässt sich ein Punkt im Dialogfenster zeichnen.

Bevor man aber Punkte im Dialogfeld zeichnen kann, muss man den Gerätekontext für das Dialogfeld holen. Dazu wird eine neue Instanz der Klasse CClientDC erzeugt. Diese Klasse kapselt den Gerätekontext und die meisten darauf ausführbaren Operationen, einschließlich aller Operationen, die auf den Bildschirm zeichnen. Praktisch ist der Gerätekontext die Leinwand, auf der Sie mit Ihrer Anwendung zeichnen: Keine Leinwand - kein Gemälde. Nachdem das Gerätekontextobjekt erzeugt ist, können Sie dessen Funktion SetPixel aufrufen. Diese Funktion färbt Pixel (Bildpunkte) an der Position, die in den beiden ersten Argumenten angegeben ist, mit der im dritten Argument spezifizierten Farbe. Wenn Sie Ihr Programm kompilieren und ausführen, können Sie sich davon überzeugen, dass Sie mit der Maus in die Fensterfläche des Dialogfelds zeichnen können (siehe Abbildung 4.2).

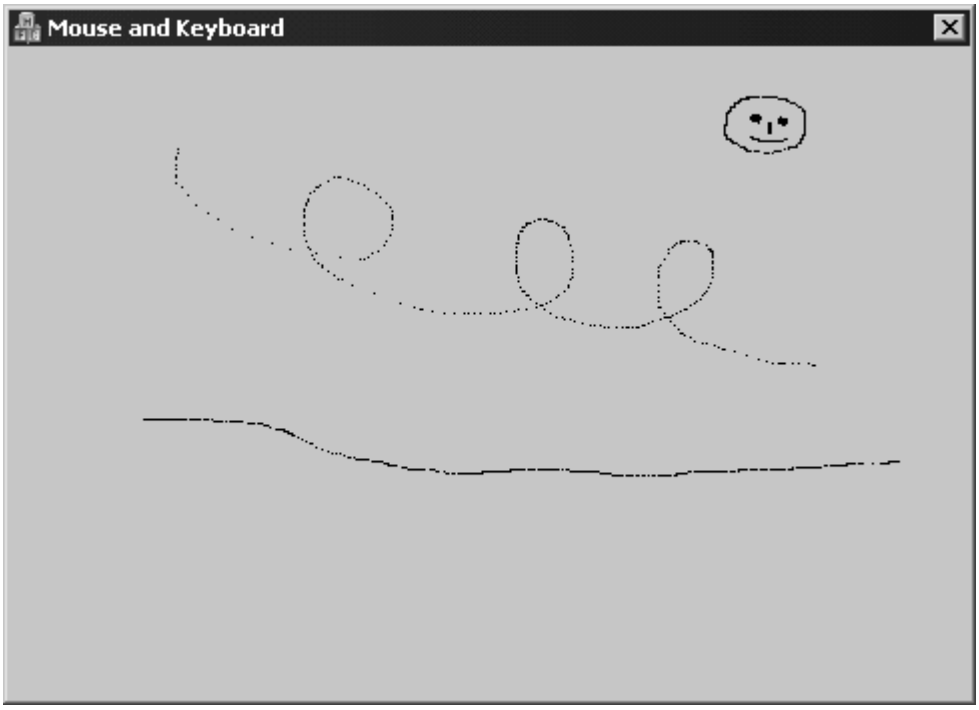


Abbildung 4.2: Mit der Maus im Fenster zeichnen

In Windows wird jede Farbe als eine große Zahl festgelegt, die sich aus drei verschiedenen Werten zusammensetzt: die Helligkeiten der roten, grünen und blauen Pixel Ihres Bildschirms. Jedes Pixel Ihrer Monitoranzeige besteht eigentlich aus drei verschiedenen Farbpixeln, die miteinander kombiniert die schließlich sichtbare Farbe ergeben. Die Funktion RGB ist ein

Makro, das die drei separaten Werte in eine einzige Zahl umwandelt. Diese Zahl lässt sich an die Funktion SetPixel oder eine andere Funktion, die einen Farbwert erfordert, übergeben. Die Werte für die Farben Rot, Grün und Blau können im Bereich zwischen 0 und 255 (jeweils einschließlich) liegen. 0 ist der dunkelste Zustand, den ein Pixel annehmen kann, 255 der hellste.

Um zu sehen, wie die drei Farbwerte zusammen eine bestimmte Farbe erzeugen, öffnen Sie das Symbol einer Anwendung im Symbol-Editor und wählen Sie Bild / Farben anpassen. Experimentieren Sie mit verschiedenen Werten zwischen 0 und 255 in den Eingabefeldern für Rot, Grün und Blau.

C++-Exkurs: Die binären Verknüpfungen AND und OR

In C++ unterscheidet man logische und binäre Verknüpfungen mit AND und OR. Die erste Kategorie verwendet man in logischen oder Bedingungsanweisungen, beispielsweise if- oder while-Anweisungen, die den Programmfluss steuern. Die binären AND- und OR- Verknüpfungen kombinieren zwei Werte auf Bitebene.

In C/C++ können Sie mithilfe der Bit-Operatoren bitweise Vergleiche und Verknüpfungen durchführen. Diese Bit-Operatoren sind in Tabelle 4.2 aufgelistet.

Operator	Beschreibung
&	And auf Bitebene
&&	Logisches And
|	Or auf Bitebene
||	Logisches Or
^	Xor auf Bitebene (ausschließendes Or)
~	Komplement (Negation) auf Bitebene
>>	Verschiebung nach rechts (Bitebene)
<<	Verschiebung nach links (Bitebene)

Tabelle 4.2: Bit-Operatoren in C/C++

Die AND-Verknüpfung wird durch das kaufmännische Und-Zeichen (&) repräsentiert. Ein einzelnes Und-Zeichen (&) steht für ein binäres AND, ein doppeltes Und-Zeichen (&&) kennzeichnet ein logisches AND.

Die Arbeitsweise der logischen AND-Verknüpfung entspricht dem Schlüsselwort AND in Visual Basic oder PowerBuilder. Diese Verknüpfung verwendet man beispielsweise in einer if-Anweisung, um auszudrücken »Wenn (IF) diese Bedingung UND (AND) die andere Bedingung erfüllt sind, dann ...«. Beide Bedingungen müssen erfüllt (TRUE) sein, damit die gesamte Anweisung das Ergebnis TRUE liefert. Eine binäres AND setzt man dagegen ein, um Bits auf 0 zu setzen oder bestimmte Bits einer Zahl einzeln betrachten zu können (maskieren). Wenn man zwei Werte mit einem binären AND verknüpft, behalten nur diejenigen Bit-Positionen eine 1, die in beiden Werten auf 1 gesetzt waren. Alle anderen Bits werden auf 0 gesetzt. Sehen wir uns zur Verdeutlichung zwei 8-Bit-Werte an:

```
Wert1              01011001
Wert2              00101001
```

Verknüpft man diese beiden Werte mit einem binären AND, erhält man folgendes Ergebnis:

```
Wert1 AND Wert2    00001001
```

Alle Bit-Positionen, auf denen in mindestens einem Wert eine 0 steht, sind im Ergebnis auf 0 gesetzt. Bits, die in beiden Werten auf 1 stehen, bleiben auch im Ergebnis auf 1.

Die OR-Verknüpfung wird durch das Pipe-Zeichen (|) symbolisiert. Analog zur AND- Verknüpfung steht ein einzelnes Pipe-Zeichen (|) für ein binäres OR, während zwei Pipe- Zeichen (||) ein logisches OR kennzeichnen. Das logische OR kann man ebenfalls in Bedingungsanweisungen wie if- oder while-Anweisungen einsetzen, um den logischen Programmfluss zu steuern. Es verhält sich genau wie das Schlüsselwort OR in Visual Basic oder PowerBuilder. Diese Verknüpfung verwendet man beispielsweise in einer if- Anweisung, um auszudrücken »Wenn (IF) diese Bedingung ODER (OR) die andere Bedingung erfüllt ist, dann ...«. Wenn eine der beiden Bedingungen TRUE ist, liefert auch die gesamte Anweisung das Ergebnis TRUE. Mit dem binären OR lassen sich Werte auf Bitebene kombinieren. Ist in einer OR-Verknüpfung die Bit-Position in einem der Werte gleich 1, ist auch das entsprechende Bit im Ergebniswert auf 1 gesetzt. Eine 0 im Ergebnis tritt bei einer binären OR-Verknüpfung nur dann auf, wenn beide Bits in den Operanden gleich 0 sind. Mit den beiden obigen Beispielwerten

```
Wert1              01011001
Wert2              00101001
```

erhält man das Ergebnis einer binären OR-Verknüpfung zu:

```
Wert1 OR Wert2     01111001
```

Hier wird jedes Ergebnisbit zu 1, das in mindestens einem der beiden Werte gleich 1 ist. Nur die Bit-Positionen im Ergebnis sind 0, die in beiden Ausgangswerten gleich 0 sind.

XOR (eXclusive OR, ausschließendes Oder oder Entweder/Oder) wird von einem Zirkumflex (^) repräsentiert. Dieser Operator funktioniert nur in der binären Ebene. Wenn man zwei binäre Werte hat, untersucht der Operator jedes Bit in beiden Werten und gibt eine 1 in den Bits zurück, in denen die Werte der beiden Bit unterschiedlich sind. Das Bit erhält das Ergebnis 0, wenn dieses Bit in beiden Werten identisch gesetzt ist. Wenn Sie beispielsweise unsere Beispielwerte durch XOR verknüpfen,

```
Wert 1             01011001
Wert 2             00101001
```

erhalten Sie diesen Wert:

```
Wert 1 XOR Wert 2  01110000
```

Der Komplement-Operator wird durch eine Tilde (~) repräsentiert. Dieser Operator arbeitet mit einem einzelnen binären Wert. Er kehrt einfach die binären Werte aller Bits in einem Wert um. Wenn ein Bit auf 1 gesetzt ist, setzt der Komplement-Operator es auf 0 und umgekehrt. Um zu sehen, wie das funktioniert, nehmen wir den ersten unserer Beispielwerte:

```
Wert 1             01011001
```

Sein Komplement ist der folgende Wert:

```
Komplement-Wert    10100110
```

Die letzten beiden Operatoren auf Bitebene sind die Shift-Operatoren. Diese beiden Operatoren werden verwendet, um die Bits in einem Wert um eine bestimmte Anzahl von Bits nach rechts oder links zu verschieben. Wenn Sie beispielsweise den ersten Wert um drei Bits nach rechts verschieben wollen, werden Sie das so schreiben:

```
wert1 = wert1 >> 3;
```

Das funktioniert so, dass der Wert

```
Wert 1             01011001
```

um drei Bits nach rechts verschoben wird, was folgenden Wert ergibt:

```
Verschobener Wert  00001011
```

Soll der gleiche Wert um drei Bits nach links verschoben werden, schreiben Sie das so:

```
wert1 = wert1 << 3;
```

Das bewirkt, dass der Wert

```
Wert 1              01011001
```

um drei Bits nach links verschoben und zu diesem Wert wird:

```
Verschobener Wert   11001000
```

C++-Exkurs: Binäre Attributflags

Die binären Verknüpfungen mit AND und OR verwendet man in C++ unter anderem, um Attributflags zu setzen und zu lesen. Attributflags sind Werte, bei denen jedes einzelne Bit angibt, ob eine bestimmte Option ein- oder ausgeschaltet ist. Damit kann der Programmierer definierte Flags verwenden. Unter einem definierten Flag versteht man einen Wert, bei dem nur ein Bit auf 1 gesetzt ist, oder eine Kombination von anderen Werten, bei denen eine bestimmte Bit-Kombination auf 1 gesetzt ist, sodass sich mehrere Optionen mit einem einzelnen Wert ausdrücken lassen. Die Flags, mit denen man verschiedene Optionen steuert, werden OR-verknüpft. Damit erhält man einen zusammengesetzten Wert, der angibt, welche Optionen eingeschaltet und welche ausgeschaltet sind.

Wenn zwei Flags, die bestimmte Bedingungen spezifizieren, auf zwei verschiedenen Bit- Positionen in einem Byte definiert sind, lassen sich diese beiden Flags häufig mit einer OR- Verknüpfung wie folgt zusammenfassen:

```
Flag1               00001000
Flag2               00100000
OR-Kombination      00101000
```

Auf diese Weise kombiniert man Flags, um eine Anzahl von Einstellungen in einem begrenzten Speicherbereich festzulegen. Praktisch passiert das gleiche mit den meisten Wahr/Falsch-(Ein/Aus-) Einstellungen in den Fenster- und Eigenschaftsdialogfeldern von Steuerelementen. Diese Ein-/Aus-Einstellungen werden mit OR verknüpft und bilden dann eine oder zwei Gruppen von Flags, aus denen Windows ablesen kann, wie das Fenster oder das Steuerelement anzuzeigen ist und wie es sich verhalten soll.

Um nun zu ermitteln, ob ein bestimmtes Flag in der Kombination enthalten ist, kann man die Flagkombination mit dem gesuchten Flag in einer AND-Verknüpfung folgendermaßen testen:

```
Kombination         00101000
Flag1               00001000
Ergebnis            00001000
```

Das Ergebnis dieser Operation lässt sich mit dem zum Filtern der Flagkombination verwendeten Flag vergleichen. Handelt es sich um dasselbe Ergebnis, ist das Flag in der Kombination enthalten. Bei einer anderen Lösung prüft man, ob das gefilterte Kombinationsflag ungleich Null ist. Wenn das zur Filterung - man spricht auch von Maskieren - verwendete Flag nicht in der Kombination enthalten ist, liefert das Ergebnisflag den Gesamtwert 0. Nach diesem Verfahren kann man sich den Vergleich in der if-Anweisung des obigen Codes sparen. Übrig bleibt eine if-Anweisung, die folgendermaßen aussieht:

```
if (nFlags & MK_LBUTTON)
```

Man kann diese Lösung auch modifizieren und testen, ob das interessierende Flag nicht in der Kombination enthalten ist:

```
if (!(nFlags & MK_LBUTTON))
```

Funktionell sind beide Versionen gleich. Vielleicht bevorzugen Sie die eine oder andere Version oder verwenden beide in Ihrem Code.

MFC-Exkurs: Der Gerätekontext

In Windows interagieren Sie niemals direkt mit dem Monitor oder anderen an den Computer angeschlossenen Geräten. Stattdessen interagieren Sie mit etwas, das Gerätekontext genannt wird. Dabei handelt es sich um eine Abstraktion der Benutzeranzeige oder anderer derzeit verwendeter Ausgabegeräte. So können Sie zum Zeichnen den gleichen Code verwenden, ob Sie nun auf dem Monitor zeichnen oder auf dem Drucker, wenn die Ausgabe Ihres Programms ausgedruckt wird.

Der Gerätekontext ist in der Klasse CDC verkapselt, die von der Klasse CClientDC in Listing 4.1 ererbt wird. Die Klasse CDC stellt Funktionalität für das Zeichnen von Formen und weitere grundlegende Zeichenfunktionalität, die Sie benötigen könnten, bereit. Die Klasse CClientDC bietet Funktionalität für einige vor dem Zeichnen notwendige Vorbereitungen und für das Aufräumen nach dem Zeichen.

Sie können nicht einfach eine Instanz der Klassen CDC oder CClientDC deklarieren und anfangen zu malen. Sie müssen da, wo Sie zeichnen wollen, den Gerätekontext abfragen. In Listing 4.1 wird dem Konstruktor der Klasse CClientDC als einziges Argument this übergeben. Die Variable this bezieht sich in diesem Fall auf das Dialogfenster. Sie übergeben also eigentlich einen Zeiger auf das Dialogfenster an den Konstruktor der Klasse CClientDC, wodurch die Instanz der Klasse CClientDC mit dem aktuellen Gerätekontext des Dialogfensters initialisiert wird. In den nächsten Tagen werden Sie noch viel mehr über den Gerätekontext und seine Verwendung zum Zeichnen verschiedener Figuren in Fenstern erfahren.

C++-Exkurs: Konstruktoren und Destruktoren

In jeder C++-Klasse gibt es zwei Methoden, die immer Mitglieder der Klasse sind. Dabei handelt es sich um die Methoden constructor und destructor. Die Methode constructor hat immer denselben Namen wie die Klasse (CMyClass), während die Methode destructor immer mit dem Klassennamen mit einer vorangestellten Tilde benannt ist (~CMyClass). Der constructor wird automatisch beim Erzeugen einer Instanz der Klasse aufgerufen und der destructor beim Zerstören dieser Instanz. Sie müssen keine der beiden Methoden erzeugen, wenn Sie sie nicht benötigen, da der Compiler Standardmethoden liefert, wenn Sie eine oder beide davon weglassen.

Sie können für eine Klasse mehrere Konstruktoren erstellen, von denen jeder einen anderen Satz Parameter übernimmt. Es gibt zwei Methoden, diese Parameter an den Konstruktor zu übergeben, abhängig davon, ob Sie eine Instanz in der Variablendeklaration erzeugen oder eine Instanz, die einem Zeiger zugewiesen ist. Wenn Sie eine Instanz in der Variablendeklaration erzeugen, übergeben Sie die Parameter wie folgt in der Deklaration an die Variable:

```
CMyClass foo(param);
```

Wenn Sie eine neue Instanz einer Klasse erzeugen und in einem Zeiger speichern, übergeben Sie die Parameter nach der Deklaration des Variablentyps:

```
CMeineKlasse *pFoo;
pFoo = new CMyClass(param);
```

Sie können nie mehr als einen destructor haben und er hat niemals Parameter.

C++-Exkurs: Objekte dynamisch erzeugen und zerstören

Manchmal müssen Sie in Ihren Anwendungen Objekte dynamisch erzeugen und zerstören. Ja, Sie können sie zu diesem Zweck als Variablen deklarieren, doch in vielen Situationen brauchen Sie etwas dynamischeres. In diesen Fällen benutzen Sie die Schlüsselworte new und delete.

Das Schlüsselwort new wird in C++ verwendet, um eine Instanz eines Datentyps oder einer Klasse zu erzeugen. Das Schlüsselwort delete wird verwendet, um mit dem Schlüsselwort new erzeugte Variablen oder Objekte zu löschen. Sie sind selbst dafür verantwortlich, allen Objekten und Variablen auf der Spur zu

bleiben, die Sie mit dem Schlüsselwort new erzeugen, und sie mit delete wieder zu zerstören. Wenn Sie nicht alle davon wieder zerstören, haben Sie am Ende eine Anwendung mit Speicherleck (Memory Leak). Möglicherweise bemerken Sie das Leck während der Entwicklung und des Testens gar nicht, doch wenn die Anwendung lange genug läuft, braucht sie zuletzt den gesamten auf dem System verfügbaren Speicher auf und lässt alle Anwendungen einfrieren.

Die Syntax des Schlüsselworts new ist das Schlüsselwort, gefolgt von dem Datentyp oder der Klasse, den oder die Sie erzeugen möchten. Der Rückgabewert ist ein Zeiger auf den erzeugten Datentyp oder die Klasse. Um ein Integer zu erzeugen, wird new beispielsweise folgendermaßen verwendet:

```
int *pInt;
pInt = new int;
```

Analog würde man es zur Erzeugung einer Klasse so verwenden:

```
CMyClass *pMyClass;
pMyClass = new CMyClass;
```

Wenn der Konstruktor der Klasse Parameter benötigt, müssen Sie diese wie folgt übergeben:

```
pMyClass = new CMyClass(param1, param2, ...);
```

Wenn Sie die Klammern nach dem Klassennamen weglassen, wird der Standard- Konstruktor aufgerufen.

Die Syntax für den delete-Operator ist ebenso einfach. Sie setzen hinter das Schlüsselwort einen Zeiger, der auf die Variable oder Klasseninstanz verweist, die Sie löschen möchten:

```
delete pInt;
delete pMyClass;
```

Wenn Sie für einen Zeiger wie folgt ein Array allozieren,

```
int *pInt = new int[8];
```

müssen Sie mit eckigen Klammern zwischen dem delete-Operator und dem Zeiger den allozierten Speicher löschen:

```
delete [] pInt;
```

Wenn Sie die eckigen Klammern weglassen, wird nicht der gesamte allozierte Speicher freigegeben.

Nachdem Sie eine Variable oder Klasseninstanz gelöscht haben, versuchen Sie keinesfalls, noch einmal darauf zuzugreifen. Der Versuch würde eine allgemeine Schutzverletzung in Ihrer Anwendung verursachen. Ihre Anwendung würde sich sofort beenden, alle Arbeit, die der Benutzer bisher geleistet hätte, würde verloren gehen.

Es wird als gute Programmierpraxis angesehen, Zeiger mit NULL zu initialisieren, sobald die Variable erzeugt wird. Wenn dem Zeiger zugewiesene Objekte gelöscht werden, sollte der Zeiger ebenfalls wieder auf NULL zurückgesetzt werden. Vor jeder Verwendung des Zeigers

sollten Sie dann mit einem einfachen

```
if (pointer)   // Wahr? Dann ist der Zeiger gültig
```

oder

```
if (!pointer)  // Wahr? Dann ist der Zeiger NICHT gültig
```

überprüfen, ob er gültig ist. Dieser einfache Test kann sehr viel dazu beitragen, Ihre Anwendung vor allgemeinen Schutzverletzungen zu bewahren.

4.2 Das Zeichenprogramm verbessern

Bei der Ausführung Ihres Programms haben Sie vielleicht ein kleines Problem bemerkt. Um eine durchgehende Linie zu zeichnen, mussten Sie die Maus sehr langsam bewegen. Wie lösen andere Zeichenprogramme dieses Problem? Sie zeichnen einfach eine Linie zwischen die beiden Punkte, die mit der Maus gesetzt werden. Auch wenn das ein wenig nach Mogelei aussieht, arbeiten die gängigen Zeichenprogramme nach diesem Verfahren.

Während Sie die Maus über den Bildschirm verschieben, prüft der Computer die Position der Maus alle paar Taktzyklen. Da der Computer auf diese Weise nicht ständig darüber informiert ist, wo sich die Maus gerade befindet, muss er diesbezügliche Annahmen treffen. Das geschieht dadurch, dass der Computer die bekannten Punkte verwendet und Linien zwischen ihnen zieht. Wenn Sie mit dem Freihandwerkzeug in Paint Linien zeichnen, übernimmt der Computer die Verbindung der einzelnen Punkte.

Wir wissen nun, dass die üblichen Zeichenprogramme Linien zwischen jedem Punktepaar ziehen. Was brauchen wir, um unsere Zeichenanwendung mit dieser Technik auszustatten? Zuerst einmal muss man die vorherige Position der Maus festhalten. Das bedeutet, dass man dem Dialogfeld zwei Variablen spendieren muss, um die vorherigen x- und y-Koordinaten zu speichern. Das lässt sich in folgenden Schritten realisieren:

1. Wählen Sie die Klassenansicht.

2. Markieren Sie die Dialogfeldklasse - in diesem Fall CMouseDlg.
3. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.
4. Geben Sie als Variablentyp int und als Variablenname m_iPrevY ein und legen Sie den Zugriff mit private fest, wie es Abbildung 4.3 zeigt.

Abbildung 4.3: Das Dialogfeld Assistent zum Hinzufügen von Membervariablen

5. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.
6. Wiederholen Sie die Schritte 3 bis 5, wobei Sie als Variablenname m_iPrevX für die zweite Variable eingeben.

Nachdem Sie die Variablen zum Speichern der vorherigen Mausposition hinzugefügt haben, können Sie die notwendigen Änderungen an der Funktion OnMouseMove gemäß Listing 4.2 vornehmen.

Listing 4.2: Die überarbeitete Funktion OnMouseMove

```
void CMouseDlg::OnMouseMove(UINT nFlags, CPoint point)
{
  // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
   // und/oder benutzen Sie den Standard.
  // Linke Maustaste gedrückt?
  if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
  {
    // Gerätekontext holen
    CClientDC dc(this);
    // Linie vom letzten zum aktuellen Punkt zeichnen
    dc.MoveTo(m_iPrevX, m_iPrevY);
    dc.LineTo(point.x, point.y);
    // Aktuellen Punkt als letzten Punkt speichern
    m_iPrevX = point.x;
    m_iPrevY = point.y;
  }
  CDialog::OnMouseMove(nFlags, point);
}
```

Der Code zum Zeichnen der Linie vom vorherigen Punkt zum aktuellen Punkt sieht folgendermaßen aus:

```
dc.MoveTo(m_iPrevX, m_iPrevY);
dc.LineTo(point.x, point.y);
```

Es ist also zunächst eine Bewegung zur ersten Position auszuführen (MoveTo) und dann eine Linie zum zweiten Punkt zu ziehen (LineTo). Der erste Schritt ist deshalb wichtig, da man sonst Windows nicht mitteilen könnte, wo der Startpunkt liegt. Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, lässt es sich etwas besser zeichnen. Allerdings tritt nun ein eigentümliches Verhalten zutage. Sobald Sie die linke Maustaste drücken, um etwas mehr zu zeichnen, zieht die Anwendung zu diesem Punkt eine Linie vom Endpunkt der letzten gezeichneten Linie, wie es Abbildung 4.4 verdeutlicht.

Sie könnten zwei Integer-Variablen durch eine einzelne CPoint-Variable ersetzen. Damit könnten Sie den Code vereinfachen, indem Sie die Variable wie folgt setzen:

```
m_pPrevPoint = point;
```
und die Zeichnung folgendermaßen ausführen:

```
dc.MoveTo(m_pPrevPoint);
dc.LineTo(point);
```

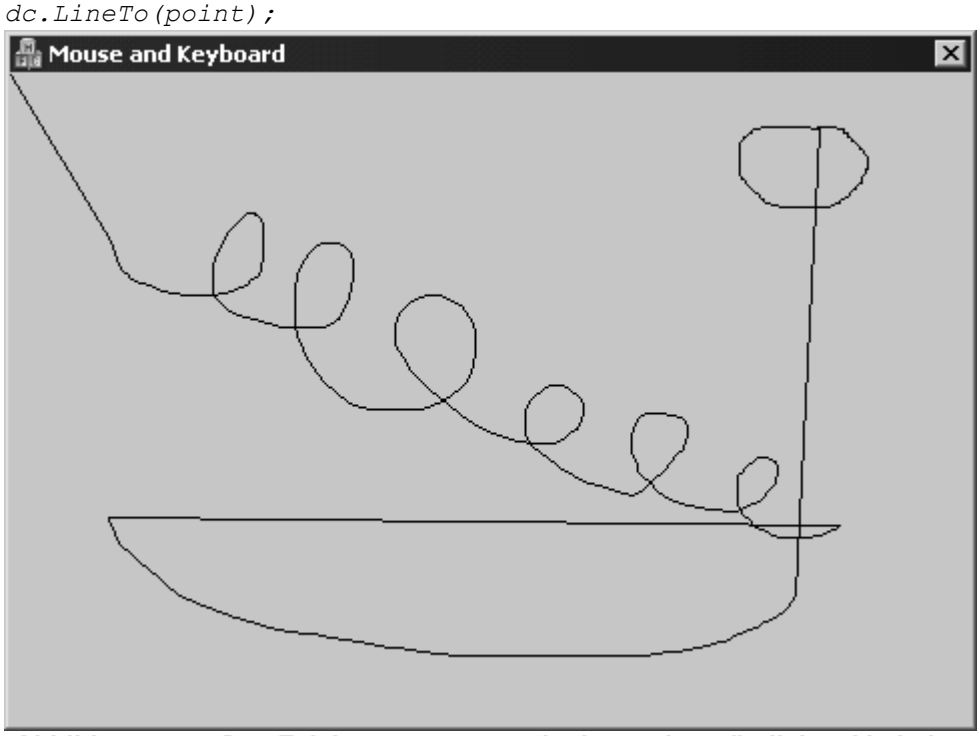


Abbildung 4.4: Das Zeichenprogramm mit einem eigentümlichen Verhalten

Letzte Anpassungen

Die Anwendung realisiert sämtliche Zeichenfunktionen auf das Ereignis »Verschieben der Maus«, wenn die linke Maustaste gedrückt ist. Initialisiert man die Variablen für die vorherige Position der Maus mit den Koordinaten, an der sich die Maus gerade befindet, wenn der Benutzer die linke Taste drückt, sollte sich das Verhalten der Anwendung korrigieren lassen. Probieren Sie die Lösung mit den folgenden Schritten aus:

1. Nehmen Sie mithilfe des Modus Meldungen in der Eigenschaftenansicht eine Funktion für die Nachricht WM_LBUTTONDOWN in das Dialogfeldobjekt auf. Die Dialognachrichten sind nur verfügbar, wenn der Dialog sich im Ressourcen-Editor befindet und sowohl ausgewählt als auch aktiv ist.

2. Fügen Sie in die eben erstellte Funktion OnLButtonDown den Code von Listing 4.3 ein.

Listing 4.3: Die Funktion OnLButtonDown

```
1: void CMouseDlg::OnLButtonDown(UINT nFlags, CPoint point)
```

```
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:   // und/oder benutzen Sie den Standard.
 5:
 5:   // Aktuellen Punkt als Anfangspunkt setzen
 7:   m_iPrevX = point.x;
 8:   m_iPrevY = point.y;
 9:
10:   CDialog::OnLButtonDown(nFlags, point);
11: }
```

Wenn Sie die Anwendung kompilieren und ausführen, sollten Sie schon fast so zeichnen können, wie man es von einem Zeichenprogramm erwartet (siehe Abbildung 4.5).

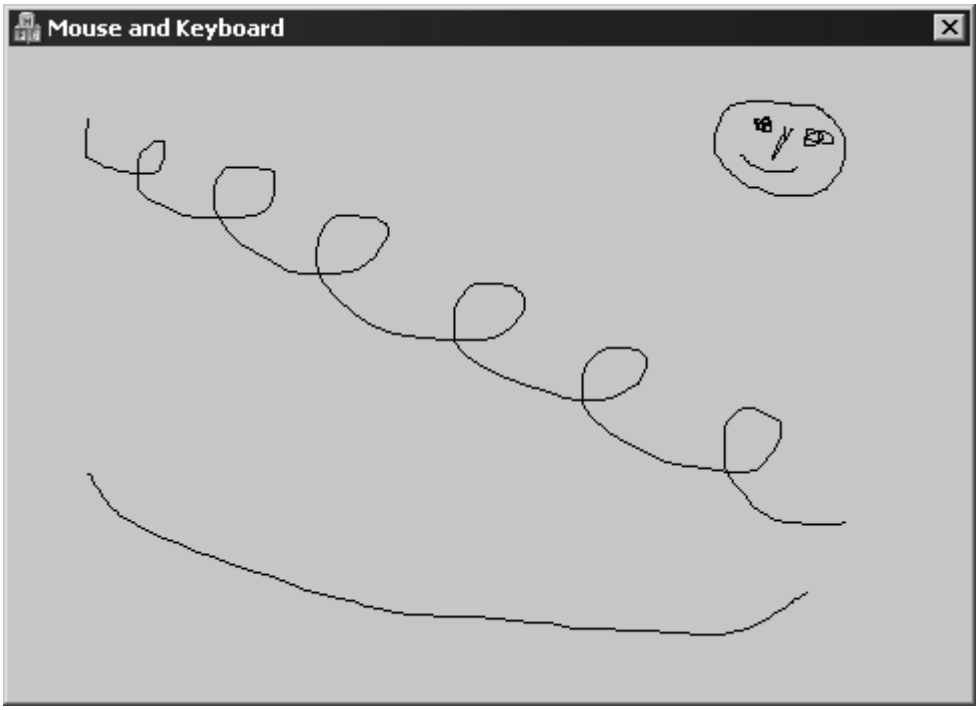


Abbildung 4.5: Das fertig gestellte Zeichenprogramm

4.3 Tastaturereignisse auffangen

Tastaturereignisse lassen sich fast genauso lesen wie Mausereignisse. Analog zur Maus werden Nachrichten verschickt, wenn man eine Taste drückt und wenn man sie wieder loslässt. Diese Ereignisse sind in Tabelle 4.3 aufgeführt.

Nachricht	Beschreibung
WM_KEYDOWN	Eine Taste wurde gedrückt.
WM_KEYUP	Eine Taste wurde losgelassen.
WM_SYSKEYDOWN	(F10) oder (Alt) wurde zusammen mit einer anderen Taste gedrückt.
WM_SYSKEYUP	Eine Tastenkombination mit (F10) oder (Alt) wurde losgelassen.

Tabelle 4.3: Nachrichten für Tastaturereignisse

Offensichtlich gibt es für die Tastatur weniger Nachrichten als für die Maus und man kann mit der Tastatur aus programmtechnischer Sicht dementsprechend weniger anfangen. Die Nachrichten sind im Dialogfeldobjekt verfügbar und werden nur ausgelöst, wenn keine aktivierten Steuerelemente im Fenster

vorhanden sind. Alle aktivierten Steuerelemente im Fenster haben den Eingabefokus, sodass alle Tastaturereignisse zu ihnen gelangen. Aus diesem Grund sollten Sie für die Zeichenanwendung alle Steuerelemente aus dem Hauptdialogfeld entfernen.

Den Mauszeiger zum Zeichnen ändern

Um eine Vorstellung davon zu bekommen, wie man Nachrichten in Bezug auf Tastaturereignisse nutzen kann, legen wir bestimmte Tasten fest, um den Mauszeiger in der Zeichenanwendung zu verändern. Die Taste (S) ändert den Cursor in den Standardpfeil, mit dem die Anwendung auch startet. Die Taste (B) verwenden wir, um den Mauszeiger in den I-Balken-Mauszeiger zu verwandeln, während die Taste (U) für den Sanduhrzeiger vorgesehen ist. Die dementsprechende Funktionalität realisieren Sie in folgenden Schritten:

1. Nehmen Sie mithilfe des Modus Meldungen in der Eigenschaftenansicht eine Funktion für die Nachricht WM_KEYDOWN in das Dialogfeldobjekt auf. Die Dialognachrichten sind nur verfügbar, wenn sich der Dialog im Ressourcen-Editor befindet und sowohl ausgewählt als auch aktiv ist.

2. In die eben erstellte Funktion OnKeyDown fügen Sie den Code gemäß Listing 4.4 ein.

Listing 4.4: Die Funktion OnKeyDown

```
 1: void CMouseDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:   // und/oder benutzen Sie den Standard.
 5:
 6:   char cChar;                 // Zeichen der gedrückten Taste
 7:   HCURSOR hCursor = NULL;     // Handle zum anzuzeigenden Cursor
 8:   HCURSOR hPrevCursor = NULL; // Handle zum letzten Cursor
 9:
10:   // Code der gedrückten Taste in Zeichen umwandeln
11:   cChar = char(nChar);
12:
13:   // Ist Zeichen ein "S"?
14:   if (cChar == 'S')
15:   {
16:     // Pfeilcursor laden
17:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
18:     // Bildschirmcursor setzen
19:     hPrevCursor = SetCursor(hCursor);
20:   }
21:
22:   // Ist Zeichen ein "B"?
23:   if (cChar == 'B')
24:   {
25:     // Balkencursor laden
26:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
27:     // Bildschirmcursor setzen
28:     hPrevCursor = SetCursor(hCursor);
29:   }
30:
31:   // Ist Zeichen ein "U"?
32:   if (cChar == 'U')
33:   {
34:     // Sanduhrcursor laden
35:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
36:     // Set the screen cursor
37:     hPrevCursor = SetCursor(hCursor);
38:   }
39:
40:     // Letzten Cursor zerstören, um Ressourcen freizugeben
41:     if (hPrevCursor)
42:       DestroyCursor(hPrevCursor);
```

```
43:
44:     // Ist Zeichen ein "X"?
45:     if (cChar == 'X')
46:     {
47:       // Pfeilcursor laden
48:       hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
49:       // Bildschirmcursor setzen
50:      hPrevCursor = SetCursor(hCursor);
51:       // Letzen Cursor zerstören, um Ressourcen freizugeben
52:       if (hPrevCursor)
53:          DestroyCursor(hPrevCursor);
54:       // Anwendung beenden
55:       OnOK();
56:   }
57:
58:   CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
59: }
```

Die Funktionsdefinition zeigt, dass die Funktion OnKeyDown drei Argumente übernimmt. Das erste, nChar, gibt die gedrückte Taste an. Es handelt sich hierbei um den Zeichencode, der in der ersten Code-Zeile in das Zeichen umgewandelt wird. Nach der Umwandlung des Zeichens kann man direkte Vergleichsoperationen ausführen, um die gedrückte Taste zu bestimmen:

```
void CMausDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
```

Das zweite Argument an die Funktion OnKeyDown, nRepCnt, liefert die Anzahl, wie oft die Taste gedrückt wurde. Normalerweise ist dieser Wert gleich 1, wenn man die Taste drückt und wieder loslässt. Hält man die Taste dagegen nieder, tritt ein Wiederholungszähler für diese Taste in Kraft. Der entsprechende Wert sagt also aus, wie oft Windows in diesem Fall ein wiederholtes Drücken der Taste annimmt.

Das dritte Argument an die Funktion OnKeyDown, nFlags, ist ein Kombinationsflag, aus dem sich ermitteln lässt, ob die (Alt)-Taste gleichzeitig mit einer anderen Taste gedrückt wurde oder ob es sich bei der gedrückten Taste um eine erweiterte Taste handelt. Über die Tasten (Shift) oder (Strg) gibt dieses Argument keine Auskunft.

Sie müssen nicht sowohl auf groß als auch auf klein geschriebene Zeichen prüfen, da die Ereignisnachricht WM_KEYDOWN nur den Code für das groß geschriebene Zeichen liefert. Die Umschalt- und Feststelltasten werden als separate Zeichencodes übergeben. Wenn Sie wissen möchten, ob ein Zeichen groß oder klein geschrieben wurde, müssen Sie verfolgen, wann die Umschalt- und Feststelltasten gedrückt und losgelassen wurden.

Sobald Sie ermittelt haben, dass eine bestimmte Taste gedrückt wurde, ändern Sie den Cursor in die Form, die für die jeweilige Taste vorgesehen ist. Dieser Vorgang besteht aus zwei Teilen. Im ersten Schritt wird der Cursor in den Speicher geladen. Das realisieren Sie mit der Funktion LoadStandardCursor. Diese Funktion lädt einen Standardcursor von Windows und gibt einen Handle auf den Cursor zurück.

Die verwandte Funktion LoadCursor übernimmt den Datei- oder Ressourcennamen eines benutzerdefinierten Cursors, sodass man eigene Cursor erzeugen und laden kann. Wenn Sie mit dem Ressourcen-Editor von Visual C++ einen Cursor entwerfen, können Sie den Cursornamen als einziges Argument an die Funktion LoadCursor übergeben. Haben Sie zum Beispiel einen Cursor unter dem Namen IDC_MYCURSOR erzeugt, können Sie ihn mit der

folgenden Code-Zeile laden:

```
lhCursor = AfxGetApp()->LoadCursor(IDC_MYCURSOR);
```

Nachdem Sie diesen Cursor geladen haben, können Sie den Mauszeiger mit der Funktion SetCursor auf Ihren Cursor setzen, wie Sie es auch bei einem Standardcursor tun.

Nachdem die Funktion den Cursor in den Speicher geladen hat, wird der Handle auf diesen Cursor an die Funktion SetCursor übergeben. Damit nimmt der Mauszeiger die Form des Cursors an, auf den der Handle zeigt. SetCursor gibt den Handle an den letzten Cursor zurück. Sie können den letzten Cursor mit der Funktion DestroyCursor zerstören. Wenn Sie die Anwendung kompilieren und ausführen, können Sie eine der festgelegten Tasten drücken, um die Form des Cursors zu ändern, wie es Abbildung 4.6 zeigt. Sobald Sie aber die Maus verschieben, um etwas zu zeichnen, nimmt der Cursor wieder die Form des Standardpfeils an. Der folgende Abschnitt beschreibt, wie man die Änderung dauerhaft machen kann.

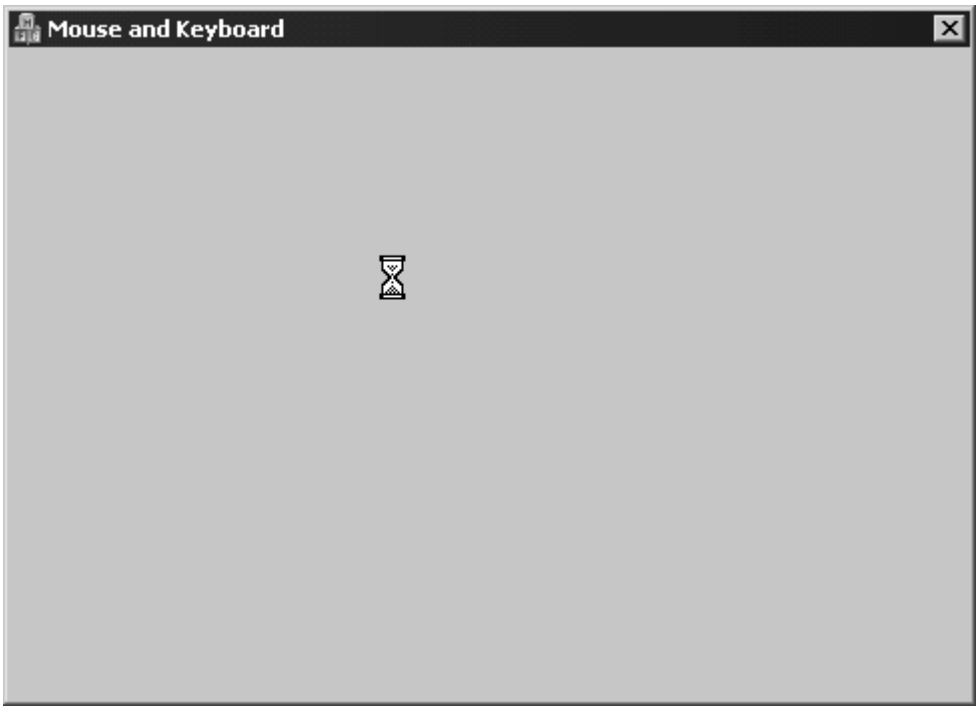


Abbildung 4.6: Die Form des Cursors lässt sich mit festgelegten Tasten ändern.

MFC-Exkurs: Die Funktion AfxGetApp

In Listing 4.4 sehen Sie, wie Sie die Funktion AfxGetApp zusammen mit der Funktion LoadStandardCursor aufrufen können:

```
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
```

Da diese Funktion einen Zeiger verwendet, um die Funktion LoadStandardCursor aufzurufen, kann man getrost annehmen, dass sie einen Zeiger auf irgendeine Klasse zurückgibt. AfxGetApp ist eine globale Funktion, welche die Instanz der Anwendungsklasse für die aktuelle Anwendung zurückgibt. Die Anwendungsklasse ist der Abkömmling der Klasse CWinApp in der aktuellen Anwendung. In unserer Anwendung ist es die Klasse CMouseApp. Wenn Sie auf in CWinApp oder in ihrem aktuellen Abkömmling verkapselte Funktionalität zugreifen möchten, können Sie mit AfxGetApp einen Zeiger darauf erhalten.

Wenn Sie in Ihrem Quellcode graben, finden Sie eine global deklarierte Variable für Ihre Anwendungsklasse:

```
CMouseApp theApp;
```

Trotz dieser im Code existierenden Variablendeklaration können Sie diese Variable nicht direkt verwenden

oder auf sie verweisen. Sie müssen die Funktion AfxGetApp verwenden, um das Anwendungsobjekt aufzurufen oder darauf zu verweisen.

C++-Exkurs: Verwendung von Anführungszeichen

In Listing 4.4 haben Sie folgenden Code verwendet, um die gedrückte Taste mit verschiedenen Zeichen zu vergleichen:

```
if (lsChar == 'A')
```

Es gibt hier etwas Wichtiges zu beachten: die Anführungszeichen, in denen das Zeichen steht. Hier werden einfache Anführungszeichen (') verwendet, weil Sie ein einzelnes Zeichen mit einem anderen einzelnen Zeichen vergleichen. Das ist kein String-, sondern ein Zeichen-Vergleich. Wäre das Zeichen in doppelte Anführungszeichen eingeschlossen ("), würde es der Compiler als String ansehen und in dieser Code-Zeile einen Fehler auslösen. Sie können ein einzelnes Zeichen nicht mit einem String vergleichen, nur mit anderen einzelnen Zeichen. Wenn ein einzelnes Zeichen in einfachen Anführungszeichen steht, wird es vom Compiler als einzelnes Zeichen erkannt und sein binärer Wert mit dem binären Wert der Zeichenvariable verglichen.

Änderung beibehalten

Dem Zeichenprogramm haftet momentan der Mangel an, dass sich der Cursor sofort wieder in den Standardpfeil zurückverwandelt, wenn man die Maus bewegt. Es muss eine Möglichkeit geben, dieses Verhalten zu unterdrücken.

Windows schickt immer eine WM_SETCURSOR-Nachricht an Ihre Anwendung, wenn der Cursor neu zu zeichnen ist - weil die Maus bewegt wurde, weil ein über der Anwendung liegendes Fenster geschlossen wurde oder aus welchen Gründen auch immer. Wenn Sie das native Verhalten Ihrer Anwendung für dieses Ereignis überschreiben, bleibt der von Ihnen festgelegte Cursor erhalten, bis Sie ihn ausdrücklich wieder ändern. Führen Sie dazu die folgenden Schritte aus:

1. Nehmen Sie in die Klasse CMouseDlg eine neue Variable auf, wie Sie es für die Variablen der vorherigen Position ausgeführt haben. Dieses Mal deklarieren Sie den Typ als BOOL, nennen die Variable m_bCursor und legen den Zugriff als private fest, wie es Abbildung 4.7 zeigt.

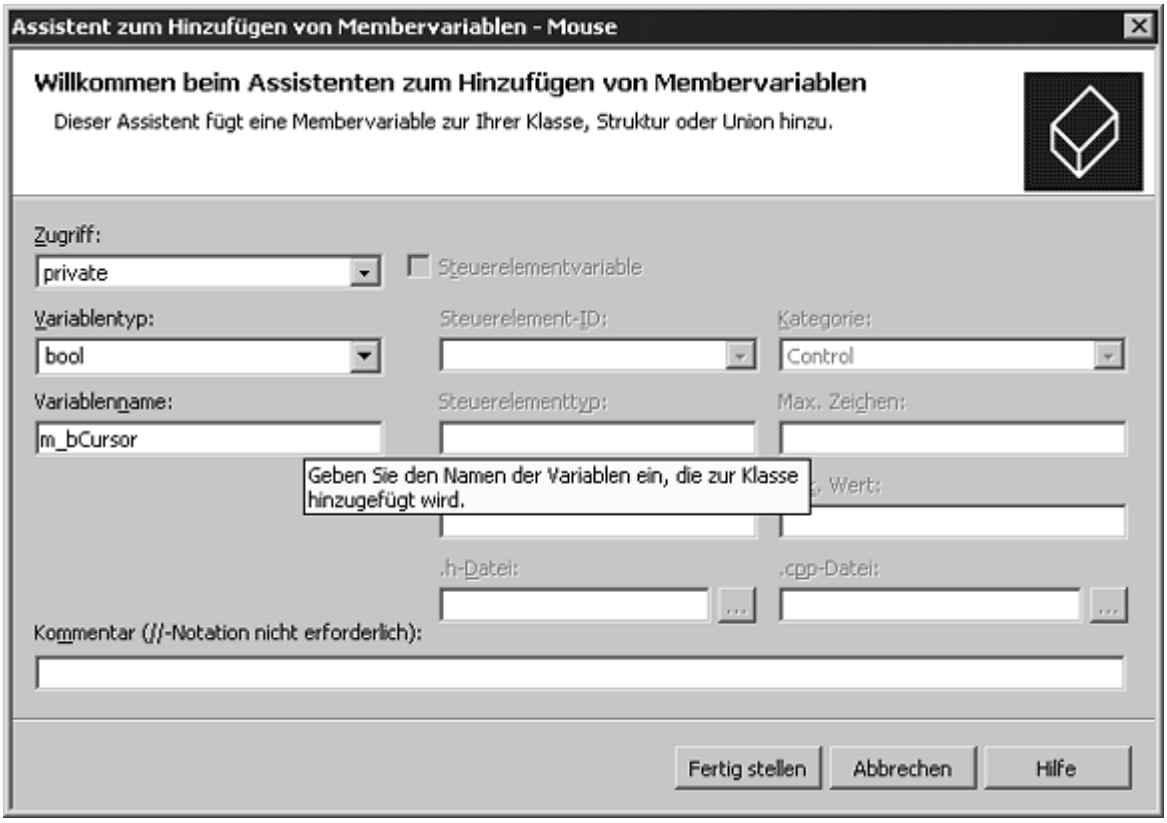


Abbildung 4.7: Eine Member-Variable definieren

2. Initialisieren Sie die Variable m_bCursor in der Funktion OnInitDialog mit dem Code in Listing 4.5.

Listing 4.5: Die Funktion OnInitDialog

```
 1: BOOL CMouseDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
 5:   // Hinzufügen des Menübefehls "Info..." zum Systemmenü.
 6:
 7:   // IDM_ABOUTBOX muss sich im Bereich der Systembefehle
 8:   // befinden.
 9:   ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
10:   ASSERT(IDM_ABOUTBOX < 0xF000);
11:
...
29:
30:   // TODO: Hier zusätzliche Initialisierung einfügen
31:   // Cursor als Pfeil initialisieren
32:   m_bCursor = FALSE;
33:
34:   return TRUE;  // Geben Sie TRUE zurück, außer ein
35:                 // Steuerelement soll den Fokus erhalten
36: }
```

3. Ändern Sie die Funktion OnKeyDown gemäß Listing 4.6, um das Flag m_bCursor auf TRUE zu setzen, wenn Sie den Cursor wechseln.

Listing 4.6: Die Funktion OnKeyDown

```
 1: void CMouseDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
 2:  {
 3:    // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:    // und/oder benutzen Sie den Standard.
 5:
 6:    char cChar;              // Zeichen der gedrückten Taste
 7:    HCURSOR hCursor = NULL;  // Handle zum anzuzeigenden Cursor
 8:    HCURSOR hPrevCursor = NULL; // Handle zum letzten Cursor
 9:
10:    // Code der gedrückten Taste in Zeichen umwandeln
11:    cChar = char(nChar);
12:
13:    // Ist Zeichen ein "S"?
14:    if (cChar == 'S')
15:      // Pfeilcursor laden
16:      hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
17:
18:    // Ist Zeichen ein "B"?
19:    if (cChar == 'B')
20:      // Balkencursor laden
21:      hCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
22:
23:    // Ist Zeichen ein "U"?
24:    if (cChar == 'U')
25:      // Sanduhrcursor laden
26:      hCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
27:
28:      // Ist Zeichen ein "X"?
29:    if (cChar == 'X')
30:    {
31:      // Pfeilcursor laden
```

```
32:       hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
33:       // Cursorflag setzen
34:       m_bCursor = TRUE;
35:       // Bildschirmcursor setzen
36:       hPrevCursor = SetCursor(hCursor);
37:       // Letzten Cursor zerstören, um Ressourcen freizugeben
38:       if (hPrevCursor)
39:          DestroyCursor(hPrevCursor);
40:       // Anwendung beenden
41:       OnOK();
42:     }
43:     else
44:     {
45:       // Bildschirmcursor setzen
46:       if (hCursor)
47:       {
48:         hPrevCursor = SetCursor(hCursor);
49:         // Cursorflag setzen
50:         m_bCursor = TRUE;
51:         // Letzen Cursor zerstören, um Ressourcen freizugeben
52:         if (hPrevCursor)
53:           DestroyCursor(hPrevCursor);
54:       }
55:     }
56:
57:     CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
58:   }
```

4. Fügen Sie mithilfe des Modus Meldungen in der Eigenschaftenansicht eine Funktion für die Nachricht WM_SETCURSOR in das Dialogfeldobjekt ein. Die Dialognachrichten sind nur verfügbar, wenn sich der Dialog im Ressourcen-Editor befindet und sowohl ausgewählt als auch aktiv ist.
5. Nehmen Sie den Code aus Listing 4.7 in die eben erstellte Funktion OnSetCursor auf.

Listing 4.7: Die Funktion OnSetCursor

```
 1: BOOL CMouseDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest,
 2:                             UINT message)
 3: {
 4:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 5:   // und/oder benutzen Sie den Standard.
 6:
 7:   if (m_bCursor)
 8:     // TRUE zurückgeben
 9:     return TRUE;
10:   return CDialog::OnSetCursor(pWnd, nHitTest, message);
11: }
```

Die Funktion OnSetCursor muss immer TRUE zurückgeben oder ansonsten die Funktion der Basisklasse aufrufen. Die Funktion der Basisklasse setzt den Cursor zurück und muss aufgerufen werden, wenn die Anwendung das erste Mal startet. Aus diesem Grund müssen Sie die Variable mit FALSE initialisieren, damit die Standardverarbeitung von OnSetCursor abläuft, bis der Benutzer eine Taste drückt, um den Cursor zu wechseln. Nachdem der Benutzer den Cursor geändert hat, übergehen Sie die Standardverarbeitung und liefern stattdessen den Wert TRUE zurück. Damit kann der Benutzer mit dem Cursor, den er ausgewählt hat, zeichnen. Dazu gehört auch der Sanduhrzeiger, wie es Abbildung 4.8 zeigt.

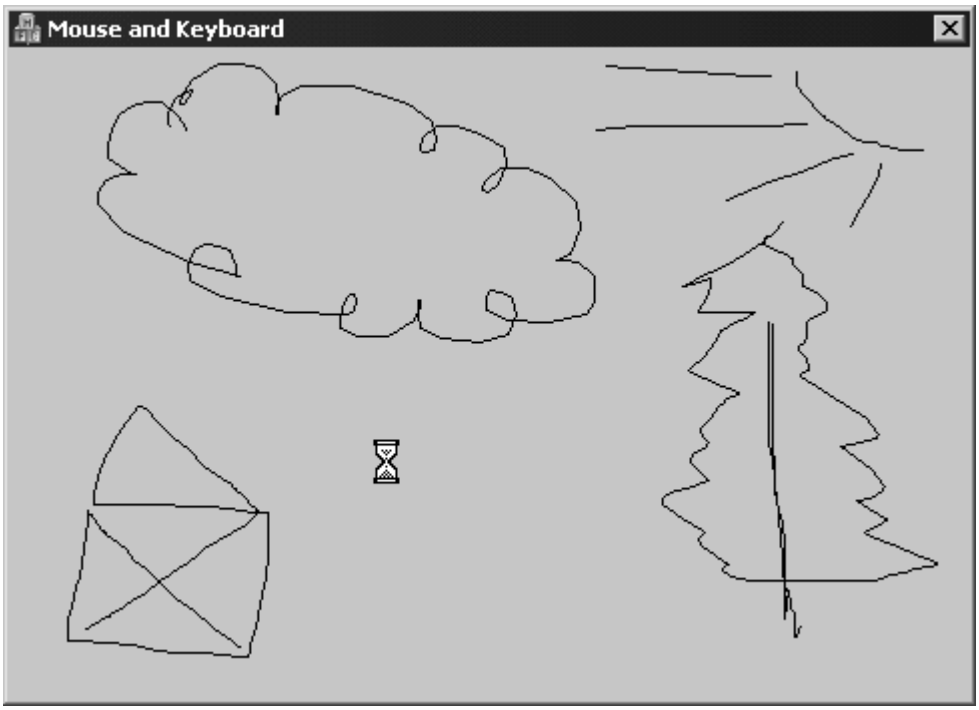


Abbildung 4.8: Mit dem Sanduhrzeiger zeichnen

Am häufigsten ändert man in einer Anwendung die Form des Mauszeigers in eine Sanduhr, um darauf hinzuweisen, dass das Programm eine länger dauernde Berechnung erledigt. In MFC sind für diese Aufgabe eigentlich zwei Funktionen verfügbar. Die erste heißt BeginWaitCursor. Diese Funktion zeigt den Sanduhrzeiger für den Benutzer an. Die zweite Funktion, EndWaitCursor, stellt den Standardzeiger wieder her. Beide Funktionen sind Elemente der Klasse CCmdTarget, von der alle MFC-Fenster- und Steuerelementklassen abgeleitet sind.

Haben Sie in einer einzigen Funktion alle Aufgaben zusammengefasst, für die Sie den Sanduhrzeiger anzeigen müssen, und muss dieser nach Abschluss dieser Funktion nicht mehr angezeigt werden, deklarieren Sie eine Variable der Klasse CWaitCursor am Beginn der Funktion. Daraufhin zeigt Windows automatisch den Sanduhrzeiger an. Sobald das Programm die Funktion verlässt, wird der Cursor wieder in den vorherigen zurückverwandelt.

4.4 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie man Nachrichten von Mausereignissen auffängt und basierend auf diesen Ereignissen einfache Verarbeitungen durchführt. Mithilfe von Mausereignissen haben Sie ein einfaches Zeichenprogramm erstellt, mit dem sich Freihandfiguren im einem Dialogfeld zeichnen lassen.

Weiterhin wurde gezeigt, wie man Tastaturereignisse auffängt und die gedrückte Taste ermittelt. Anhand dieser Information haben Sie festgelegt, welcher Cursor zum Zeichnen zu verwenden ist. In diesem Zusammenhang haben Sie sich mit dem standardmäßigen Zeichnen des Cursors in MFC-Anwendungen beschäftigt und gelernt, wie man dieses Verhalten in den eigenen Code integriert, damit sich die Anwendung in der gewünschten Weise verhält.

Aufbauend auf diesen Kenntnissen erfahren Sie in der nächsten Lektion, wie man den Windows-Timer einsetzt, um Ereignisse in regelmäßigen Abständen auszulösen. Sie werden auch lernen, wie sich mit zusätzlichen Dialogfeldern Rückmeldungen vom Benutzer einholen lassen, sodass man auf der Basis dieser Eingaben das Verhalten der Anwendung steuern kann. Danach werden Sie lernen, wie man Menüs für eine Anwendung erstellt.

4.5 Workshop

Fragen und Antworten

Frage:
Wie kann ich die Art der zu zeichnenden Linie ändern? Ich möchte eine dickere Linie mit einer anderen Farbe zeichnen.

Antwort:
Wenn man mit einem der Standardbefehle für Gerätekontexte auf den Bildschirm zeichnet, kommt ein so genannter Stift zum Einsatz, genau wie wenn man mit einem Stift auf Papier zeichnet. Um dickere Linien oder andere Farben darzustellen, muss man einen neuen Stift auswählen. Das lässt sich realisieren, indem man den Code in der Funktion OnMouseMove an der Stelle anpasst, wo der Gerätekontext geholt wird. Der folgende Code bewirkt, dass die Linie mit einem dicken roten Stift gezeichnet wird:

```
// Gerätekontext holen
CClientDC dc(this);
// Einen neuen Stift erzeugen
CPen lpen(PS_SOLID, 16, RGB(255, 0, 0));
// Den neuen Stift verwenden
dc.SelectObject(&lpen);
// Eine Linie vom letzten zum aktuellen Punkt zeichnen
dc.MoveTo(m_iPrevX, m_iPrevY);
dc.LineTo(point.x, point.y);
```

Frage:
Wie kann man ermitteln, ob die Tasten (Shift) oder (Strg) gedrückt sind, wenn man die Nachricht WM_KEYDOWN empfängt?

Antwort:
Zu diesem Zweck ruft man die Funktion ::GetKeyState mit einem bestimmten Tastencode auf, um die gedrückten Tasten zu ermitteln. Wenn der Rückgabewert der Funktion ::GetKeyState negativ ist, wird die Taste niedergehalten. Bei einem nicht negativen Rückgabewert ist die Taste nicht gedrückt. Zum Beispiel können Sie mit dem folgenden Code ermitteln, ob die (Shift)-Taste gedrückt ist:

```
if (::GetKeyState(VK_SHIFT) < 0)
   MessageBox("Umschalt-Taste gedrückt!");
```

Antwort:
In Windows sind für alle Sondertasten spezielle virtuelle Tastencodes definiert. Mit diesen Codes kann man auf spezielle Tasten prüfen, ohne sich um die OEM- Scancodes oder andere Tastenfolgen kümmern zu müssen. Die virtuellen Tastencodes kann man in der Funktion ::GetKeyState verwenden und sie an die Funktion OnKeyDown als Argument nChar übergeben. Eine Liste der virtuellen Tastencodes finden Sie in der Dokumentation zu Visual C++.

Quiz

1. Für welche Mausnachrichten kann man Funktionen hinzufügen?

2. Wie kann man ermitteln, ob bei der Nachricht WM_MOUSEMOVE die linke Maustaste gedrückt ist?
3. Wie kann man verhindern, dass sich der Mauszeiger in den Standardzeiger zurückverwandelt, nachdem man einen anderen Zeiger festgelegt hat?

Übungen

1. Modifizieren Sie das Zeichenprogramm, so dass beim Drücken der linken Maustaste in Rot, definiert als RGB(255, 0, 0), und bei der rechten Maustaste in Blau, definiert als RGB(0, 0, 255), gezeichnet wird.

2. Erweitern Sie die Funktion OnKeyDown in Listing 4.4, um einige der folgenden Standardcursors aufzunehmen:

- IDC_CROSS
- IDC_UPARROW
- IDC_SIZEALL
- IDC_SIZENWSE
- IDC_SIZENESW
- IDC_SIZEWE
- IDC_SIZENS
- IDC_NO
- IDC_APPSTARTING
- IDC_HELP

Tag 5

Timer

Oftmals ist eine Anwendung zu erstellen, die eine bestimmte Aktion in regelmäßigen Abständen wiederholen soll - beispielsweise alle 30 Minuten nach E-Mails sehen oder eine Sicherungsdatei alle fünf Minuten speichern. Derartige Aktionen finden Sie in vielen bekannten Anwendungen, mit denen Sie täglich zu tun haben. Zu den wiederkehrend auszuführenden Aufgaben gehören auch ständige Prüfungen auf Ressourcen, wie man sie von Ressourcen- oder Performancemonitoren her kennt. Diese Beispiele zeigen nur einen Ausschnitt der Situationen, in denen man auf die Timer - oder Zeitgeber - des Betriebssystems Windows zurückgreift.

Heute lernen Sie, wie ...

- man Timer in Visual C++-Anwendungen steuert und einsetzt,
- sich mehrere Timer mit unterschiedlichen Zeitintervallen einrichten lassen,
- man ermittelt, welcher Timer ausgelöst wurde,
- Sie diese wichtige Ressource in alle Ihre Visual C++-Anwendungen einbinden können.

5.1 Funktionsweise von Windows-Timern

> *Windows-Timer stellen einen Mechanismus bereit, über den man einen oder mehrere Timer mit einer bestimmten Anzahl von Millisekunden auslösen kann. Wenn man einen Timer für ein Intervall von 1.000 Millisekunden einrichtet, wird er jede Sekunde ausgelöst. Beim Auslösen eines Timers schickt Windows eine WM_TIMER-Nachricht an die Anwendung. Mit dem Klassen-Assistenten können Sie eine Funktion in Ihre Anwendung einbauen, um diese Timer-Nachricht zu behandeln.*

Timer-Ereignisse werden nur dann in die Nachrichtenwarteschlange gestellt, wenn die Warteschlange leer ist und die Anwendung im Leerlauf arbeitet. Sollte die Anwendung beschäftigt sein, stellt Windows keine Timer-Nachrichten in die Nachrichtenwarteschlange. Sollte Ihre Anwendung in diesem Fall mehrere Timer-Nachrichten verpasst haben, stellt Windows nur eine einzige in die Nachrichtenwarteschlange und sendet Ihrer Anwendung nicht alle, die während der Laufzeit der Anwendung aufgetreten sind. Es spielt keine Rolle, wie viele Timer- Nachrichten Ihre Anwendung verpasst hat - Windows stellt immer nur eine einzige in Ihre Warteschlange.

Beim Starten oder Anhalten eines Timers geben Sie einen ganzzahligen Wert als Timer- ID an. Die Anwendung kann anhand der Timer-ID ermitteln, welcher Timer ausgelöst hat, außerdem kann er Timer starten und stoppen. Eine bessere Vorstellung von dieser Arbeitsweise bekommen Sie, wenn Sie sich mit der heute zu erstellenden Anwendung beschäftigen.

5.2 Die Anwendung mit einer Uhr ausstatten

In der heutigen Beispielanwendung kommen zwei Timer zum Einsatz. Der erste Timer verwaltet eine Uhr im Fenster. Dieser Timer läuft immer, solange die Anwendung läuft. Der zweite Timer lässt sich über das Dialogfeld vom Benutzer für beliebige Intervalle konfigurieren. Der Benutzer kann den Timer je nach Bedarf starten und stoppen. Gehen wir also an die Arbeit.

Das Projekt und die Anwendung erstellen

Die heutige Beispielanwendung erstellen Sie in drei Phasen:

1. Nehmen Sie die Steuerelemente auf, die für die gesamte Anwendung erforderlich sind.

2. Fügen Sie den ersten der beiden Timer hinzu. Dieser Timer steuert die Uhr im Dialogfeld der Anwendung.
3. Fügen Sie den zweiten Timer hinzu, den der Benutzer entsprechend seinen Wünschen einstellen, starten und stoppen kann.

Führen Sie die folgenden Schritte aus, um die Anwendung zu erstellen:

1. Erstellen Sie ein neues MFC-Anwendung Visual C++-Projekt namens Timers.

2. Geben Sie im MFC Anwendungs-Assistent an, dass es sich um eine dialogbasierende Anwendung handelt.
3. Verwenden Sie die Voreinstellungen im MFC Anwendungs-Assistent.
4. Gestalten Sie das Dialogfeld entsprechend Abbildung 5.1 mit den Eigenschaften der Steuerelemente, die in Tabelle 5.1 aufgeführt sind.

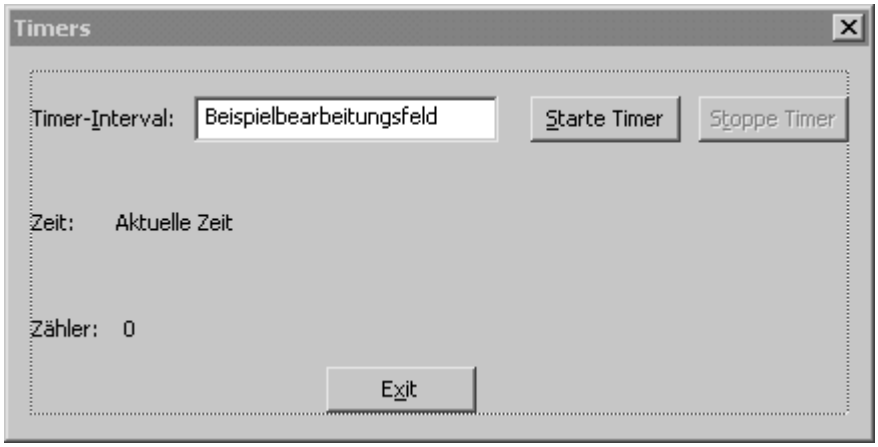


Abbildung 5.1: Layout des Dialogfelds für die Timer-Anwendung

Objekt	Eigenschaft	Einstellung
Text	Beschriftung (Titel)	`Timer-&Interval:`
Eingabefeld	ID	`IDC_EINTERVAL`
Schaltfläche	ID Beschriftung	`IDC_BSTARTTIME` `&Starte Timer`

Schaltfläche	ID
	Beschriftung
	Deaktiviert (Disabled)
		`IDC_BSTOPTIMER`
		`S&toppe Timer`
		`TRUE`

Text	Beschriftung	`Zeit:`
Text	ID Beschriftung	`IDC_STATICTIME` `Aktuelle Zeit`
Text	Beschriftung	`Zähler:`
Text	ID Beschriftung	`IDC_STATICCOUNT` `0`
Schaltfläche	ID Beschriftung	`IDC_BEXIT` `E&xit`

Tabelle 5.1: Einstellungen der Steuerelementeigenschaften

5. Legen Sie die Tabulator-Reihenfolge fest, wie Sie es am Tag 3, »Steuerelemente«, gelernt haben.
6. Fügen Sie für die Schaltfläche Exit Code zum Schließen der Anwendung wie am Tag 3 hinzu.

Die Timer-IDs hinzufügen

Da in dieser Anwendung zwei Timer zum Einsatz kommen, nehmen Sie auch zwei IDs in Ihre Anwendung auf, um die beiden Timer zu identifizieren. Führen Sie dazu die folgenden Schritte aus:

1. Rechtsklicken Sie in der Ressourcenansicht auf den Ordner Timers.rc am oberen Ende des Ressourcenbaums. Wählen Sie aus dem Kontextmenü Select Resource Symbols.

2. Im Dialogfeld Ressourcen Symbole klicken Sie auf die Schaltfläche Neu ....
3. Im Dialogfeld Neues Symbol geben Sie ID_CLOCK_TIMER als Symbolname und 1 als Wert ein, wie es Abbildung 5.2 zeigt. Klicken Sie auf Ok, um das Symbol zum Projekt hinzuzufügen.

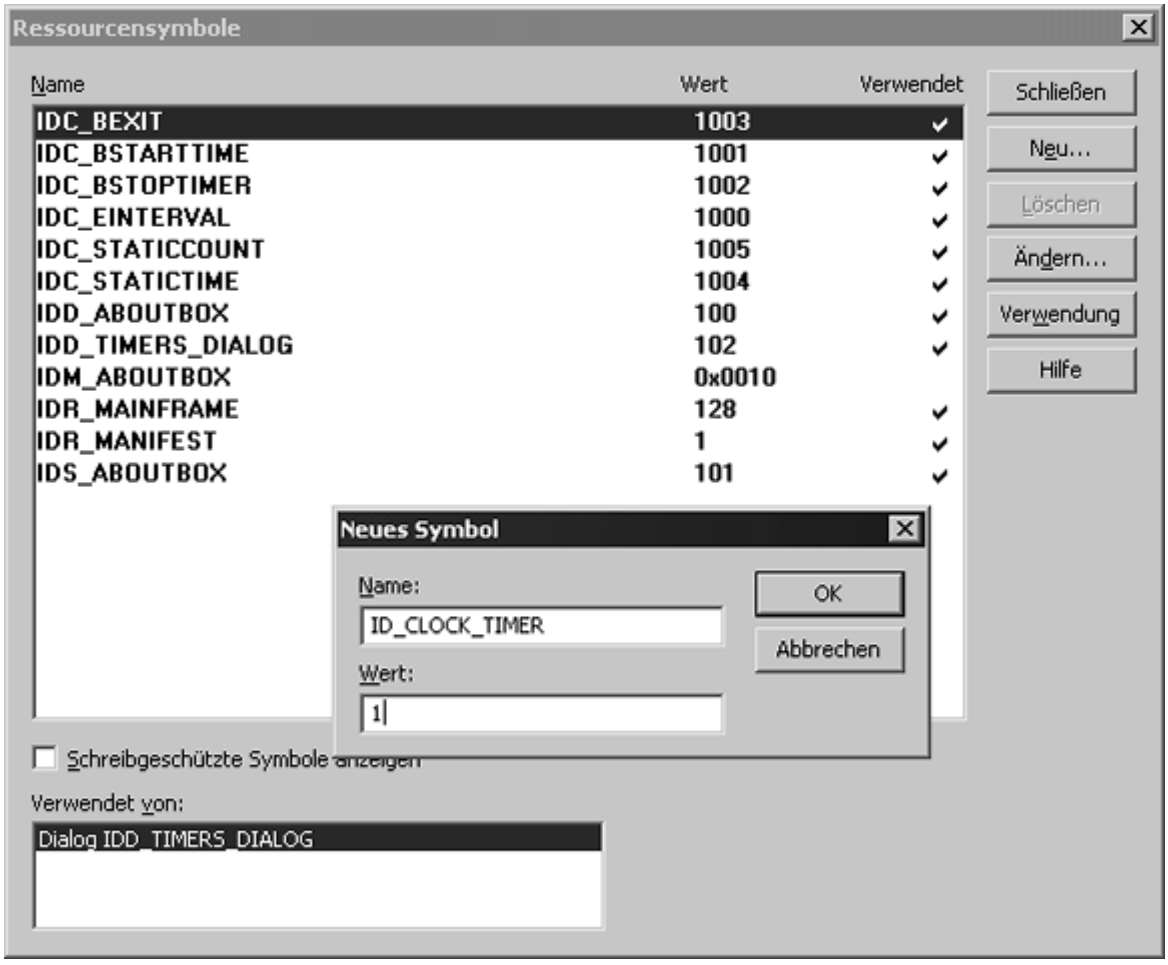


Abbildung 5.2: Ein neues Ressourcensymbol hinzufügen

4. Wiederholen Sie die Schritte 2 und 3, wobei Sie ID_COUNT_TIMER als Symbolname und 2 als Wert spezifizieren.
5. Klicken Sie auf die Schaltfläche Schliessen, um das Dialogfeld Ressourcen Symbole zu schließen. Die beiden Timer-IDs gehören nun zu Ihrer Anwendung und warten auf ihren Einsatz.

Den Uhren-Timer starten

Um den Timer für die Uhr zu starten, bearbeiten Sie die Funktion OnInitDialog, wie Sie es in den vergangenen beiden Tagen getan haben. Nehmen Sie den neuen Code aus Listing 5.1 in die Funktion auf.

Listing 5.1: Die Funktion OnInitDialog

```
 1: BOOL CTimersDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
 5:   // Hinzufügen des Menübefehls "Info..." zum Systemmenü.
 6:
 7:   // IDM_ABOUTBOX muss sich im Bereich der Systembefehle
 8:   // befinden.
 9:   ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
10:   ASSERT(IDM_ABOUTBOX < 0xF000);
11:
...
30:
31:   // TODO: Hier zusätzliche Initialisierung einfügen
32:   // Timer für die Uhr starten
33:   SetTimer(ID_CLOCK_TIMER, 1000, NULL);
34:
35:   return TRUE;  // Geben Sie TRUE zurück, außer ein
36:                 // Steuerelement soll den Fokus erhalten
37: }
```

Der Code in diesem Listing startet den Uhren-Timer mit der Funktion SetTimer. Das erste an die Funktion SetTimer übergebene Argument ist die ID für den Uhren-Timer. Das zweite Argument gibt an, wie oft Sie das Ereignis auslösen möchten. Im vorliegenden Fall wird das Timer-Ereignis alle 1.000 Millisekunden, d.h. jede Sekunde, ausgelöst. Im dritten Argument wird die Adresse einer optionalen Callback-Funktion übergeben, die Sie festlegen können, um das Ereignis WM_TIMER zu umgehen. Wenn Sie für dieses Argument den Wert NULL übergeben, stellt Windows das Ereignis WM_TIMER in die Nachrichtenwarteschlange der Anwendung.

Unter einer Callback-Funktion versteht man eine vom Programmierer erstellte Funktion, die das Betriebssystem Windows direkt aufruft. Callback-Funktionen weisen spezielle Definitionen für die Argumente auf, je nachdem, welches Subsystem die Funktion aufruft und aus welchen Gründen. Im Anschluss an die Funktionsdefinition haben Sie aber wieder freie Hand, wie Sie die Funktion gestalten möchten oder müssen.

Die Funktionsweise einer Callback-Funktion beruht darauf, dass man die Adresse der Funktion als Argument an eine Windows-Funktion, die Callback-Funktionen als Argumente akzeptiert, übergibt. Nachdem Sie die Funktionsadresse an Windows übergeben haben, wird die Funktion jedesmal direkt aufgerufen, wenn Windows auf Grund der entsprechenden Bedingungen die Callback-Funktion aufrufen muss.

Das Timer-Ereignis der Uhr behandeln

Nachdem Sie nun einen Timer gestartet haben, müssen Sie noch den Code aufnehmen, der die Timer-Nachrichten behandelt. Dazu führen Sie folgende Schritte aus:

1. Nehmen Sie eine Variable für das Steuerelement IDC_STATICTIME vom Typ CString mit dem Namen m_sTime auf.

2. Fügen Sie eine Funktion für die Nachricht WM_TIMER für das Objekt CTimersDlg hinzu.
3. Übernehmen Sie den Code aus Listing 5.2 in die Funktion OnTimer.

Listing 5.2: Die Funktion OnTimer

```
 1: void CTimersDlg::OnTimer(UINT nIDEvent)
```

```
 2: {
 3:    // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:    // und/oder benutzen Sie den Standard.
 5:    // Aktuelle Zeit holen
 6:    CTime curTime = CTime::GetCurrentTime();
 7:
 8:    // Aktuelle Zeit anzeigen
 9:    m_sTime = curTime.Format("%H:%M:%S");
10:
11:    // Dialogfeld aktualisieren
12:    UpdateData(FALSE);
13:
14:    CDialog::OnTimer(nIDEvent);
15: }
```

Der Code in diesem Listing deklariert eine Instanz der Klasse CTime und initialisiert sie mit der aktuellen Systemzeit. Als Nächstes setzt die Funktion den String m_sTime auf die aktuelle Uhrzeit und verwendet dabei die Methode Format, um die Zeit in der üblichen Form mit Stunden, Minuten und Sekunden (HH:MM:SS) zu formatieren. Schließlich wird das Dialogfeld mit der aktuellen Uhrzeit aktualisiert. Wenn Sie die Anwendung jetzt kompilieren und ausführen, wird eine laufende Uhr in der Mitte des Dialogfelds zu sehen sein, wie es Abbildung 5.3 zeigt.

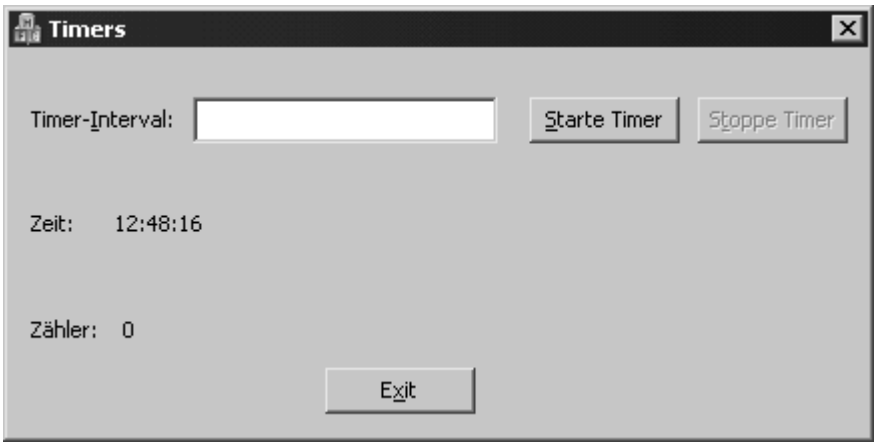


Abbildung 5.3: Im Dialogfeld der Anwendung ist eine laufende Uhr zu sehen.

MFC-Exkurs: Die Klasse CTime

Die Klasse CTime verkapselt einen großen Teil der Datum-/Zeit-Funktionalität, die Sie in Ihren Anwendungen benötigen könnten. CTime wurde um die Coordinated Universal Time (UTC) herum erstellt, die früher auch als Greenwich Mean Time (GMT) bezeichnet wurde. So können Sie problemlos sowohl mit der Ortszeit (basierend auf der Konfiguration Ihres Computers) als auch mit UTC arbeiten.

Datum und Zeit erzeugen und initialisieren

Wenn Sie ein CTime-Objekt nur mit einer einfachen Deklaration erzeugen, erhalten Sie ein leeres CTime-Objekt, das erst verwendet werden kann, wenn Sie es mit einer Zeit initialisiert haben. Daher ist das keine sehr brauchbare Methode, ein CTime-Objekt zu erzeugen. Es gibt einige Methoden zum Erstellen von CTime-Objekten, die es gleichzeitig initialisieren. Eine dieser Methoden wird in Listing 5.2 verwendet:

```
CTime CurTime = CTime::GetCurrentTime();
```

Hier verwendet der Code die Methode GetCurrentTime, die ein CTime-Objekt zurückgibt, um die neue Instanz mit dem aktuellen Datum und der aktuellen Zeit zu initialisieren. Eine weitere Möglichkeit, das CTime-Objekt zu erzeugen und zu initialisieren, wäre die Verwendung der folgenden Version des Konstruktors:

```
CTime(int iJahr, int iMonat, int iTag,
      int iStd, int iMin, int iSek, int iSZ);
```

Diese Version des Konstruktors übergibt die Werte für Jahr, Monat, Tag, Stunde, Minute und Sekunde, mit denen Sie das Objekt initialisieren möchten. Der letzte Parameter, iSZ, gibt an, ob Sommerzeit verwendet wird. Wenn iSZ = 0 ist, gilt die Standardzeit; ist es größer als 0, gilt Sommerzeit; ist iSZ kleiner als 0 (die Standardeinstellung, wenn Sie diesen Parameter nicht übergeben), wird automatisch anhand der Rechnerkonfiguration ermittelt, ob Sommerzeit gilt oder nicht.

Datum und Zeit abfragen

Nachdem Sie ein CTime-Objekt erzeugt und initialisiert haben, müssen Sie in der Lage sein, Datum und Zeit aus ihm abzufragen. In Tabelle 5.2 sind einige Funktionen aufgelistet, die das ermöglichen.

Funktion	Beschreibung
GetYear	Gibt das Jahr zurück
GetMonth	Gibt den Monat zurück
GetDay	Gibt den Tag im Monat zurück
GetHour	Gibt die Stunde zurück
GetMinute	Gibt die Minute zurück
GetSecond	Gibt die Sekunde zurück
GetDayOfWeek	Gibt den Wochentag zurück (1 = Sonntag, 7 = Samstag)
Format	Gibt einen CString der Ortszeit zurück, der nach dem als Parameter an diese Funktion übergebenen Formatstring formatiert ist. In Tabelle 5.3 sind die Formatcodes aufgelistet.
FormatGmt	Gibt einen CString der GMT zurück, der nach dem als Parameter an diese Funktion übergebenen Formatstring formatiert ist. In Tabelle 5.3 sind die Formatcodes aufgelistet.

Tabelle 5.2: Funktionen zum Abfragen von Datum und Zeit in CTime

Formatcode	Beschreibung
%a	Der abgekürzte Name des Wochentags
%A	Der ausgeschriebene Name des Wochentags
%b	Der abgekürzte Monatsname
%B	Der ausgeschriebene Monatsname
%c	Datum und Zeit in der Standardformatierung für den konfigurierten Standort
%d	Der Tag des Monats
%D	Die Summe der Tage im Ctime

Tabelle 5.3: Formatcodes für Datum und Zeit in CTime

Formatcode	Beschreibung
%H	Die Stunde im 24-Stunden-Format
%I	Die Stunde im 12-Stunden-Format
%j	Der Tag des Jahres
%m	Der Monat als Zahl von 01 bis 12
%M	Die Minute
%p	Die AM/PM-Markierung
%S	Die Sekunden

%U	Die Woche des Jahres als Zahl, wobei Sonntag als erster Wochentag gilt
%w	Der Wochentag als Zahl, wobei 0 dem Sonntag entspricht
%W	Die Woche des Jahres als Zahl, wobei Montag als erster Wochentag gilt
%x	Das Datum im örtlichen Standardformat
%X	Die Zeit im örtlichen Standardformat
%y	Das Jahr ohne Jahrhundert (zweistellig)
%Y	Das Jahr mit Jahrhundert
%z, %Z	Der Name bzw. die Abkürzung der Zeitzone
%%	Ein Prozentzeichen

Tabelle 5.3: Formatcodes für Datum und Zeit in CTime (Forts.)

5.3 Einen zweiten Timer in die Anwendung aufnehmen

Wie Sie sich überzeugen konnten, lässt sich ein einzelner Timer ziemlich leicht in eine Anwendung einbauen. Man braucht dazu nur die Funktion SetTimer aufzurufen und dann den Timer-Code in die Funktion OnTimer zu schreiben. Manchmal benötigt man aber mehrere Timer, die gleichzeitig in ein und derselben Anwendung laufen. In diesem Fall ist die Sache ein wenig komplizierter.

Die Variablen der Anwendung hinzufügen

Bevor Sie den zweiten Timer in die Anwendung einbauen, sind den Steuerelementen ein paar Variablen zuzuordnen. Für den Uhren-Timer war nur eine einzelne Variable erforderlich, um die Zeitanzeige zu aktualisieren. Jetzt brauchen wir zusätzliche Variablen für die anderen Steuerelemente. In Tabelle 5.4 sind diese Variablen zusammengefasst.

Folgen Sie beim Einfügen der Variable für das Steuerelement IDC_INTERVAL diesen Schritten:

1. Geben Sie in den beiden Eingabefeldern einen minimalen Wert von 1 und einen maximalen Wert von 100000 ein (siehe Abbildung 5.4).

Objekt	Name	Kategorie	Typ
IDC_STATICCOUNT	m_sCount	Value (Wert)	CString
IDC_BSTARTTIME	m_cStartTime	Control	CButton
IDC_BSTOPTIMER	m_cStopTime	Control	CButton
IDC_EINTERVAL	m_iInterval	Value	int

Tabelle 5.4: Variablen für die Steuerelemente

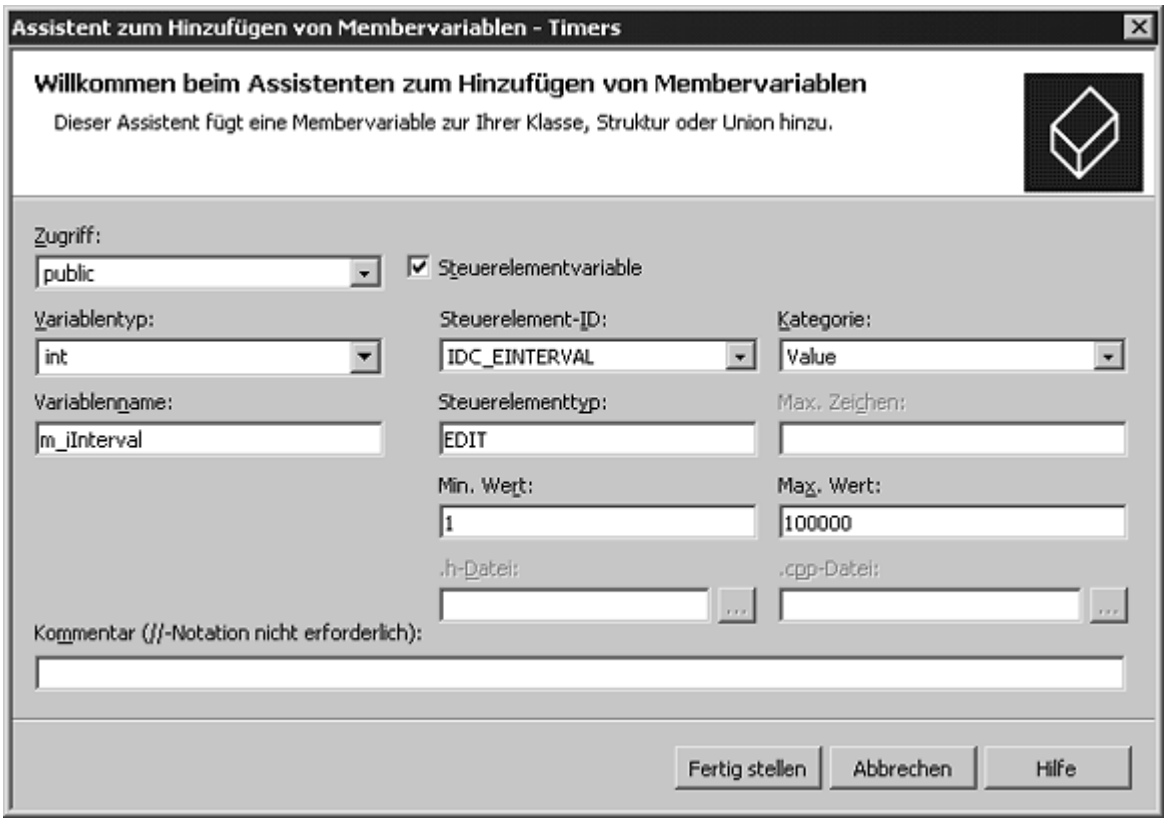


Abbildung 5.4: Den zu prüfenden Bereich für eine Variable festlegen

2. In der Klassenansicht nehmen Sie eine Member-Variable in die Klasse CTimersDlg auf, wie Sie es gestern gelernt haben. Legen Sie den Variablentyp mit int, den Namen als m_iCount und den Zugriff als private fest.
3. Fügen Sie eine Funktion für die Nachricht EN_CHANGE für die Objekt-ID IDC_INTERVAL (das Eingabefeld) hinzu. In diese Funktion übernehmen Sie den Code aus Listing 5.3.

Listing 5.3: Die Funktion OnEnChangeEinterval

```
 1: void CTimersDlg::OnEnChangeEinterval()
 2: {
 3:    // TODO:  Falls dies ein RICHEDIT-Steuerelement ist, wird das
 4:    // Kontrollelement
 5:    // diese Benachrichtigung nicht senden, es sei denn, Sie
 6:    // setzen den CDialog::OnInitDialog() außer Kraft.
 7:    // Funktion und Aufruf CRichEditCtrl().SetEventMask()
 8:    // mit dem ENM_CHANGE-Flag ORed in der Eingabe.
 9:
10:    // TODO:  Fügen Sie hier Ihren Code für die
11:    // Kontrollbenachrichtigungsbehandlung ein.
12:    // Variablen aktualisieren
13:    UpdateData(TRUE);
14: }
```

Wenn Sie für die Intervallvariable des Timers einen Wertebereich festlegen und der Benutzer einen Wert außerhalb des spezifizierten Bereichs eingibt, fordert Visual C++ den Benutzer automatisch zur Eingabe eines Wertes im zulässigen Bereich auf. Diese Aufforderung wird durch den Aufruf der Funktion UpdateData in OnEnChangeEinterval ausgelöst. Die als Letztes über den Arbeitsbereich hinzugefügte Variable dient als eigentlicher Zähler, der mit jedem Timer-Ereignis inkrementiert wird.

Den Timer für den Zähler starten und stoppen

Damit der zweite Timer funktionsfähig wird, müssen Sie

- die Variable m_iInterval initialisieren,
- den Timer starten, wenn der Benutzer auf die Schaltfläche IDC_BSTARTTIME klickt,
- bei jedem Timer-Ereignis die Variable m_iCount inkrementieren und das Dialogfeld aktualisieren,
- den Timer anhalten, wenn der Benutzer auf die Schaltfläche IDC_BSTOPTIMER klickt.

Diese zusätzliche Funktionalität realisieren Sie in folgenden Schritten:

1. Bearbeiten Sie die Funktion OnInitDialog entsprechend dem Code von Listing 5.4.

Listing 5.4: Die aktualisierte Funktion OnInitDialog

```
 1: BOOL CTimersDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:     ...
 5:   // TODO: Add extra initialization here
 6:   // Zählerintervall initialisieren
 7:   m_iInterval = 100;
 8:
 9:   // Dialogfeld aktualisieren
10:   UpdateData(FALSE);
11:
12:   // Timer für die Uhr starten
13:   SetTimer(ID_CLOCK_TIMER, 1000, NULL);
14:
15:   return TRUE;  // Geben Sie TRUE zurück, außer ein
16:                 // Steuerelement soll den Fokus erhalten
17: }
```

2. Nehmen Sie eine Funktion für die Nachricht BN_CLICKED für die Schaltfläche IDC_BSTARTTIME auf und fügen Sie den Code gemäß Listing 5.5 in die Funktion OnBnClickedBstarttime ein.

Listing 5.5: Die Funktion OnBnClickedBstarttime

```
 1: void CTimersDlg::OnBnClickedBstarttime()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen aktualisieren
 6:   UpdateData(TRUE);
 7:
 8:   // Zähler initialisieren
 9:   m_iCount = 0;
10:   // Zähler für Anzeige formatieren
11:   m_sCount.Format("%d", m_iCount);
12:
13:   // Dialogfeld aktualisieren
14:   UpdateData(FALSE);
15:   // Timer starten
16:   SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
17: }
```

3. Fügen Sie eine Funktion für die Nachricht BN_CLICKED für die Schaltfläche IDC_BSTOPTIMER hinzu und übernehmen Sie in die Funktion OnBnClickedBstoptimer den Listing 5.6 entsprechenden Code.

Listing 5.6: Die Funktion OnBnClickedBstoptimer

```
1: void CTimersDlg::OnBnClickedBstoptimer()
2: {
3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
4:   // Benachrichtigung ein.
5:   // Timer anhalten
```

```
6:   KillTimer(ID_COUNT_TIMER);
7: }
```

4. Aktualisieren Sie die Funktion OnTimer gemäß dem Code aus Listing 5.7.

Listing 5.7: Die aktualisierte Funktion OnTimer

```
 1: void CTimersDlg::OnTimer(UINT nIDEvent)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:   // und/oder benutzen Sie den Standard.
 5:   // Aktuelle Zeit holen
 6:   CTime curTime = CTime::GetCurrentTime();
 7:
 8:   // Welcher Timer hat dieses Ereignis ausgelöst?
 9:   switch (nIDEvent)
10:   {
11:     // Der Uhren-Timer?
12:     case ID_CLOCK_TIMER:
13:       // Aktuelle Zeit anzeigen
14:       m_sTime = curTime.Format("%H:%M:%S");
15:     break;
16:     // Der Zähler-Timer?
17:     case ID_COUNT_TIMER:
18:       // Zähler inkrementieren
19:       m_iCount++;
20:       // Zähler formatieren und anzeigen
21:       m_sCount.Format("%d", m_iCount);
22:       break;
23:   }
24:
25:   // Dialogfeld aktualisieren
26:   UpdateData(FALSE);
27:
28:   CDialog::OnTimer(nIDEvent);
29: }
```

Die Funktion OnInitDialog initialisiert jetzt die Variable m_iInterval mit einem Anfangswert von 100. Um diese Initialisierung im Dialogfenster widerzuspiegeln, wird die Funktion UpdateData aufgerufen.

Die Funktion OnBnClickedBstarttime synchronisiert zuerst die Variablen mit den Werten der Steuerelemente. Damit lässt sich die aktuelle Einstellung der Variablen m_iInterval ermitteln. Als Nächstes initialisiert die Funktion die Variable m_iCount mit dem Anfangswert 0 und formatiert dann den Wert in der CString-Variablen m_sCount. Diese Variable wird im Dialogfenster aktualisiert. Als Letztes ist der Timer zu starten, wobei man die ID ID_COUNT_TIMER und das Intervall aus der Variablen m_iInterval spezifiziert.

In der Funktion OnBnClickedBstoptimer muss man lediglich den Timer anhalten. Dazu ruft man die Funktion KillTimer auf und übergibt ihr die Timer-ID als einziges Argument.

Die eigentlich interessanten Dinge passieren in der Funktion OnTimer. Hier steht momentan nur der Code zur Behandlung des Timer-Ereignisses für die Uhr. Um die Funktionalität für den Zähler-Timer hinzuzufügen, ist zu ermitteln, welcher Timer diese Funktion ausgelöst hat. Das einzige Argument an die Funktion OnTimer ist aber gerade die Timer-ID. Diese ID lässt sich in einer switch-Anweisung testen, um den Timer herauszufinden, der diese Funktion aufgerufen hat, und um zu steuern, welcher Code daraufhin auszuführen ist. Der Code für den Uhren-Timer bleibt gemäß Listing 5.2 erhalten. Der Code für den Zähler-Timer wird an der entsprechenden Stelle in der switch- Anweisung untergebracht. Er inkrementiert den Zähler und aktualisiert dann die Variable m_sCount auf den neuen Wert. Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie ein Timer-Intervall festlegen und den Timer starten, wie es Abbildung 5.5 zeigt.

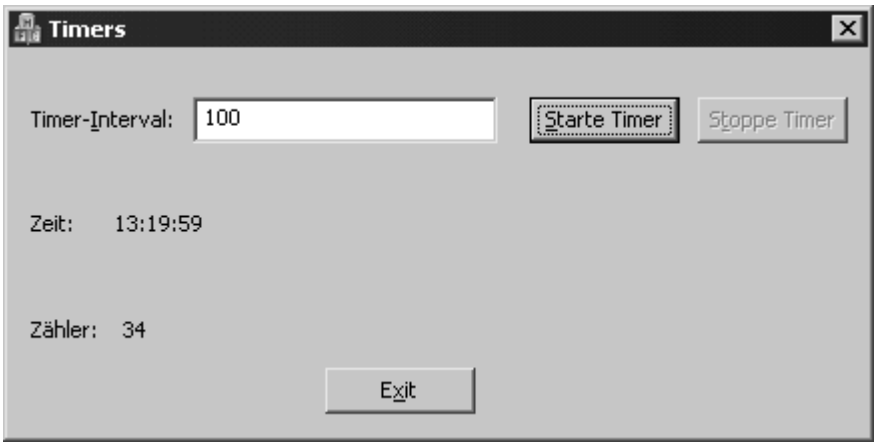


Abbildung 5.5: Im Dialogfeld der Anwendung ist ein laufender Zähler zu sehen.

Es gibt einen »Bug« in der Anwendung. Wenn Sie im Eingabefeld einen Wert eingeben, der sich nicht in den vorgesehenen Grenzen befindet, wird ein Warnfenster angezeigt, um Sie darüber zu informieren. Während diese Nachricht geöffnet ist, wird der Zähler-Timer ausgelöst und versucht, die Steuerelemente im Dialogfeld zu aktualisieren. Da die Funktion UpdateData für die Aktualisierung der Steuerelemente aufgerufen wird, während sie noch versucht, die Variablen aus den Steuerelementen zu aktualisieren, führt das zu einem Wiederaufruf-Konflikt (reentrant - Funktionen werden erneut aufgerufen bevor sie erstmals beendet sind), der das Programm zum Absturz bringt. Um dieses Problem zu beheben, fügen Sie eine Boolesche Variable in die Dialogklasse ein, mit der Sie das Zähler-Ereignis an der Aktualisierung der Steuerelemente hindern, solange die Änderungen im Eingabefeld nicht vollständig abgearbeitet sind.

Die Funktion OnEnChangeEinterval würde dann so aussehen:

Listing 5.8: Variante der Funktion OnEnChangeEinterval

```
 1: void CTimersDlg::OnEnChangeEinterval()
 2: {
 3:   ...
 4:   // TODO:  Fügen Sie hier Ihren Code für die
 5:   // Kontrollbenachrichtigungsbehandlung ein.
 6:   // Aktualisierung der Anzeige verhindern
 7:   m_bPreventUpdate = TRUE;
 8:   // Variablen aktualisieren
 9:   UpdateData(TRUE);
10:   // Aktualisierung der Anzeige zulassen
11:   m_bPreventUpdate = FALSE;
12: }
```

und die Funktion OnTimer so:

Listing 5.9: Variante der Funktion OnTimer

```
 1: void CTimersDlg::OnTimer(UINT nIDEvent)
 2: {
 3: ...
 4:   // Kann die Anzeige aktualisiert werden?
 5:   if (!m_bPreventUpdate)
 6:   // Dialogfeld aktualisieren
 7:     UpdateData(FALSE);
```

```
 8:
 9:   CDialog::OnTimer(nIDEvent);
10: }
```

Wenn die Applikation jetzt gestartet wird kann es vorkommen, daß am Anfang keine laufende Zeit angezeigt wird. Warum? In der Funktion OnTimer steuert der Wert der Variablen m_bPreventUpdate ob die Anzeige aktualisiert wird. Diese Variable ist aber zunächst noch nicht mit einem Wert initialisiert worden, ihr Wert ist also zufällig. Deshalb muss die Variable in der Funktion OnInitDialog noch mit dem Wert FALSE initialisiert werden.

Zuletzt wollen Sie verhindern, dass der Timer gestartet wird, also wird die Funktion OnBnClickedBstarttime dann so aussehen:

Listing 5.10: Variante der Funktion OnTimer

```
void CTimersDlg::OnBnClickedBstarttime()
{
  // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
  // Benachrichtigung ein.
  // Aktualisierung der Anzeige verhindern
  m_bPreventUpdate = TRUE;
  // Variablen aktualisieren
  UpdateData(TRUE);
  // Aktualisierung der Anzeige zulassen
  m_bPreventUpdate = FALSE;
  // Bei Werten außerhalb des Bereichs weitere Verarbeitung
  // verhindern
  if ((m_iInterval < 1) || (m_iInterval > 100000))
    return;
...
}
```

Die Schaltfläche Stop Timer aktivieren

Die Anwendung läuft zwar schon ganz gut, weist aber noch ein kleines Problem auf. Nachdem Sie den zweiten Timer gestartet haben, können Sie ihn nicht mehr anhalten. Beim Festlegen der Eigenschaften für die Steuerelemente haben Sie die Schaltfläche Stoppe Timer deaktiviert. Um den Timer anhalten zu können, müssen Sie diese Schaltfläche aktivieren.

Es ist sinnvoll, nach dem Start des Timers die Schaltfläche Stoppe Timer zu aktivieren und die Schaltfläche Starte Timer zu deaktivieren. Sobald der Benutzer den Timer angehalten hat, kehren Sie das Ganze um. Das lässt sich auf die gleiche Weise erreichen, wie Sie die Steuerelemente an Tag 3 aktiviert und deaktiviert haben. Diese Lösung kann man auch ein wenig modifizieren.

Beim Hinzufügen der Variablen für die Steuerelemente haben Sie auch Variablen für die Schaltflächen Starte Timer und Stoppe Timer aufgenommen. Es handelt sich hierbei nicht um normale Variablen, sondern um Steuerelementvariablen. Statt nun einen Zeiger auf die Steuerelemente über deren IDs zu ermitteln, können Sie direkt mit den Steuerelementvariablen arbeiten. Überarbeiten Sie dazu die Funktionen OnBnClickedBstarttime und OnButtonClickedBstoptimer gemäß Listing 5.11.

Listing 5.11: Die überarbeiteten Funktionen OnBnClickedBstarttime und OnBnClickedBstoptimer

```
 1: void CTimersDlg::OnBnClickedBstarttime()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
      // Benachrichtigung ein.
 4:   // Variablen aktualisieren
 5:   UpdateData(TRUE);
 6:
 7:   // Zähler initialisieren
 8:   m_iCount = 0;
 9:   // Zähler für Anzeige formatieren
```

```
10:    m_strCount.Format("%d", m_iCount);
11:
12:    // Dialogfeld aktualisieren
13:    UpdateData(FALSE);
14:    // Timer starten
15:    SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
16:
17:    // Schaltfläche Stop Timer aktivieren
18:    m_cStopTime.EnableWindow(TRUE);
19:    // Schaltfläche Start Timer deaktivieren
20:    m_cStartTime.EnableWindow(FALSE);
21: }
22:
23: void CTimersDlg::OnBnClickedBstoptimer()
24: {
25:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
       // Benachrichtigung ein.
26:    // Timer anhalten
27:    KillTimer(ID_COUNT_TIMER);
28:
29:    // Schaltfläche Stop Timer deaktivieren
30:    m_cStopTime.EnableWindow(FALSE);
31:    // Schaltfläche Start Timer aktivieren
32:    m_cStartTime.EnableWindow(TRUE);
33: }
```

Wenn Sie jetzt die Anwendung kompilieren und ausführen, können Sie den Zähler-Timer starten und anhalten, wie es Abbildung 5.6 zeigt. Damit kann der Benutzer mit dem Timer-Intervall experimentieren - verschiedene Intervalle ausprobieren und die Unterschiede beobachten, wobei die Uhr zum Vergleich weiterläuft.

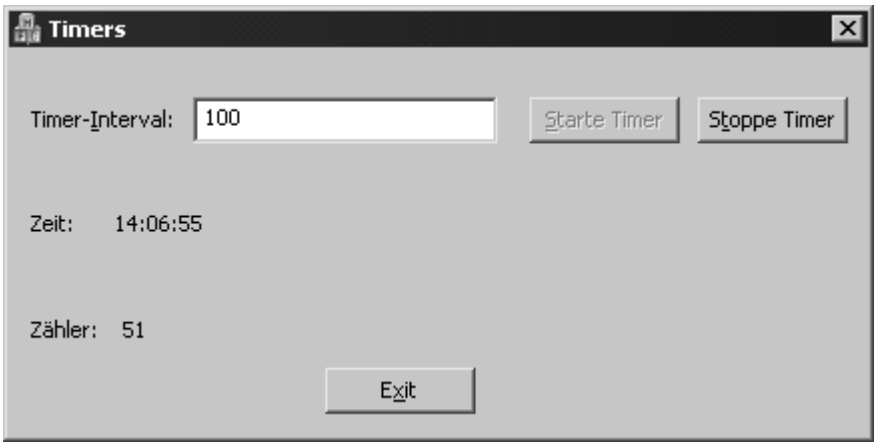


Abbildung 5.6: Die fertig gestellte Anwendung.

5.4 Zusammenfassung

Die heutige Lektion hat sich mit dem Einsatz von Timern im Betriebssystem Windows beschäftigt. Timer erlauben es, bestimmte Funktionen in einer Anwendung zeitgesteuert auszulösen. Es wurde gezeigt, wie man mehrere Timer in derselben Anwendung einsetzt, sie gleichzeitig laufen lässt und dabei unterschiedliche Aktionen auslöst.

In den folgenden Tagen gehen wir darauf ein, wie man zusätzliche Dialogfelder einbindet, um Rückmeldungen vom Benutzer zu erhalten. Die Eingaben des Benutzers kann man auswerten und damit das Verhalten der Anwendung kontrollieren. Im Anschluss daran lernen Sie, wie sich Menüs in einer Anwendung realisieren lassen. Anschließend steht die Arbeit mit Text und Schriften auf der Tagesordnung.

5.5 Workshop

Fragen und Antworten

Frage:
Wie groß ist der Bereich, den ich für das Timer-Intervall in meinen Anwendungen festlegen kann?

Antwort:
Der verfügbare Bereich beginnt bei 55 Millisekunden und reicht bis 232 - 1 Millisekunden, d.h. gut 49½ Tage.

Frage:
Wie viele Timer kann man in einer Anwendung gleichzeitig laufen lassen?

Antwort:
Diese Frage lässt sich nicht allgemein gültig beantworten. Allen Anwendungen steht im Betriebssystem Windows eine begrenzte Anzahl von Timern zur Verfügung. Obwohl diese Zahl mehr als ausreichend sein dürfte, wenn alle Anwendungen zusammen nicht mehr als eine Handvoll Timer verwenden, können diese Ressourcen dennoch knapp werden, wenn eine Anwendung eine größere Anzahl Timer für sich in Anspruch nimmt. Möglicherweise werden dann Ihrer Anwendung Timer verweigert oder andere Anwendungen erhalten keine. Als Faustregel gilt, dass man nicht mehr als etwa zwei oder drei Timer gleichzeitig betreiben sollte. Falls die Anwendung eine größere Zahl erfordert, empfiehlt es sich, den Entwurf der Anwendung noch einmal kritisch unter die Lupe zu nehmen und so umzubauen, dass sie mit weniger Timern auskommt.

Frage:
Gibt es eine Möglichkeit, eine Aktion in der Anwendung auszulösen, wenn sich die Anwendung im Leerlauf befindet, statt einen Timer einzusetzen und die Aktion zu starten, wenn man meint, dass die Anwendung im Leerlauf arbeitet?

Antwort:
Ja, die gibt es. Alle Windows-Anwendungen verfügen über eine OnIdle-Funktion, mit der man eine Verarbeitung während der Leerlaufphase auslösen kann. Die Funktion OnIdle kommt an Tag 17, »Multitasking«, zur Sprache.

Quiz

1. Auf welche Weise haben Sie die beiden Timer-IDs zu den Ressourcensymbolen hinzugefügt?

2. Welche andere Möglichkeit gibt es, um diese beiden IDs in die Anwendung aufzunehmen?
3. Wie kann man zwei Timer in der Funktion OnTimer auseinander halten?
4. Wie viele Timer-Ereignisse empfängt Ihre Anwendung, wenn der Timer für das Intervall von einer Sekunde eingerichtet wird, die Anwendung eine Minute beschäftigt ist und somit den Empfang von Timer-Nachrichten unterbindet?

Übung

Überarbeiten Sie Ihre Anwendung dahingehend, dass der Uhren-Timer beim Starten des Zähler-Timers auf das gleiche Intervall wie der Zähler-Timer gesetzt wird. Hält der Benutzer den Zähler-Timer an, soll der Uhren-Timer wieder mit einem Intervall von 1 Sekunde laufen.

Tag 6

Dialogboxen

In den meisten Anwendungen gibt es zahlreiche Situationen, in denen die Anwendung Informationen vom Benutzer anfordert, beispielsweise wie die Anwendung konfiguriert werden soll oder ob ein Dokument vor dem Verlassen der Anwendung zu speichern ist. In den meisten derartigen Fällen öffnet die Anwendung ein neues Fenster, um die entsprechenden Fragen zu stellen. Man bezeichnet diese Fenster als Dialogboxen.

Dialogboxen enthalten in der Regel ein oder mehrere Steuerelemente und erklären mit etwas Text, welche Informationen die Anwendung vom Benutzer erwartet. In Dialogboxen ist normalerweise kein größerer leerer Arbeitsbereich vorgesehen, wie er etwa in den Hauptfenstern von Textverarbeitungen oder Programm-Editoren zu finden ist. Alle in den vergangenen Tagen erstellten Anwendungen waren als Dialogboxen konzipiert, was sich auch in den unmittelbar folgenden Lektionen nicht ändert.

Die bisher erstellten Dialogboxen waren durchweg Anwendungen mit einem einzigen Dialogfenster.

Heute lernen Sie, wie man ...

- Dialogboxen flexibler einsetzt,
- andere Dialogboxen aufruft und die vom Benutzer in diesen Fenstern eingegebenen Informationen an das Hauptfenster zurückgibt, um sie von der Anwendung verarbeiten zu lassen,
- sowohl Standarddialogboxen verwendet - wie etwa die in den vergangenen Tagen eingesetzten Meldungsboxen - als auch benutzerdefinierte Dialogboxen, die Sie selbst erstellt haben.

6.1 Vordefinierte (oder System-) Dialogboxen

Das Betriebssystem Windows stellt eine Reihe von vordefinierten Dialogboxen bereit. Einfache Dialogboxen, die so genannten Meldungsboxen, präsentieren dem Benutzer eine Nachricht und enthalten eine bis drei Schaltflächen, über die der Benutzer seine Reaktion an die Anwendung übermitteln kann. Komplexere Dialogboxen wie etwa die Dialogboxen Öffnen, Speichern unter oder Drucken werden ebenfalls durch Windows bereitgestellt. Diese System- (oder allgemeinen) Dialogboxen werden zusammen mit einer Kombination aus einer Variablendeklaration einer C++-Klasse und einer Reihe von Interaktionen mit der Klasseninstanz erzeugt und verwendet.

Meldungsboxen

In den vergangenen Tagen haben Sie bereits erfahren, dass sich Meldungsboxen sehr einfach einsetzen lassen: Man ruft einfach eine Funktion auf und übergibt den Meldungstext als einziges Argument. Es erscheint eine Meldungsbox mit einem Symbol und dem Meldungstext sowie einer Schaltfläche, die der Benutzer anklickt, wenn er die Meldung zur Kenntnis genommen hat. Aus anderen Windows-Anwendungen ist Ihnen sicherlich bekannt, dass es eine ganze Palette von Meldungsboxen mit verschiedenartigen Kombinationen von Schaltflächen und Symbolen gibt.

Die Funktion MessageBox

Wie Sie bereits aus den letzten Tagen wissen, kann die Funktion MessageBox ein, zwei oder drei Argumente übernehmen. Das erste Argument ist die für den Benutzer anzuzeigende Meldung, während der Wert des optionalen zweiten Arguments in der Titelleiste der Meldungsbox erscheint. Mit einem ebenfalls optionalen dritten Argument lassen sich die Schaltflächen, die der Benutzer wählen kann, und das in der Meldung anzuzeigende Symbol festlegen. Zusätzlich zu diesen drei Argumenten liefert die Funktion MessageBox einen Ergebniswert zurück, der die vom Benutzer angeklickte Schaltfläche kennzeichnet. Über die Kombination von drittem Argument und Rückgabewert kann man in Visual C++- Anwendungen mit der Funktion MessageBox ein breites Funktionsspektrum realisieren.

Wenn Sie mit dem dritten Argument an die Funktion MessageBox die Schaltflächen oder das anzuzeigende Symbol spezifizieren, ist das zweite Argument (der Titel des Meldungsfelds) nicht mehr optional. In diesem Fall müssen Sie einen Wert für die Titelleiste des Meldungsfelds bereitstellen.

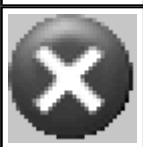
ID	Schaltflächen
MB_ABORTRETRYIGNORE	Abbrechen, Wiederholen, Ignorieren
MB_OK	OK
MB_OKCANCEL	OK, Abbrechen
MB_RETRYCANCEL	Wiederholen, Abbrechen
MB_YESNO	Ja, Nein
MB_YESNOCANCEL	Ja, Nein, Abbrechen
MB_CANCELTRYCONTINUE	Abbrechen, Wiederholen, Weiter

Tabelle 6.1: IDs zur Festlegung der Schaltflächenkombination in der Funktion MessageBox

Bezüglich der Kombination von Schaltflächen, die in der Meldungsbox erscheinen, sind Ihnen enge Grenzen gesetzt. Eigene Schaltflächenkombinationen lassen sich nicht festlegen. Sollten Sie mit den vordefinierten Schaltflächen oder den zulässigen Kombinationen nicht auskommen, bleibt Ihnen nichts anderes übrig, als eine benutzerdefinierte Dialogbox zu erstellen, die der gewünschten Meldungsbox entspricht. Tabelle 6.1 listet die verfügbaren Kombinationen von Schaltflächen auf, die man in der Funktion MessageBox spezifizieren kann.

Die Messagebox des Typs MB_CANCELTRYCONTINUE ist nur verfügbar, wenn im Quellcode die Konstante WINVER auf einen Wert >= 0x0500 gesetzt ist. Durch den Projektassistenten für MFC-Applikationen wird jedoch diese Konstante in der Datei stdafx.h auf den Wert 0x0400 festgelegt. Sofern Sie diesen Wert ändern müssen Sie beachten, dass die Applikation danach nur noch auf Windows 2000/XP oder höheren Version störungsfrei läuft. Ältere Windows-Version können eine Messagebox des Typs MB_CANCELTRYCONTINUE nicht anzeigen.

Um das anzuzeigende Symbol festzulegen, addiert man die ID des Symbols zur ID der Schaltflächenkombination. Die verfügbaren Symbole sind in Tabelle 6.2 aufgeführt. Möchten Sie nur das Symbol oder nur die Schaltflächenkombination festlegen und für das jeweils andere Element den Standardwert übernehmen, geben Sie einfach nur die gewünschte ID an.

ID	Symbol	
MB_ICONINFORMATION		Info-Symbol
MB_ICONQUESTION		Fragezeichen-Symbol
MB_ICONSTOP		Stopp-Symbol

MB_ICONEXCLAMATION		Ausrufezeichen-Symbol

Tabelle 6.2: IDs zur Festlegung der Symbole in der Funktion MessageBox

Wenn man eine Schaltflächenkombination angibt, liegt es nahe, auch den Rückgabewert zu verwenden, damit man ermitteln kann, auf welche Schaltfläche der Benutzer geklickt hat. Der Rückgabewert ist als Integer-Wert definiert. Die verfügbaren Werte sind in Tabelle 6.3 aufgelistet.

ID	**Gewählte Schaltfläche**
0	Es trat ein Fehler beim Anzeigen der Messagebox auf.
IDABORT	Abbrechen
IDRETRY	Wiederholen
IDIGNORE	Ignorieren
IDYES	Ja
IDNO	Nein
IDOK	OK
IDCANCEL	Abbrechen
IDTRYAGAIN	Noch einmal versuchen
IDCONTINUE	Fortfahren

Tabelle 6.3: Rückgabewerte der Funktion MessageBox

Die beiden Werte IDTRYAGAIN und IDCONTINUE sind nur bei WINVER >= 0x0500 verfügbar.

Die Funktion MessageBox ist ein Mitglied der Klasse CWnd und damit nur für Abkömmlinge dieser Klasse verfügbar. Das heißt, diese Funktion kann nur von einer Fenster- oder Steuerelement-Klasse aus aufgerufen werden. Wenn Sie eine Meldungsbox von einer Klasse aus öffnen müssen, die sich nicht aus CWnd ableitet, müssen Sie eine andere Version dieser Funktion namens AfxMessageBox verwenden. Dieser zweiten Funktion fehlt der Parameter, mit dem der Titel der Meldungsbox festgelegt werden kann, man kann nur die anzuzeigende Meldung und die Symbol-/Schaltflächenparameter angeben.

Eine Dialogfeld-Anwendung erstellen

Damit Sie kennen lernen, wie man in einer Anwendung mit der Funktion MessageBox vom Benutzer Informationen einholt, erstellen Sie eine einfache Beispielanwendung, in der die Funktion MessageBox in verschiedenen Varianten zum Einsatz kommt. Die Anwendung verfügt über zwei separate Schaltflächen, über die sich zwei verschiedene Versionen der Funktion MessageBox aufrufen lassen, damit Sie die Unterschiede und Gemeinsamkeiten der verschiedenen Optionen der Funktion studieren können. Weiter hinten in der heutigen Lektion fügen Sie der Anwendung die Standarddialogbox Datei öffnen hinzu. Dieses Beispiel zeigt, wie der Benutzer einen Dateinamen festlegen oder eine andere der vordefinierten Funktionen ausführen kann. Schließlich erzeugen Sie eine benutzerdefinierte Dialogbox, in die der Benutzer verschiedene Arten von Werten eingeben kann. In diesem Beispiel sehen Sie, wie man die vom Benutzer in die Hauptdialogbox der Anwendung eingegebenen Werte lesen kann, nachdem der Benutzer die benutzerdefinierte Dialogbox

geschlossen hat.

Den ersten Teil der Anwendung erstellen Sie mit folgenden Schritten:

1. Erstellen Sie ein neues MFC-Anwendung Visual C++-Projekt und nennen Sie es Dialogs.

2. Wählen Sie die gleichen Einstellungen wie in den Anwendungen der letzten Lektionen.
3. Gestalten Sie das Hauptdialogfeld der Anwendung entsprechend Abbildung 6.1 und legen Sie die Eigenschaften gemäß Tabelle 6.4 fest.

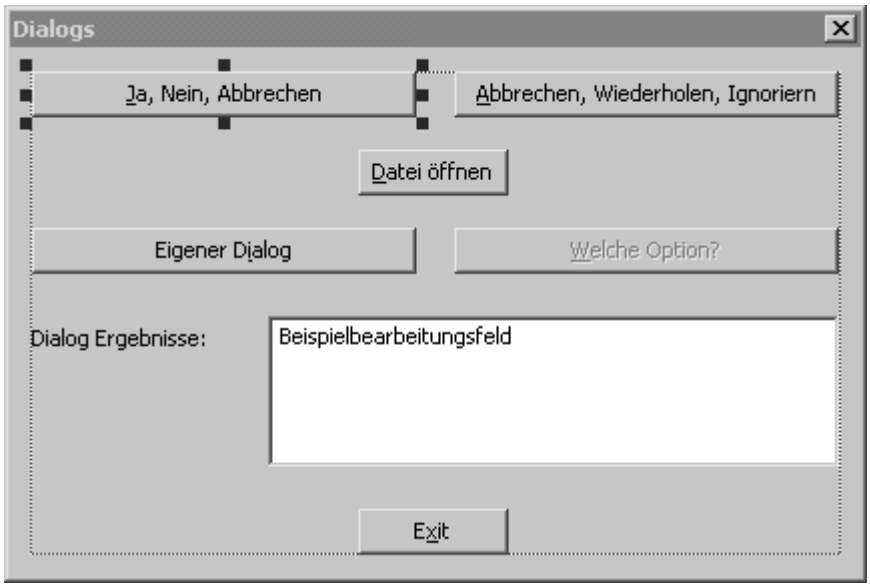


Abbildung 6.1: Das Layout des Hauptdialogfelds der Anwendung

Objekt	Eigenschaft	Einstellung
Schaltfläche	ID Beschriftung (Titel)	IDC_YESNOCANCEL &Ja, Nein, Abbrechen
Schaltfläche	ID Beschriftung	IDC_ABORTRETRYIGNORE &Abbrechen, Wiederholen, Ignoriern
Schaltfläche	ID Beschriftung	IDC_FILEOPEN &Datei öffnen
Schaltfläche	ID Beschriftung	IDC_BCUSTOMDIALOG Eigener D&ialog
Schaltfläche	ID Beschriftung Deaktiviert	IDC_BWHICHOPTION &Welche Option? True
Schaltfläche	ID Beschriftung	IDC_BEXIT E&xit
Text	Beschriftung	Dialog Ergebnisse:
Eingabefeld	ID Mehrfachzeile	IDC_RESULTS True

	Automatischer vertikaler Bildlauf	True

Tabelle 6.4: Eigenschaften der Steuerelemente für das Hauptdialogfeld

4. Verbinden Sie mit der an Tag 3 gelernten Methode die Variablen mit den in Tabelle 6.5 aufgeführten Steuerelementen.

Objekt	Name	Kategorie	Typ
IDC_ERESULTS	m_strResults	Value	CString
IDC_BWHICHOPTION	m_cWhichOption	Control	CButton

Tabelle 6.5: Variablen der Steuerelemente

5. Verbinden Sie wie an den vorhergegangenen Tagen eine Funktion für die Ereignisbehandlung von BN_CLICKED mit der Schaltfläche Exit. Fügen Sie den an den letzten Tagen verwendeten Code zum Schließen der Anwendung ein.

Die Meldungsboxen programmieren

Für die erste Schaltfläche (Ja, Nein, Abbrechen):

1. Erstellen Sie wie an den bisherigen Tagen mithilfe des Abschnitts Steuerelementereignisse in der Eigenschaftenansicht eine Funktion für das Klickereignis. Fügen Sie in die Funktion für diese Schaltfläche den Code aus Listing 6.1 ein.

Listing 6.1: Die Funktion OnBnClickedByesnocancel

```
 1: void CDialogsDlg::OnBnClickedByesnocancel(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   int iResults; // Variable für Schaltflächenauswahl
 6:
 7:   // Benutzer fragen
 8:   iResults = MessageBox("Drücken Sie Ja, Nein oder Abbrechen",
 9:                         "Ja, Nein, Abbrechen Dialog",
10:                         MB_YESNOCANCEL | MB_ICONINFORMATION);
11:
12:   // Angeklickte Schaltfläche ermitteln
13:   // Benutzer die angeklickte Schaltfläche melden
14:   switch (iResults)
15:   {
16:     case IDYES:     // Schaltfläche Ja?
17:       m_strResults = "Ja! Ja! Ja!";
18:       break;
19:     case IDNO:      // Schaltfläche Nein?
20:       m_strResults = "Nein, nein, nein und nochmals nein.";
21:       break;
22:     case IDCANCEL:  // Schaltfläche Abbrechen?
23:       m_strResults = "Traurig abgebrochen.";
24:       break;
25:   }
26:
27:   // Dialogfeld aktualisieren
28:   UpdateData(FALSE);
29: }
```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie verfolgen, wie sich durch Wahl der verschiedenen Schaltflächen im Meldungsfeld der Ablauf der Aktionen in der Anwendung bestimmen lässt.

2. Nehmen Sie eine Funktion für die Klickereignisse der Schaltflächen Abbrechen, Wiederholen, Ignorieren auf und geben Sie den gleichen Code wie in Listing 6.1 ein. Jetzt verwenden Sie jedoch die Werte MB_ABORTRETRYIGNORE und MB_ICONQUESTION und ändern die Aufforderungen und Meldungen.

Sie können sich nun davon überzeugen, wie sich diese andere Schaltflächenkombination in der gleichen Weise verwenden lässt.

Die Behandlungsroutinen für die Nachrichten der beiden Steuerelemente sind praktisch gleich. In jeder Funktion nimmt die deklarierte Integer-Variable den Rückgabewert der Funktion MessageBox auf. Als Nächstes wird die Funktion MessageBox mit einer anzuzeigenden Meldung, einem Titel für das Meldungsfeld und einer Kombination von Schaltflächen-ID und Symbol-ID aufgerufen.

Nachdem man den Rückgabewert der Funktion MessageBox in der Variablen gesichert hat, wird dieser Wert in einer switch-Anweisung ausgewertet, um die gedrückte Schaltfläche zu bestimmen. Eine Nachricht weist den Benutzer darauf hin, welche Schaltfläche er im Meldungsfeld angeklickt hat. Die vom Benutzer gewählte Schaltfläche ließe sich auch leicht mit einer oder zwei if-Anweisungen auswerten, um die Programmausführung zu steuern, da aber der Rückgabewert ganzzahlig ist, bietet sich eine switch-Anweisung wie von selbst an.

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie auf die beiden oberen Schaltflächen klicken und eine Meldung wie in Abbildung 6.2 sehen. Nachdem Sie auf eine der Schaltflächen des Meldungsfelds geklickt haben, erscheint im Eingabefeld des Hauptdialogfelds eine Meldung, die auf die gewählte Schaltfläche hinweist (siehe Abbildung 6.3).

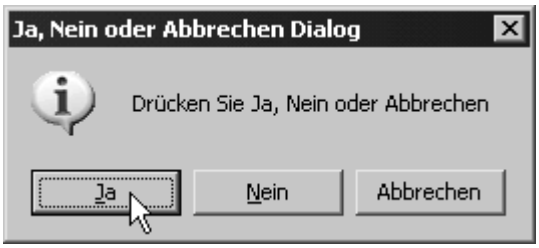


Abbildung 6.2: Das Meldungsfeld mit drei Auswahloptionen

Abbildung 6.3: Je nach angeklickter Schaltfläche wird eine Meldung angezeigt.

C++-Exkurs: Die switch-Anweisung

Wenn Sie schon andere Programmiersprachen verwendet haben, besteht die Wahrscheinlichkeit, dass Sie ein der switch-Anweisung in C/C++ ähnliches Flusskontrolle-Statement gesehen und verwendet haben. Die switch-Anweisung wird verwendet, wenn für eine Variable mit mehreren möglichen Werten abhängig vom jeweiligen Wert jeweils eine andere Verarbeitungslogik ausgeführt werden muss. Zwischen der switch-Anweisung in C/C++ und ähnlichen Statements in anderen Sprachen (wie dem select case in Visual Basic) gibt es einen wichtigen Unterschied: In der switch- Anweisung von C/C++ müssen alle zu vergleichenden Werte numerischer Natur sein. Sie können zur Kontrolle des logischen Flusses keine Strings oder Rückgabewerte von Funktionen, die etwas anderes als Integer-Werte zurückliefern, verwenden. Der auszuwertende Wert muss genauso wie jeder Fall (case) numerisch sein.

C++-Regel

Die switch-Anweisung kann zur Kontrolle des Verarbeitungsflusses nur numerische Werte vergleichen. Einzelne Zeichen können mit der switch-Anweisung verglichen werden, da im Vergleich der ASCII-Wert (oder Unicode-Wert) des Zeichens verwendet wird.

Die switch-Anweisung besitzt folgende Syntax:

```
switch (wert)
{
  case 0:
    verarbeitungslogik
    break;
  case 1:
    verarbeitungslogik
    break;
  default:
    verarbeitungslogik
    break;
}
```

Die switch-Anweisung wird von dem in allen Vergleichen zu verwendenden Wert gefolgt. Dieser Wert steht immer in Klammern. Danach stehen in Klammern ({}) eingeschlossen alle zu überprüfenden Fälle. Das Schlüsselwort case kennzeichnet jeden einzelnen Wert, der einem eigenen Satz Verarbeitungslogik entspricht. Es wird von dem numerischen Wert, der angibt, dass diese Logik ausgeführt werden soll, und einem Doppelpunkt (:) gefolgt. Wenn ein Satz Verarbeitungslogik abgeschlossen ist, müssen Sie das Schlüsselwort break verwenden, um die switch-Anweisung zu verlassen. Wenn Sie das nicht tun, wird die

Verarbeitung mit der Verarbeitungslogik im nächsten case in der Anweisung fortgesetzt. Durch dieses Verhalten von C/C++ können Sie einem einzelnen Satz Verarbeitungslogik mehrere Werte zuordnen:

```
switch (wert)
{
  case 0:
    verarbeitungslogik
    break;
  case 1:
  case 2:
  case 3:
    verarbeitungslogik
    break;
  default:
    verarbeitungslogik
    break;
}
```

In diesem Fall wird der zweite Satz Logik ausgeführt, wenn der Wert gleich 1, 2 oder 3 ist. Das Schlüsselwort default kennzeichnet die Programmlogik, die anzuwenden ist, wenn keiner der angegebenen Fälle zutrifft. Das Schlüsselwort default sollte immer am Ende der switch-Anweisung stehen, hinter allen anderen Fällen. Trifft keiner der mit case bezeichneten Fälle und ist kein default angegeben, so wird kein Befehl ausgeführt.

Wenn Sie in der Verarbeitungslogik eines case Variablen deklarieren, sollten Sie die gesamte Logik dieses case als Code-Block in Klammern ({}) einschließen. So gelten die Variablen nur lokal in diesem case und es werden Compilerfehler verhindert, wenn Sie die gleichen Variablennamen an anderer Stelle verwenden.

Es ist allgemein üblich, mit der Präprozessor-Direktive #define Konstanten zu deklarieren, die in der switch-Anweisung an Stelle numerischer Werte verwendet werden. So können Sie lesbare Konstantennamen verwenden, die sich auf die funktionale Bedeutung des zu vergleichenden Wertes beziehen. In Listing 6.1 werden solche definierten Konstanten verwendet. In diesem Fall waren die Konstanten allerdings schon vorher definiert, sodass Sie sich diese Arbeit nicht machen mussten.

C++-Exkurs: Die break-Anweisung

Mit der break-Anweisung, die jeden Fall in der switch-Anweisung beendet, können Sie die aktuelle Ebene der Verarbeitungslogik verlassen. Sie können mit break auch eine Schleife verlassen. Wenn Sie beispielsweise diese Logik verwenden würden:

```
for (x = 0; x < 10; x++)
{
  for (y = 0; y < 10; y++)
  {
    if (x == y)
      break;
    // Hier wird der Wert von x und y ausgegeben
    ...
  }
```

```
}
```

würde die Verarbeitung jedes Mal, wenn die Zähler in beiden Schleifen gleich sind, die innere Schleife verlassen und alle dort folgende Verarbeitungslogik überspringen. Da sich das break in der inneren Schleife befindet, würde die äußere Schleife nie unterbrochen. Wenn Sie in der Verarbeitungslogik die Werte beider Zähler anzeigten, würde die Ausgabe so aussehen:

```
x = 1 y = 0
x = 2 y = 0
x = 2 y = 1
x = 3 y = 0
x = 3 y = 1
x = 3 y = 2
x = 4 y = 0
x = 4 y = 1
...
x = 9 y = 7
x = 9 y = 8
```

Sie können mit der break-Anweisung auch do- und while-Schleifen verlassen. Diese beiden Schleifenarten lernen Sie an späteren Tagen kennen.

Standarddialogboxen

Standarddialogboxen lassen sich nicht so einfach einsetzen wie die Funktion MessageBox, trotzdem bleibt alles noch überschaubar. Die Microsoft Foundation Classes (MFC) bieten mehrere C++-Klassen für Windows-Standarddialogboxen. Tabelle 6.6 bringt eine Übersicht zu diesen Klassen.

Klasse	Dialogfeldtyp
CFileDialog	Dateiauswahl
CFontDialog	Schriftauswahl
CColorDialog	Farbauswahl
CPageSetupDialog	Seite zum Drucken einrichten
CPrintDialog	Drucken
CFindReplaceDialog	Suchen und Ersetzen

Tabelle 6.6: Klassen für Standarddialogboxen

Den in diesen Klassen verkapselten Standarddialogboxen begegnen Sie auf Schritt und Tritt in den meisten Windows-Anwendungen, wenn Sie zum Beispiel Dateien öffnen, Druckeroptionen einstellen, drucken oder Begriffe suchen und ersetzen wollen. Neben diesen Auswahlen bieten verschiedene OLE-Standarddialogklassen mehrere Standardfunktionen für OLE- oder ActiveX-Komponenten und Anwendungen.

Alle derartigen Dialogboxen setzt man in der gleichen Weise ein, auch wenn einzelne Eigenschaften und Klassenfunktionen je nach der Funktionalität des Dialogfelds variieren können. Um ein Standarddialogfeld zu verwenden, führen Sie folgende Schritte aus:

1. Deklarieren Sie eine Variable des Klassentyps.

2. Legen Sie alle erforderlichen Eigenschaften fest, bevor Sie das Dialogfeld dem Benutzer anzeigen.
3. Rufen Sie die Methode DoModal der Klasse auf, um das Dialogfeld anzuzeigen.
4. Übernehmen Sie den Rückgabewert der Methode DoModal, um zu ermitteln, ob der Benutzer auf OK oder Abbrechen geklickt hat.
5. Wenn der Benutzer auf die Schaltfläche OK geklickt hat, lesen Sie alle Eigenschaften, die der Benutzer eventuell im Dialogfeld gesetzt oder geändert hat.

Um die Funktionsweise besser verstehen zu können, fügen Sie der Beispielanwendung die Klasse CFileDialog hinzu. Nehmen Sie dazu eine Funktion für das Klickereignis der Schaltfläche Datei öffnen auf. Schreiben Sie in diese Funktion den Code aus Listing 6.2.

Listing 6.2: Die Funktion OnBnClickedBfileopen

```
 1: void CDialogsDlg::OnBnClickedBfileopen()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   CFileDialog ldFile(TRUE);
 6:
 7:   // Dialogfeld File Open zeigen und Ergebnis auffangen
 8:   if (ldFile.DoModal() == IDOK)
 9:   {
10:     // Gewählten Dateinamen ermitteln
11:     m_strResults = ldFile.GetFileName();
12:     // Dialogfeld aktualisieren
13:     UpdateData(FALSE);
14:   }
15: }
```

Diese Funktion deklariert zuerst eine Instanz der Klasse CFileDialog. Dieser Instanz wird TRUE als Argument an den Klassenkonstruktor übergeben. Der Klasse ist damit bekannt, dass es sich um eine Dialogbox Datei öffnen handelt. Bei Übergabe von FALSE erscheint eine Dialogbox Datei speichern. Zwischen beiden Dialogboxen gibt es keinen funktionellen, sondern nur einen optischen Unterschied. An den Konstruktor können Sie noch weitere Argumente übergeben und damit die anzuzeigenden Dateinamenserweiterungen, eine Startdatei und deren Standort sowie Filter zur Anzeige ausgewählter Dateien festlegen. Näheres findet sich in der Hilfe zu Visual C++.

Nachdem Sie die Instanz der Dialogbox Datei öffnen erzeugt haben, können Sie deren Funktion DoModal aufrufen. Es handelt sich dabei um eine Member-Funktion der Basisklasse CDialog. Diese Funktion ist in allen Dialogboxen verfügbar. Die Funktion DoModal zeigt die Dialogbox Datei öffnen wie in Abbildung 6.4 dargestellt an. Aus dem Rückgabewert der Funktion DoModal lässt sich ableiten, auf welche Schaltfläche der Benutzer geklickt hat. Die Wahl der Schaltfläche Datei öffnen liefert den Wert IDOK zurück, genau wie bei der Funktion MessageBox. Je nach Benutzerauswahl können Sie nun anhand des Rückgabewerts festlegen, wie das Programm fortzusetzen ist.

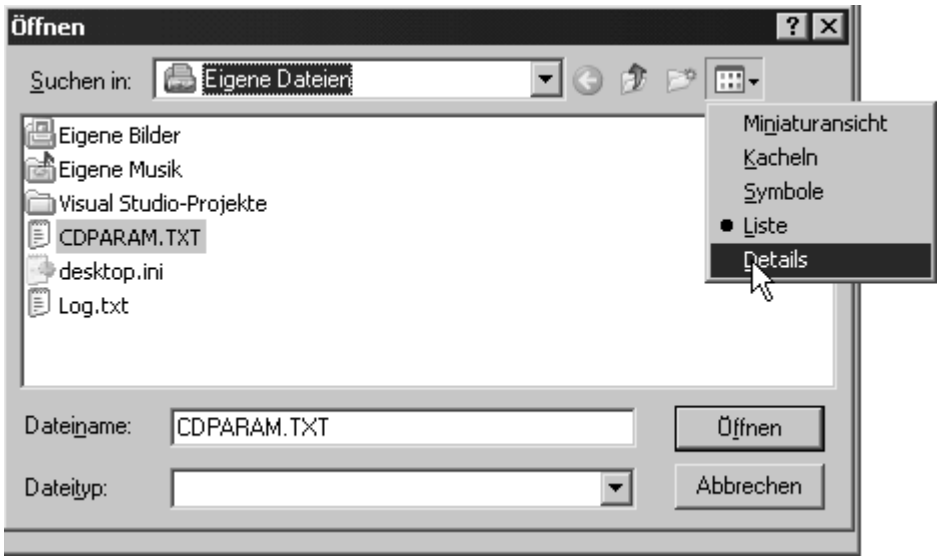


Abbildung 6.4: Die Dialogbox Öffnen

Abhängig vom verwendeten Betriebssystem kann sich die Dialogbox Datei öffnen von der in Abbildung 6.4 gezeigten unterscheiden. Die Screenshots in diesem Buch wurden alle unter Windows XP gemacht. Wenn Sie ein anderes Betriebssystem verwenden, werden die Standarddialogboxen angezeigt, die in dieser Version von Windows verfügbar sind.

Um den Namen der ausgewählten Datei anzuzeigen, setzen Sie die Variable m_strResults auf den Rückgabewert der Methode GetFileName der Klasse CFileDialog. Diese Methode liefert lediglich den Dateinamen ohne Verzeichnispfad oder Laufwerksbezeichnung, wie es Abbildung 6.5 verdeutlicht. Mit anderen Methoden der Klasse kann man auch den Verzeichnispfad (GetPathName) und die Dateinameserweiterung (GetFileExt) ermitteln.

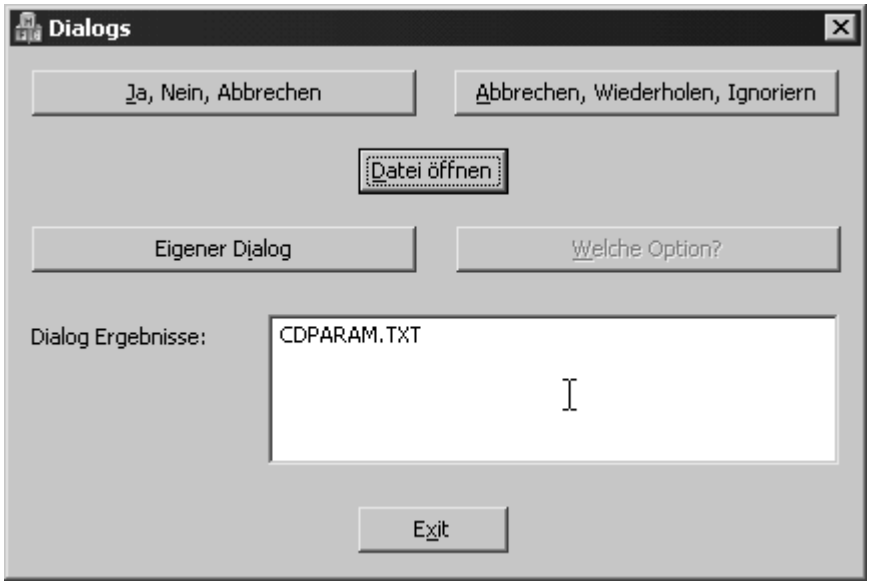


Abbildung 6.5: Den ausgewählten Dateinamen anzeigen

Sie können Dialogboxen in zwei Modi anzeigen: modal und nicht-modal. Eine modale Dialogbox verhindert während ihrer Anzeige jede andere Benutzerinteraktion mit der aufrufenden Anwendung. Der Benutzer kann in der Anwendung nichts anderes tun, bis die Dialogbox geschlossen wird. Ein gutes Beispiel für eine modale Dialogbox ist eine Meldungsbox, bei der die Benutzer erst auf die Schaltfläche klicken müssen, bevor sie in der Anwendung weiterarbeiten können.

Eine Variante hiervon bildet die systemmodale Dialogbox. Das Anzeigen einer solchen Dialogbox führt dazu, dass alle laufenden Anwendungen gestoppt bleiben, solange diese Dialogbox nicht geschlossen worden ist. Diese Funktion kann mit der Funktion MessageBox erzeugt werden, die dazu notwendigen Parameter finden sich in der Online-Hilfe. Diese Dialogboxen werden oftmals verwendet, um auf besonders kritische Zustände hinzuweisen.

Eine nicht-modale Dialogbox kann offen sein, während der Benutzer die Anwendung anderweitig verwendet; der Benutzer wird nicht am Erledigen anderer Aufgaben gehindert, während die Dialogbox geöffnet ist. Gute Beispiele für nicht-modale Dialogboxen sind die Dialoge Suchen und Suchen und Ersetzen in Microsoft Word. Sie können diese Dialogboxen offen auf dem Bildschirm behalten, während Sie im durchsuchten Dokument weiterarbeiten.

MFC-Exkurs: Die Klasse CFileDialog

Die Klasse CFileDialog verkapselt die Funktionalität der in Windows eingebauten Dialogboxen Datei öffnen und Datei speichern. Mit ihnen können Sie den Benutzern Ihrer Anwendung die gleichen Dialogboxen zur Verfügung stellen, die auch in den meisten anderen Windows-Anwendungen verwendet werden. In Tabelle 6.7 sind die wichtigsten Funktionen zusammengestellt, die mit der Klasse CFileDialog verwendet werden:

Funktion	Beschreibung
GetPathName	Gibt den vollständigen Pfad der angegebenen Datei zurück
GetFileName	Gibt den angegebenen Dateinamen zurück
GetFileExt	Gibt die Dateinamenserweiterung der angegebenen Datei zurück
GetFileTitle	Gibt den Dateinamen ohne Erweiterung zurück (wenn Sie beispielsweise die Datei »MeineDatei.txt« auswählen, gibt diese Funktion »MeineDatei« zurück

Tabelle 6.7: Wichtige Member-Funktionen von CFileDialog

Die in Tabelle 6.7 aufgelisteten Methoden sollten nicht aufgerufen werden, bevor die Methode DoModal aufgerufen wurde und den Rückgabewert IDOK zurückgeliefert hat. Die gleiche Regel gilt für alle in den nächsten Tabellen aufgelisteten Methoden für gebräuchliche Dialoge, sofern nichts anderes angegeben wird.

MFC-Exkurs: Die Klasse CFontDialog

Die Klasse CFontDialog verkapselt die in Windows eingebaute Standarddialogbox für die Schriftartenauswahl. Mithilfe dieser Klasse können Sie den Benutzern Ihrer Anwendung eine Standard-Schriftendialogbox zur Verfügung stellen, an die sie aus anderen Windows- Anwendungen gewöhnt sind. In Tabelle 6.8 sind die wichtigsten Member-Funktionen der Klasse CFontDialog zusammengestellt.

Funktion	Beschreibung
GetFaceName	Gibt den Namen der ausgewählten Schriftart zurück
GetStyleName	Gibt den Stil der ausgewählten Schriftart zurück (ein Schriftartenstil kann mehrere Schriftarten zur Verfügung stellen)
GetSize	Gibt die für die ausgewählte Schriftart festgelegte Größe zurück
GetColor	Gibt die für die ausgewählte Schriftart festgelegte Farbe zurück
GetWeight	Gibt die für die ausgewählte Schriftart festgelegte Gewichtung (Strichstärke) zurück
IsStrikeOut	Gibt einen Booleschen Wert zurück, der anzeigt, ob für die ausgewählte Schriftart das Attribut »Durchgestrichen« festgelegt wurde
IsUnderLine	Gibt einen Booleschen Wert zurück, der anzeigt, ob für die ausgewählte Schriftart das Attribut »Unterstrichen« festgelegt wurde
IsBold	Gibt einen Booleschen Wert zurück, der anzeigt, ob für die ausgewählte Schriftart das Attribut »Fett« festgelegt wurde
IsItalic	Gibt einen Booleschen Wert zurück, der anzeigt, ob für die ausgewählte Schriftart das Attribut »Kursiv« festgelegt wurde

Tabelle 6.8: Wichtige Member-Funktionen von CFontDialog

MFC-Exkurs: Die Klasse CColorDialog

Die Klasse CColorDialog verkapselt die Standarddialogbox für die Farbauswahl in vielen Windows-Anwendungen. Diese Dialogbox wird verwendet, um den RGB-Wert der angegebenen Farbe abzufragen, der dann an eine GDI-Methode übergeben werden kann, die einen Farbwert benötigt (mehr darüber lernen Sie in ein paar Tagen). In Tabelle 6.9 sind die in der Klasse CColorDialog verwendeten gebräuchlichen Methoden zusammengefasst.

Funktion	Beschreibung
GetColor	Gibt die ausgewählte Farbe zurück
GetSavedCustomColors	Gibt ein Array mit von Benutzern erstellten Farben zurück
SetCurrentColor	Stellt die aktuelle Farbauswahl ein, die angezeigt wird, wenn die Dialogbox erscheint. Rufen Sie diese Methode vor DoModal auf.

Tabelle 6.9: Wichtige Member-Funktionen von CColorDialog

MFC-Exkurs: Die Klasse CPageSetupDialog

Die Klasse CPageSetupDialog verkapselt die Dialogbox Seite einrichten, die oft mit Druckfunktionalität zusammen verwendet wird. Sie kann aufgerufen werden, um den Benutzern die Angabe des zu verwendenden Druckers, der Seitengröße und der Druckränder zu ermöglichen. In Tabelle 6.10 sind die mit der Klasse CPageSetupDialog verwendeten gebräuchlichen Methoden zusammengefasst.

Funktion	Beschreibung
CreatePrinterDC	Gibt einen Gerätekontext zurück, der zum Drucken verwendet werden kann

GetDeviceName	Gibt den Namen des ausgewählten Druckers zurück
GetDevMode	Gibt eine Struktur zurück, die Informationen über den ausgewählten Drucker und seine Fähigkeiten enthält (z.B. Farbe oder schwarz-weiß)
GetMargins	Sie übergeben dieser Methode zwei Zeiger auf entweder eine CRect-Klasse oder einer RECT-Struktur und sie füllt die Klasse oder Strukturen mit den Ausdehnungen der Ränder und des Druckbereichs.
GetPaperSize	Diese Methode gibt eine CSize-Klasse zurück, die die aktuelle Papiergröße angibt.
GetDriverName	Gibt den Namen des ausgewählten Druckertreibers zurück
GetPortName	Gibt den Namen des ausgewählten Ausgabeports zurück

Tabelle 6.10: Wichtige Member-Funktionen von CPageSetupDialog

MFC-Exkurs: Die Klasse CPrintDialog

Die Klasse CPrintDialog verkapselt den in den meisten Windows-Anwendungen verwendeten Standard-Druckdialog. Diese Klasse enthält die meisten Methoden aus CPageSetupDialog, bis auf GetPaperSize und GetMargin. Die restlichen Funktionen aus Tabelle 6.10 sind in der Klasse CPrintDialog verfügbar. In Tabelle 6.11 sind die anderen wichtigen Funktionen dieser Klasse zusammengestellt.

Funktion	Beschreibung
GetCopies	Gibt die Anzahl der auszudruckenden Kopien zurück
GetFromPage	Gibt die Nummer der Startseite zurück, wenn ein Druckbereich festgelegt wurde

Tabelle 6.11: Wichtige Member-Funktionen von CPrintDialog

Funktion	Beschreibung
GetToPage	Gibt die Nummer der letzten Seite zurück, wenn ein Druckbereich festgelegt wurde
GetPrinterDC	Gibt einen Handle zum Gerätekontext des angegebenen Druckers zurück
PrintAll	Gibt einen Booleschen Wert zurück, der anzeigt, ob alle Seiten des aktuellen Dokuments zu drucken sind
PrintCollate	Gibt einen Booleschen Wert zurück, der anzeigt, ob der Benutzer die auszudruckenden Seiten sortiert haben möchte
PrintRange	Gibt einen Booleschen Wert zurück, der anzeigt, ob ein Druckbereich angegeben wurde
PrintSelection	Gibt einen Booleschen Wert zurück, der anzeigt, ob nur die markierten Elemente oder Seiten gedruckt werden sollen

Tabelle 6.11: Wichtige Member-Funktionen von CPrintDialog (Forts.)

Anders als bei CPageSetupDialog können Sie die Methode CreatePrinterDC aufrufen, ohne vorher die Methode DoModal aufgerufen zu haben. Sie müssen die Dialogbox gar nicht anzeigen, können die Klasse jedoch trotzdem verwenden, um die Druckerinformationen und den Gerätekontext abzufragen, sodass Sie auf dem aktuellen Drucker drucken können, ohne den

Dialog anzuzeigen.

MFC-Exkurs: Die Klasse CFindReplaceDialog

Die Klasse CFindReplaceDialog verkapselt den Standarddialog zum Suchen und Ersetzen, der in vielen Windows-Anwendungen verwendet wird. Im Gegensatz zu den anderen Dialogboxen ist diese nicht-modal, sodass der Benutzer jederzeit mit dem Vaterfenster interagieren kann, solange die Dialogbox sichtbar ist. Wegen dieses Unterschieds rufen Sie nicht die Methode DoModal auf, um diesen Dialog anzuzeigen, sondern die Methode Create.

Die Methode Create erfordert zwei Parameter, kann jedoch bis zu fünf Parameter übernehmen:

- Der erste Parameter, ein Boolescher Wert, gibt an, ob es sich bei der Dialogbox nur um einen Suchen-Dialog oder um Suchen und Ersetzen handelt. Wird als erster Parameter TRUE übergeben, wird ein Suchen-Dialog angezeigt. Bei der Übergabe von FALSE wird ein Suchen und Ersetzen-Dialog angezeigt.
- Der zweite Parameter ist der String, nach dem gesucht werden soll.
- Der dritte (optionale) Parameter ist der String, durch den ersetzt werden soll.
- Der vierte Parameter gibt die Suchrichtung an. Die Standardrichtung ist abwärts, angegeben mit der Konstante FR_DOWN. Die Richtung aufwärts wird festgelegt, indem man als vierten Parameter 0 übergibt.
- Der letzte Parameter ist ein Zeiger auf das übergeordnete Fenster.

Die für die Verwendung der Klasse CFindReplaceDialog erforderlichen Programmierkonzepte sind etwas fortgeschrittener als bei den anderen gebräuchlichen Dialogklassen. Wenn die Programmiersprache C++ neu für Sie ist, werden Sie es nach weiterer Lektüre dieses Buchs leichter verstehen. Wenn Sie Schwierigkeiten haben, die Verwendung dieser Dialogklasse zu verstehen, sollten Sie diesen Abschnitt markieren und zurückkehren, nachdem Sie zumindest Tag 14 »DLLs« beendet haben.

Da es sich bei CFindReplaceDialog um einen nicht-modalen Dialog handelt, muss er in Ihrer Anwendung anders behandelt werden. Zuerst deklarieren Sie keine Variable CFindReplaceDialog, sondern einen Zeiger auf eine CFindReplaceDialog-Instanz und verwenden dann das Schlüsselwort new, um die Instanz des Dialogs zu erzeugen:

```
// In der Header-Deklaration der Klasse (eine Variable auf Klassen-
// Ebene, die in der Header-Datei deklariert wird)
CFindReplaceDialog *m_pFRDlg;
...
// Im Quellcode der Klasse (an beliebiger Stelle in der Quellcode-
// Datei, jedoch vor der ersten Verwendung)
m_pFRDlg = new CFindReplaceDialog;
```

Zweitens müssen Sie das Vaterfenster anmelden, um die Meldungen von CFindReplaceDialog zu empfangen. Sie müssen nahe dem oberen Ende der Quellcode-Datei für die Klasse, die die Meldungen erhalten soll, eine Meldungs-ID-Variable deklarieren:

```
static UINT WM_FINDREPLACE = ::RegisterWindowMessage(FINDMSGSTRING);
```

Nachdem Sie die Meldung von CFindReplaceDialog angemeldet haben, fügen Sie die Funktion zum Empfang und zur Behandlung dieser Meldungen ein. Zuerst müssen Sie die Funktion zu Ihrer Klasse hinzufügen, die die Meldungen empfängt. Die Funktion sollte mit einem Rückgabewert des Datentyps long, mit dem Zugriff protected und mit zwei Parametern definiert werden: ein Parameter vom Typ WPARAM der ohne Funktionalität ist und einen Parameter vom Typ LPARAM. Über den Wert des zweiten Parameters erhält die Meldungsfunktion eine Verbindung zur Dialogbox. Nachdem Sie Ihrer Klasse die Funktion hinzugefügt haben,

öffnen Sie die Klassendefinition (die Header-Datei) und fügen Sie vor der Funktionsdefinition das Schlüsselwort afx_msg ein. Die Klassenheader- Definition für diese Funktion sollte wie folgt aussehen:

```
class CMyClass : public CWnd
{
...
  protected:
    afx_msg long OnFindReplace(WPARAM wParam, LPARAM lParam);
    // Weitere von MFC-Assistenten hinzugefügte afx_msg-Funktionen
    DECLARE_MESSAGE_MAP();
...
};
```

Zuletzt müssen Sie, um Ihre Klasse für den Empfang von Meldungen der Klasse CFindReplaceDialog einzurichten, in der Nachrichtenzuordnungstabelle nahe dem oberen Ende Ihrer Quellcode-Datei einen Eintrag hinzufügen. Wenn Sie die Nachrichtenzuordnungstabelle betrachten, finden Sie viele Einträge, die bereits von MFC-Assistenten hinzugefügt wurden. Sie fügen mit dem Makro ON_REGISTERED_MESSAGE einen neuen Eintrag ein:

```
BEGIN_MESSAGE_MAP(CMyClass, CWnd)
...
  ON_REGISTERED_MESSAGE(WM_FINDREPLACE, OnFindReplace)
  END_MESSAGE_MAP()
```

Nun ist Ihre Klasse bereit, Ereignismeldungen von der Klasse CFindReplaceDialog zu empfangen und zu verarbeiten. Das funktioniert so, dass jedes Mal Meldungen vom Suchen/Ersetzen-Dialog ausgesandt werden, wenn ein Benutzer beispielsweise auf eine der Schaltflächen Suchen oder Ersetzen klickt. Diese Meldungen werden an die Nachrichtenzuordnungstabelle weitergeleitet und durch das Makro ON_REGISTERED_MESSAGE zu der Funktion, die Sie für die Behandlung dieser Meldungen definiert haben (siehe Abbildung 6.6) weitergereicht.

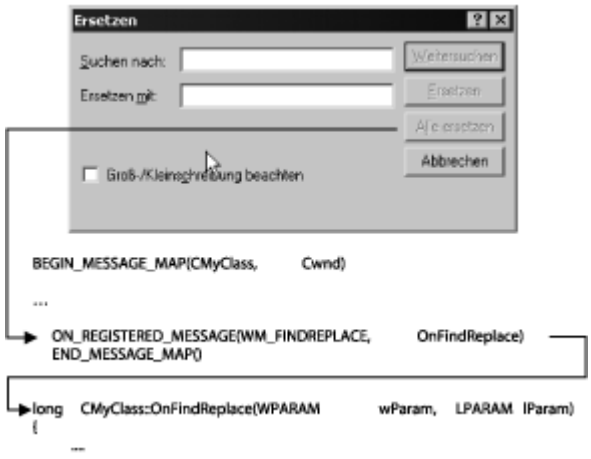


Abbildung 6.6: Die Weiterleitung einer Suchen/Ersetzen-Meldung

Die wichtigsten Member-Funktionen der Klasse CFindReplaceDialog, die Sie in ihrer Funktion zur Behandlung der Meldungen verwenden werden, um festzustellen, welche Aktion der Benutzer ausgeführt hat, sind in Tabelle 6.12 zusammengestellt.

Funktion	Beschreibung
FindNext	Gibt einen Booleschen Wert zurück, der anzeigt, ob der Benutzer die nächste Stelle finden möchte, an welcher der angegebene String auftritt
GetFindString	Gibt den vom Benutzer eingegebenen zu suchenden String zurück
GetReplaceString	Gibt den vom Benutzer eingegebenen String zurück, durch den der zu suchende String ersetzt werden soll
IsTerminating	Gibt einen Booleschen Wert zurück, der anzeigt, ob der Benutzer den Suchen/Ersetzen-

	Dialog schließen will
MatchCase	Gibt einen Booleschen Wert zurück, der anzeigt, ob bei der Suche Groß-/Kleinschreibung berücksichtigt werden soll
MatchWholeWord	Gibt einen Booleschen Wert zurück, der anzeigt, ob nur ganze Worte gesucht werden sollen, die dem Suchstring entsprechen
ReplaceAll	Gibt einen Booleschen Wert zurück, der anzeigt, ob der Suchstring an allen Stellen seines Auftretens ersetzt werden soll
ReplaceCurrent	Gibt einen Booleschen Wert zurück, der anzeigt, ob der Suchstring an der aktuellen Stelle seines Auftretens ersetzt werden soll
SearchDown	Gibt einen Booleschen Wert zurück, der anzeigt, ob von der aktuellen Position in den Daten aus abwärts gesucht werden soll

Tabelle 6.12: Wichtige Member-Funktionen des CFindReplaceDialogs

6.2 Eigene Dialogboxen erstellen

Mit Standarddialogboxen können Sie nun umgehen. Was aber, wenn Sie ein benutzerdefiniertes Dialogfeld brauchen? Diese Aufgabe ist einfach zu lösen, da es sich im Wesentlichen um eine Kombination der Schritte handelt, die Sie bereits vom Erstellen des Hauptdialogfelds der Beispielanwendungen und vom Einsatz von Standarddialogboxen her kennen. Es sind zwar ein paar zusätzliche Schritte erforderlich. Diese sollten Ihnen jedoch prinzipiell geläufig sein.

Die Dialogbox erzeugen

Die benutzerdefinierte Dialogbox für die Beispielanwendung stellt dem Benutzer ein Eingabefeld bereit, in das er Text eingeben sowie eine Gruppe von Optionsfeldern, aus denen er eine Option auswählen kann. Wenn der Benutzer auf OK klickt, zeigt die Anwendung den vom Benutzer eingegebenen Text im Anzeigebereich der Hauptdialogbox der Anwendung an. Über eine weitere Schaltfläche kann sich der Benutzer anzeigen lassen, welches der Optionsfelder ausgewählt ist. Diese Übung zeigt, wie man mit benutzerdefinierten Dialogboxen Informationen vom Benutzer einholt und die vom Benutzer getroffenen Angaben auswertet, nachdem die Dialogbox geschlossen wird.

Um eine benutzerdefinierte Dialogbox für die Anwendung zu erstellen, sind folgende Aufgaben zu erledigen:

- eine weitere Dialogbox zu den Ressourcen der Anwendung hinzufügen,
- das Layout der Dialogbox gestalten,
- die Basisklasse deklarieren, von der die Dialogbox abgeleitet wird,
- Variablen mit den Steuerelementen in der Dialogbox verbinden.

Danach können Sie die benutzerdefinierte Dialogbox in Ihrer Anwendung einsetzen. Die genannten Aufgaben führen Sie in folgenden Schritten aus:

1. Wählen Sie die Ressourcenansicht aus.

2. Klicken Sie mit der rechten Maustaste auf den Ordner Dialog und wählen Sie Dialog einfügen.
3. Ändern Sie die Objekt-ID für das neue Dialogfeld in IDD_MESSAGEDLG.
4. Wenn Sie die neue Dialogbox bearbeiten, behalten Sie die Schaltflächen OK und Abbrechen bei. Verschieben Sie sie auf eine andere Position, wie es Abbildung 6.7 zeigt und fügen Sie die weiteren Dialogelemente hinzu.
5. Gestalten Sie mit den Objekteigenschaften in Tabelle 6.13 den Rest des Fensters.

Objekt	**Eigenschaft**	**Einstellung**
Dialog (gesamtes Fenster)	Beschriftung (Titel)	Nachrichten und Optionen Dialog
Text	Beschriftung	Schreiben Sie eine &Nachricht

Eingabefeld	ID Mehrfachzeile Auto Vscroll	IDC_EMESSAGE True True
Gruppenfeld	Titel	Wählen Sie eine Option
Optionsfeld	ID Beschriftung Gruppe	IDC_ROPTION1 &Option 1 True
Optionsfeld	ID Beschriftung	IDC_ROPTION2 O&ption 2
Optionsfeld	ID Beschriftung	IDC_ROPTION3 Op&tion 3
Optionsfeld	ID Beschriftung	IDC_ROPTION4 Opt&ion 4

Tabelle 6.13: Eigenschaftseinstellungen der Steuerelemente für das benutzerdefinierte Dialogfeld

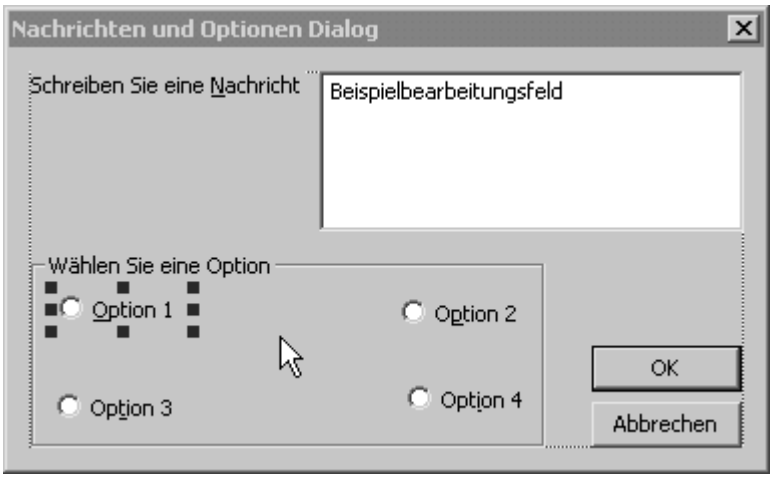


Abbildung 6.7: Das Layout des benutzerdefinierten Dialogfelds

6. Wenn Sie den Entwurf der Dialogbox abgeschlossen haben, klicken Sie mit der rechten Maustaste und wählen Sie aus dem Kontextmenü Klasse hinzufügen.
7. Tragen Sie in das Feld Name die Bezeichnung CMsgDlg ein und vergewissern Sie sich, dass im Listenfeld Basisklasse der Eintrag CDialog ausgewählt ist (siehe Abbildung 6.8).
8. Lassen Sie die anderen Einstellungen im Dialogfeld unverändert und klicken Sie auf Fertig stellen.
9. Nachdem die neue Klasse erstellt ist, weisen Sie den Steuerelementen in der neuen Dialogbox die Variablen entsprechend Tabelle 6.14 zu.

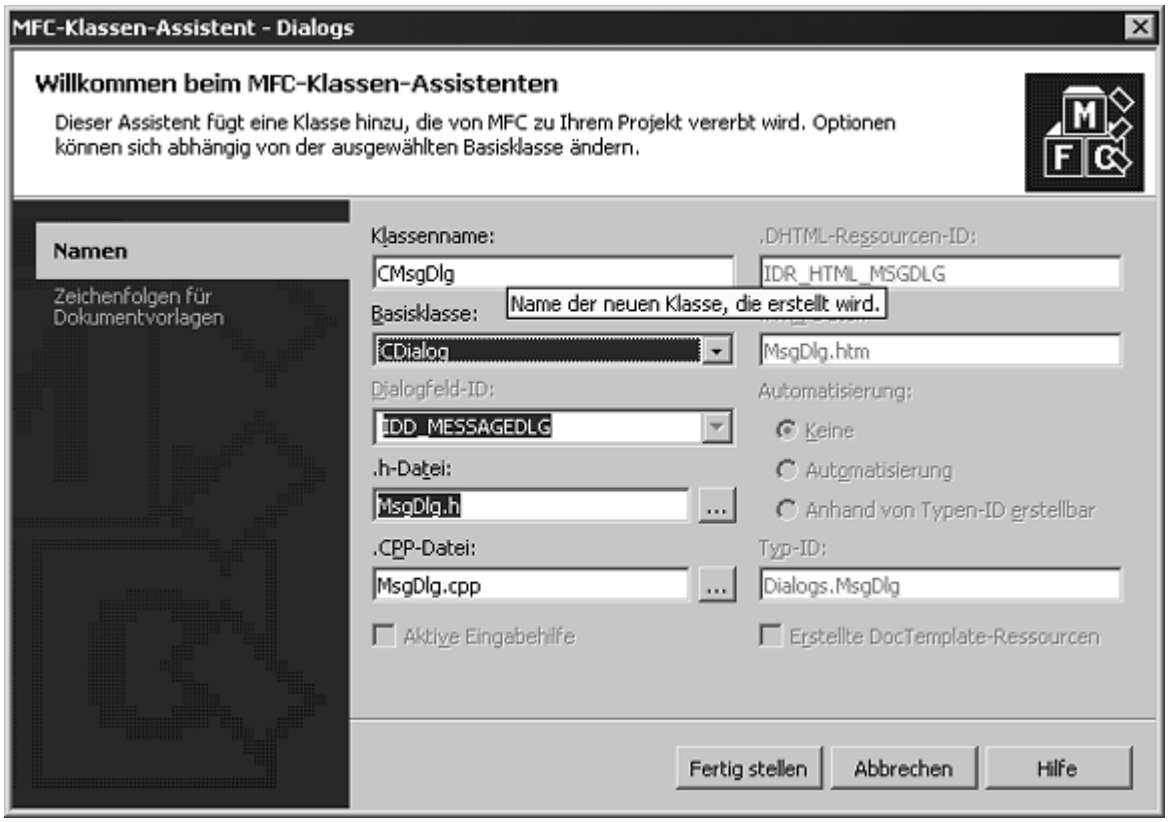


Abbildung 6.8: Das Dialogfeld MFC-Klassenassistent

Objekt	Name	Kategorie	Typ	Zugriff
IDC_EMESSAGE	m_strMessage	Value	CString	public
IDC_ROPTION1	m_iOption	Value	int	public

Tabelle 6.14: Variablen der Steuerelemente

Zwei Dinge sind zu beachten, wenn Sie die Steuerelementeigenschaften und Variablen im benutzerdefinierten Dialogfeld konfigurieren. Zum einen sollten Sie nur für das erste Optionsfeld die Eigenschaft Gruppe einschalten. Damit wird gekennzeichnet, dass alle sich anschließenden Optionsfelder zu einer einzigen Gruppe gehören, in der nur ein Optionsfeld zu einem bestimmten Zeitpunkt ausgewählt sein kann. Wenn Sie die Eigenschaft Gruppe für alle Optionsfelder einschalten, sind die Optionsfelder voneinander unabhängig, sodass man mehrere Optionen gleichzeitig auswählen kann. Damit entspricht das Verhalten der Optionsfelder dem von Kontrollkästchen, wobei der Hauptunterschied darin besteht, dass der Benutzer die Optionen nicht wieder ausschalten kann. Das widerspricht dem Standardverhalten, nach dem immer nur eine Option markiert ist. Der andere Unterschied bezieht sich auf das Erscheinungsbild. Optionsfelder weisen runde Auswahlbereiche auf, während Kontrollkästchen quadratisch sind.

Wenn der Gruppe von Optionsfeldern noch weitere Steuerelemente folgen, ist es eine gute Idee, die Option Gruppe beim ersten Steuerelement nach den Optionsfeldern als TRUE zu kennzeichnen. So machen Sie klar, dass keine weiteren Steuerelemente zur Gruppe der Optionsfelder gehören.

Als Nächstes fällt auf, dass nur eine einzige Integer-Variable für die Optionsfelder deklariert wurde und zwar für das Optionsfeld, für das die Eigenschaft Gruppe eingeschaltet ist. Der Wert dieser Variablen hängt davon

ab, welches Optionsfeld ausgewählt ist. Handelt es sich um das erste Optionsfeld, hat die Variable den Wert 0, beim zweiten Optionsfeld den Wert 1 usw. Umgekehrt können Sie bestimmen, welches Optionsfeld ausgewählt sein soll, indem Sie den Wert der Variablen festlegen (auf die Nummer des Optionsfelds in der Gruppe minus 1).

In der Programmiersprache C++ beginnen alle Nummerierungen mit 0 und nicht mit 1. Die erste Position in einem Array (Feld) oder in einer Gruppe von Steuerelementen ist demnach an Position 0, die zweite an Position 1, das dritte Element hat den Index 2 usw.

Es sind nun alle Arbeiten abgeschlossen, um die Dialogbox in der Anwendung einzusetzen. Vielleicht erwarten Sie, dass die Funktion UpdateData im Code für die Dialogbox aufzurufen ist. Da Sie aber die Schaltflächen OK und Abbrechen nicht aus der Dialogbox entfernt haben, findet der Aufruf von UpdateData automatisch statt, sobald der Benutzer auf OK klickt. Damit brauchen Sie für die zweite Dialogbox überhaupt keinen Code zu verfassen, sondern nur für die erste.

Die Dialogbox in der Anwendung einsetzen

Nachdem Sie Ihre benutzerdefinierte Dialogbox fertig gestellt haben, können Sie sie genauso einsetzen wie die zu Windows gehörenden Standarddialogboxen.

1. Deklarieren Sie eine Instanz der benutzerdefinierten Dialogboxklasse, die den Konstruktor der Klasse aufruft und eine Instanz der Klasse erzeugt.

2. Rufen Sie die Methode DoModal der Dialogbox auf und speichern Sie den Rückgabewert der Funktion in einer Variablen.
3. Werten Sie die Variablen aus, die Sie den Steuerelementen der Dialogbox zugeordnet haben.

Die Instanz des Dialogfelds erzeugen

Bevor Sie das benutzerdefinierte Dialogfeld in der Anwendung nutzen können, müssen Sie dem Hauptdialogfeld der Anwendung das benutzerdefinierte Dialogfeld sowie dessen Variablen und Methoden bekannt machen und festlegen, wie das Hauptdialogfeld mit ihm in Wechselwirkung treten kann. Das lässt sich erreichen, indem Sie die Header-Datei für das benutzerdefinierte Dialogfeld in die Quelldatei für das Hauptdialogfeld der Anwendung einbinden. Führen Sie die folgenden Schritte aus:

1. Wählen Sie die Ansicht Projektmappen-Explorer.

2. Erweitern Sie die Ordner Dialogs (der Applikationsname den sie vorhin im Anwendungsassistenten eingegeben hatten) und danach Quelldateien.
3. Doppelklicken Sie auf die Datei DialogsDlg.cpp. Daraufhin erscheint der Quellcode für das Hauptdialogfeld der Anwendung im Bearbeitungsbereich von Visual Studio.
4. Gehen Sie an den Beginn des Quellcodes, wo sich die #include-Anweisungen befinden, und fügen Sie eine #include-Anweisung für die Datei MsgDlg.h vor DialogsDlg.h ein, wie es Listing 6.3 zeigt.

Listing 6.3: Die benötigten #include-Anweisungen

```
1: // DialogsDlg.cpp : Implementierungsdatei
2: //
3:
4: #include "stdafx.h"
5: #include "Dialogs.h"
6: #include "MsgDlg.h"
7: #include "DialogsDlg.h"
```

Die #include-Anweisung für die Datei MsgDlg.h muss vor der #include-Anweisung für die Datei DialogsDlg.h

stehen, weil Sie eine Variablendeklaration für die benutzerdefinierte Dialogbox in die Klasse des Hauptdialogfelds in der Header-Datei des Hauptdialogfelds aufnehmen. Wenn Sie die Header-Datei MsgDlg.h nach derjenigen für das Hauptdialogfeld einbinden, beschwert sich der Compiler und weigert sich, Ihre Anwendung zu kompilieren, bis Sie die #include-Anweisung der Datei MsgDlg.h vor die #include- Anweisung der Datei DialogsDlg.h verschieben.

In den Programmiersprachen C und C++ bezeichnet man die #include-Anweisung als Präprozessor-Direktive. Sie weist den Präprozessor an, den Inhalt der angegebenen Datei in den zu kompilierenden Quellcode einzulesen. Man verwendet diese Direktiven, um Deklarationen von Klassen, Strukturen und Funktionen in separate Dateien zu schreiben, die sich in jeden Quellcode aufnehmen lassen, der auf die Informationen in der Header-Datei angewiesen ist. Weitere Informationen zur Funktionsweise der #include-Anweisungen und warum man sie verwendet, finden Sie im Kapitel zu Tag 2.

Nachdem Sie nun dem Hauptdialogfeld der Anwendung die benutzerdefinierte Dialogbox bekannt gemacht haben, müssen Sie eine Variable Ihrer benutzerdefinierten Dialogbox deklarieren. Führen Sie dazu folgende Schritte aus:

1. Wählen Sie die Klassenansicht aus.

2. Klicken Sie mit der rechten Maustaste auf die Klasse CDialogsDlg.
3. Wählen Sie aus dem Kontextmenü Hinzufügen / Variable hinzufügen.
4. Legen Sie den Variablentyp mit CMsgDlg, den Variablennamen mit m_dMsgDlg und den Zugriffsstatus als private fest. Klicken Sie auf Fertig stellen, um die Variable in Ihr Hauptdialogfeld aufzunehmen.

Wenn Sie die Klasse CDialogsDlg in der Baumansicht erweitern, ist die Instanz der benutzerdefinierten Dialogbox als Element der Hauptdialogfeldklasse der Anwendung zu sehen. Das bedeutet, dass Sie nun die benutzerdefinierte Dialogbox in Ihrer Anwendung nutzen können.

Das Dialogfeld aufrufen und die Variablen lesen

Sie haben nun die benutzerdefinierte Dialogbox in das Hauptdialogfeld der Anwendung aufgenommen und zwar als Variable, die immer verfügbar ist, und nicht einfach als lokale Variable, die nur innerhalb einer einzigen Funktion gültig ist (wie bei der Variablen CFileDialog). Jetzt können Sie den Code hinzufügen, um das Dialogfeld tatsächlich zu nutzen. Führen Sie dazu die folgenden Schritte aus:

1. Fügen Sie eine Funktion für das Klickereignis (BN_CLICKED) der Schaltfläche IDC_BCUSTOMDIALOG hinzu.

2. Fügen Sie eine Funktion für das Klickereignis (BN_CLICKED) der Schaltfläche IDC_BWHICHOPTION hinzu.
3. Übernehmen Sie in die Funktion OnBnClickedBcustomdialog den Code aus Listing 6.4.

Listing 6.4: Die Funktion OnBnClickedBcustomdialog

```
 1: void CDialogsDlg::OnBnClickedBcustomdialog()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Meldungsdialog anzeigen und Ergebnis auffangen
 6:   if (m_dMsgDlg.DoModal() == IDOK)
 7:   {
 8:     // OK geklickt, Nachricht anzeigen, die Benutzer im
 9:     // Nachrichtendialog eingegeben hat
10:     m_strResults = m_dMsgDlg.m_strMessage;
11:     // Dialogfeld aktualisieren
```

```
12:     UpdateData(FALSE);
13:     // Schaltfläche Welche Option aktualisieren
14:     m_cWhichOption.EnableWindow(TRUE);
15:   }
16: }
```

4. Übernehmen Sie in die Funktion OnBnClickedBwhichoption den Code aus Listing 6.5.

Listing 6.5: Die Funktion OnBnClickedBwhichoption

```
 1: void CDialogsDlg::OnBnClickedBwhichoption()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Gewähltes Optionsfeld ermitteln und mit einer
 6:   // Meldung die gewählte Option anzeigen.
 7:   switch(m_dMsgDlg.m_iOption)
 8:   {
 9:     case 0:    // Erstes Optionsfeld gewählt?
10:       m_strResults = "Die erste Option wurde selektiert.";
11:       break;
12:     case 1:    // Zweites Optionsfeld gewählt?
13:       m_strResults = "Die zweite Option wurde selektiert.";
14:       break;
15:     case 2:    // Drittes Optionsfeld gewählt?
16:       m_strResults = "Die dritte Option wurde selektiert.";
17:       break;
18:     case 3:    // Viertes Optionsfeld gewählt?
19:       m_strResults = "Die vierte Option wurde selektiert.";
20:       break;
21:     default:   // Keines der Optionsfelder ausgewählt?
22:       m_strResults = "Keine Option wurde selektiert.";
23:       break;
24:   }
25:
26:   // Dialogfeld aktualisieren
27:   UpdateData(FALSE);
28: }
```

Der Code in Listing 6.4 ruft die Methode DoModal der benutzerdefinierten Dialogbox auf. Diese Methode zeigt die Dialogbox an und wartet darauf, dass der Benutzer auf eine der beiden Schaltflächen in der Dialogbox klickt, wie es Abbildung 6.9 zeigt. Wählt der Benutzer die Schaltfläche OK, kopiert der Code die vom Benutzer in die benutzerdefinierte Dialogbox eingegebene Nachricht in die Variable des Eingabefelds, das die Nachricht für den Benutzer anzeigen soll. Nach der Aktualisierung der Dialogboxanzeige mit den neuen Variablenwerten aktiviert die Funktion die Schaltfläche Welche Option wie in Abbildung 6.10. Entscheidet sich der Benutzer für die Schaltfläche Abbrechen, wird gar nichts unternommen. Die Dialogboxanzeige ändert sich nicht.

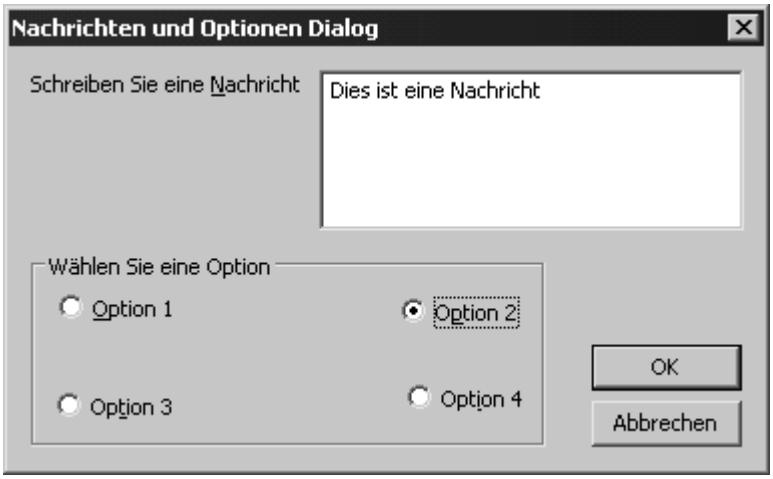


Abbildung 6.9: In der benutzerdefinierten Dialogbox kann der Benutzer eine Nachricht eintippen.

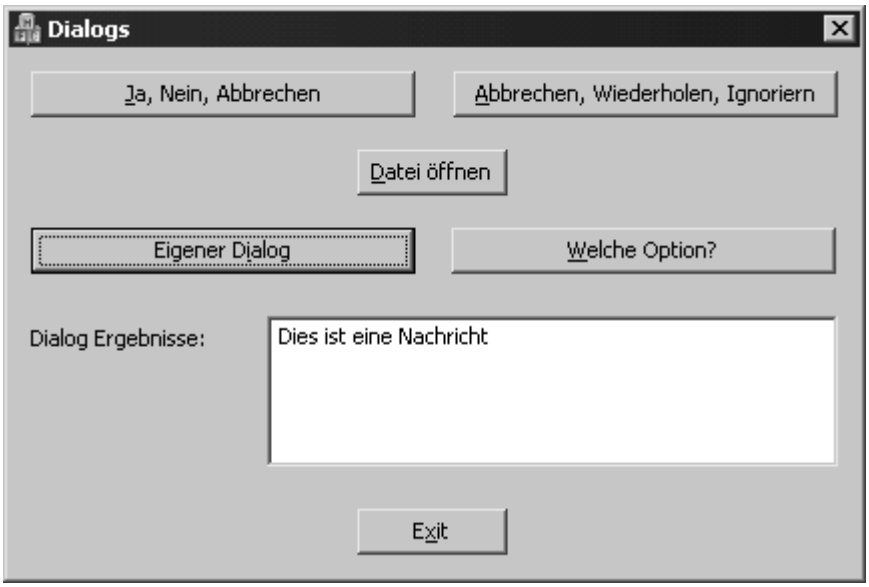


Abbildung 6.10: Die in der benutzerdefinierten Dialogbox eingegebene Nachricht wird dem Benutzer angezeigt.

Wenn der Benutzer auf die Schaltfläche Welche Option klickt, übergeben Sie die Variable des Optionsfelds in der benutzerdefinierten Dialogbox an eine switch-Anweisung, um eine Meldung auszuwählen, die dem Benutzer das gewählte Optionsfeld anzeigt (siehe Abbildung 6.11). Beachten Sie, dass man in beiden Funktionen direkt auf die Variablen der Steuerelemente in der benutzerdefinierten Dialogbox vom Hauptdialogfeld aus zugreifen kann. Das ist darauf zurückzuführen, dass die mit den Steuerelementen verbundenen Variablen als public deklariert sind und sie somit für einen Zugriff von außerhalb der Dialogfeldklasse zugänglich gemacht wurden. Dieses Verhalten können Sie ändern, indem Sie in der Klassenheader-Datei den Zugriffsspezifizierer public durch private ersetzen.

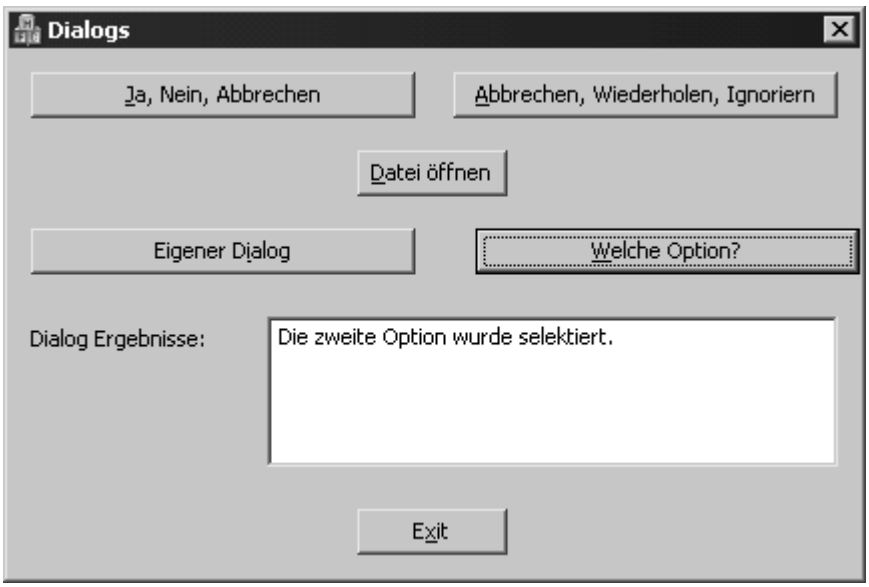


Abbildung 6.11: Die in der benutzerdefinierten Dialogbox gewählte Option wird dem Benutzer angezeigt.

6.3 Zusammenfassung

Heute haben Sie gelernt, wie man zusätzliche Dialogboxen in eine Anwendung einbaut, um die Interaktion des Benutzers mit der Anwendung zu ermöglichen. Es wurde gezeigt, welche Optionen für die einfache MessageBox-Funktion verfügbar sind, wie man dem Benutzer verschiedene Schaltflächenkombinationen

anbietet und wie man die vom Benutzer getroffene Auswahl bestimmt. Anhand dieser Informationen lässt sich der weitere Ablauf eines Programms festlegen.

Im nächsten Abschnitt ging es um Standarddialogboxen, die in das Betriebssystem Windows integriert sind. Es wurde dargestellt, wie diese Standarddialoge in C++-Klassen der MFC-Klassenbibliothek verkapselt sind. Sie haben gelernt, wie man dem Benutzer die Auswahl von Dateien mit dem Standarddialogfeld Datei öffnen ermöglicht und wie man ermitteln kann, welche Datei der Benutzer ausgewählt hat.

Schließlich haben Sie erfahren, wie man eigene Dialogboxen entwirft und sie in eine Anwendung einbaut, um Informationen vom Benutzer einzuholen und diese Informationen in der Anwendung auszuwerten.

6.4 Workshop

Fragen und Antworten

Frage:
Für das benutzerdefinierte Dialogfeld wurde kein Code geschrieben. Muss ich meine benutzerdefinierten Dialogboxen immer auf diese Weise erstellen oder kann ich für die Dialogboxen Code vorsehen?

Antwort:
Die benutzerdefinierten Dialogboxen unterscheiden sich nicht von den Hauptdialogboxen, die Sie in den bisherigen Anwendungen eingesetzt haben. Wenn Sie das Verhalten des Dialogfelds interaktiv beeinflussen möchten, können Sie so viel Code hinzufügen wie Sie wollen. Im benutzerdefinierten Dialogfeld der heutigen Beispielanwendung war kein Code zu schreiben, da er einfach nicht erforderlich war. Das Dialogfeld hatte lediglich die Funktion UpdateData aufzurufen, was aber die Funktion OnOK automatisch beim Schließen des Dialogfelds übernimmt. Da Sie weder die Schaltfläche OK noch die Schaltfläche Cancel aus dem Dialogfeld gelöscht haben, ist diese Funktionalität bereits vorhanden.

Frage:
Was passiert, wenn ich zwei oder mehr Schaltflächenkombinationen im selben Aufruf der Funktion MessageBox angebe?

Antwort:
Die Anwendung lässt sich problemlos kompilieren, aber beim Aufruf der Funktion MessageBox ereignet sich manchmal überhaupt nichts, manchmal wird eine der ausgewählten Schaltflächenkombinationen angezeigt. Denken Sie daran, dass nur eine Schaltflächenkombination angezeigt wird, weshalb es sinnlos ist, zwei oder mehr Kombinationen mit OR zu verknüpfen.

Frage:
Wie kann ich das Dialogfeld Datei öffnen in meine Anwendung einbauen, damit es bereits mit einem vorgegebenen Verzeichnis geöffnet wird?

Antwort:
Die Klasse CFileDialog verfügt über eine öffentliche (public) Eigenschaft namens m_ofn. Diese Eigenschaft ist als Struktur realisiert, die mehrere Attribute des Dialogfelds Datei öffnen enthält. Dazu gehört auch das anfänglich angezeigte Verzeichnis. Die Struktur ist als OPENFILENAME-Struktur in Listing 6.6 definiert, eine genau Erklärung der einzelnen Strukturelemente findet sich in der Online- Hilfe.

Listing 6.6: Die Struktur OPENFILENAME

```
typedef struct tagOFN {  // ofn
  DWORD         lStructSize;
  HWND          hwndOwner;
  HINSTANCE     hInstance;
  LPCTSTR       lpstrFilter;
  LPTSTR        lpstrCustomFilter;
  DWORD         nMaxCustFilter;
  DWORD         nFilterIndex;
  LPTSTR        lpstrFile;
  DWORD         nMaxFile;
```

```
  LPTSTR        lpstrFileTitle;
  DWORD         nMaxFileTitle;
  LPCTSTR       lpstrInitialDir;
  LPCTSTR       lpstrTitle;
  DWORD         Flags;
  WORD          nFileOffset;
  WORD          nFileExtension;
  LPCTSTR       lpstrDefExt;
  DWORD         lCustData;
  LPOFNHOOKPROC lpfnHook;
  LPCTSTR       lpTemplateName;
}  OPENFILENAME;
```

Antwort:
Diese Attribute können Sie festlegen, bevor Sie die Methode DoModal der Dialogfeldklasse aufrufen, um das Verhalten des Dialogfelds Datei öffnen zu bestimmen. Wenn Sie zum Beispiel wie in Listing 6.7 das Startverzeichnis auf C:\Temp setzen, bevor Sie DoModal aufrufen, erscheint das Dialogfeld Öffnen mit diesem ausgewählten Verzeichnis.

Listing 6.7: Die überarbeitete Funktion OnBnClickedBfileopen

```
 1: void CDialogsDlg::OnBnClickedBfileopen(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
      // Benachrichtigung ein.
 4:   CFileDialog ldFile(TRUE);
 5:
 6:   // Anfangsverzeichnis initialisieren
 7:   ldFile.m_ofn.lpstrInitialDir = "C:\\Temp\\";
 8:
 9:   // Dialogfeld anzeigen und Ergebnis auffangen
10:   if (ldFile.DoModal() == IDOK)
11:   {
12:     // Gewählten Dateinamen ermitteln
13:     m_strResults = ldFile.GetFileName();
14:     // Dialogfeld aktualisieren
15:     UpdateData(FALSE);
16:   }
17: }
```

Wenn Sie die Verwendung der doppelten Backslashes im String mit dem Verzeichnisnamen im obigen Listing irritiert, lesen Sie in Tabelle 2.3 im Kapitel zu Tag 2 die Erklärung zu Sonderzeichen in C-/C++-Strings nach. Um eine möglichst große Quellcode-Kompatibilität zwischen UNIX und Windows herzustellen akzeptieren alle Dateifunktionen auch den Schrägstrich '/' in Verzeichnisangaben, die obige Zeile könnte somit auch wie folgt geschrieben werden:

```
ldFile.m_ofn.lpstrInitialDir = "C:/Temp/";
```

Quiz

1. Wie lauten die möglichen Rückgabewerte, die Ihre Anwendung aus einem Aufruf der Funktion MessageBox erhält, wenn Sie die Schaltflächenkombination MB_RETRYCANCEL übergeben?

2. Welche Standarddialogboxen des Betriebssystems Windows sind als MFC-Klassen definiert?
3. Worin liegt der Unterschied zwischen einer modalen und einer nicht-modalen Dialogbox?
4. Wie lässt sich statt der in der heutigen Beispielanwendung eingesetzten Dialogbox Datei öffnen die

Dialogbox Datei speichern anzeigen?

5. Warum war es nicht erforderlich, für die benutzerdefinierte Dialogbox Funktionen oder Code zu erstellen?

Übungen

1. Modifizieren Sie Ihre Anwendung, damit das Verzeichnis zusammen mit dem Dateinamen erscheint. (Hinweis: Die Funktion GetPathName liefert den Pfad und den Dateinamen, die in der Dialogbox Datei öffnen ausgewählt wurden.)

2. Nehmen Sie in die benutzerdefinierte Dialogbox eine Schaltfläche auf, die den Aufruf der Funktion MessageBox mit den Auswahlen Ja und Nein bewirkt. Übergeben Sie das Ergebnis an das Hauptdialogfeld der Anwendung.

Tag 7

Menüs

Die meisten Windows-Anwendungen lassen sich über Pulldown-Menüs bedienen. Auf diese Weise kann man dem Benutzer eine umfangreiche Palette von Funktionen bereitstellen, ohne dass dafür Schaltflächen im Fenster vorhanden sein müssen. Der wertvolle Platz auf dem Bildschirm lässt sich damit für andere Dinge freihalten.

Heute lernen Sie, wie man ...

- Menüs für eine Visual C++-Anwendung erstellt,
- ein Menü mit dem Hauptdialogfeld einer Anwendung verbindet,
- Funktionen der Anwendung über ein Menü aufruft,
- ein Kontextmenü erzeugt, das sich durch Drücken der rechten Maustaste öffnen lässt,
- Zugriffstasten einrichtet, um die Menübefehle über die Tastatur aufzurufen.

7.1 Menüs

Als die ersten Computer-Terminals eingeführt wurden und sich die Benutzer mit Computer-Software auseinander zu setzen begannen, stellten sogar die Entwickler von großen Mainframe-Systemen fest, dass man dem Benutzer in irgendeiner Form Menüs anbieten sollte, aus denen sich die von der Software auszuführenden Funktionen auswählen lassen. Diese frühen Menüs waren im Vergleich mit heutigen Standards primitiv und ließen sich nur schwer bedienen. Seit diesen Anfangstagen haben sich die Menüs enorm verbessert. Außerdem haben sich bestimmte Standards für die Gestaltung und Bedienung durchgesetzt, die das Einarbeiten in neue Programme erleichtern.

Die Software-Entwickler, die den Gedanken der grafischen Benutzeroberflächen (Graphical User Interface - GUI) umsetzen, zielen mit einheitlichen Verhaltensmustern der Elemente einer Anwendung darauf ab, Computersysteme und Anwendungen verständlicher und benutzerfreundlicher zu gestalten. Menüs zur Auswahl der Anwendungsfunktionen bilden dabei einen Teil der grafischen Benutzeroberfläche, der sich leichter erlernen lässt, wenn alle in gleicher Weise arbeiten. Das Ergebnis war die Entwicklung einer Reihe von Menüstilen.

Menüstile

Als erste Menüstile wurden Pulldown- und überlappende Menüs standardisiert. Bei diesen Menüs sind die Kategorien in einer Zeile am oberen Rand des Anwendungsfensters angeordnet. Wenn man eine dieser Kategorien auswählt, öffnet sich eine Liste unter der Kategorie. Hier kann man aus einer Anzahl von Einträgen (oder Befehlen) wählen, die verschiedene Funktionen in der Anwendung auslösen.

Eine Variante dieses Menüstils ist das überlappende Menü, bei dem sich ein weiteres Untermenü rechts neben einem Menüeintrag öffnet. Dieses Untermenü ist dem Pulldown-Menü ähnlich und weist eine Reihe von Einträgen auf, die Anwendungsfunktionen auslösen. Die Menüentwickler haben keine Grenzen gesetzt, wie tief die überlappenden Menüs verschachtelt werden können. Allerdings wurde schnell klar, dass mehr als zwei überlappende Menüebenen etwas unhandlich sind. Ein Beispiel für mehrstufige Schachtelungen ist das Startmenü von Windows.

Schließlich wurde ein dritter Menüstil entwickelt, das so genannte Popup- oder Kontextmenü - ein Menü, das sich unter dem Mauszeiger öffnet und frei über den gesamten Arbeitsbereich der Anwendung verschiebbar ist. Die Bezeichnung Kontextmenü rührt daher, dass das spezielle Menü vom markierten Objekt oder Arbeitsbereich abhängig ist, bei dem sich der Cursor oder Mauszeiger gerade befindet.

Tastenkombinationen - Menüauswahlen aktivieren

Wenn ein Benutzer in einer Anwendung vorrangig mit der Tastatur arbeitet, beispielsweise bei einer Textverarbeitung, ist eine sinkende Produktivität festzustellen, wenn er die Hände von der Tastatur nehmen muss, um einen Menübefehl mit der Maus auszuwählen. Aus diesem Grund sehen die Software-Entwickler bestimmte Tastenkombinationen für verschiedene Menübefehle vor (insbesondere für die am häufigsten genutzten Menüauswahlen). Bei diesen speziellen Tasten und Tastenkombinationen spricht man von Schnelltasten (accelerator keys), Zugriffstasten und Hotkeys.

Die Hotkeys werden in einem Menübefehl durch unterstrichene Buchstaben gekennzeichnet. Wenn man die (Alt)-Taste und dazu die Taste des unterstrichenen Buchstabens drückt, wählt man den Menüeintrag aus, dem dieser Buchstabe zugeordnet ist. Auf diese Weise kann man durch die Anwendungsmenüs navigieren, ohne die Hände von der Tastatur nehmen zu müssen.

In neueren Versionen von Windows wird die Unterstreichung gewöhnlich erst angezeigt, wenn man die (Alt)-Taste drückt.

Für erfahrene Benutzer haben die Anwendungsentwickler Acceleratorkeys (Schnelltasten) oder Zugriffstasten vorgesehen. Eine Zugriffstaste ist eine einzelne Tastenkombination, über die man eine Anwendungsfunktion direkt auslösen kann, statt sich durch die Anwendungsmenüs zu arbeiten. Erfahrene Benutzer können auf diese Weise zügiger arbeiten, weil sie häufig genutzte Anwendungsfunktionen nicht über den Umweg einer Menüauswahl aufrufen müssen. Damit sich der Benutzer die Zugriffstasten praktisch im Vorübergehen einprägen kann, sind die Tastenkombinationen rechts neben dem betreffenden Menüeintrag aufgeführt. Oftmals werden die Funktionstasten (F1) bis (F12) für diese Funktionen verwendet.

Standards und Konventionen für Menüs

Es gibt zwar keine eigentlichen Standards, wie Menüs zu entwerfen sind, aber es haben sich eine Reihe von Konventionen durchgesetzt, die sich auf den Entwurf und die Organisation von Menüs beziehen. Diese Konventionen können Sie bei Microsoft unter http://msdn.microsoft.com/ui/ einsehen. Die Konventionen lauten wie folgt:

- In der obersten Menüleiste sind die Kategorien mit einzelnen Worten zu bezeichnen. Eine aus zwei Worten bestehende Kategorie kann der Benutzer leicht für zwei Einzelwortkategorien halten.
- Das Menü Datei ist als erstes Menü von links anzuordnen. Es enthält alle dateiorientierten Funktionen (wie Neu, Öffnen, Speichern, Drucken usw.) sowie den Befehl Beenden. Der Menübefehl Beenden ist am unteren Rand des Menüs zu platzieren und von den übrigen Menüeinträgen durch ein Trennzeichen abzusetzen[1].
- Rechts an das Menü Datei schließt sich das Menü Bearbeiten an. Dieses Menü enthält alle Funktionen zur Bearbeitung wie Kopieren, Ausschneiden, Einfügen, Rückgängig, Wiederholen usw.
- Im Menü Ansicht sind die Menüeinträge vorgesehen, mit denen sich das Aussehen des Arbeitsbereichs einer Anwendung steuern lässt.
- Das Menü Fenster kommt in Anwendungen mit mehreren Dokumenten (MDI- Anwendungen, siehe Tag 10) zum Einsatz. Es enthält Befehle, mit denen sich untergeordnete Fenster steuern, das aktuelle Fenster auswählen und das Layout ändern lassen. Dieses Menü ist immer als vorletztes Menü (d.h. zweites von rechts) in der Menüleiste zu platzieren.
- Das Menü ? (Hilfe) ist das letzte (ganz rechts befindliche) Menü in der Menüleiste. Es enthält Menübefehle, um Anweisungen oder Informationen zur Anwendung aufrufen zu können. Wenn die Anwendung einen Copyright-Vermerk oder Unternehmensinformationen anzeigen muss oder möchte, sollte man sie als letzten Eintrag in diesem Menü unter der Bezeichnung Info über <Anwendungsname> unterbringen.
- Anwendungsspezifische Menüs werden gewöhnlich zwischen den Menüs Ansicht und Fenster untergebracht.

Nicht in jeder Anwendung sind alle Standardmenüs enthalten. Bei den genannten Regeln handelt es sich um Richtlinien, nach denen jedes Menü zu platzieren ist, wenn es in Ihrer Anwendung enthalten ist. Wenn der Funktionsbereich eines dieser Menüs nicht zu einer bestimmten Anwendung passt, passt auch das Menü nicht und sollte nicht verwendet werden.

Menüs entwerfen

Menüs sind in Visual C++-Anwendungen als Ressource definiert. Daher kann man sie im Editor von Visual C++ über die Registerkarte Ressourcenansicht im Arbeitsbereich entwerfen. Wenn Sie eine dialogbasierte Anwendung neu erstellen, ist im Ressourcenzweig noch kein Menüordner vorhanden, was Sie aber ändern können.

Verschiedene Aspekte von Windows-Anwendungen werden als Ressourcen betrachtet. Dazu gehören das Fensterlayout, Menüs, Symbolleisten, Bilder, Strings, Zugriffstasten usw. Alle diese Merkmale sind in einer so genannten Ressourcendatei organisiert, auf die der Visual C++-Compiler zurückgreift, um diese Objekte aus deren Definitionen zu erzeugen. Die Ressourcendatei ist eine Textdatei mit der Dateinamenserweiterung .rc. Sie enthält eine textuelle Beschreibung der verschiedenen Objekte einschließlich der IDs, Beschriftungen, Abmessungen usw.

Manche Ressourcen wie etwa Bilder und Klänge lassen sich nicht in Textform beschreiben, sondern müssen in einem Binärformat gespeichert werden. Diese Ressourcen sind in separaten Dateien abgelegt, wobei Dateiname und Standort in die Ressourcendatei eingebunden werden.

Die Ressourcendatei ist zwar dafür gedacht, alle Anwendungsressourcen zu speichern, doch muss man nicht alle Ressourcen in dieser Datei speichern. Menüs können dynamisch erzeugt werden, Dialogfenster können unter Verwendung externer Vorlagen oder Skins (Oberflächen) gezeichnet werden.

Ein Vorteil von Ressourcendateien ist, dass sie in DLLs eingefügt und dann für die Lokalisierung Ihrer Anwendung verwenden werden können. Indem Sie den gesamten Text in Ihrer Anwendung in der String-Tabelle in der Ressourcendatei unterbringen, können Sie mehrere DLLs erzeugen, welche die String-Tabelle in verschiedenen Sprachen enthalten. So können Sie Ihre Anwendung in verschiedenen Ländern verkaufen, indem Sie die DLL mit der passenden Sprache zu ihrer Applikation hinzufügen. Am Tag 14 erfahren Sie mehr über die Verwendung von DLLs.

7.2 Ein Menü erstellen

Ein Menü lässt sich problemlos erstellen. Dazu sind folgende allgemeine Schritte auszuführen:

1. Die Anwendung erstellen, die den Rahmen für das Menü bildet

2. Dem Projekt eine Menüressource hinzufügen
3. Die Menüressource anpassen, um die jeweiligen Menübefehle für die Anwendung einzubinden
4. Das Menü mit Funktionalität ausstatten, indem man entsprechende Routinen mit den Menübefehlen verbindet

Die Anwendung erstellen

Für die Beispielanwendung in diesem Kapitel erzeugen Sie eine einfache dialogbasierte Anwendung, die eine einzelne Schaltfläche und ein Menü enthält:

1. Erstellen Sie ein neues MFC-Anwendung Visual C++-Projekt. Nennen Sie das Projekt Menus.

2. Übernehmen Sie wie an den vergangenen Tagen die Standardeinstellungen des Anwendungs-Assistenten.
3. Wenn der Anwendungs-Assistent das Gerüst erstellt hat, löschen Sie alle Steuerelemente aus dem Dialogfeld.
4. Nehmen Sie in das Dialogfeld eine einzelne Schaltfläche auf. Als Namen (ID) legen Sie für diese Schaltfläche IDC_EXIT fest und geben als Beschriftung E&xit ein.
5. Fügen Sie für das BN_CLICKED-Ereignis der Schaltfläche Beenden eine Funktion hinzu. Schreiben Sie in diese Funktion den Code, der die Funktion OnOK aufruft. (Wie Sie wissen, bewirkt die Funktion OnOK das Schließen der Anwendung.)

Wenn Sie nicht mehr genau wissen, wie Sie die Funktion OnOK hinzufügen, sehen Sie sich am besten noch einmal das Beispiel im Abschnitt »Die Anwendung schließen« von Tag 3, »Steuerelemente«, an.

Ein Menü hinzufügen und anpassen

Nachdem Sie die zu Grunde liegende Anwendung erstellt haben, können Sie für die Anwendung ein neues Menü erzeugen. Fügen Sie zuerst eine Menüressource in das Projekt ein. Dabei ruft Visual C++ automatisch den Menü-Editor auf, der Ihnen die Anpassung des Menüs erlaubt. Die folgenden Schritte zeigen, wie Sie ein Menü hinzufügen und anpassen:

1. Wählen Sie die Ressourcenansicht aus.

2. Markieren Sie den Ordner der Projektressourcen am Beginn der Strukturansicht, in unserem Beispiel Menus.
3. Klicken Sie mit der rechten Maustaste, um das Kontextmenü zu öffnen und wählen Sie dann Hinzufügen / Ressource hinzufügen.
4. Im Dialogfeld Ressource hinzufügen markieren Sie Menu in der Liste verfügbarer Ressourcen, wie es Abbildung 7.1 zeigt. Klicken Sie auf die Schaltfläche Neu.

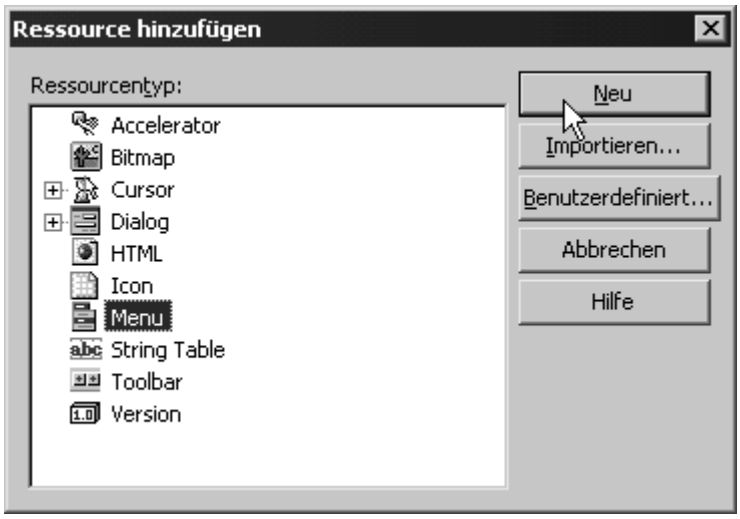


Abbildung 7.1: Das Dialogfeld Ressource hinzufügen

Der Menü-Designer öffnet sich im Bearbeitungsbereich von Visual Studio. Der erste Menübefehl ist hervorgehoben (siehe Abbildung 7.2).

Abbildung 7.2: Ein leeres Menü

Die Menüressource ist nun angelegt und Sie können sie mit Menübefehlen beleben. Einen Menübefehl fügen Sie in den folgenden Schritten hinzu:

1. Klicken Sie auf den hervorgehobenen Bereich, um an der Menüposition Text eingeben zu können.

2. Geben Sie den Titel des Menüs ein, in diesem Beispiel &Datei.

Wenn Sie in Visual C++ .NET Menüeinträge definieren, geben Sie den anzuzeigenden Text an der Position des Menüs ein. Sie können nicht auf Eigenschaften zugreifen, bevor Sie einen Namen für den Menüeintrag an dieser Position eingegeben haben. Da für die Einträge oberster Ebene, die Sie hinzufügen, das Kontrollkästchen Popup aktiviert ist (das ist für alle Menüeinträge oberster Ebene der Standard), löst dieses Menüelement keine Funktionalität in der Anwendung aus und benötigt daher keine Objekt-ID.

3. Die erste Position des Dropdown-Menüs ist hervorgehoben. Klicken Sie wieder auf den hervorgehobenen Bereich und geben Sie den Text &Hallo ein.
4. Wenn Sie rechtsklicken und IDs Bearbeiten wählen, sehen Sie, dass die ID des Menüs bereits als ID_DATEI_HALLO initialisiert wurde. Rechtsklicken Sie und deaktivieren Sie IDs Bearbeiten, um das Menü wiederherzustellen.

Momentan verfügen Sie über ein Menü mit einem einzigen Menübefehl. Sie können nun weitere Menübefehle hinzufügen, indem Sie die obigen Schritte 3 und 4 für jeden hervorgehobenen Bereich wiederholen. In das Menü lassen sich auch Trennzeichen einfügen. Um ein Trennzeichen hinzuzufügen, markieren Sie den hervorgehobenen Bereich, in dem Sie die Trennzeichen platzieren möchten. Rechtsklicken Sie und wählen Sie Trennzeichen einfügen aus dem Kontextmenü (siehe Abbildung 7.3). Sie könnten auch einfach ein Minuszeichen (-) eingeben, um festzulegen, dass es sich bei dem Menüeintrag um ein Trennzeichen handeln soll.

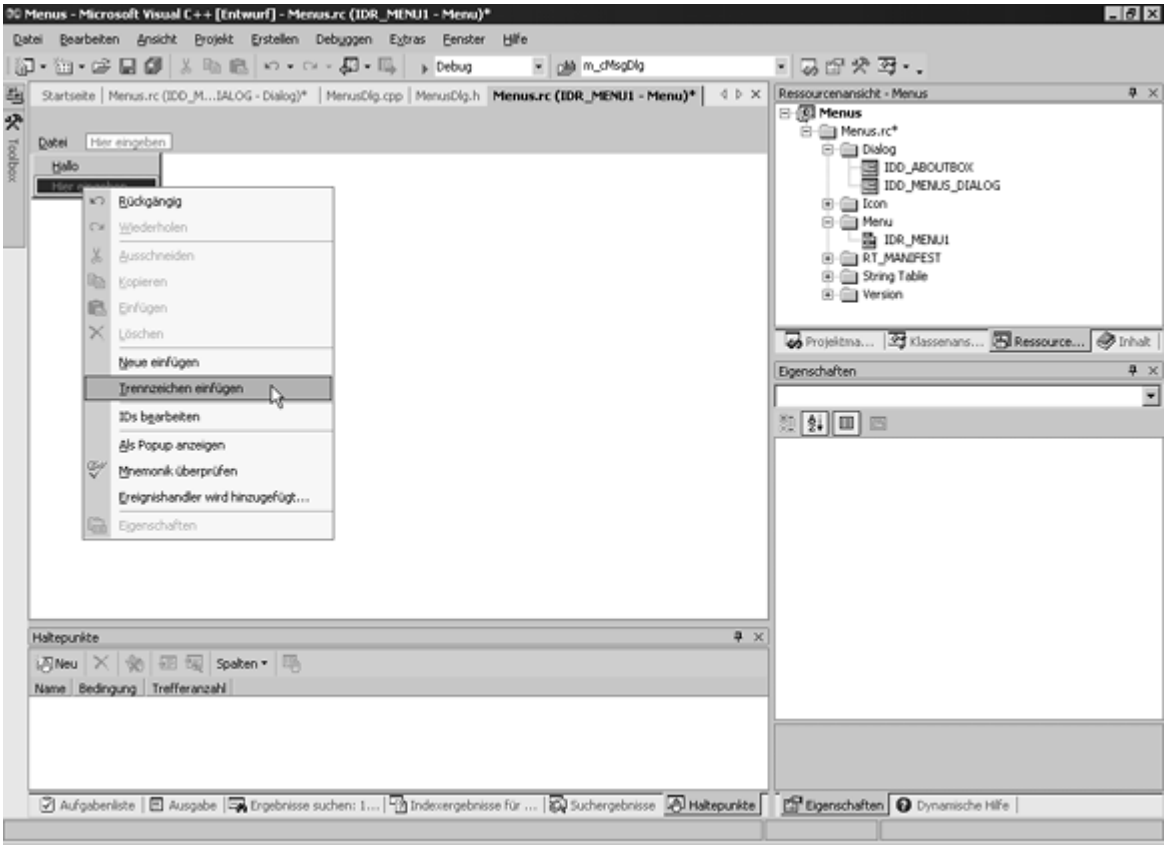


Abbildung 7.3: Ein Trennzeichen festlegen

Um das Beispielprogramm zu vervollständigen, nehmen Sie mit den gleichen Schritten wie oben beschrieben einen Menübefehl Beenden in das Menü Datei auf sowie ein zweites Menü namens ? (Hilfe) mit einem Menübefehl Info über Menus. Die folgenden Schritte, die der obigen Schrittfolge ähnlich sind, zeigen im Detail das Hinzufügen der zusätzlichen Elemente:

1. Klicken Sie auf die dritte Dropdown-Position und geben Sie den Text &Beenden als Titel des Menüeintrags ein.

2. Klicken Sie auf die zweite Position in der Menüleiste. Legen Sie den Titel mit ? (&Hilfe) fest.
3. Klicken Sie auf die erste Position unter dem Menü ? (Hilfe). Geben Sie den Text &Info über Menus als Titel des Menüeintrags ein.

Das Menü ist damit erstellt. Allerdings fehlt noch die Verbindung zur Anwendung.

Das Menü mit dem Dialogfeld verbinden

Sie verfügen nun über ein Menü, das Sie in Ihrer Anwendung einsetzen können. Wenn Sie aber die Anwendung in der jetzigen Entwicklungsphase kompilieren und ausführen, erscheint das Menü nicht. Das Menü ist noch mit dem Dialogfeld zu verbinden. Das erreichen Sie in folgenden Schritten:

1. Öffnen Sie den Dialog-Editor, indem Sie auf das Hauptdialogfeld der Anwendung im Ordner Dialog der Ressourcenansicht doppelklicken. Im Beispiel doppelklicken Sie auf IDD_MENUS_DIALOG.

2. Markieren Sie das gesamte Dialogfeld. Achten Sie darauf, dass keine Steuerelemente markiert sind. (Sie sollen hier die Einstellung der Eigenschaften für das Dialogfeld an sich vornehmen und nicht für irgendein Steuerelement des Dialogfelds.)
3. Wählen Sie aus der Dropdown-Liste Menü das eben erstellte Menü IDR_MENU1 aus, wie es Abbildung 7.4 zeigt.

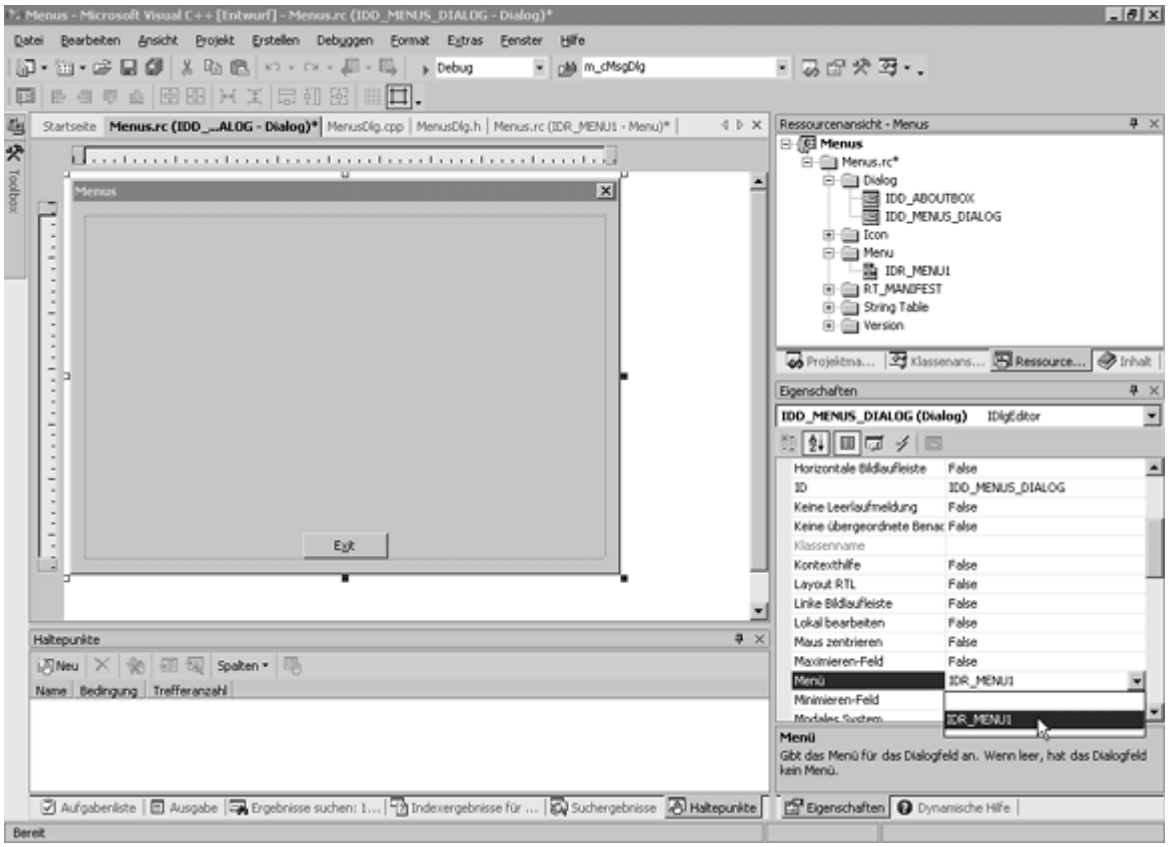


Abbildung 7.4: Das Menü mit dem Dialogfeld verbinden

Wenn Sie die Anwendung kompilieren und ausführen, ist das Menü mit dem Dialogfeld der Anwendung verbunden, wie es aus Abbildung 7.5 hervorgeht. Wie bei jeder anderen Windows-Anwendung können Sie Menübefehle auswählen. Allerdings gibt es einen kleinen Unterschied. Wenn Sie momentan einen der Menübefehle wählen, passiert überhaupt nichts. Sie müssen noch die Funktionalität für die Menübefehle realisieren.

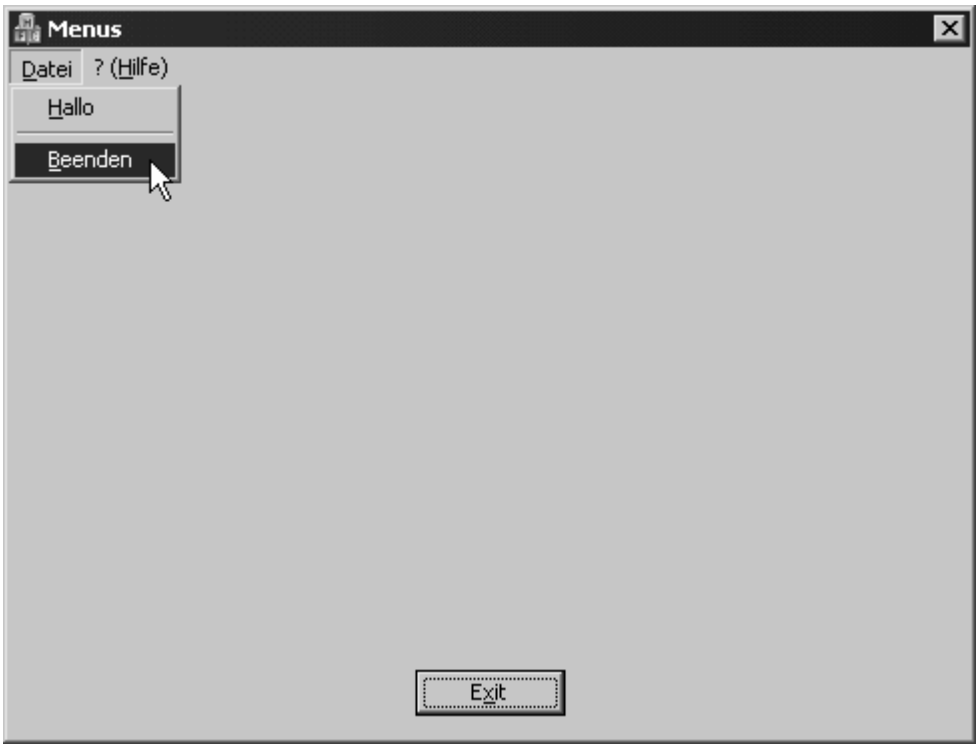


Abbildung 7.5: Das Menü ist nun Teil des Anwendungsdialogfelds.

Menübefehle mit Funktionalität ausstatten

Nunmehr verfügen Sie über ein Menü als Teil der Anwendung. Bevor Ihr Menü irgendeine Aktion bewirken kann, müssen Sie - genau wie überall in Ihren Visual C++- Anwendungen - festlegen, was zu tun ist. Um das Menü der Beispielanwendung mit etwas Funktionalität auszustatten, führen Sie die folgenden Schritte aus:

1. Öffnen Sie den Menü-Designer für Ihr Menü.

2. Wählen Sie den mit Hallo beschrifteten Eintrag aus (unter dem Menü Datei).
3. Rechtsklicken Sie und wählen Sie Ereignishandler wird hinzugefügt aus dem Kontextmenü.
4. Wählen Sie als Meldungstyp COMMAND und die Klasse, die die Ereignismeldung verarbeiten soll. In diesem Beispiel wählen Sie aus der Klassenliste CMenusDlg (siehe Abbildung 7.6). Klicken Sie auf die Schaltfläche Hinzufügen, um die Funktion zur Nachrichtenverarbeitung zu erzeugen.
5. Bearbeiten Sie die Funktion und geben Sie den Code aus Listing 7.1 ein.

Listing 7.1: Die Funktion OnFileHello

```
1: void CMenusDlg::OnFileHello()
2: {
3:    // // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:
5:    // Meldung anzeigen
6:    MessageBox("Hallo da draussen vor den Bildschirmen", "Hallo");
7: }
```

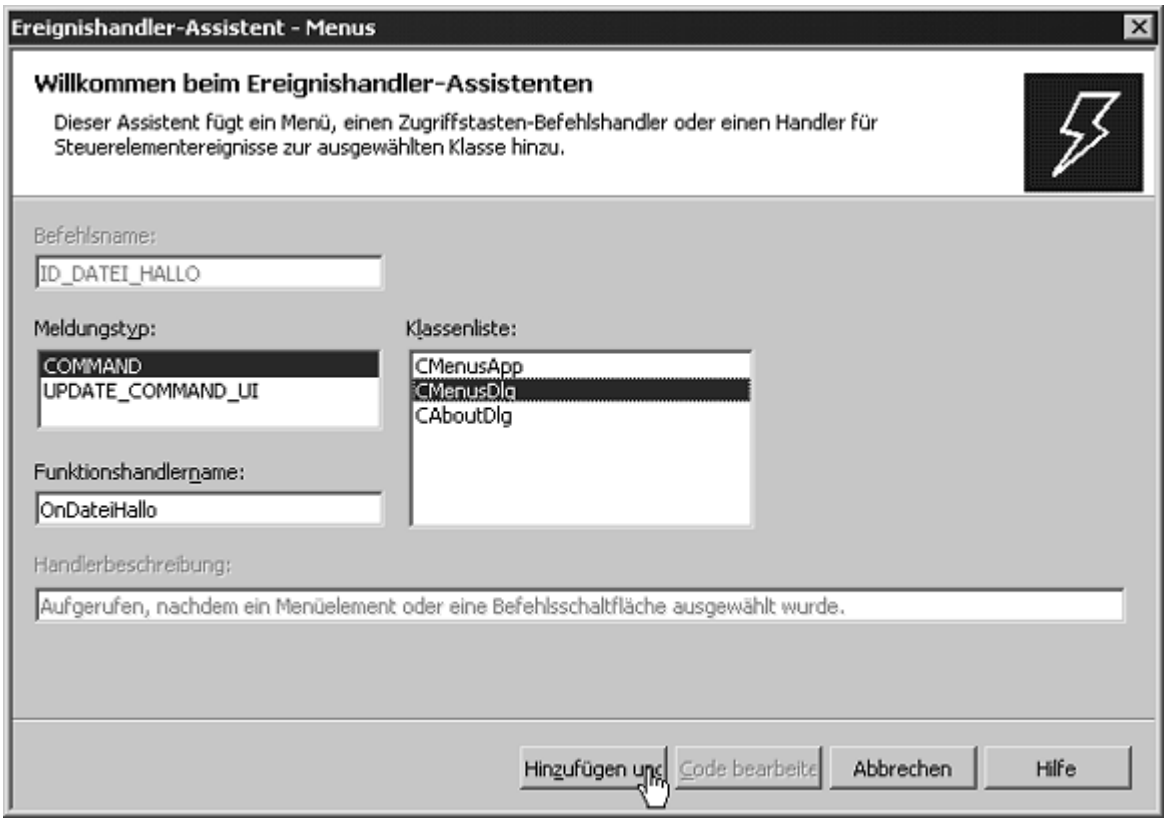


Abbildung 7.6: Der Dialog Ereignis-Assistent

Die Nachricht COMMAND erhält das Anwendungsfenster, wenn ein Menübefehl ausgewählt wird. Wenn man eine Funktion für diese Nachricht vorsieht, hat das die gleiche Wirkung wie eine Funktion zur Auswahl des Menübefehls. Die eigentliche Ereignisnachricht heißt WM_COMMAND, sie wird im Ereignishandler Assistenten jedoch abgekürzt.

In früheren Versionen von C++ konnten Sie bereits vorhandene Behandlungsroutinen von Menübefehlen aufrufen, indem Sie die vorhandene Funktion zum Menüereignis COMMAND hinzufügten. In Visual C++ .NET mit dem Ereignishandler Assistent können Sie das nicht mehr tun. Daher müssen Sie eine Behandlungsroutine für das Menü hinzufügen und in dieser Funktion die andere Behandlungsroutine aufrufen, die Sie wiederverwenden möchten.

Um die Funktion OnBnClickedExit für den Menübefehl Beenden erneut zu verwenden, folgen Sie diesen Schritten:

1. Öffnen Sie wieder den Menü-Designer, wählen Sie den Menüeintrag Beenden und fügen Sie eine Behandlungsroutine ein.

2. Fügen Sie eine Funktion für die Nachricht COMMAND hinzu. Übernehmen Sie den vom Klassenassistent vorgeschlagenen Funktionsnamen und klicken Sie auf Hinzufügen.
3. Bearbeiten Sie die Funktion und fügen Sie den Code aus Listing 7.2 ein, um die Behandlungsroutine der Schaltfläche Beenden aufzurufen.

Listing 7.2: Die Funktion OnFileExit

```
1: void CMenusDlg::OnFileExit()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Behandlungsroutine der Schaltfläche Exit aufrufen
```

```
5:    OnBnClickedExit();
6: }
```

4. Um die Funktionalität Ihres Beispiels abzurunden, fügen Sie dem Menüeintrag Info über eine Funktion für die Nachricht COMMAND hinzu. Bearbeiten Sie die Funktion wie in Listing 7.3.

Listing 7.3: Die Funktion OnHelpAboutmenus

```
1: void CMenusDlg::OnHelpAboutmenus()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Instanz des About-Menus-Fensters deklarieren
5:    CAboutDlg dlgAbout;
6:
7:    // About-Fenster anzeigen
8:    dlgAbout.DoModal();
9: }
```

Den Menübefehl Datei / Beenden haben Sie mit einer vorhandenen Funktion verbunden, die Ihre Anwendung

Wenn Sie die Anwendung kompilieren und ausführen, können Sie sich davon überzeugen, dass alle Menüeinträge funktionieren. Wenn Sie ? (Hilfe) / Info über Menus wählen, wie es Abbildung 7.7 zeigt, erscheint das Dialogfeld Info über Menus (siehe Abbildung 7.8). Wählen Sie Datei / Hallo, so sehen Sie eine Meldungsbox mit dem Text Hallo da draussen vor den Bildschirmen (siehe Abbildung 7.9). Und mit Datei / Beenden lässt sich die Anwendung schließen.

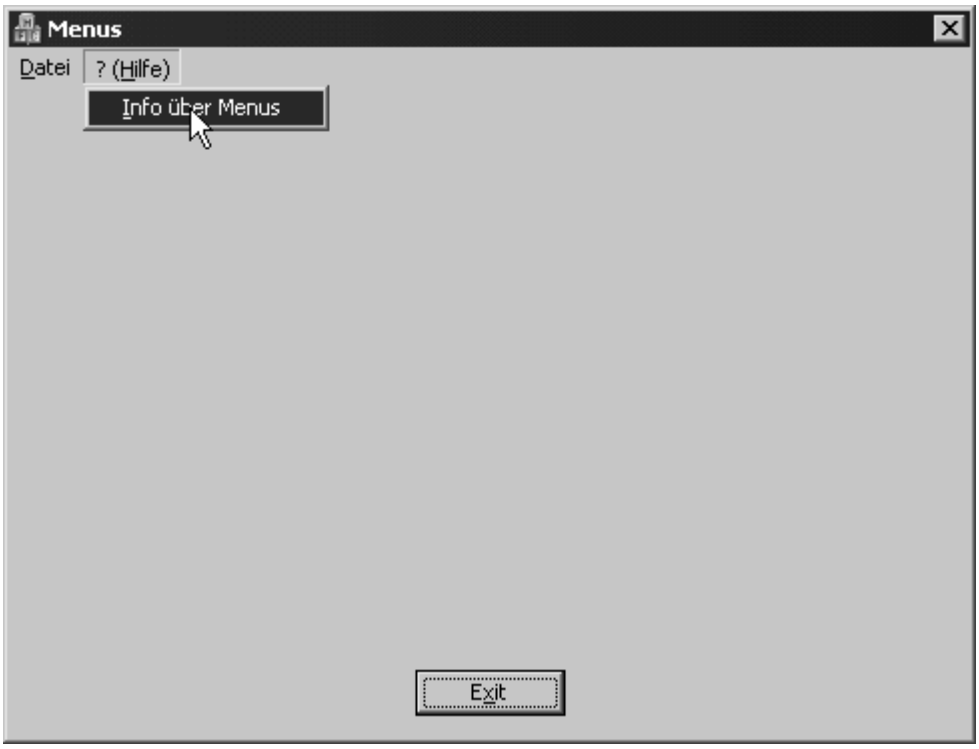


Abbildung 7.7: Der Menüeintrag ? (Hilfe) / Info über Menus

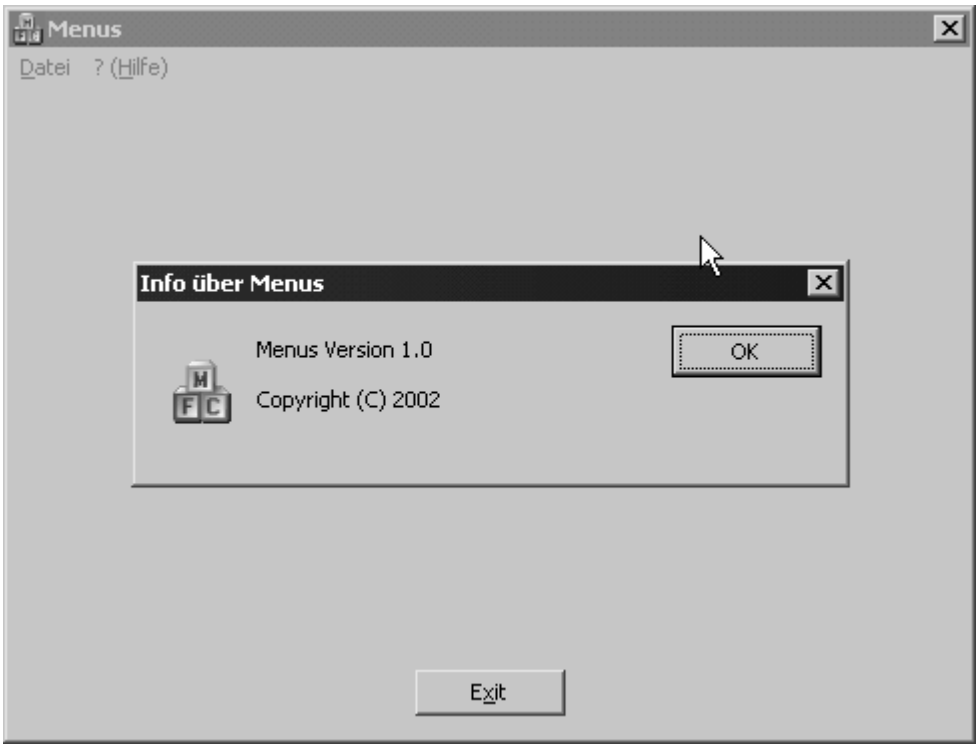


Abbildung 7.8: Das Dialogfeld Info über Menus

Abbildung 7.9: Das Meldungsfeld Hallo

MFC-Exkurs: Die Klasse CMenu

Die Klasse CMenu verkapselt einen Großteil der Menüfunktionalität. Der größte Teil dieser Funktionalität beschäftigt sich allerdings mit der Erstellung von Menüs durch Ihren Code, nicht mit der Arbeit mit Menüs, die Sie mit dem Menü-Designer erstellt haben. Einige Methoden ermöglichen es Ihnen aber, mit vordefinierten Menüs zu arbeiten.

Das erste Menü, das Sie in Ihrer Anwendung verwenden (oder zumindest das erste Menü, das der Benutzer sieht), hängen Sie natürlich an das Hauptfenster an. Daher bedarf dieses Menü keiner weiteren Verarbeitung. Wenn Sie es ändern wollen, während die Anwendung läuft, müssen Sie allerdings selbst etwas Programmierarbeit in Angriff nehmen. Die erste Methode, die Sie dafür benötigen, ist LoadMenu. Diese Methode übernimmt einen einzigen Parameter des Menüs - die Ressourcen-ID, das Sie laden möchten. Nachdem Sie das Menü geladen haben, können Sie die Methode SetMenu der Klasse CWnd für das Fenster aufrufen, dessen Menüs Sie ändern möchten. Die Methode SetMenu übernimmt einen Zeiger auf das neue Menü als einzigen Parameter. Die Aufgabe ließe sich folgendermaßen erledigen:

Listing 7.4: Austauschen eines Menüs durch ein anderes

```
 1: // m_MyNewMenu ist eine CMenu-Variable auf Klassenebene im
 2: // Fenster für das Sie this als neues Menü setzen möchten.
 3: // Es ist KEINE lokale Variable in der Funktion,
 4: // in der Sie umschalten
 5: m_MyNewMenu.LoadMenu(IDR_NEW_MENU);
 6: // Altes Menü durch Übergabe von NULL statt CMenu-Zeiger löschen
 7: SetMenu(NULL);
 8: // Menü über das Handle zerstören
 9: // m_hMenuHandle ist ebenfalls eine HANDLE-Variable auf
10: // Klassenebene in der Fensterklasse
11: // Zeiger auf das alte Menü abrufen
12: CMenu *pMenu = CMenu::FromHandle(m_hMenuHandle);
13: if (pMenu)
14:   // Menü zerstören;
15:   // alle von ihm verwendeten Ressourcen freigeben
16:   pMenu->DestroyMenu();
17: // Neues Menü durch Übergabe der
18: // CMenu-Variablen-Adresse hinzufügen
19: SetMenu(&m_MyNewMenu);
20: // Neues Menü-Handle der Handle-Variable zuweisen
21: m_hMenuHandle = m_MyNewMenu.m_hMenu;
```

In diesem Code wurde ein Zeiger auf das Menü aus dem Menü-Handle abgefragt und zu dessen Zerstörung verwendet, nachdem das alte Menü nicht mehr mit dem Fenster verbunden war. Das geschah durch die Member-Funktion FromHandle, die einen CMenu- Handle als einzigen Parameter übernimmt und einen Zeiger auf das Menü zurückgibt. Nach der Rückgabe eines Zeigers auf das Menü wurde die Methode DestroyMenu aufgerufen. Sie löscht das alte Menü aus dem Speicher und gibt die von ihm verwendeten Ressourcen frei. Man hätte das auch mit der Windows-API-Funktion DestroyMenu erreichen können, die den Menü-Handle als Parameter übernimmt:

```
// Altes Menü über Handle zerstören
::DestroyMenu(m_hMenuHandle);
```

Auf den nächsten Seiten werden Sie die Verwendung von zwei weiteren Member- Methoden der Klasse CMenu erlernen. Diese sind für die Erstellung von Kontextmenüs nützlich. Es handelt sich bei diesen CMenu-Funktionen um GetSubMenu, mit dem man einen Zeiger auf den Dropdown-Teil eines Menüs abfragen kann, und um TrackPopupMenu, das ein Menü als Kontextmenü anzeigt. Sie werden diese Funktionen mit der Funktion GetMenu der Klasse CWnd zusammen verwenden, die einen Zeiger auf das aktuell vom Fenster verwendete Menü zurückgibt.

MFC-Exkurs: Menü-Ereignisnachrichten

Als Sie Ihren Menüeinträgen die Behandlungsroutine hinzufügten, sind Ihnen wahrscheinlich zwei Ereignisnachrichten aufgefallen, denen Sie Behandlungsroutinen zuweisen konnten. Die erste Nachricht, COMMAND, ist die Nachricht, die an das Fenster übergeben wird, um zu signalisieren, dass der Benutzer diesen bestimmten Teil des Menüs ausgewählt hat. Bei der anderen Ereignisnachricht, UPDATE_COMMAND_UI, ist nicht ganz so offensichtlich, was sie signalisiert. Die Nachricht UPDATE_COMMAND_UI wird direkt vor der Anzeige des Menüeintrags an die Fensterklasse gesendet. Diesem Ereignis müssen Sie jede Verarbeitung hinzufügen, von der die Anzeige des Menüeintrags beeinflusst wird. Wenn sich der Menüeintrag wie ein Kontrollkästchen verhalten und neben ihm eine Markierung angezeigt werden soll, fügen Sie diese Markierung an dieser Stelle ein. Soll ein Menüeintrag

deaktiviert sein, fügen Sie hier die entsprechende Funktionalität ein. In Kapitel 10, »SDI- und MDI-Anwendungen« werden Sie viel detaillierter mit dieser Ereignisnachricht arbeiten.

7.3 Kontextmenüs erstellen

Die meisten Windows-Anwendungen verfügen über so genannte Popup- oder Kontextmenüs. Diese ruft man auf, indem man mit der rechten Maustaste auf ein Objekt klickt. Die Bezeichnung Popup (etwa: aufspringen) rührt daher, dass diese Menüs mitten im Anwendungsbereich erscheinen und nicht an eine Menüleiste, den Fensterrahmen oder etwas anderes auf dem Bildschirm (ausgenommen den Mauszeiger) gebunden sind. Man spricht auch oft von Kontextmenüs, weil der Inhalt dieses Menüs vom Kontext abhängt, in dem es geöffnet wird. Die verfügbaren Befehle im Kontextmenü hängen von der momentanen Auswahl in der Anwendung oder der aktuellen Position des Mauszeigers auf einem bestimmten Objekt ab.

Um ein Kontextmenü in der Anwendung bereitzustellen, gibt es zwei Lösungsansätze. Entweder erstellen Sie ein Menü, das speziell als Kontextmenü vorgesehen ist, oder Sie verwenden eines der Pulldown-Menüs aus dem Hauptmenü, das Sie bereits erstellt haben. Wenn Sie ein Menü speziell als Kontextmenü entwerfen, überspringen Sie die oberste Ebene der Menüleiste, indem Sie ein Leerzeichen oder anderen Text, der nicht auf dem Bildschirm erscheint, als Titel eingeben (es macht Sinn, eine Beschriftung zu verwenden, die den Kontext des Menüs beschreibt).

Jeder Menübefehl eines Dropdown-Menüs lässt sich seinerseits als Kontextmenü verwenden. Zu diesem Zweck müssen Sie einen Handle auf das Untermenü (den Dropdown-Teil des Menüs) ermitteln und dann die Funktion TrackPopupMenu auf dem Untermenü aufrufen. Die übrige Funktionalität des Kontextmenüs wurde bereits behandelt, als Sie die anderen Menüs erstellten und programmierten. Um ein Kontextmenü in die Beispielanwendung einzubauen, führen Sie die folgenden Schritte aus:

1. Fügen Sie mit dem Meldungen-Modus der Eigenschaftenansicht eine Funktion für die Nachricht WM_CONTEXTMENU in das Dialogfeld der Anwendung ein.

Ein Kontextmenü kann man über zwei verschiedene Nachrichten für Behandlungsfunktionen auslösen. Es liegt zunächst auf der Hand, die Nachricht WM_RBUTTONDOWN zu verwenden. Windows sendet diese Nachricht, wenn der Benutzer mit der rechten Maustaste klickt. Weiterhin kann man (und sollte man) die Nachricht WM_CONTEXTMENU verwenden, die speziell dafür vorgesehen ist, ein Kontextmenü zu initiieren. Dieses Ereignis entsteht bei einer Reihe von Benutzeraktionen: eine davon ist das Loslassen der rechten Maustaste, eine andere das Drücken der Kontextmenü-Taste auf einer der neueren Windows-Tastaturen.

2. Nehmen Sie in die Funktion den Code aus Listing 7.5 auf. Achten Sie darauf, die Kommentarzeichen vor und hinter den an die Funktion übergebenen Parametern zu entfernen, da der Funktionsrumpf, der vom Assistenten bereitgestellt wird, nicht funktional ist:

Listing 7.5: Die Funktion OnContextMenu

```
void CMenusDlg::OnContextMenu(CWnd* /*pWnd*/, CPoint /*point*/)

 1: void CMenusDlg::OnContextMenu(CWnd* pWnd, CPoint point)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
 4:   CMenu *pMenu;
 5:
 6:   // Zeiger auf Menü abfragen
 7:   pMenu = GetMenu();
 8:   // Zeiger auf das Untermenü abfragen
 9:   pMenu = pMenu->GetSubMenu(0);
10:   // Als Kontextmenü öffnen
11:   pMenu->TrackPopupMenu(TPM_CENTERALIGN | TPM_LEFTBUTTON,
```

```
12:                          point.x, point.y, pWnd, NULL);
13: }
```

Die Funktion in Listing 7.5 ermittelt mit der Funktion GetMenu einen Zeiger auf das Fenstermenü. Dieser Zeiger sollte immer ein lokaler innerhalb der Funktion sein, in der Sie ihn verwenden, da sich der Ort des Menüs (im Speicher) in der laufenden Anwendung ändern kann. Über den Menüzeiger ermitteln Sie als Nächstes mit der Funktion GetSubMenu einen Zeiger auf das erste Dropdown-Menü (die Nummerierung der Untermenüs beginnt mit 0 wie bei allen anderen Elementen in C/C++). Nachdem Sie einen Zeiger auf das Untermenü besitzen, können Sie ihn als reguläre Klasseninstanz von CMenu behandeln.

Das letzte Teilchen in diesem Puzzle ist der Aufruf der Member-Funktion TrackPopupMenu der Klasse CMenu. Diese Funktion verwendet fünf Argumente und bestimmt damit, wo und wie das Kontextmenü anzuzeigen ist. Das erste Argument kombiniert zwei Flags. Das erste Flag, TPM_CENTERALIGN, zentriert das Kontextmenü unter dem Mauszeiger. Sie können stattdessen auch TPM_LEFTALIGN oder TPM_RIGHTALIGN verwenden. Diese Flags richten den linken bzw. rechten Rand des Kontextmenüs an der Mausposition aus. Der zweite Teil dieser Flag-Kombination ist TPM_LEFTBUTTON. Dieses Flag bewirkt, dass das Kontextmenü über das Drücken der linken Maustaste aufgerufen wird. Mit TPM_RIGHTBUTTON lässt sich das Kontextmenü mit der rechten Maustaste aktivieren.

Die Argumente zwei und drei der Funktion TrackPopupMenu legen die Bildschirmposition - nicht die relative Position im Fensterbereich - für das Kontextmenü fest. Das vierte Argument ist ein Zeiger auf das Fenster, das die Nachrichten der Menübefehle erhält. Das letzte Argument beschreibt ein Rechteck, in das der Benutzer klicken kann, ohne das Kontextmenü zu schließen. Übergibt man hier NULL, wird das Kontextmenü geschlossen, wenn der Benutzer außerhalb des Kontextmenüs klickt. Mit diesem Code können Sie in Ihrer Anwendung ein Kontextmenü gemäß Abbildung 7.10 realisieren.

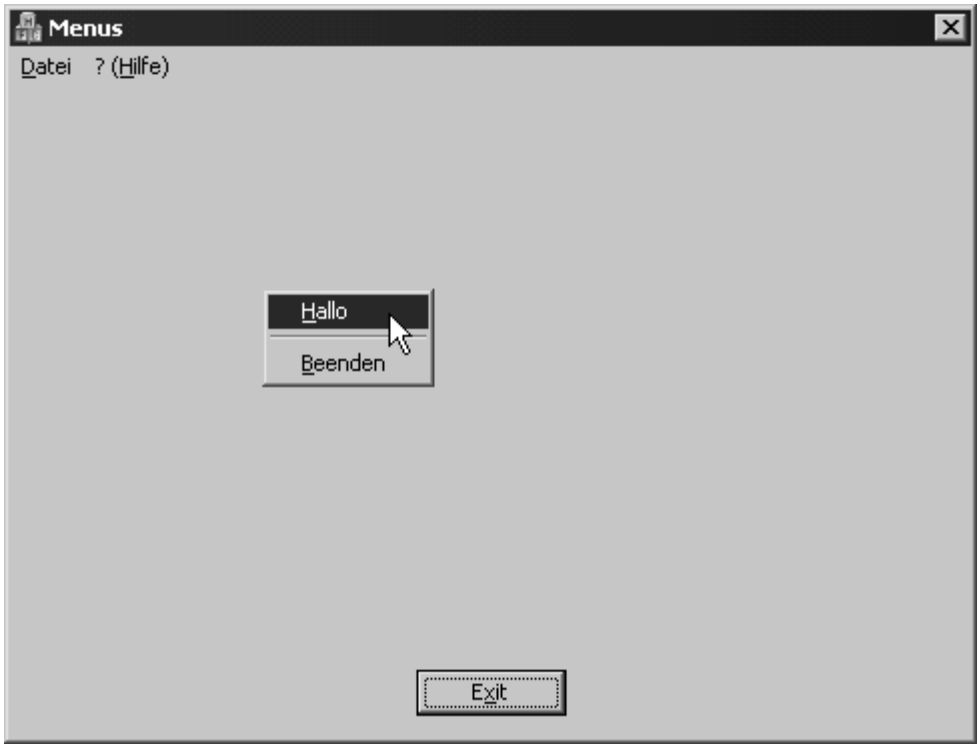


Abbildung 7.10: Das Kontextmenü in der Praxis

Positionen auf dem Bildschirm kann man als absolute Position oder als relative Position angeben. Sie werden als horizontale Position (x) und vertikale Position (y) bestimmt. Die Position wird in Pixeln von der oberen linken Ecke des Bildschirms (absolut) oder von der oberen linken Ecke des Fensters (relativ) aus gemessen. Durch eine Kombination der absoluten und relativen Werte können Sie herausfinden, wo im Fenster und über welchem Element sich der Mauszeiger befindet. So können Sie ein Kontextmenü öffnen, das zu dem Element unter dem Mauszeiger passt.

7.4 Ein Menü mit Zugriffstasten

Zu den bereits von Anfang an vorhandenen Schnelltasten für die Auswahl von Menübefehlen gehören die Zugriffstasten. Wie bereits weiter vorn in diesem Kapitel erwähnt, sind Zugriffstasten spezielle Tastenkombinationen - gewöhnlich eine Kombination der (Strg)- Taste mit einer anderen Taste - oder Funktionstasten, die innerhalb der gesamten Anwendung eindeutig sind. Jede dieser Tastenkombinationen löst eine bestimmte Menüfunktion aus.

Die Funktionsweise der Zugriffstasten ist mit der von Menüs vergleichbar. Es handelt sich ebenfalls um eine Anwendungsressource, die in einer Tabelle auf der Registerkarte Ressourcen im Arbeitsbereich definiert ist. Jeder Tabelleneintrag verfügt über eine Objekt-ID und den Code für eine Tastenkombination. Nachdem Sie die Zugriffstasten definiert haben, können Sie den Objekt-IDs Funktionalität zuordnen. Einer Zugriffstaste lässt sich dieselbe Objekt-ID wie dem korrespondierenden Menüeintrag zuweisen, sodass in der Nachrichtenzuordnungstabelle der Anwendung nur ein Eintrag zu definieren ist.

Nachdem Sie alle Zugriffstasten definiert haben, können Sie die Tastenkombination im Menübefehl angeben, damit der Benutzer über sie informiert ist. Hängen Sie ein »\t« am Ende der Beschriftung eines Menübefehls an und geben Sie anschließend die Tastenkombination an. Die Zeichenfolge »\t« erscheint in der Menüanzeige als Tabulatorin Menüs, der die Beschriftung des Menüs von der Tastenkombination absetzt.

Leider funktionieren Zugriffstasten nicht bei auf Dialogfeldern basierenden Fenstern, sodass sie in der heutigen Anwendung nicht eingesetzt werden können. Wie man Zugriffstasten mit Menüs verbindet, lernen Sie in Kapitel 16 bei der Behandlung von COM-Schnittstellen in MFC-Anwendungen.

7.5 Zusammenfassung

Schwerpunkt des heutigen Tages waren Menüs in Visual C++-Anwendungen. Sie haben gelernt, wie man mit den Werkzeugen von Visual Studio ein Menü für eine Anwendung erstellt und dann das Menü mit einem Fenster der Anwendung verbindet. Anschließend wurde gezeigt, wie man den verschiedenen Menübefehlen Funktionalität zuordnet. Gegen Ende der heutigen Lektion haben Sie erfahren, wie man einen Teil des Menüs als Popup- oder Kontextmenü verwenden kann. Schließlich wurde erwähnt, wie sich Zugriffstasten in die meisten Anwendungen einbinden lassen. Morgen werden Sie lernen, wie Sie sich die Fähigkeit von Windows, mit Schriftarten umzugehen, zu Nutzen machen, sodass Sie Benutzern die Konfiguration Ihrer Anwendungen ermöglichen können.

7.6 Workshop

Fragen und Antworten

Frage:

Muss ich meine Menüs in der gleichen Weise benennen, wie es bei anderen Anwendungen üblich ist? Zum Beispiel verfügen viele Anwendungen über die Menüs Datei und Hilfe. Kann ich meine Menüs irgendwie anders bezeichnen?

Antwort:
Die Menübefehle in der Menüleiste können Sie beliebig benennen. Allerdings gibt es allgemein anerkannte Konventionen, die alle dateiorientierten Funktionen unter einem Menü Datei zusammenfassen und alle hilfebezogenen Funktionen unter einem Menü Hilfe. Wenn Sie ein Menü mit Einträgen wie Broccoli, Erbsen und Möhren haben, werden Sie das Menü wahrscheinlich mit Gemüse bezeichnen, wobei man dafür auch Lebensmittel oder Pflanzen verwenden könnte. Im Allgemeinen sollten Sie darauf achten, dass sich die Benutzer leicht in Ihre Anwendung einarbeiten können, sodass die Bezeichnungen der Menüs möglichst die Einträge im Pulldown-Teil des Menüs charakterisieren.

Frage:
Warum kann ich nicht eine einzelne Taste als Zugriffstaste festlegen?

Antwort:
Eine einzelne Taste löst die Nachricht WM_KEY aus und nicht die Menünachrichten. Als die Entwickler von Windows die Arbeitsweise von Zugriffstasten festgelegt haben, gingen sie davon aus, dass einzelne Tasten normalerweise eine Eingabe in die aktive Anwendung darstellen. Wenn einzelne Tasten als Zugriffstasten erlaubt wären, könnte Windows nicht bestimmen, ob das Zeichen eine Eingabe oder eine Schnelltaste darstellt. Durch die erforderlichen Tastenkombinationen (mit Ausnahme der Funktionstasten) haben die Entwickler sichergestellt, dass Windows diese Entscheidung nicht treffen muss.

Quiz

1. Welche Nachricht sendet eine Menüauswahl an die Nachrichtenwarteschlange der Anwendung?

2. Wie verbinden Sie ein Menü mit einem Dialogfeld?
3. Welche vorhandene Klasse legen Sie für die Behandlung von Nachrichten für das Menü fest?
4. Durch welche Nachricht sollte man ein Kontextmenü auslösen?

Übungen

1. Nehmen Sie in das Hauptfenster eine Schaltfläche auf und lassen Sie sie dieselbe Funktion aufrufen wie der Menübefehl Hallo.

2. Verändern Sie das Kontextmenü in Ihrer Anwendung, sodass es das Dropdown-Menü Hilfe als Kontextmenü verwendet.

1

Somit sollten wir eigentlich die verwendete Schaltfläche Exit" in Beenden" umbenennen.

Woche 1 im Rückblick

Die erste Woche liegt hinter Ihnen. Sie haben nun einen ersten Eindruck bekommen, welche Möglichkeiten sich mit Visual C++ beim Erstellen von Anwendungen bieten. Blicken wir noch einmal zurück, was behandelt wurde und was Sie bisher gelernt haben sollten.

Es empfiehlt sich an dieser Stelle, dass Sie ein paar einfache Anwendungen auf eigene Faust entwerfen und erstellen, um Ihre Kenntnisse zu den bisher behandelten Elementen zu vertiefen. Bauen Sie die verschiedensten Steuerelemente in die Anwendungen ein und fügen Sie weitere Dialogfelder hinzu, damit Sie mit diesen Themen vertraut werden. Erst bei der praktischen Arbeit zeigt sich, ob Sie alle Konzepte verstanden haben. Vielleicht vertiefen Sie sich auch in die MFC-Dokumentation und probieren kompliziertere - und hier nicht behandelte - Funktionen in eigenen Anwendungen aus.

Vor allem verstehen Sie jetzt, wie man Steuerelemente und Dialogfelder in Anwendungen einbindet, um Informationen vom Benutzer einzuholen und sie ihm anzuzeigen. Das ist ein fundamentaler Teil jeder Windows-Anwendung, da fast jede Anwendung mit dem Benutzer auf irgendeine Weise in Wechselwirkung tritt. Die Standardsteuerelemente können Sie problemlos im Anwendungsfenster unterbringen und in die Anwendung einbeziehen. Das Gleiche gilt für das Standardmeldungsfeld sowie die von Windows bereitgestellten Dialogfelder. Weiterhin können Sie eigene, benutzerdefinierte Dialogfelder erstellen und in eine geplante Anwendung einbinden. Wiederholen Sie bei Bedarf den Stoff zu Tag 3, »Fehlerbeseitigung«, um sich mit Steuerelementen vertraut zu machen, und/oder vertiefen Sie Ihre Kenntnisse zu Standard- und benutzerdefinierten Dialogfeldern in der Lektion von Tag 6, »Dialogfelder«.

Ein weiterer wichtiger Punkt, dem Sie in der Mehrheit Ihrer Anwendungen begegnen, sind die Menüs. Dazu müssen Sie genau wissen, wie man ein gutes Menü entwirft, wie man sicherstellt, dass keine Konflikte mit den Zugriffstasten auftreten, und wie man die Funktionalität der Anwendung mit den Menübefehlen verbindet. Die von Ihnen erstellten Menüs und die darüber ausgelösten Befehle können Sie nun problemlos in Ihre Anwendungen integrieren. Wenn Sie mit diesem Thema nicht hundertprozentig vertraut sind, wiederholen Sie den Stoff von Tag 7, »Menüs«.

In verschiedenen Situationen müssen Sie Aktionen in regelmäßigen Abständen auslösen oder ermitteln, wie lange bestimmte Abläufe dauern. In diesen und anderen Fällen kommen die Timer ins Spiel. Sollten Sie noch Probleme haben, wie man Timer in Anwendungen einbaut, gehen Sie einfach zu Tag 5, »Timer«, zurück.

Je nach der Art Ihrer Anwendungen kann es erforderlich sein, die vom Benutzer ausgelösten Maus- und Tastaturereignisse aufzufangen. Wenn Sie etwa eine Zeichenanwendung erstellen, sind diese Informationen von grundlegender Bedeutung. Das Gleiche gilt bei einer Anwendung, die Sie mit Drag&Drop-Fähigkeiten ausstatten. Es gibt eine Reihe von Situationen, bei denen diese Funktionalität in Ihren Anwendungen unabdingbar ist. Mit Abschluss des dritten Tages sind Sie damit vertraut, wie man die verschiedenen Mausereignisse auffängt und ermittelt, welche Maustaste beim jeweiligen Ereignis gedrückt wurde. Außerdem können Sie Tastaturereignisse auffangen, wenn die Steuerelemente im Anwendungsfenster nicht selbst diese Tastatureingaben verarbeiten. Wiederholen Sie einfach den Stoff des vierten Tages. »Maus und Tastatur«, falls Ihnen diesbezüglich noch nicht alles geläufig ist.

Schließlich kennen Sie mittlerweile die Entwicklungsumgebung von Visual C++ - Visual Studio - zur Genüge und wissen, welcher Bereich der Umgebung wofür gedacht ist und wie man die verschiedenen Werkzeuge und Hilfsprogramme beim Erstellen der Anwendung einsetzt. Sie können jetzt problemlos über den Arbeitsbereich durch Ihr Anwendungsprojekt navigieren und die verschiedenen Editoren lokalisieren und starten. Weiterhin können Sie das Symbol, das Ihre Anwendung repräsentiert, finden und selbst gestalten. Insbesondere wissen Sie, wo die einzelnen Elementfunktionen oder Variablen in den Klassen Ihrer Anwendung untergebracht sind. Sie sollten mit dem Debugger von Visual C++ vertraut sein und wie er es Ihnen ermöglicht, Ihren Code Schritt für Schritt zu durchlaufen und zu beobachten, ob alles so funktioniert wie es soll. Verwenden Sie den Debugger bei jedem Beispiel, das Sie in diesem Buch erstellen. Nach 21 Tagen sollten Sie einiges an Erfahrung mit dem Debugger gesammelt haben.

Nach Abschluss der ersten Woche arbeiten Sie bereits problemlos mit Visual C++. Wenn Sie mit den bisher

behandelten Themen einigermaßen vertraut sind, fahren Sie fort und lernen Sie mehr darüber, was sich mit Visual C++ als Ihrem Programmierwerkzeug schlechthin alles realisieren lässt. Also auf zur zweiten Woche ...

Woche 2 im Überblick

In der zweiten Woche tauchen Sie in verschiedene kompliziertere Themen ein. Dennoch gehören diese Themen zum Kern beim Erstellen von Windows-Anwendungen.

Die Woche beginnt mit Tag 8, »Text und Schriften«, der sich mit der Infrastruktur der Schriften unter Windows beschäftigt und zeigt, wie man in einer Visual C++-Anwendung darauf zugreift. In diesem Zusammenhang erstellen Sie eine Liste der verfügbaren Schriften und bringen Text in ausgewählten Schriften auf den Bildschirm.

Am Tag 9, »Bilder, Zeichnungen und Bitmaps«, lernen Sie, wie man in einer Windows-Anwendung Grafiken erstellt. Sie lernen, wie man einfache Linien, Rechtecke und Ellipsen zeichnet. Es wird gezeigt, wie man einen Gerätekontext beim Zeichnen von Grafiken einsetzt, ohne sich um die konkrete Grafikhardware kümmern zu müssen, mit der der Benutzer Ihrer Anwendung arbeitet.

Am Tag 10, »SDI- und MDI-Anwendungen«, lernen Sie, wie man eine einfache Anwendung mit einem Dokument - eine so genannte SDI (Single Document Interface)-Anwendung - erstellt. In diesem Zusammenhang wird die Dokument-/Ansicht-Architektur behandelt, die man bei Visual C++ für derartige Anwendungen einsetzt, und Sie erfahren, wie man auf der Basis dieser Architektur eigene Anwendungen erstellt. Sie wenden Ihr Wissen zum Erstellen von SDI-Anwendungen dann auf Anwendungen mit mehreren Dokumenten - so genannten MDI (Multiple Document Interface)-Anwendungen - an. Sie sehen, wie man mit der gleichen Dokument-/Ansicht-Architektur MDI-Anwendungen erstellt, die zu den gebräuchlichsten Formen der heute verfügbaren Windows-Anwendungen gehören.

Am elften Tag, »Symbolleisten und Statusleisten«, erstellen und modifizieren Sie Ihre eigenen Symbolleisten und Statusleisten. Es wird gezeigt, wie man die Schaltflächen einer Symbolleiste mit Menüs in der Anwendung verbindet und zusätzliche Symbolleisten hinzufügt. Weiterhin bringen Sie Ihre eigenen Informationselemente in der Statusleiste, die man in den meisten Windows-Anwendungen am unteren Rand des Fensters findet, unter, und Sie lernen, wie man die Statusleiste aktualisiert, um den jeweiligen Zustand der Anwendung widerzuspiegeln.

Die Dokument-/Ansicht-Architektur ist ebenfalls am Tag 12, »Dateizugriff«, ein Schwerpunktthema. Hier setzen Sie diese Architektur ein, um die in der Anwendung erzeugten Daten zu speichern und wiederherzustellen. Dabei lernen Sie die Flexibilität dieser Einrichtung kennen und erfahren, wie man die unterschiedlichsten Datentypen in derselben Datei speichern und in der Anwendung in ihrer ursprünglichen Form wiederherstellen kann.

An Tag 13, »Datenbanken per ADO bearbeiten«, geht es um die neueste Datenbanktechnologie von Microsoft, die ActiveX Data Objects (ADO) und wie man sie in Visual C++ einbindet, um dem Benutzer den Datenbankzugriff zu ermöglichen.

Zur Abrundung der Woche an Tag 14, »DLLs«, lernen Sie ein weiteres Mittel kennen, um anderen Programmierern Ihren Code zugänglich zu machen: Dynamic Link Libraries (DLLs). Dabei erstellen Sie zwei verschiedene Arten von DLLs: Der eine Typ lässt sich nur zusammen mit anderen Visual C++-Anwendungen einsetzen, den zweiten kann man mit jeder anderen Entwicklungssprache von Windows nutzen.

Mit Abschluss dieser Woche beherrschen Sie die grundlegenden Aufgaben bei der Anwendungsentwicklung mit Visual C++. An dieser Stelle sollten Sie eine kleine Pause einlegen und etwas experimentieren. Versuchen Sie, die verschiedenen Arten von Anwendungen zu erstellen, Ihre Kenntnisse zu erweitern und herauszufinden, wo Ihre Grenzen liegen, bevor Sie sich der letzten Woche mit komplizierteren Themen zuwenden.

Tag 8

Text und Schriften

In den meisten Windows-Anwendungen brauchen Sie sich nicht darum zu kümmern, welche Schriften in welcher Stärke, Höhe usw. zu verwenden sind. Wenn Sie die Schrift nicht explizit festlegen, nimmt Windows eine Standardschrift für die Anwendung an. Wollen Sie eine bestimmte Schrift einsetzen, können Sie sie für ein bestimmtes Dialogfeld über die Eigenschaften des Dialogfelds spezifizieren. Manchmal möchte man aber die in der Anwendung einzusetzende Schrift selbst steuern können oder dem Benutzer die Möglichkeit geben, eine Schrift für eine bestimmte Situation auszuwählen. Für diese Fälle lernen Sie heute, wie man Schriften ändert und auflistet.

Insbesondere geht es heute darum, wie man ...

- eine Liste der verfügbaren Schriften erstellt,
- eine zu verwendende Schrift auswählt,
- Schriften dynamisch ändert.

8.1 Schriften suchen und verwenden

Bei der Arbeit mit Schriften muss man zunächst wissen, dass nicht jedes System, auf dem die Anwendung läuft, über die gleichen installierten Schriften verfügt. Schriften sind in Dateien festgelegt, die man in Windows-Systemen relativ leicht installieren und wieder entfernen kann. Jeder Computerbenutzer kann sein System mit einer beliebigen Auswahl und Anzahl von Schriften ausstatten. Wenn Sie eine Schrift festlegen, die in einem System nicht existiert, wählt Windows entweder die Standardschrift des Systems oder eine möglichst ähnliche alternative Schrift.

Man kann aber die verfügbaren Schriften des Betriebssystems abfragen. Bei dieser Methode können Sie selbst entscheiden oder den Benutzer darüber entscheiden lassen, welche der verfügbaren Schriften zu verwenden sind. Wenn Sie die verfügbaren Schriften abfragen, können Sie auch die aufgelisteten einschränken oder alle auflisten und dann verschiedene Schriften auf der Basis verschiedener Attribute auswählen.

Die verfügbaren Schriften auflisten

Um eine Liste der verfügbaren Schriften auf einem Computer zu erhalten, rufen Sie die Windows-API-Funktion (API - Application Programming Interface, Anwendungsprogrammierschnittstelle) EnumFontFamiliesEx auf. Diese Funktion teilt Windows mit, dass Sie eine Liste der Schriften im System wünschen. Bevor Sie diese Funktion verwenden und erwarten, dass sie Ihnen eine große Liste der verfügbaren Schriften liefert, müssen Sie verstehen, auf welche Weise die Funktion die Liste zurückgibt.

Callback-Funktionen

Eines der Schlüsselargumente der Funktion EnumFontFamiliesEx ist die Adresse einer anderen Funktion. Diese zweite Funktion ist eine so genannte Callback-Funktion (oder Anwendungsrückruf-Funktion), die durch das Betriebssystem aufgerufen wird. Bei fast jeder Aufzählungsfunktion im Betriebssystem Windows übergibt man die Adresse einer Callback-Funktion als Argument, da sie einmal für jedes Element in der Aufzählungsliste aufgerufen wird.

Mit anderen Worten: Man muss eine Funktion in die Anwendung einbinden, um jede einzelne Schrift des Systems zu empfangen, und dann muss man selbst eine Liste aufbauen.

Wenn man diese Funktion erstellt, um jede Schrift zu empfangen und die Liste zu erstellen, kann man die Callback-Funktion nicht nach Gutdünken selbst festlegen. Alle Callback-Funktionen sind bereits in der Windows-API vordefiniert. Man muss einen bestimmten Typ von Callback-Funktion verwenden, um die Liste der Schriften zu empfangen. Der betreffende Funktionstyp lautet EnumFontFamExProc. Dieser Funktionstyp legt fest, wie Ihre Funktion definiert sein muss, welche Argumente vorhanden sein müssen und welcher Typ des Rückgabewertes zurückzugeben ist. Der Funktionstyp legt nicht fest, wie die Funktion zu benennen ist oder wie sie intern arbeitet. Diese Aspekte liegen vollkommen in Ihrer Hand.

Die Funktion EnumFontFamiliesEx

Die Funktion EnumFontFamiliesEx, die Sie aufrufen, um die Liste der verfügbaren Schriften abzufragen, übernimmt fünf Argumente. Ein typischer Einsatzfall dieser Funktion sieht folgendermaßen aus:

```
// Eine Gerätekontextvariable erzeugen
CClientDC dc (this);
// Eine LOGFONT-Struktur deklarieren
LOGFONT lLogFont;
// Zeichensatz festlegen
lLogFont.lfCharSet = DEFAULT_CHARSET;
// Alle Schriften spezifizieren
lLogFont.lfFaceName[0] = '\0';
// Muss DEFAULT_PITCH | FF_DONTCARE sein um alle anzuzeigen
// beide Werte sind aber als 0 definiert
lLogFont.lfPitchAndFamily = 0;
// Schriftfamilien auflisten
::EnumFontFamiliesEx((HDC) dc, &lLogFont,
(FONTENUMPROC) MyEnumFontFamExProc, (LPARAM) this, 0);
```

Das erste Argument ist ein Gerätekontext, der eine Instanz der Klasse CClientDC sein kann. Jede Anwendung, die im Betriebssystem Windows läuft, verfügt über einen Gerätekontext. Dieser stellt dem Betriebssystem eine Reihe notwendiger Informationen - was für die Anwendung verfügbar ist und was nicht - bereit.

Im zweiten Argument übergibt man einen Zeiger auf eine LOGFONT-Struktur. Diese enthält die Informationen über die Schriftarten, die man auflisten möchte. In der Struktur kann man in den Strukturelementen lfCharset (Zeichensatz), lfFaceName (Schriftartname) und lfPitchAndFamily (Schriftartfamilie oder ob eine proportionale Schrift gewünscht ist) angeben, welche Eigenschaften der Zeichensatz der aufzulisten ist haben soll. Wenn Sie an allen Schriften des Systems interessiert sind, übergeben Sie für das Argument lfFaceName eine leere Zeichenkette (im obigen Beispiel wird das erste Zeichen der Zeichenkette mit einem Null-Zeichen als Endmarkierung gesetzt). Möchten Sie keine besondere LOGFONT-Struktur festlegen so übergeben Sie den Wert NULL.

Das dritte Argument gibt die Adresse der Callback-Funktion an, über die die Liste der Schriften erstellt wird. Die Adresse einer Funktion übergibt man einfach, indem man den Funktionsnamen als Argument verwendet. Der Visual C++-Compiler ersetzt den Funktionsnamen durch die Adresse der Funktion. Allerdings müssen Sie den Typ der Funktion in den Typ derjenigen Callback-Funktion umwandeln, auf die die Funktion zurückgreift.

Das vierte Argument stellt einen LPARAM-Wert dar, der an die Callback-Funktion weitergereicht wird. Windows verwendet diesen Parameter nicht, er liefert aber der Callback-Funktion einen Kontext, in dem die Schriftliste zu erstellen ist. Im Beispiel ist der übergebene Wert ein Zeiger auf das Fenster, in dem der Code ausgeführt wird. Auf diese Weise kann die Callback-Funktion diesen Zeiger verwenden, um auf eine beliebige Struktur zuzugreifen, die sie für den Aufbau der

Schriftliste benötigt. Der Zeiger kann auch der erste Knoten in einer verketteten Liste von Schriften oder ähnlichen Strukturen sein.

Das letzte Argument ist für zukünftige Versionen von Windows reserviert und muss momentan auf 0 gesetzt werden, damit die Anwendung einen Wert übergibt, der kein Fehlverhalten der Funktion bewirkt.

MFC-Exkurs: Die Struktur LOGFONT

Die Struktur LOGFONT enthält Informationen über die anzuzeigende Schriftart. Diese Informationen schließen Höhe, Breite und Gewichtung der Schrift ein und ob sie fett oder kursiv ist. Die Struktur LOGFONT ist wie folgt definiert:

```
typedef struct tagLOGFONT
{
    LONG      lfHeight;
    LONG      lfWidth;
    LONG      lfEscapement;
    LONG      lfOrientation;
    LONG      lfWeight;
    BYTE      lfItalic;
    BYTE      lfUnderline;
    BYTE      lfStrikeOut;
    BYTE      lfCharSet;
    BYTE      lfOutPrecision;
    BYTE      lfClipPrecision;
    BYTE      lfQuality;
    BYTE      lfPitchAndFamily;
    TCHAR     lfFaceName[LF_FACESIZE];
} LOGFONT;
```

Die Struktur LOGFONT entspricht den an die Funktion CreateFont übergebenen Parametern. Die Elemente der Struktur werden in Kürze zusammen mit der Funktion CreateFont detailliert besprochen. Der einzige wirkliche Unterschied zwischen der Struktur LOGFONT und den Parametern von CreateFont besteht im letzten Element, lfFaceName. Bei der Funktion CreateFont ist dieser letzte Parameter ein Zeiger auf einen String, der den Namen der Schriftart enthält. In der Struktur LOGFONT ist ein tatsächlicher String enthalten, dessen Länge auf 32 Zeichen beschränkt ist, also auf den Wert, als der LF_FACESIZE derzeit definiert ist.

C++-Exkurs: Strukturen in C++

In C++ hören Sie viel von Klassen und Strukturen. Beide scheinen sich in ihrer Funktionsweise und der Art der Interaktion miteinander sehr ähnlich zu sein. Trotz dieser Ähnlichkeiten unterscheiden sie sich jedoch. Strukturen stammen aus der Programmiersprache C. Sie sind eine Methode, eine Gruppe von Werten zu definieren, die immer zusammenbleiben sollen. Eine Struktur enthält gewöhnlich eine Anzahl von Datenelementen, die gemeinsam etwas beschreiben. Klassen tun das Gleiche. Ein entscheidender Unterschied zwischen Klassen und Strukturen ist, dass Strukturen keine Funktionalität enthalten - sie enthalten ausschließlich Datenelemente. Der technische Unterschied zwischen Strukturen und Klassen ist, dass der Standardzugriff auf die Elemente von Strukturen immer public ist, für Klassen jedoch private. Der wirkliche Unterschied zeigt sich allerdings im Gebrauch, da Strukturen nur als Behälter für Datenstrukturen verwendet werden.

Die Grundsyntax für die Definition einer Struktur ist folgende:

```
struct MeineStruktur
{
  // Datenelemente
};
```

Mit dieser Definition können Sie eine Instanz dieser Struktur über die folgende Variablendeklaration deklarieren:

```
struct MeineStruktur msMeineVariable;
```

Beide Teile können auch zusammengefasst werden:

```
struct MeineStruktur
{
  // Datenelemente
} msMeineVariable;
```

Oftmals werden die Strukturdefinitionen in den Header-Dateien eines Programmes vorgenommen.

C++-Exkurs: Benutzerdefinierte Datentypen

Wenn Sie mit Strukturen arbeiten, wollen Sie oft eine Struktur als Datentyp deklarieren, sodass Sie die Deklaration und Verwendung der Struktur in Ihrer Anwendung vereinfachen können. Dazu fügen Sie vor der Strukturdefinition das Schlüsselwort typedef ein:

```
typedef struct tMeineStruktur
{
  // Datenelemente
} MeineStruktur;
```

So können Sie die Variablendeklaration wie folgt vereinfachen:

```
MeineStruktur msMeineVariable;
```

Indem Sie einen Datentyp Ihrer Struktur deklarieren, müssen Sie nicht bei jeder Deklaration einer Instanz dieser Struktur angeben, dass es sich um eine Struktur handelt. Sie müssen sich bei der Deklaration von Datentypen nicht auf Strukturen beschränken, Sie können auch Ihre eigenen Datentypen deklarieren, die anderen Basisdatentypen gleichwertig sind:

```
typedef long MeinLong;
```

So etwas tun Sie gewöhnlich, wenn Sie einen ständig gebrauchten Datentyp haben und den Zweck der Variablen im Datentyp widerspiegeln wollen. Viele Windows-Datentypen sind auf diese Art definiert, beispielsweise DWORD, HANDLE und BYTE. All diese Standardvariablen wurden mit dem Schlüsselwort typedef definiert.

Sie können auch Datentypen als Zeiger auf andere Datentypen und Strukturen deklarieren, indem Sie vor dem Typnamen einen Stern einfügen:

```
typedef long* PLONG;
```

In diesem Fall ist der Datentyp PLONG als Zeiger auf eine long-Variable definiert.

Es ist allgemein üblich, bei der Deklaration eines Zeiger-Datentyps mit typedef einen Namen für den Datentyp zu wählen, der mit »P« (für Pointer - Zeiger) oder »LP« (Long Pointer, ein Überbleibsel aus 16-Bit-Tagen) beginnt, gefolgt vom Variablentyp, gewöhnlich in Großbuchstaben. Wenn Sie beispielsweise einen Datentyp deklarieren, bei dem es sich um einen Zeiger auf char handelt, würden Sie ihm den Namen PCHAR geben. Wenn Sie einen Datentyp als Zeiger auf eine Struktur deklarieren, ist der Name gewöhnlich »P« gefolgt vom Namen der Struktur, alles in Großbuchstaben (beispielsweise wäre PMEINESTRUKTUR ein Zeiger auf eine Instanz der Struktur MeineStruktur).

Man darf das nicht mit den #define- und const-Deklarationen verwechseln, die gewöhnlich ebenfalls groß geschrieben werden. Es ist nicht das Gleiche, doch es ist üblich, in all diesen Fällen Großschreibung zu verwenden.

Der Funktionstyp EnumFontFamProc

Ihre Callback-Funktion muss als unabhängige Funktion definiert sein und nicht als Element irgendeiner C++-Klasse. Eine typische Funktionsdeklaration von EnumFontFamExProc sieht folgendermaßen aus:

```
int CALLBACK EnumFontFamProc(
  LPENUMLOGFONTEX lpelf,
  LPNEWTEXTMETRICEX lpntm,
  DWORD nFontType,
  LPARAM lParam)
{
  // Einen Zeiger auf das Dialogfenster erzeugen
  CMyDlg* pWnd = (CMyDlg*) lParam;
  if (pWnd)
  {
    // Den Schriftnamen in das Listenfeld hinzufügen
    pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
    // 1 zurückgeben, um die Aufzählung fortzusetzen
    return 1;
  }
  // Zeiger auf Fenster ist NULL, Auflistung beenden
  return 0;
}
```

Das erste Argument an diese Funktion ist ein Zeiger auf eine ENUMLOGFONTEX-Struktur. Diese enthält Informationen über die logischen Attribute der Schrift einschließlich Schriftname, Stil und Schriftform. Unter ein und demselben Schriftnamen können mehrere Schriften mit unterschiedlichen Stilen - Standard, Fett, Kursiv und Fett Kursiv - und Sprachversionen - Arabisch, Baltisch, viele andere und zum Schluss Western - erscheinen.

Das zweite Argument ist ein Zeiger auf eine NEWTEXTMETRICEX-Struktur, die Informationen über die physikalischen Attribute der Schrift wie Höhe, Breite und Zeichenabstand enthält. Es handelt sich dabei um relative Werte, da sie je nach Größe der Schrift zu skalieren sind.

Das dritte Argument ist ein Flag, das den Typ der Schrift spezifiziert. Dieses Flag kann eine Kombination der folgenden Werte darstellen:

DEVICE_FONTTYPE	Gibt an, dass der Zeichensatz ein vom Ausgabegerät bereitgestellter Zeichensatz ist (z.B. eingebauter Druckerzeichensatz) oder das Ausgabegerät die Übermittlung von Zeichensätzen unterstützt. Bei Rasterausgabegeräten wie Bildschirm oder Matrixdrucker ist dieses Flag immer 0.
RASTER_FONTTYPE	Gibt an, dass der Zeichensatz ein gerasterter Bitmap-Zeichensatz ist. Diese Zeichensätze sind nur in bestimmten Größen im System vorhanden (z.B. MS Serif).
TRUETYPE_FONTTYPE	Gibt an, dass der Zeichensatz ein TrueType-Zeichensatz ist. Ist weder RASTER_FONTTYPE noch TRUETYPE_FONTTYPE gesetzt, so handelt es sich um einen Vektor-Zeichensatz (z.B. Roman).

Schließlich gibt das vierte Argument den Wert an, der an die Funktion EnumFontFamiliesEx übergeben wurde. Im Beispiel war das ein Zeiger auf das Dialogfeld, in dem die Liste der Schriften zu erstellen ist. Wenn man diesen Wert als Zeiger auf das Dialogfeld umwandelt, kann

die Funktion auf ein Listenfeld zugreifen, um die Schriftnamen hinzuzufügen.

Der Rückgabewert aus dieser Funktion bestimmt, ob sich die Liste der Schriften fortsetzt. Liefert die Funktion 0, beendet das Betriebssystem die Auflistung der verfügbaren Schriften. Wenn die Funktion einen von 0 verschiedenen Wert liefert, setzt das Betriebssystem die Liste der verfügbaren Schriften fort.

MFC-Exkurs: Die Struktur ENUMLOGFONTEX

Die Struktur ENUMLOGFONTEX enthält Informationen über aufgelistete Schriftarten. Die Struktur ist wie folgt definiert:

```
typedef struct tagENUMLOGFONTEX
{
    LOGFONT     elfLogFont;
    TCHAR       elfFullName[LF_FULLFACESIZE];
    TCHAR       elfStyle[LF_FACESIZE];
    TCHAR       elfScript[LF_FACESIZE];
} ENUMLOGFONTEX, FAR *LPENUMLOGFONTEX;
```

Das Element elfLogFont ist eine LOGFONT-Struktur. Es enthält Informationen über die bestimmte Schriftart und ihre Fähigkeiten.

Das Element elfFullName ist ein String, der einen eindeutigen Namen für die Schriftart enthält, außer unter Windows 95/98/ME. Hier gibt das Element den eindeutigen Namen nur für TrueType-Schriftarten an. Für Nicht-TrueType-Schriften unter Windows 95/98/ ME ist der eindeutige Schriftartname das Element lfFaceName der LOGFONT-Struktur elfLogFont.

Das Element elfStyle ist ein String, der die Auszeichnung der Schrift beschreibt. Beispiele dafür wären "Standard", "Fett", "Kursiv" oder "Fett Kursiv". Die möglichen Werte variieren je nach Sprachversion von Windows.

Das Element elfScript ist ein String, der den von der Schriftart verwendeten Zeichensatz angibt. Beispiele wären "Westeuropäisch" oder "Kyrillisch".

MFC-Exkurs: Die Struktur NEWTEXTMETRICEX

Die Struktur NEWTEXTMETRICEX enthält Informationen über eine Schriftart. Alle Informationen aus zwei weiteren Strukturen werden kombiniert, um die Struktur NEWTEXTMETRICEX zu erstellen:

```
typedef struct tagNEWTEXTMETRICEX
{
    NEWTEXTMETRIC   ntmTm;
    FONTSIGNATURE   ntmFontSig;
} NEWTEXTMETRICEX;
```

Die erste dieser beiden Strukturen, NEWTEXTMETRIC, ist wie folgt definiert:

```
typedef struct tagNEWTEXTMETRIC
{
    LONG        tmHeight;
    LONG        tmAscent;
    LONG        tmDescent;
    LONG        tmInternalLeading;
    LONG        tmExternalLeading;
    LONG        tmAveCharWidth;
    LONG        tmMaxCharWidth;
    LONG        tmWeight;
    LONG        tmOverhang;
```

```
    LONG        tmDigitizedAspectX;
    LONG        tmDigitizedAspectY;
    TCHAR       tmFirstChar;
    TCHAR       tmLastChar;
    TCHAR       tmDefaultChar;
    TCHAR       tmBreakChar;
    BYTE        tmItalic;
    BYTE        tmUnderlined;
    BYTE        tmStruckOut;
    BYTE        tmPitchAndFamily;
    BYTE        tmCharSet;
    DWORD       ntmFlags;
    UINT        ntmSizeEM;
    UINT        ntmCellHeight;
    UINT        ntmAvgWidth;
} NEWTEXTMETRIC;
```

Die Struktur NEWTEXTMETRIC enthält Informationen über die Schriftart, zum Beispiel deren Größe, mittlere und maximale Zeichenbreite und wie viel Überhang sie hat (wie weit Zeichen wie g, q und p unter die Linie reichen, auf der alle Zeichen sitzen).

Die zweite Struktur, FONTSIGNATURE, ist wie folgt definiert:

```
typedef struct tagFONTSIGNATURE
{
    DWORD fsUsb[4];
    DWORD fsCsb[2];
} FONTSIGNATURE;
```

Die beiden Elemente dieser Struktur zeigen auf Informationen über in dieser Schriftart enthaltene Glyphen. Diese Informationen sind unabdingbar, um herauszufinden, welche Sprachen eine bestimmte Schriftart verwenden können. Das ist elementar, um Windows in verschiedenen Sprachen (zum Beispiel Umlaute oder Akzente) und Zeichensätze (zum Beipiel Chinesisch oder Koreanisch) lokalisieren zu können.

Eine Glyphe ist die Form eines Schriftzeichens. Die Form, die Glyphen annehmen, hängt von dem Schrifttyp ab. In Rasterschriften ist eine Glyphe die vom System zur Anzeige eines Zeichens oder Symbols in der Schriftart verwendete Bitmap. In Vektorschriften ist eine Glyphe die Sammlung von Punkten, die verbunden werden, um die Zeichen oder Symbole in der Schriftart zu zeichnen (im Grunde werden die Zeichen durch »verbinde die Punkte« dargestellt). Bei TrueType- und Opentype-Schriften ist ein Glyphe eine Sammlung von Befehlen zum Zeichnen von Linien und Kurven, die ausgeführt werden, um die Zeichen oder Symbole einer Schriftart zu zeichnen.

Eine Schrift verwenden

Um eine bestimmte Schrift in einer Anwendung einzusetzen, ruft man eine Instanz der Klasse CFont auf. Über die Methode CreateFont lässt sich die zu verwendende Schriftart zusammen mit deren Größe, Stil und Ausrichtung festlegen. Nachdem Sie eine Schrift erstellt haben, können Sie ein Steuerelement oder Fenster anweisen, diese Schrift zu verwenden, indem Sie die Methode SetFont des jeweiligen Objekts aufrufen. Dieser Ablauf sieht beispielsweise wie folgt aus:

```
CFont m_fFont;    // Die zu verwendende Schrift

// Die zu verwendende Schrift erzeugen
m_fFont.CreateFont(12, 0, 0, 0, FW_NORMAL,
        0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
```

```
        CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH | FF_DONTCARE,
         m_sFontName);
// Die Schrift für den Anzeigebereich festlegen
m_ctlDisplayText.SetFont(&m_fFont);
```

Die im obigen Code verwendete CFont-Variable sollte man als Member-Variable der Klasse, in der der Code steht, deklarieren. Im Beispielcode ist die Variable unmittelbar über der Stelle, wo sie verwendet wird, deklariert, um die Art und Weise ihrer Deklaration zu zeigen. Die Variable sollte nicht als lokale Variable in einer Funktion deklariert oder verwendet werden.

Es sieht so einfach aus - Sie müssen lediglich zwei Funktionen aufrufen. Aber an die Funktion CreateFont ist eine furchtbar große Zahl von Argumenten zu übergeben. Gerade diese Argumente machen die Methode CreateFont zu einer flexiblen Funktion mit umfangreicher Funktionalität. Nachdem man die Schrift erstellt hat, kann man sie ganz einfach einsetzen, indem man die Schrift an die Methode SetFont übergibt. Diese Methode ist eine Elementfunktion der Klasse CWnd und demzufolge für alle Fenster- und Steuerelementklassen in Visual C++ verfügbar. Das bedeutet, dass man dieses Verfahren für alle visuellen Objekte in einer Visual C++-MFC-Anwendung einsetzen kann.

Um die Arbeitsweise der Funktion CreateFont zu verstehen, sehen wir uns die einzelnen Argumente an, die an die Funktion zu übergeben sind. Die Funktion ist folgendermaßen definiert:

```
BOOL CreateFont(
    int nHeight,
    int nWidth,
    int nEscapement,
    int nOrientation,
    int nWeight,
    BYTE bItalic,
    BYTE bUnderline,
    BYTE cStrikeOut,
    BYTE nCharSet,
    BYTE nOutPrecision,
    BYTE nClipPrecision,
    BYTE nQuality,
    BYTE nPitchAndFamily,
    LPCTSTR lpszFaceName);
```

Das erste dieser Argumente, nHeight, legt die Höhe der zu verwendenden Schrift fest. Dieser logische Wert wird in Geräte-Einheiten übersetzt. Ist der Wert gleich 0, kommt ein plausibler Standardwert zur Anwendung. Wenn der Wert größer als 0 ist, wird der Wert gegen die Zellenhöhe des Zeichensatzes verglichen, ein negativer Wert wird gegen die Zeichenhöhe verglichen.

Die Zeichenhöhe (character height) ist der Abstand zwischen der Spitze eines großen "A" und dem unteren Bogen eines kleinen "g". Bei der Zellenhöhe (cell height) wird der Abstand um den Platzbedarf von Akzenten ergänzt (»Â«), dieser Bereich wird auch internal lead genannt.

Das zweite Argument, nWidth, spezifiziert die durchschnittliche Breite der Zeichen in der Schrift. Ein Wert von 0 spezifiziert keine besondere Breite. Dieser logische Wert wird in einen physikalischen Wert konvertiert, genau wie es bei der Höhe der Fall ist.

Das dritte Argument, nEscapement, bestimmt den Winkel gegenüber der x-Achse, in dem eine Textzeile ausgegeben wird. Dieser Wert ist in Schritten von 0,1 Grad entgegen dem Uhrzeigersinn spezifiziert. Möchte man senkrechten Text von unten nach oben ausgeben, gibt man 900 für dieses Argument an. Der Wert 0 steht für normalen waagerecht von links nach rechts verlaufenden Text.

Das vierte Argument, nOrientation, bestimmt den Winkel der einzelnen Zeichen in der Schrift. Das funktioniert genau wie beim vorherigen Argument, steuert aber die zeichenweise Ausgabe und nicht die zeilenweise Textausgabe. Um Zeichen kopfüber auszugeben, setzt man diesen Wert auf 1800. Mit dem Wert 900 werden die Zeichen liegend ausgegeben. Die Richtung der Drehung ändert sich mit der Richtung der y-Achse.

Im fünften Argument, nWeight, gibt man die Gewichtung oder die Strichstärke der Schrift an. Dieser Wert umfasst einen Bereich von 0 bis 1000, gemessen in der Anzahl der geschwärzten Pixel pro 1000. Für dieses Argument sind Konstanten definiert, damit man diesen Wert einfach und einheitlich steuern kann. Tabelle 8.2 listet diese Konstanten auf.

Konstante	Wert
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_BLACK	900
FW_HEAVY	900

Tabelle 8.1: Konstanten für die Gewichtung der Schrift

Sie haben bereits mehrere Sätze konstanter Werte gesehen und Sie werden in diesem Buch noch einige mehr sehen. Vielleicht wundern Sie sich über die Namensgebung dieser Konstanten. Beachten Sie, dass alle ein Präfix aus zwei bis drei Buchstaben besitzen. Dieses Präfix zeigt gewöhnlich an, wofür die Konstante verwendet wird, oder es ist die Abkürzung der Verwendung, für den die Konstante gedacht ist. In diesem Fall tragen alle Konstanten die Abkürzung FW, die für font weight (Gewichtung der Schrift) steht. Betrachten Sie bei allen Konstanten in diesem Buch und in der Windows-API (wie auch in der MFC-Klassenbibliothek) die Präfixe und vergleichen Sie sie mit der Verwendung der Konstante, um herauszufinden, wofür sie stehen.

Die tatsächliche Interpretation und Verfügbarkeit dieser Gewichte hängt von der Schrift ab.

Einige Schriften haben nur die Gewichte FW_NORMAL, FW_REGULAR und FW_BOLD. Wenn man FW_DONTCARE angibt, wird ein Standardwert wie bei den meisten der übrigen Argumente eingesetzt.

Das sechste Argument, bItalic, legt fest, ob die Schrift kursiv auszugeben ist. Es handelt sich um einen Booleschen Wert, wobei 0 für nicht kursive Schrift und jeder andere Wert für eine kursive Schrift steht.

Das siebente Argument, bUnderline, ist ebenfalls ein Boolescher Wert und kennzeichnet mit einer 0, dass die Schrift nicht unterstrichen ist, während jeder andere Wert für eine unterstrichene Schrift steht.

Diese Werte sind zwar von ihrer Natur her Boolesche Werte, werden jedoch als BYTE-Datentypen deklariert. Bis der Datentyp bool in den C++-Standard aufgenommen wurde, waren Boolesche Datentypen als Integer definiert, wobei 0 als FALSE galt und jeder andere Wert als TRUE. So können Boolesche Ausdrücke in Bedingungsanweisungen funktionieren - man überprüft den Wert eines Zeigers, um seine Gültigkeit festzustellen. Wurde der Zeiger mit NULL initialisiert, bedeutet jeder Wert ungleich Null, dass der Zeiger auf etwas gesetzt wurde. In diesen Parametern wird die gleiche Form Boolescher Werte verwendet. Wenn man sie als Boolesche Werte behandelt, ist ein Nullwert FALSE und jeder andere Wert TRUE.

Das achte Argument, cStrikeOut, gibt an, ob die Zeichen in der Schrift mit einer Linie durch das Zeichen - also durchgestrichen - darzustellen sind. Auch dieser Boolesche Wert bedeutet TRUE - durchgestrichen - bei allen von 0 verschiedenen Werten und FALSE - nicht durchgestrichen - bei 0.

Das neunte Argument, nCharSet, legt den Zeichensatz der Schrift fest. Einige verfügbaren Konstanten für diesen Wert sind in hier aufgeführt. Zusätzlich zu den hier aufgeführten finden Sie in der Online-Hilfe noch weitere Zeichensätze, einige sind jedoch nur bei bestimmten Sprachversionen von Windows zu verwenden. Der Wert DEFAULT_CHARSET kann bei Windows 95/98/ME zu unerwünschten Ergebnissen führen, unter Windows NT/2000/XP wird ein zur aktuellen Regionaleinstellung passender Zeichensatz verwendet.

- ANSI_CHARSET
- DEFAULT_CHARSET
- SYMBOL_CHARSET
- SHIFTJIS_CHARSET
- OEM_CHARSET

Das System, auf dem Ihre Anwendung läuft, kann andere Zeichensätze verwenden. Darüber hinaus ist der OEM-Zeichensatz systemabhängig. Das verkompliziert die ganze Sache für Systeme unterschiedlicher Hersteller. Wenn Sie einen dieser Zeichensätze verwenden, ist es riskant, die auszugebenden Strings zu manipulieren, sodass es sich empfiehlt, einfach den anzuzeigenden String zu übergeben.

Das zehnte Argument, nOutPrecision, legt fest, wie nahe die Ausgabe den angeforderten Werten für Höhe, Breite, Zeichenausrichtung, Textwinkel und Zeichenabstand der Schrift entsprechen muss. Die verfügbaren Werte für dieses Argument lauten:

- OUT_CHARACTER_PRECIS (wird nicht verwendet)
- OUT_DEFAULT_PRECIS (keine besonderen Angaben gewünscht)
- OUT_DEVICE_PRECIS
- OUT_OUTLINE_PRECIS (erst ab Windows NT verfügbar)
- OUT_RASTER_PRECIS
- OUT_STRING_PRECIS
- OUT_STROKE_PRECIS (wird nicht verwendet)
- OUT_TT_ONLY_PRECIS
- OUT_TT_PRECIS

Die Werte OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS und OUT_TT_PRECIS steuern, welche Schrift zu wählen ist, wenn mehrere Schriften mit dem gleichen Namen existieren. Wenn Sie zum Beispiel OUT_TT_PRECIS verwenden und eine Schrift sowohl als TrueType- als auch als Rasterversion spezifizieren, kommt die TrueType-Version zum Einsatz. Praktisch erzwingt OUT_TT_PRECIS die Verwendung einer TrueType-Schrift, selbst wenn die angegebene Schrift nicht in einer TrueType-Version vorliegt.

Das elfte Argument, nClipPrecision, legt fest, wie die Zeichen, die teilweise außerhalb des Anzeigebereichs liegen, abzuschneiden sind. Die Werte für dieses Argument lauten:

- CLIP_CHARACTER_PRECIS
- CLIP_DEFAULT_PRECIS
- CLIP_EMBEDDED
- CLIP_ENCAPSULATE
- CLIP_LH_ANGLES
- CLIP_MASK
- CLIP_STROKE_PRECIS
- CLIP_TT_ALWAYS

Diese Werte lassen sich mit OR verknüpfen, um eine Kombination der Clipping-Verfahren zu spezifizieren, eine Erläuterung zu den Werten findet sich in der Online-Hilfe.

Das zwölfte Argument, nQuality, spezifiziert die Ausgabequalität und gibt an, wie genau das GDI (Graphics Device Interface) versuchen muss, die logischen Attribute in die physikalische Schriftausgabe umzusetzen. Die verfügbaren Werte für dieses Argument lauten:

- ANTIALIASED_QUALITY (auf dem Bildschirm werden die Schriftarten geglättet dargestellt)
- CLEARTYPE_QUALITY (nur in Windows XP verfügbar)
- DEFAULT_QUALITY
- DRAFT_QUALITY
- NONANTIALIASED_QUALITY
- PROOF_QUALITY

Im dreizehnten Argument, nPitchAndFamily, legt man den Zeichenabstand und die Schriftfamilie fest. Dieser Wert besteht eigentlich aus zwei Werten, die per OR verknüpft sind, um einen Kombinationswert zu bilden. Der erste Satz von verfügbaren Werten lautet:

- DEFAULT_PITCH
- VARIABLE_PITCH (die einzelnen Buchstaben sind unterschiedlich breit)
- FIXED_PITCH (alle Buchstaben haben die gleiche Breite)

Diese Werte geben den Zeichenabstand der Schrift an. Die zweite Gruppe von verfügbaren Werten gibt die Schriftfamilie an. Die verfügbaren Werte für diesen Teil des Arguments heißen:

- FF_DECORATIVE (dekorative Schriftart)
- FF_DONTCARE
- FF_MODERN (nicht proportionale Schriftart)
- FF_ROMAN (proportionale Schriftart mit Serifen)

- FF_SCRIPT (eine wie Handschrift aussehende Schriftart)
- FF_SWISS (proportionale Schriftart ohne Serifen)

Die Schriftfamilie beschreibt die allgemeine Erscheinung einer Schrift. Man kann mit dem Wert für die Schriftfamilie eine alternative Schrift wählen, wenn eine bestimmte Schrift nicht im System vorhanden ist.

Das letzte Argument, lpszFacename, ist ein normaler C-String, der den Namen der zu verwendenden Schrift angibt. Dieser Schriftname stammt aus der Schriftinformation, die die Callback-Funktion EnumFontFamExProc zurückgibt. Wird anstelle eines C-Strings der Wert NULL übergeben, so gibt die Funktion die erste Schriftart zurück, auf welche die anderen Angaben passen.

8.2 Schriften verwenden

Heute erstellen Sie eine Anwendung, die dem Benutzer die Möglichkeit gibt, die anzuzeigende Schrift aus einer Liste der verfügbaren Schriften auszuwählen. Der Benutzer gibt den anzuzeigenden Text in der ausgewählten Schrift ein, um sich vom Aussehen derselben ein Bild machen zu können.

Das Anwendungsgerüst erstellen

Die heutige Anwendung erstellen Sie mit den folgenden Schritten:

1. Erzeugen Sie ein neues MFC-Anwendung Visual C++-Projekt. Nennen Sie das Projekt TextFonts.

2. Übernehmen Sie die Standardeinstellungen, die Sie bereits in den Projekten der vergangenen Tage verwendet haben. Als Titel der Anwendung geben Sie Schriften ein.
3. Entwerfen Sie das Hauptdialogfeld gemäß Abbildung 8.1 mit den Eigenschaften nach Tabelle 8.2.

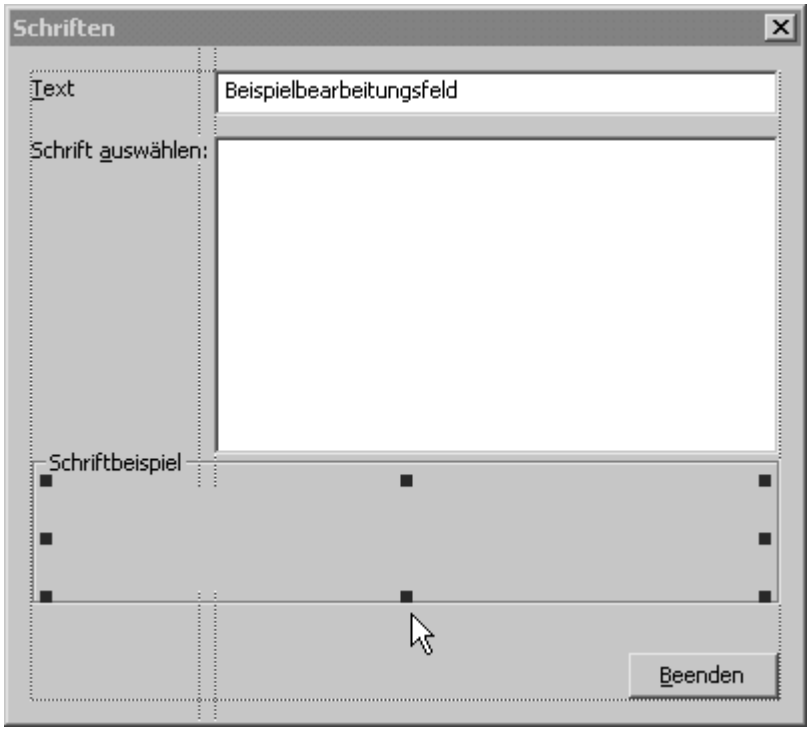


Abbildung 8.1: Der Entwurf des Hauptdialogfelds

Objekt	Eigenschaft	Einstellung
Text	Beschriftung	&Text eingeben:
Eingabefeld	ID	IDC_ESAMPTEXT

Text	Beschriftung	Schrift &auswählen:
Listenfeld	ID	IDC_LFONTS
Gruppenfeld	Beschriftung	Schriftbeispiel
Text (innerhalb des Gruppenfeldes; so anpassen, dass das Gruppenfeld ausgefüllt wird)	ID Beschriftung	IDC_DISPLAYTEXT. Leerer String
Schaltfläche	ID Beschriftung	IDC_EXIT &Beenden

Tabelle 8.2: Eigenschaftseinstellungen der Steuerelemente

4. Verwenden Sie die gleichen Aktionen wie an den vorhergegangenen Tagen, um die Variablen in Tabelle 8.4 zu den Steuerelementen hinzuzufügen.

Objekt	Name	Kategorie	Typ
IDC_DISPLAYTEXT (statisches Text-Steuerelement ohne Beschriftung im Gruppenfeld)	m_ctlDisplayText m_strDisplayText	Control Value	CStatic CString
IDC_LFONTS	m_ctlFontList m_strFontName	Control Value	CListBox CString
IDC_ESAMPTEXT	m_strSampText	Value	CString

Tabelle 8.3: Variablen der Steuerelemente

Denken Sie daran, zuerst die Kategorie zu setzen, bevor Sie den Typ aus dem Kombinationsfeld verfügbarer Typen auswählen.

5. Fügen Sie wie in den gestrigen Anwendungen der Schaltfläche IDC_EXIT eine Funktion hinzu, um die Anwendung zu schließen.

Eine Schriftliste aufbauen

Damit Sie die Liste der Schriften erstellen können, fügen Sie eine Callback-Funktion hinzu. Damit lassen Sie alle Schriften auflisten und in das Listenfeld des Dialogfelds schreiben. Fügen Sie zu diesem Zweck am Anfang der Header-Datei TextFontsDlg.h die Funktionsdeklaration aus Listing 8.1 ein. Diese Funktion lässt sich nicht mit den in Visual C++ verfügbaren Werkzeugen hinzufügen. Sie müssen die Datei öffnen und die Funktion in eigener Regie eintragen.

Listing 8.1: Die Deklaration der Callback-Funktion in der Header-Datei TextFontsDlg.h

```
1: #pragma once
2: #include "afxwin.h"
3:
4: int CALLBACK MyEnumFontProc(ENUMLOGFONTEX* lpelf,
5:     NEWTEXTMETRICEX* lpntm, DWORD nFontType, LPARAM lParam);
```

Nachdem Sie die Funktionsdeklaration in die Header-Datei aufgenommen haben, öffnen Sie die Quellcodedatei TextFontsDlg.cpp, gehen an das Ende der Datei und fügen die Funktionsdefinition aus Listing 8.2 ein.

Listing 8.2: Die Definition der Callback-Funktion in der Quelldatei TextFontsDls.cpp

```
 1: int CALLBACK MyEnumFontProc(ENUMLOGFONTEX* lpelf,
 2:     NEWTEXTMETRICEX* lpntm, DWORD nFontType, LPARAM lParam)
 3: {
 4:   // Zeiger auf Dialogfenster erzeugen
 5:   CTextFontsDlg* pWnd = (CTextFontsDlg*) lParam;
 6:
 7:   if (pWnd)
 8:   {
 9:     // Schriftname in Listenfeld einfügen
10:     pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
11:     // 1 zurückgeben, um Schriftauflistung fortzusetzen
12:     return 1;
13:   }
14:   return 0;
15: }
```

In anderen Code-Listings ist der Teil, den Sie eingeben müssen, fett gedruckt. In Listing 8.2 ist nichts fett gedruckt. Das liegt daran, dass Sie das gesamte Listing eingeben müssen. Keiner der codegenerierenden Assistenten in der Visual C++-Umgebung hat schon einen Teil des Codes hinzugefügt.

Damit haben Sie die Callback-Funktion definiert und müssen nun noch eine Funktion hinzufügen, um die Liste der Schriften vom Betriebssystem abzurufen. Führen Sie dazu die folgenden Schritte aus:

1. Wählen Sie die Klassenansicht.

2. Markieren Sie die Klasse CTextFontsDlg, klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.
3. Legen Sie den Funktionstyp mit void, den Funktionsnamen als FillFontList und den Zugriff als private fest. Klicken Sie auf Fertig stellen, um das Dialogfeld zu schließen und die Funktion hinzuzufügen.
4. Bearbeiten Sie die Funktionsdefinition gemäß Listing 8.3.

Listing 8.3: Die Funktion FillFontList

```
 1: void CTextFontsDlg::FillFontList(void)
 2: {
 3:   LOGFONT lf;
 4:
 5:   lf.lfCharSet = DEFAULT_CHARSET;
 6:   // Alle Schriftarten festlegen
 7:   lf.lfFaceName[0] = '\0';
 8:   // Muss DEFAULT_PITCH | FF_DONTCARE sein um alle anzuzeigen
 9:   // beide Werte sind aber als 0 definiert
10:   lf.lfPitchAndFamily = 0;
11:   // Listenfeld leeren
12:   m_ctlFontList.ResetContent();
13:   // Gerätekontextvariable erzeugen
14:   CClientDC dc (this);
15:   // Schriftfamilien aufzählen
16:   ::EnumFontFamiliesEx((HDC) dc, &lf,
```

```
17:          (FONTENUMPROC) MyEnumFontProc, (LPARAM) this, 0);
18: }
```

5. Bearbeiten Sie die Funktion OnInitDialog gemäß Listing 8.4, um die Funktion FillFontList aufzurufen.

Listing 8.4: Die bearbeitete Funktion OnInitDialog

```
 1: BOOL CTextFontsDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
 5:   // Symbol für dieses Dialogfeld festlegen.
 6:   // Wird automatisch erledigt, wenn das
 7:   // Hauptfenster der Anwendung kein Dialogfeld ist
 8:
 9:   SetIcon(m_hIcon, TRUE);     // Großes Symbol verwenden
10:   SetIcon(m_hIcon, FALSE);    // Kleines Symbol verwenden
11:
12:   // TODO: Hier zusätzliche Initialisierung einfügen
13:   // Listenfeld für Schriften füllen
14:   FillFontList();
15:
16:   return TRUE;  // Geben Sie TRUE zurück, außer ein
17:                 // Steuerelement soll den Fokus erhalten
18: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollte das Listenfeld mit den Namen aller im System verfügbaren Schriften gefüllt sein. Allerdings gibt es einen Aspekt dieser Liste, den Sie in Ihrer Anwendung wahrscheinlich nicht haben möchten. Abbildung 8.2 zeigt viele doppelte Einträge in der Liste der Schriften im Listenfeld. Es wäre schön, wenn man diese Duplikate ausblenden könnte, um nur noch eine Zeile pro Schriftart anzuzeigen.

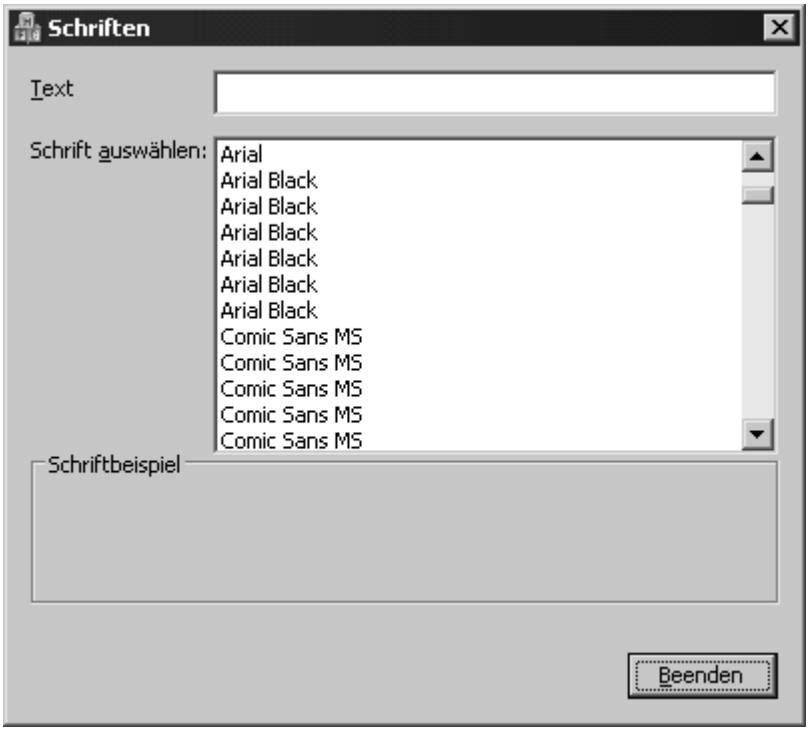


Abbildung 8.2: Alle Schriften im System auflisten

Es zeigt sich, dass der Funktionsaufruf von EnumFontFamiliesEx von Haus aus synchron ist. Das bedeutet, dass die Funktion erst dann zurückkehrt, wenn alle Schriften im System in Aufrufen Ihrer Callback-Funktion aufgelistet sind. Mit etwas Code in der Funktion FillFontList lassen sich aber alle doppelten Einträge eliminieren, sobald das Listenfeld gefüllt ist. Dazu modifizieren Sie die Funktion FillFontList gemäß Listing 8.5.

Listing 8.5: Die modifizierte Funktion FillFontList

```
 1: void CTextFontsDlg::FillFontList(void)
 2: {
 3:   int iCurCount;             // Aktuelle Schrift
 4:   CString strCurFont;        // Aktueller Schriftname
 5:   CString strPrevFont = "";  // Vorheriger Schriftname
 6:   LOGFONT lf;
 7:
 8:   lf.lfCharSet = DEFAULT_CHARSET;
 9:   // Alle Schriftarten festlegen
10:   lf.lfFaceName[0] = NULL;
10:   // Muss außer bei Hebräisch oder Arabisch 0 sein
11:   lf.lfPitchAndFamily = 0;
12:
13:   // Listenfeld leeren
14:   m_ctlFontList.ResetContent();
15:   // Gerätekontextvariable erzeugen
16:   CClientDC dc (this);
17:   // Schriftfamilien aufzählen
18:   ::EnumFontFamiliesEx((HDC) dc, &lf,
29:         (FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
21:   // Schleife vom letzten zum ersten Eintrag im Listenfeld,
22:   // um doppelte Einträge zu suchen und zu löschen
23:   for (iCurCount = m_ctlFontList.GetCount(); iCurCount > 0;
24:        iCurCount--)
25:   {
26:     // Aktuellen Schriftnamen holen
27:     m_ctlFontList.GetText((iCurCount - 1), strCurFont);
28:     // Mit vorherigem Schriftnamen identisch?
29:     if (strCurFont == strPrevFont)
30:     {
31:       // Wenn ja, dann löschen
32:       m_ctlFontList.DeleteString((iCurCount - 1));
33:     }
34:     else
35:     {
36:       // Vorherigen auf aktuellen Schriftnamen setzen
37:       strPrevFont = strCurFont;
38:     }
39:   }
40: }
```

Beachten Sie, dass die for-Schleife am Ende der Liste beginnt und rückwärts zum Anfang arbeitet. Damit kann man den aktuellen Eintrag löschen, ohne sich um den Schleifenzähler kümmern zu müssen. Ansonsten müsste man den Zähler korrigieren, um keine Zeile zu überspringen. Wenn Sie jetzt die Anwendung kompilieren und ausführen, sollten keine doppelten Einträge mehr in der Liste der verfügbaren Schriften zu sehen sein.

C++-Exkurs: Zeiger umwandeln

Eine der Stärken von C/C++ ist die Fähigkeit, Zeiger umzuwandeln. Wenn Sie einen Zeiger als bestimmten Daten- oder Klassentyp deklarieren, behandelt er alles, worauf er zeigt, als diesen Typ. Wenn Sie einen Zeiger als Zeiger auf eine CWnd-Klasse deklarieren und er auf eine Instanz der Klasse CDialog zeigt, können Sie nur die Member-Funktionen der Klasse CWnd aufrufen und haben auch nur Zugriff auf deren Variablen. Sie können aber diesen Zeiger in einen Zeiger auf eine CDialog-Klasse umwandeln und so auf die Member-Funktionen und Variablen dieser Klasse zugreifen. Das Gleiche kann man mit Zeigern auf andere Datentypen tun, beispielsweise einen Zeiger auf char in einen Zeiger auf integer umwandeln, um auf den ASCII-Wert des Zeichens zuzugreifen.

Sie können einen Zeiger in einen anderen Typ umwandeln, indem Sie den gewünschten Datentyp in Klammern einschließen, zusammen mit einem Stern, der anzeigt, dass Sie den Zeiger immer noch als Zeiger

umwandeln:

```
(NeuerTyp*)pPtr;
```

Wenn Sie beispielsweise einen Zeiger auf eine CWnd-Klasse haben und daraus einen Zeiger auf eine CDialog-Klasse machen möchten, können Sie das folgendermaßen tun:

```
CDialog* pDlg = (CDialog*)pWndPtr;
```

Sie können auch direkt Methoden umwandeln und aufrufen, ohne den Zeiger auf einen anderen Zeiger zu setzen:

```
((CDialog*)pWndPtr)->DoModal();
```

Text für Schriftprobe festlegen

Bevor man die Schrift für den Benutzer anzeigen kann, muss man etwas Text in den Anzeigebereich schreiben. In das Eingabefeld im oberen Teil des Dialogfelds gibt der Benutzer den Text ein, der in der ausgewählten Schrift anzuzeigen ist. Führen Sie die folgenden Schritte aus, um die Schriftprobe darzustellen:

1. Übernehmen Sie in die Funktion OnInitDialog den Code aus Listing 8.6, um das Eingabefeld zu initialisieren und Text anzuzeigen.

Listing 8.6: Die modifizierte Funktion OnInitDialog

```
 1: BOOL CTextFontsDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
...
 5:   // Symbol für dieses Dialogfeld festlegen.
 6:   // Wird automatisch erledigt
 7:   // wenn das Hauptfenster der Anwendung kein Dialogfeld ist
 8:   SetIcon(m_hIcon, TRUE);         // Großes Symbol verwenden
 9:   SetIcon(m_hIcon, FALSE);        // Kleines Symbol verwenden
10:
11:   // TODO: Hier zusätzliche Initialisierung einfügen
12:   // Listenfeld für Schriften füllen
13:   FillFontList();
14:
15:   // Einzugebenden Text initialisieren
16:   m_strSampText = "Ein Mustertext in C++";
17:   // Text in Feld für Schriftbeispiel kopieren
18:   m_strDisplayText = m_strSampText;
19:   // Dialogfeld aktualisieren
20:   UpdateData(FALSE);
20:
20:   return TRUE;  // Geben Sie TRUE zurück, außer ein
21:                 //Steuerelement soll den Fokus erhalten
22: }
```

2. Fügen Sie mit dem Modus Steuerelementereignisse im Eigenschaftenfenster eine Funktion für die Nachricht EN_CHANGE und das Eingabefeld IDC_ESAMPTEXT hinzu.
3. Schreiben Sie in die Funktion den Code aus Listing 8.7.

Listing 8.7: Die Funktion OnEnChangeEsamptext

```
 1: void CTextFontsDlg::OnEnChangeEsamptext(void)
 2: {
 3: // TODO: Falls dies ein RICHEDIT-Steuerelement ist,
 4: // wird das Kontrollelement
```

```
 5:   // diese Benachrichtigung nicht senden, es sei denn,
 6:   // Sie setzen den CDialog::OnInitDialog() außer Kraft.
 7:   // Funktion und Aufruf CRichEditCtrl().SetEventMask()
 8:   // mit dem ENM_CHANGE-Flag ORed in der Eingabe.
 9:
10:   // TODO: Fügen Sie hier Ihren Code für die
11:   // Kontrollbenachrichtigungsbehandlung ein.
12:   // Variablen mit Steuerelementen des Dialogfelds aktualisieren
13:   UpdateData(TRUE);
14:   // Aktuellen Text in Schriftbeispiel kopieren
15:   m_strDisplayText = m_strSampText;
16:   // Dialogfeld mit Variablen aktualisieren
17:   UpdateData(FALSE);
18: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie in das Eingabefeld Text eingeben können, der gleichzeitig im Anzeigebereich für die Schrift im darunter liegenden Gruppenfeld erscheint.

Anzuzeigende Schrift auswählen

Bevor Sie die Schrift für den Anzeigebereich ändern können, brauchen Sie eine CFont- Elementvariable der Dialogfeldklasse, über die Sie die anzuzeigende Schrift festlegen und ändern können. Führen Sie die folgenden Schritte aus, um diese Variable hinzuzufügen:

1. Klicken Sie in der Klassenansicht mit der rechten Maustaste auf die Klasse CTextFontsDlg und wählen Sie Hinzufügen / Variable hinzufügen.

2. Legen Sie den Typ der Variablen als CFont, den Variablennamen mit m_fSampFont und den Zugriff als private fest. Klicken Sie auf Fertig stellen, um das Dialogfeld zu schließen und die Variable hinzuzufügen.

Wenn Sie den Code hinzufügen, um die ausgewählte Schrift zu verwenden, schreiben Sie dafür eine eigene Funktion, die nicht mit einem Steuerelement verbunden ist, da Sie diese Funktion von mehreren verschiedenen Funktionen aus aufrufen werden. Um die Funktion für die Anzeige und Verwendung der ausgewählten Schrift hinzuzufügen, führen Sie die folgenden Schritte aus:

1. Klicken Sie in der Klassenansicht mit der rechten Maustaste auf die Klasse CTextFontsDlg und wählen Sie Hinzufügen / Funktion hinzufügen.

2. Legen Sie den Funktionstyp als void, den Funktionsnamen als SetMyFont und den Zugriffsstatus als private fest. Klicken Sie auf Fertig stellen, um das Dialogfeld zu schließen und die Funktion hinzuzufügen.
3. Übernehmen Sie in die Funktion den Code aus Listing 8.8.

Listing 8.8: Die Funktion SetMyFont

```
 1: void CTextFontsDlg::SetMyFont(void)
 2: {
 3:   CRect rRect;        // Rechteck des Anzeigebereichs
 4:
 5:   // Wurde eine Schrift ausgewählt?
 6:   if (m_strFontName.GetLength() > 0)
 7:   {
 8:     // Abmessungen des Feldes für Schriftbeispiel ermitteln
 9:     m_ctlDisplayText.GetWindowRect(&rRect);
10:     // Aktuelle Schrift freigeben
11:     m_fSampFont.Detach();
12:     // Zu verwendende Schrift erzeugen
13:     m_fSampFont.CreateFont((rRect.Height()- 5), 0, 0, 0,
14:                            FW_NORMAL,
15:             0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
16:             CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
```

```
17:             DEFAULT_PITCH | FF_DONTCARE, m_strFontName);
18:
19:     // Schrift für Anzeigefeld der Schriftprobe festlegen
20:     m_ctlDisplayText.SetFont(&m_fSampFont);
21:   }
22: }
```

4. Fügen Sie eine Funktion für die Nachricht LBN_SELCHANGE und für das Listenfeld IDC_LFONTS hinzu und übernehmen Sie in diese Funktion den Code aus Listing 8.9.

Listing 8.9: Die Funktion OnLbnSelchangeLfonts

```
 1: void CTextFontsDlg::OnLbnSelchangeLfonts()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen mit Steuerelementen des Dialogfelds aktualisieren
 6:   UpdateData(TRUE);
 7:
 8:   // Schrift für Beispiel festlegen
 9:   SetMyFont();
10: }
```

Die Funktion SetMyFont prüft zuerst, ob eine Schrift ausgewählt wurde. Als Nächstes wird der Bereich des statischen Textfelds abgerufen, der für die Anzeige der Schriftprobe vorgesehen ist. Damit lässt sich eine Schrifthöhe festlegen, die etwas kleiner als die Höhe des Bereichs ist, der für die Anzeige der Schrift zur Verfügung steht. Zuletzt wird die ausgewählte Schrift erzeugt und das statische Textfeld angewiesen, die neu erstellte Schrift zu verwenden.

Die Funktion OnLbnSelchangeLfonts kopiert die Steuerelementwerte in die zugeordneten Variablen und ruft dann die Funktion SetMyFont auf, um die ausgewählte Schrift zu verwenden. Wenn Sie die Anwendung kompilieren und ausführen, sollten Sie eine Schrift auswählen und das Ergebnis im Anzeigebereich verfolgen können, wie es Abbildung 8.3 zeigt.

Abbildung 8.3: Die ausgewählte Schrift anzeigen

8.3 Zusammenfassung

Heute haben Sie gelernt, wie man Schriften in Visual C++-Anwendungen verwendet. Dabei haben Sie eine Liste der im System verfügbaren Schriften erstellt und eine Schrift für ein Anzeigeobjekt erzeugt. Weiterhin hat die Lektion gezeigt, wie man Callback- Funktionen erstellt und einsetzt, um eine Liste der Ressourcen vom Betriebssystem Windows zu erhalten. Außerdem haben Sie gelernt, wie man auf Steuerelemente von der Callback-Funktion aus zugreift und dabei einen Fensterzeiger verwendet, den man an die Funktion, die die Ressourcenliste abruft, übergeben hat.

8.4 Workshop

Fragen und Antworten

Frage:
Der Funktion CreateFont muss man eine Vielzahl von Argumenten übergeben. Kann man auch auf eine alternative Funktion zurückgreifen?

Antwort:
Eine derartige Funktion gibt es, obwohl man trotzdem noch alle Informationen festlegt. Die Struktur LOGFONT enthält die gleichen Attribute, die man an die Funktion CreateFont übergibt. Man kann eine Instanz dieser Struktur deklarieren, die Attribute mit Standardwerten initialisieren und dann diese Struktur an die Funktion CreateFontIndirect übergeben. Wenn umfangreiche Schriftänderungen vorzunehmen sind, ist diese Methode zu bevorzugen, weil man dieselbe Instanz der Struktur verwenden kann und nur diejenigen Attribute modifiziert, die sich gegenüber den aktuellen Einstellungen geändert haben. Damit erstellt man dann die verschiedenartigen Schriften.

Antwort:
Um diese alternative Lösung zum Erzeugen einer Schrift zu verwenden, deklariert man eine Instanz der Struktur LOGFONT als Element der Dialogfeldklasse und initialisiert dann alle Attribute, bevor man die Funktion SetMyFont aufruft. In der Funktion SetMyFont nehmen Sie die Modifikationen gemäß Listing 8.10 vor.

Listing 8.10: Die modifizierte Funktion SetMyFont

```
 1: void CTextFontsDlg::SetMyFont()
 2: {
 3:
 4:   // Wurde eine Schrift ausgewählt?
 5:   if (m_strFontName,GetLength() > 0)
 6:   {
 7:     // Annahme, dass Schriftgröße bereits in Struktur
 8:     // m_lLogFont initialisiert wurde. Damit ist nur
 9:     // noch der Schriftname zu spezifizieren
10:     _tcscpy(m_lLogFont.lfFaceName, m_strFontName);
11:     // Aktuelle Schrift freigeben
12:     m_fSampFont.Detach();
13:     // Zu verwendende Schrift erzeugen
14:     m_fSampFont.CreateFontIndirect(&m_lLogFont);
15:
16:     // Schrift für Anzeigefeld der Schriftprobe festlegen
17:     m_ctlDisplayText.SetFont(&m_fSampFont);
18:   }
19: }
```

Frage:
Wie kann ich die Schriften in meiner Liste ausschließlich auf TrueType-Schriften begrenzen?

Antwort:
Testen Sie das an die Callback-Funktion übergebene Argument nFontType, um den Schrifttyp zu bestimmen. Wenn Sie zum Beispiel nur TrueType-Schriften in die Schriftliste aufnehmen wollen, modifizieren Sie die Callback-Funktion, um das Argument nFontType mit der Konstanten TRUETYPE_FONTTYPE zu maskieren, und prüfen dann, ob das Ergebnis gleich TRUETYPE_FONTTYPE ist, wie es das folgende Listing 8.11 zeigt:

Listing 8.11: Callback-Funktion zur Suche nach TrueType-Schriftarten

```
 1: int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
 2: LPNEWTEXTMETRIC lpntm, DWORD nFontType, LPARAM lParam)
 3: {
 4:   // Zeiger auf Dialogfenster erzeugen
 5:   CTextFontsDlg* pWnd = (CTextFontsDlg*) lParam;
 6:
 7:   // Liste auf TrueType-Schriften begrenzen
 8:   if (nFontType & TRUETYPE_FONTTYPE)
 9:   {
10:     // Schriftname in Listenfeld einfügen
11:     pWnd->m_ctlFontList.AddString(
12:                     lpelf->elfLogFont.lfFaceName);
13:   }
14:   // 1 zurückgeben, um Schriftauflistung fortzusetzen
15:   return 1;
16: }
```

Quiz

1. Wie kann man festlegen, dass der Text unterstrichen wird?

2. Wie lässt sich Text von oben nach unten ausgeben?
3. Wie oft wird die Callback-Funktion EnumFontFamProc durch das Betriebssystem aufgerufen?

Übungen

1. Fügen Sie gemäß Abbildung 8.4 ein Kontrollkästchen hinzu, um wählen zu können, ob der eingegebene Text die Schrift repräsentieren soll oder der Schriftname in der ausgewählten Schrift anzuzeigen ist.

Abbildung 8.4: Ausgewählte Schrift mit Bezeichnung anzeigen

2. Fügen Sie gemäß Abbildung 8.5 ein Kontrollkästchen hinzu, um die Textprobe kursiv anzuzeigen.

Abbildung 8.5: Die ausgewählte Schrift kursiv anzeigen

Tag 9

Bilder, Zeichnungen und Bitmaps

Vielleicht ist Ihnen schon aufgefallen, dass viele Anwendungen Zeichnungen und Bilder anzeigen. Das verleiht einer Anwendung einen gewissen Pfiff und Schliff. Bei einigen Anwendungen sind Grafiken ein integraler Bestandteil der gebotenen Funktionalität. Wenn man unter Windows erfolgreich programmieren will, sind gute Kenntnisse notwendig, wie man die grafischen Fähigkeiten in einer Anwendung umsetzt. Sie haben an Tag 4 bereits gelernt, wie man Linien zeichnet und wie sich eine Folge von Linien zu einer geschlossenen Zeichnung verbinden lässt. Heute gehen wir über diese Möglichkeiten hinaus. Sie lernen, wie sich kompliziertere Grafikfähigkeiten in eine Anwendung aufnehmen lassen.

Insbesondere beschäftigen wir uns heute damit, wie ...

- Windows über einen Gerätekontext Zeichenbefehle in grafische Ausgaben übersetzt,
- man die grafische Ausgabe über verschiedene Abbildungsmodi beeinflussen kann,
- Windows mit Stiften und Pinseln verschiedene Teile der Grafik zeichnet,
- man Bitmaps und andere Bilder dynamisch lädt und anzeigt.

9.1 Die grafische Geräteschnittstelle (GDI)

Das Betriebssystem Windows stellt eine Reihe von Abstraktionsebenen bereit, die sich auf das Erstellen und den Einsatz von Grafiken in einer Anwendung beziehen. In den Tagen der DOS-Programmierung mussten Sie sich eingehend mit der Ansteuerung der Grafikhardware beschäftigen, um Zeichenoperationen in einer Anwendung zu realisieren. Das erforderte umfangreiches Wissen und Kenntnisse der verschiedenartigsten Grafikkarten, die ein Benutzer in seinen Computer einbauen konnte, wozu noch die verschiedensten Optionen für Monitore und Bildschirmauflösungen kamen. Es gab zwar einige Grafikbibliotheken, die man für die Anwendungsprogrammierung erwerben konnte, aber alles in allem war es ziemlich mühsam, diese Fähigkeit in eine Anwendung einzubauen.

Mit Windows hat Microsoft das Ganze vereinfacht. Zuerst einmal stellt Microsoft das virtuelle Grafikgerät GDI (graphics device interface) für alle Windows-Anwendungen bereit. Dieses virtuelle Gerät ändert sich nicht mit der Hardware, sondern bleibt für die gesamte Palette der Grafikhardware, die man eventuell beim Benutzer vorfindet, gleich. Auf Grund dieser Einheitlichkeit lassen sich beliebige Grafikaufgaben in einer Anwendung realisieren, da man sich nicht um die Konvertierung der Ausgaben in Befehle, die die Hardware versteht, kümmern muss.

Gerätekontexte

> *Bevor man irgendeine Grafik erstellen kann, muss man über den Gerätekontext verfügen, in dem die Grafiken angezeigt werden. Der Gerätekontext enthält Informationen über das System, die Anwendung und das Fenster, in dem die Ausgabe der Zeichnungen und Bilder erfolgt. Das Betriebssystem entnimmt dem Gerätekontext, in welchem Kontext eine Grafik zu zeichnen ist, wie groß der sichtbare Bereich ist und wo sich der Zeichenbereich momentan auf dem Bildschirm befindet.*

Eine Grafik geben Sie immer im Kontext eines Anwendungsfensters aus. Dieses Fenster kann jederzeit als Vollbild, minimiert, teilweise überdeckt oder vollständig unsichtbar sein. Um den jeweiligen Zustand brauchen Sie sich nicht zu kümmern, weil Sie Ihre Grafiken mithilfe des Gerätekontextes in das Fenster zeichnen.

Windows verwaltet alle Gerätekontexte und ermittelt daraus, ob die Grafiken überhaupt oder welcher Teil davon tatsächlich für den Benutzer anzuzeigen ist. Praktisch stellt der Gerätekontext, in dem Sie Ihre Grafiken zeichnen, den visuellen Kontext des Fensters dar, in dem Sie die Grafiken ausgeben.

Die meisten Funktionen in Bezug auf Zeichnungen und Bilder führt der Gerätekontext mit zwei Ressourcen aus: Stift und Pinsel. Wie ihre Entsprechungen in der realen Welt führen Stifte und Pinsel zwar ähnliche, aber dennoch unterschiedliche Aufgaben aus. Der Gerätekontext verwendet Stifte, um Linien und Figuren zu zeichnen, während Pinsel die Flächen auf dem Bildschirm ausfüllen. Das entspricht völlig der Vorgehensweise, wenn man auf Papier zunächst die Umrisse eines Bildes mit einem Stift zeichnet und dann einen Pinsel in die Hand nimmt, um die Flächen zwischen den Linien mit Farben auszufüllen.

Die Gerätekontextklasse

In Visual C++ stellt die MFC-Gerätekontextklasse (CDC für Context Device Class) zahlreiche Zeichenfunktionen für Kreise, Quadrate, Linien, Kurven usw. bereit. Diese Funktionen gehören zur Gerätekontextklasse, da sie alle auf Informationen des Gerätekontextes zurückgreifen, um in den Anwendungsfenstern zu zeichnen.

Die Instanz einer Gerätekontextklasse erstellt man mit einem Zeiger auf die Fensterklasse, die man mit dem Gerätekontext verbinden möchte. Damit lässt sich in den Konstruktoren und Destruktoren der Gerätekontextklasse der gesamte Code unterbringen, der sich mit der Zuweisung und Freigabe eines Gerätekontextes befasst.

Gerätekontext-Objekte gehören genau wie alle Arten von Zeichen-Objekten zu den Ressourcen im Betriebssystem Windows. Das Betriebssystem verfügt nur über eine begrenzte Zahl dieser Ressourcen. Obwohl die Gesamtzahl der Ressourcen in den neueren Versionen von Windows recht groß ist, kann dennoch ein Mangel an Ressourcen entstehen, wenn eine Anwendung diese zwar reserviert, aber nicht wieder korrekt freigibt. Diesen Verlust bezeichnet man als Ressourcenleck (ressource leak), der analog zu einem Speicherleck das System des Benutzer blockieren oder lahm legen kann. Es empfiehlt sich also, die Ressourcen in Funktionen zu erzeugen, wo sie zum Einsatz kommen, und sie sobald wie möglich wieder freizugeben, wenn die Arbeit mit den betreffenden Ressourcen abgeschlossen ist.

Dementsprechend nutzt man Gerätekontexte und deren Zeichenressourcen fast ausschließlich als lokale Variablen innerhalb einer einzigen Funktion. Die einzige echte Ausnahme liegt vor, wenn das Gerätekontext-Objekt von Windows erzeugt und in eine ereignisverarbeitende Funktion als Argument übergeben wird.

MFC-Exkurs: Wichtige Gerätekontext-Zeichenfunktionen

Viele Zeichenfunktionen sind Member-Funktionen der Klasse CDC. Zusätzlich ist CDC die Basisklasse für alle anderen Gerätekontextklassen, sodass diese Zeichenfunktionen unabhängig vom verwendeten Gerätekontext verfügbar sind. Die auf den nächsten Seiten aufgelisteten Funktionen stellen nur einen kleinen Ausschnitt der verfügbaren Funktionalität dar, sind aber die für die heutige Besprechung am Besten passenden.

Hintergrund- und Vordergrundfarben

Es existieren mehrere Funktionen, mit denen man kontrollieren kann, wie Text und Abbildungen angezeigt werden. Ein wichtiges Element ist hier die Angabe von Vordergrund- und Hintergrundfarben, wenn Sie Text oder andere Formen in ein Fenster setzen. In der Klasse CDC gibt es eine Gruppe von Funktionen, die diese Aspekte kontrollieren. Die folgenden Funktionen sind nur ein paar der häufiger genutzten.

- GetBkColor und SetBkColor

  Wenn Sie Text in ein Fenster zeichnen, müssen Sie sich nicht nur über die Farbe des Textes sondern auch über die Hintergrundfarbe hinter dem Text Gedanken machen. Dafür sind die Funktionen GetBkColor und SetBkColor da. Beide Funktionen geben ein COLORREF zurück (RGB, wie in Rot, Grün, Blau; das wird ein paar Seiten weiter hinten bei der Besprechung des Stifts erklärt), das die aktuelle oder vorherige Hintergrundfarbe enthält. Die Funktion SetBkColor übernimmt ein COLORREF als einzigen Parameter. Tritt ein Fehler auf, liefert SetBkColor den Wert 0x80000000 als vorherige Farbe zurück.

- GetBkMode und SetBkMode

  Sie müssen nicht nur die Hintergrundfarbe setzen, sondern auch entscheiden, ob Sie diese Farbe verwenden möchten oder ob sichtbar sein soll, was auch immer sich hinter dem Text befindet. Das wird mit den Funktionen GetBkMode und SetBkMode kontrolliert. Es gibt in Windows nur zwei Modi für das Zeichnen des Hintergrunds: OPAQUE (deckend) und TRANSPARENT. Der TRANSPARENT-Modus zeigt nicht die Hintergrundfarbe an, sondern lässt, was immer sich hinter dem Text befindet, durchscheinen. Der Hintergrundmodus OPAQUE füllt den Hintergrund mit der Hintergrundfarbe aus und verbirgt alles, was sich hinter dem Text befindet. Diese beiden Werte sind die einzigen Rückgabewerte beider Funktionen, als aktueller bzw. vorheriger Hintergrundmodus. Diese beiden Werte sind auch die einzigen, die SetBkMode akzeptiert.

- GetTextColor und SetTextColor

  Wenn Sie Text in einem Fenster zeichnen wollen, müssen Sie möglicherweise die für den Text zu verwendende Farbe angeben. Dazu verwenden Sie die Funktionen GetTextColor und SetTextColor. Diese beiden Funktionen verhalten sich genauso wie ihre beiden Hintergrund-Verwandten und geben einen COLORREF-Wert zurück, der die aktuelle bzw. vorherige Textfarbe angibt. Und genau wie sein Hintergrund-Verwandter übernimmt auch SetTextColor einen COLORREF-Wert als einziges Argument.

Funktionen zum Zeichnen von Linien

Der nächste Aspekt des Zeichnens, für den CDC Funktionalität bietet, sind die verschiedenen Zeichenfunktionen. Es handelt sich bei der folgenden Liste zwar keinesfalls um alle verfügbaren Funktionen, doch Sie sollten einen allgemeinen Eindruck davon bekommen, welche Art von Funktionalität verfügbar ist.

- GetCurrentPosition

  Viele Funktionen zum Zeichnen von Linien nehmen die aktuelle Position auf dem Bildschirm als Startpunkt an. Dieser Punkt ist möglicherweise nicht da, wo Sie ihn vermuten, also müssen Sie seine Position von Zeit zu Zeit überprüfen. Sie können den Wert mit der Funktion GetCurrentPosition abfragen. Der Rückgabewert dieser Funktion ist eine Instanz der Klasse CPoint, die aus den X- und Y-Koordinaten eines bestimmten Punkts besteht.

- MoveTo und LineTo

  Wenn Sie sich innerhalb des Zeichenbereichs bewegen wollen, können Sie die Funktionen MoveTo und LineTo verwenden, die Sie schon an Tag 4 gesehen haben. Beide Funktionen können, abhängig davon, wie Sie mit ihnen arbeiten möchten, entweder die X- und Y-Koordinaten oder ein CPoint-Objekt übernehmen.

- Arc und ArcTo

Wenn Sie eine Kurve zeichnen wollen, werden die Funktionen Arc und ArcTo besser Ihren Bedürfnissen entsprechen. Beide Funktionen zeichnen eine gebogene Linie zwischen zwei Punkten. Die Wölbung des Bogens wird durch ein begrenzendes Rechteck definiert. Der Startpunkt des Bogens ergibt sich aus dem Schnittpunkt zwischen dem Bogen und einem Strahl vom Mittelpunkt des Rechtecks zum angegebenen Startpunktparameter. Der angegebene Startpunktparameter muss nicht auf dem Bogen liegen, für den Endpunkt des Bogens gilt entsprechendes. Der Bogen wird gegen den Uhrzeigersinn vom Startpunkt zum Endpunkt gezeichnet. Sind Startpunkt und Endpunkt identisch, so wird ein kompletter Bogen gezeichnet. Die Argumente für die Funktionen sind diese:

```
Arc(int x1, int y1, int x2, int y2, // umschließendes Rechteck
      int x3, int y3,               // Startpunkt
      int x4, int y4);              // Endpunkt
ArcTo(int x1, int y1, int x2, int y3,
      int x3, int y3, int x4, int y4);
```

oder

```
Arc(LPCRECT lpRect, POINT ptStart, POINT ptEnd);
ArcTo(LPCRECT lpRect, POINT ptStart, POINT ptEnd);
```

In der ersten Syntax stellen die ersten vier Parameter den linken oberen und den rechten unteren Punkt des begrenzenden Rechtecks dar, das Sie in der zweiten Syntax als CRect-Klasse übergeben können. Die nächsten beiden Parameter in der ersten Syntax definieren den Startpunkt des Bogens, in der zweiten Syntax als CPoint oder als Struktur POINT übergeben. Die letzten beiden Parameter in der ersten Syntax und der letzte in der zweiten schließlich geben den Endpunkt des Bogens an. Der Unterschied zwischen den beiden Funktionen Arc und ArcTo ist, dass ArcTo die aktuelle Position nach dem Zeichnen des Bogens auf den Endpunkt setzt.

- Polyline und PolylineTo

  Die Funktionen Polyline und PolylineTo übernehmen eine Reihe von Punkten und zeichnen eine Reihe von Linien zwischen ihnen. Beide Funktionen übernehmen die gleichen Parameter:

```
Polyline(LPPOINT lpPoints, int iCount);
PolylineTo(LPPOINT lpPoints, int iCount);
```

  Der erste Parameter ist ein Zeiger auf ein Array von CPoint-Objekten oder POINT-Strukturen. Der zweite Parameter ist eine ganze Zahl, welche die Anzahl der Punkte im Array angibt; es müssen mindestens zwei Punkte angegeben werden. Die Funktion PolylineTo setzt die aktuelle Position auf den letzten Punkt, wenn sie zurückkehrt.

Funktionen zum Zeichnen von Formen

Linien zeichnen ist schön und gut, aber manchmal müssen Sie in der Lage sein, eine komplette Form zu zeichnen. In CDC sind eine Reihe von Funktionen zum Zeichnen von Formen enthalten. Im Folgenden sind einige der Formfunktionen beschrieben, die Sie beim Zeichnen von Grafiken verwenden können.

- Ellipse

  Die Funktion Ellipse zeichnet eine Ellipse, die durch ein begrenzendes Rechteck definiert wird. Wenn es sich beim umschließenden Rechteck um ein Quadrat handelt, zeichnet die Funktion einen Kreis. Ansonsten wird die Ellipse nach der Form des umschließenden Rechtecks gestreckt und gedreht. Die Funktion Ellipse übernimmt entweder ein CRect-Objekt oder vier Integer als Parameter, um das begrenzende Rechteck zu definieren. Die Ellipse wird mit dem aktuellen Stift und Pinsel gezeichnet (über die Sie in Kürze alles lernen werden).

```
Ellipse( int x1, int y1, int x2, int y2 );
Ellipse( LPCRECT lpRect );
```

- Pie

  Die Funktion Pie zeichnet einen Keil von der Form eines Kuchenstücks. Die Funktion übernimmt die gleichen Parameter wie die Funktionen Arc und ArcTo. Die Form des Bogens am äußeren Ende des Keils entspricht der, die bei Übergabe der gleichen Parameter an Arc oder ArcTo entstanden wäre. Die Spitze des Keils ist die Mitte des umschließenden Rechtecks.

- Polygon

  Die Funktion Polygon ähnelt den Funktionen Polyline und PolylineTo. Sie alle übernehmen die gleichen Parameter und zeichnen Linien zwischen den Punkten in einem Array von Punkten. Der Unterschied bei der Funktion Polygon ist, dass sie die Form vollendet, indem sie eine abschließende Linie zwischen dem letzten und dem ersten Punkt zieht.

- Rectangle und RoundRect

  Die Funktionen Rectangle und RoundRect zeichnen eine Rechteckform. RoundRect zeichnet das Rechteck mit abgerundeten Ecken. Diese beiden Funktionen übernehmen ein CRect-Objekt als Parameter, um das zu zeichnende Rechteck zu definieren. Die Funktion RoundRect übernimmt einen zweiten Parameter, ein CPoint- Objekt. Die X-Koordinate dieses Punkts gibt die Breite der Ellipse an, die zum Zeichnen der abgerundeten Ecken verwendet wird, die Y-Koordinate gibt die Höhe dieser Ellipse an.

```
Rectangle( int x1, int y1, int x2, int y2 );
Rectangle( LPCRECT lpRect );
```

und

```
Rectangle( int x1, int y1, int x2, int y2, int x3, int y3 );
Rectangle( LPCRECT lpRect, POINT point );
```

Bitmap-Funktionen

In CDC gibt es auch eine Reihe von Funktionen, die zum Zeichnen und Anzeigen von Bitmap-Bildern verwendet werden. Die meisten dieser Funktionen dienen dem Zweck, eine Bitmap von einem Anzeigekontext in den Speicher und von da auf einen anderen Bildschirm zu transferieren. Im Folgenden sind ein paar dieser Funktionen beschrieben.

- BitBlt

  Die Funktion BitBlt kopiert ein Bild von einem Gerätekontext zum anderen. Die Funktion hat folgende Syntax:

```
BOOL BitBlt(int xDst, int yDst, int nWidth, int nHeight,
            CDC* pSrcDC,
            int xSrc, int ySrc, DWORD dwRop)
```

  Die ersten beiden Parameter der Funktion BitBlt (xDst und yDst) sind die X- und Y-Koordinaten für die obere linke Ecke des Bilds auf dem Zielgerätekontext. Die nächsten beiden Parameter, nWidth und nHeight, geben die Breite und Höhe des Bilds auf dem Zielgerätekontext an. Der fünfte Parameter, pSrcDC, ist ein Zeiger auf den kompatiblen Gerätekontext, von dem das Bild kopiert werden soll. Die nächsten beiden Parameter, xSrc und ySrc, geben die obere linke Ecke des Bilds im Gerätekontext an, von dem das Bild kopiert wird. Der letzte Parameter, dwRop, ist ein Flag, das anzeigt, wie der Kopiervorgang ablaufen soll. In Tabelle 9.1 sind die möglichen Werte für dieses Flag aufgelistet.

Option	Beschreibung
BLACKNESS	Die gesamte Ausgabe wird schwarz.
CAPTUREBLT	Fügt alle über dem Fenster liegenden weiteren Fenster in das resultierende Bild ein (standardmäßig wird nur das Bildfenster eingefügt.)
DSTINVERT	Invertiert die Farben des angezeigten Bilds
MERGECOPY	Bewirkt mithilfe des Booleschen Operators AND eine Verschmelzung des Quellbilds mit dem Zielmuster

Tabelle 9.1: Optionsflags für BitBlt

Option	Beschreibung
MERGEPAINT	Bewirkt mithilfe des Booleschen Operators OR die Kombination des invertierten Quellbilds mit dem Zielbild
NOMIRRORBITMAP	Erlaubt keine Spiegelung der Bitmap
NOTSRCCOPY	Kopiert das invertierte Quellbild in die Zielausgabe
NOTSRCERASE	Invertiert das verschmolzene Bild. Funktioniert wie eine Kombination aus MERGEPAINT und DSTINVERT
PATCOPY	Kopiert nur das Muster ins Zielbild
PATINVERT	Kombiniert das Muster und das Quellbild mithilfe des XOR-Operators
PATPAINT	Kombiniert mithilfe des OR-Operators das invertierte Quellbild mit dem Muster. Das resultierende Bild wird mithilfe des OR-Operators mit dem Zielbild kombiniert.
SRCAND	Kombiniert Quell- und Zielbild mithilfe des AND-Operators
SRCCOPY	Kopiert das Quellbild ins Zielbild
SRCERASE	Invertiert das Zielbild und kombiniert es dann mithilfe des AND-Operators mit dem Quellbild
SRCINVERT	Kombiniert Quell- und Zielbild mithilfe des XOR-Operators
SRCPAINT	Kombiniert Quell- und Zielbild mithilfe des OR-Operators
WHITENESS	Die gesamte Ausgabe wird weiß.

Tabelle 9.1: Optionsflags für BitBlt (Forts.)

- StretchBlt

  Die Funktion StretchBlt ändert die Größe eines Bilds und kopiert es dann von einem Gerätekontext zum anderen. Die Funktion hat folgende Syntax:

```
BOOL StretchBlt(int xDst, int yDst, int nWidth, int nHeight, CDC* pSrcDC,
                int xSrc, int ySrc, int nSrcWidth, int nSrcHeight,DWORD dwRo
```

  Die ersten beiden Parameter der Funktion StretchBlt, xDst und yDst, sind die X- und Y-Koordinaten der oberen linken Ecke des Bilds auf dem Zielgerätekontext. Die nächsten beiden Parameter, nWidth und nHeight, geben die Breite und Höhe des Bilds auf dem Zielgerätekontext an. Der fünfte Parameter, pSrcDC, ist ein Zeiger auf den kompatiblen Gerätekontext, von dem das Bild kopiert werden soll. Die nächsten vier Parameter - xSrc, ySrc, nSrcWidth und nSrcHeight - geben die X- und Y-Koordinaten der oberen linken Ecke und die Breite und Höhe des Bilds im Gerätekontext an, von dem es kopiert wird. Der letzte Parameter, dwRop, gibt an, wie der Kopiervorgang ablaufen soll. In Tabelle 9.1 sind die möglichen Werte für dieses Flag aufgelistet.

Textfunktionen

Schließlich gibt es noch einige Funktionen, um Text auf die Zeichenfläche zu setzen. Diese Funktionen verwenden die ausgewählte Schriftart und Farbe. Hier besprechen wir nur zwei dieser Funktionen.

- TextOut

  Die Funktion TextOut platziert Text auf der Zeichenfläche und übernimmt drei Parameter:

  - die X-Koordinate für die Position der oberen linken Ecke des Texts,
  - die Y-Koordinate für die obere linke Ecke,
  - ein CString, der den anzuzeigenden Text enthält beziehungsweise einen String nach C-Konvention und eine Längenangabe.

  ```
  TextOut( int x, int y, LPCTSTR lpszString, int nCount );
  TextOut( int x, int y, const CString& str );
  ```

- GetTextExtent

  Die Funktion GetTextExtent berechnet die Größe des Bereichs, den ein Textstring in der aktuellen Schriftart auf der Zeichenfläche einnimmt. Diese Funktion übernimmt einen CString, der den zu berechnenden Text enthält und gibt ein CSize- Objekt zurück. Die Klasse CSize verkapselt die Struktur SIZE, die zwei Elemente enthält: cx, das die Breite des Texts angibt, und cy, das seine Höhe angibt.

```
CSize GetTextExtent( LPCTSTR lpszString, int nCount );
CSize GetTextExtent( const CString& str );
```

- SelectObject

  Die Funktion SelectObject wird verwendet, um Stifte, Pinsel, Bitmaps, Schriftarten, Regionen oder GDI (Graphical Device Interface)- Objekte auszuwählen. Sie übernimmt als einzigen Parameter einen vom Gerätekontext zu verwendenden Zeiger auf das auszuwählende Objekt und gibt einen Zeiger auf das zuvor ausgewählte Objekt des gleichen Typs zurück. Mit anderen Worten, wenn Sie SelectObject aufrufen, um einen neuen Stift zum Zeichnen auszuwählen, gibt die Funktion einen Zeiger auf den vorher ausgewählten Stift zurück. Wenn Sie einen neuen Pinsel auswählen, gibt sie einen Zeiger auf den vorher ausgewählten Pinsel zurück.

  Wenn Sie ein Objekt ausgewählt haben, wird jedes Mal, wenn Sie eine Funktion im Gerätekontext aufrufen, die das gewählte Objekt in der angeforderten Zeichenfunktion verwendet, das neue Objekt benutzt. Es wird als gute Programmierpraxis angesehen, das zuvor ausgewählte Objekt zu speichern, wenn Sie SelectObject aufrufen, um ein neues Objekt auszuwählen. Wenn Sie die Verwendung des neu ausgewählten Objekts beendet haben, rufen Sie SelectObject wieder auf und übergeben den Zeiger auf das vorherige Objekt, um den Gerätekontext in den Zustand vor Ihren Änderungen zurückzuversetzen.

MFC-Exkurs: Die Klasse CPen

Die Stiftklasse CPen haben Sie bereits eingesetzt, um Farbe und Breite für die auf dem Bildschirm zu zeichnenden Zeichen festzulegen. CPen ist das hauptsächliche Ressourcenwerkzeug, mit dem man Linien aller Art auf dem Bildschirm darstellt. Erzeugt man eine Instanz der Klasse CPen, kann man den Typ, die Farbe und die Dicke der Linie spezifizieren. Nachdem der Stift erzeugt ist, lässt er sich als aktuelles Zeichenwerkzeug für den Gerätekontext auswählen und damit für alle Zeichenbefehle an diesen Gerätekontext verwenden. Der folgende Code erzeugt einen neuen Stift und wählt ihn dann als aktuellen Zeichenstift aus:

```
// Den Gerätekontext erzeugen
CDC dc(this);
```

```
// Den Stift erzeugen
CPen pnPen(PS_SOLID, 1, RGB(0, 0, 0)));
// Den Stift als aktuellen Zeichenstift auswählen
// Den alten Stift speichern, sodass er später zurückgesetzt
// werden kann
CPen* pOldPen = dc.SelectObject(&pnPen);
```

Es stehen mehrere Stiftstile zur Verfügung, die verschiedene Muster der gezeichneten Linien ergeben. Abbildung 9.1 zeigt die grundlegenden Stile, auf die man mit jeder Farbe in einer Zeichenanwendung zurückgreifen kann.

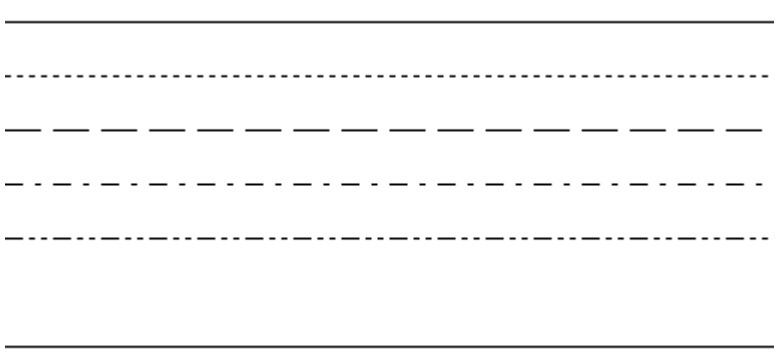


Abbildung 9.1: Stiftstile in Windows

Alle diese Linienstile ergeben bei einer Stiftdicke größer als 1 durchgehende Linien. Wenn Sie einen anderen Stil als PS_SOLID verwenden wollen, müssen Sie eine Stiftbreite von 1 angeben.

Wenn Sie einen neuen Stift in den Gerätekontext wählen, sollten Sie immer den vorherigen Stift speichern, damit Sie ihn wieder auswählen können, wenn Sie den neuen Stift nicht mehr verwenden.

Zusammen mit dem Linienstil, den der Stift zeichnen soll, kann man auch die Breite und Farbe des Stifts festlegen. Die Kombination dieser drei Variablen legt die Erscheinung der resultierenden Linien fest. Die Linienbreite kann Werte ab einschließlich 1 erhalten, obwohl sich bei einer Breite über 32 kaum noch eine vernünftige Linie zeichnen lässt.

Die Farbe ist als RGB-Wert anzugeben, der sich aus drei separaten Werten für die Helligkeit der roten, grünen und blauen Komponenten der Pixel auf dem Computerbildschirm zusammensetzt. Die einzelnen Komponenten können Werte im Bereich zwischen 0 und 255 annehmen. Die Funktion RGB fasst die Einzelwerte in einem Format zusammen, das Windows für die Ausgabe benötigt. Einige der gebräuchlicheren Farben sind in Tabelle 9.2 aufgeführt.

Farbe	Rot	Grün	Blau
Schwarz	0	0	0

Blau	0	0	255
Dunkelblau	0	0	128
Grün	0	255	0
Dunkelgrün	0	128	0
Cyan	0	255	255
Dunkelcyan	0	128	128
Rot	255	0	0
Dunkelrot	128	0	0
Magenta	255	0	255
Dunkelmagenta	128	0	128
Gelb	255	255	0

Tabelle 9.2: Gebräuchliche Windows-Farben

Farbe	Rot	Grün	Blau
Dunkelgelb	128	128	0
Dunkelgrau	128	128	128
Hellgrau	192	192	192
Weiß	255	255	255

Tabelle 9.2: Gebräuchliche Windows-Farben (Forts.)

MFC-Exkurs: Die Klasse CBrush

Mit der Pinselklasse CBrush lassen sich Pinsel erzeugen, die definieren, wie Bereiche ausgefüllt werden. Bei einer geschlossenen Figur zeichnet man den Umriss mit dem aktuellen Stift und füllt den Innenbereich mit dem aktuellen Pinsel aus. Pinsel können durchgängige Farben (mit den gleichen RGB-Werten wie die Stifte festzulegen), Linienmuster oder sogar sich wiederholende Muster, die sich aus einer kleinen Bitmap aufbauen, liefern. Wollen Sie einen Pinsel mit durchgehender Farbe erzeugen, geben Sie dazu die entsprechende Farbe an:

```
CBrush brSolidBrush(RGB(255, 0, 0));
```

Um einen gemusterten Pinsel zu erzeugen, müssen Sie nicht nur die Farbe festlegen, sondern auch das Muster:

```
CBrush brPatternBrush(HS_BSDIAGONAL, RGB(0, 0, 255));
```

Nachdem Sie einen Pinsel erzeugt haben, können Sie ihn mit dem Gerätekontext-Objekt auswählen, genau wie es bei Stiften geschieht. Nach der Auswahl eines Pinsels dient er als aktueller Pinsel, wenn man irgendein Objekt zeichnet, das einen Pinsel verwendet.

Wie bei Stiften kann man Pinsel mit einer Reihe von Standardmustern erzeugen, die Abbildung 9.2 zeigt. Neben diesen Mustern gibt es einen zusätzlichen Pinselstil, HS_BITMAP, der eine Bitmap als Muster zum Füllen des angegebenen Bereichs verwendet. Die Bitmap muss mindestens acht mal acht Pixel groß sein. Man kann auch einen Bitmap- Pinsel erzeugen, indem man eine Bitmap-Ressource für die Anwendung anlegt und ihr eine Objekt-ID zuweist. Danach können Sie einen Pinsel mithilfe des folgenden Codes erzeugen:

```
// Das Bild laden
```

```
bmpBitmap.LoadBitmap(IDB_MYBITMAP);
// Den Pinsel erzeugen
CBrush brBitmapBrush(&bmpBitmap);
```

Wenn Sie Ihre eigenen benutzerdefinierten Muster für einen Pinsel erzeugen wollen, können Sie das Muster als Bitmap der Größe acht mal acht Pixel erstellen und den Bitmap-Pinsel verwenden. Damit lässt sich die Zahl der Pinselmuster weit über die begrenzte Anzahl der Standardmuster erweitern.

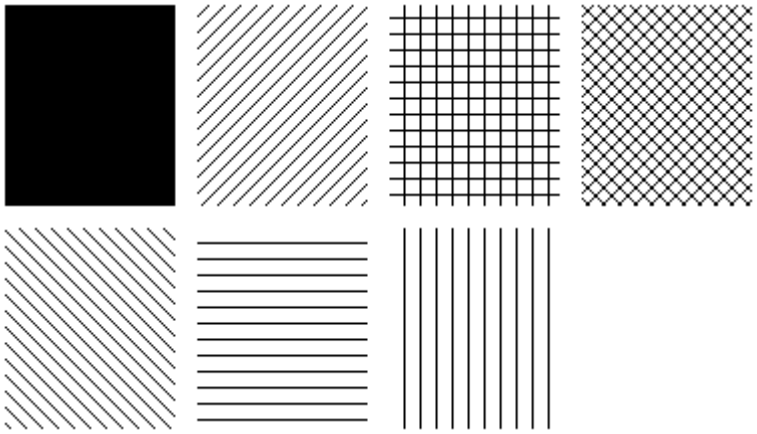


Abbildung 9.2: Standardpinselmuster

MFC-Exkurs: Die Klasse CBitmap

Es gibt verschiedene Möglichkeiten, um Bilder in einer Anwendung anzuzeigen. Feststehende Bitmaps nimmt man als Ressourcen mit zugewiesenen Objekt-IDs in die Anwendung auf und zeigt sie mit statischen Bildsteuerelementen oder ActiveX- Steuerelementen an. Man kann auch die Bitmap-Klasse CBitmap einsetzen, um die Anzeige der Bilder weitgehend beeinflussen zu können. Mithilfe der Bitmap-Klasse lassen sich Bitmaps dynamisch aus vorhandenen Dateien laden, die Bilder bei Bedarf in der Größe ändern und sie an den zugewiesenen Platz anpassen.

Wenn man die Bitmap als Ressource hinzufügt, erzeugt man eine Instanz der Klasse CBitmap und verwendet die Ressourcen-ID der Bitmap für das zu ladende Bild. Über die API-Funktion LoadImage lässt sich eine Bitmap aus einer Datei laden. Nachdem Sie die Bitmap geladen haben, können Sie das Bild über dessen Kenn-Nummer (Handle) mit der Klasse CBitmap verbinden, wie es der folgende Beispielcode zeigt:

```
// Die Bitmap-Datei laden
HBITMAP hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
                    m_strFileName, IMAGE_BITMAP, 0, 0,
                    LR_LOADFROMFILE | LR_CREATEDIBSECTION);
// Das geladene Bild dem CBitmap-Objekt zuweisen
bmpBitmap.Attach(hBitmap);
```

Nachdem Sie die Bitmap in das CBitmap-Objekt geladen haben, können Sie einen zweiten Gerätekontext erzeugen und die Bitmap in diesen Gerätekontext selektieren. Wenn Sie den zweiten Gerätekontext erzeugt haben, müssen Sie ihn mit dem ersten Gerätekontext kompatibel machen, bevor die Bitmap in ihn selektiert wird. Da Gerätekontexte durch das Betriebssystem für ein bestimmtes Ausgabegerät (Bildschirm, Drucker usw.) erzeugt werden, müssen Sie sicherstellen, dass der zweite Gerätekontext ebenfalls mit demselben Ausgabegerät wie der erste verbunden ist.

```
// Einen Gerätekontext erzeugen
```

```
CDC dcMem;
// Neuen Gerätekontext zum originalen DC kompatibel machen
dcMem.CreateCompatibleDC(dc);
// Bitmap in den neuen DC selektieren
dcMem.SelectObject(&bmpBitmap);
```

Wenn Sie die Bitmap in einen kompatiblen Gerätekontext selektieren, können Sie sie in den normalen Anzeigegerätekontext mithilfe der Funktion BitBlt kopieren:

```
// Das Bitmap in den Anzeige-DC kopieren
dc->BitBlt(10, 10, bm.bmWidth,
                  bm.bmHeight, &dcMem, 0, 0,
                  SRCCOPY);
```

Sie können das Bild auch mittels der Funktion StretchBlt kopieren und in der Größe ändern:

```
// Bitmap in den Anzeige-DC kopieren und dabei
// in der Größe ändern
dc->StretchBlt(10, 10, (lRect.Width() - 20),
                 (lRect.Height() - 20), &dcMem, 0, 0,
                  bm.bmWidth, bm.bmHeight, SRCCOPY);
```

Mit der Funktion StretchBlt lässt sich das Bild an jeden beliebigen Bereich auf dem Bildschirm anpassen.

MFC-Exkurs: Die Klasse CImage

Bitmaps waren früher das wichtigste Format für in Programme integrierte Bilder, doch das ist nicht mehr der Fall. Zum Teil dank der Beliebtheit des Internets finden Sie heutzutage öfter Web-Bildformate als Bitmaps. Wahrscheinlich sehen Sie mehr Bilder im JPEG- oder PNG-Format als Bitmaps. Verstehen Sie mich nicht falsch, Bitmaps haben noch immer ihren Platz in der Software, doch Sie werden feststellen, dass Sie oft auch andere Bildformate laden und anzeigen können müssen.

In Visual C++ können sie zwei Klassenbibliotheken verwenden, um etwas zu erstellen. MFC ist eine dieser Klassenbibliotheken, mit der Sie Anwendungen erstellen. Die andere Klassenbibliothek ist die Active Template Library (ATL), mit der in erster Linie Komponenten und Dienste erstellt werden, die ohne Benutzerschnittstelle laufen. In den letzten Tagen dieses Buchs werden Sie en wenig mit der ATL-Bibliothek arbeiten. Wenn Sie etwas erstellen wollten, mussten Sie sich immer zwischen der MFC-Bibliothek (für die Erstellung von Anwendungen) und der ATL-Bibliothek (zur Erstellung von Komponenten oder Anwendungen ohne Benutzerschnittstelle) entscheiden. Es hieß immer entweder - oder, ohne Kompromiss. Seit Visual C++.NET kann eine Reihe von gemeinsamen Klassen sowohl in MFC als auch in ATL verwendet werden. Eine dieser gemeinsamen Klassen ist CImage.

Die Klasse CImage verkapselt Funktionalität zum Schreiben und Lesen von Bildern der Formate JPEG (Joint Photographic Experts Group), GIF (Graphics Interchange Format), BMP (Bitmap) und PNG (Portable Network Graphics). Ein wichtiger Unterschied zwischen CImage und CBitmap (neben dem offensichtlichen mit den Bildformaten) ist, dass CImage ein eigenes Gerätekontext-Handle enthält und nicht mit der Methode SelectObject in einen anderen Gerätekontext gewählt werden kann. Stattdessen arbeiten Sie direkt mit dem CImage-Objekt und verwenden seine Kopierfunktionen, um ein Bild in einen Ausgabe-Gerätekontext zu kopieren. Dabei fällt der Prozess des Auswählens in einen kompatiblen Gerätekontext weg, während die für die Anzeige des Bilds erforderlichen Kopierfunktionen erhalten bleiben.

Wichtige CImage-Methoden

Die Klasse CImage stellt ein vollständiges Sortiment an Funktionalität für die Arbeit mit diesen verschiedenen Bildformaten bereit. Die hier aufgeführte Liste von Funktionen ist alles andere als vollständig, wird Ihnen jedoch genügend Vertrautheit mit der Klassenfunktionalität vermitteln, damit Sie sie in Ihren Anwendungen verwenden können. Bevor Sie versuchen, sie in einer Anwendung einzusetzen, müssen Sie ihr allerdings die entsprechende #include-Anweisung hinzufügen:

```
#include <atlimage.h>
```

Da sich CImage nicht in der Standard-MFC-Bibliothek befindet, wird es nicht automatisch in Ihre Anwendung eingeschlossen. Das heißt Sie müssen es selbst hinzufügen. Das können Sie jedoch nicht an einer beliebigen Stelle tun, da es von den MFC- Klassendeklarationen abhängt, die es verwendet. Die Header-Datei atlimage.h muss nach der Header-Datei eingeschlossen werden, in der sich die Deklaration der Klasse CString befindet. Die einfachste Methode ist, diese #include-Anweisung am Ende der Header- Datei stdafx.h einzufügen, die als Teil Ihrer Anwendung erzeugt wird. Die Header-Datei stdafx.h wird vom MFC-Application-Assistent erstellt und besteht in erster Linie aus für MFC benötigten #include-Anweisungen und Konstantendefinitionen. Da die Klasse CImage ein Teil der Klassenbibliothek ist, macht es am meisten Sinn, die Anweisung hier zu platzieren.

- Die Funktionen Attach und Detach

  Die Funktionen Attach (Anhängen) und Detach (Abhängen) tun ziemlich genau das, was man erwarten würde. Attach hängt ein geladenes Bild an eine Instanz der Klasse CImage an, ähnlich wie Sie eine Bitmap mit der Attach-Funktion an eine CBitmap- Klasse anhängen würden. Attach übernimmt zwei Parameter, wobei nur der erste Parameter erforderlich ist. Der erste Parameter ist ein Handle zu einer Bitmap, die in den Speicher geladen wurde, wie bei der Attach-Funktion von CBitmap. Der zweite Parameter ist ein Flag, das die Ausrichtung der DIB (Device Independent Bitmap - geräteunabhängige Bitmap) im Bild angibt. Dieses Flag kann folgende Werte annehmen:

Flag	**Beschreibung**
DIBOR_DEFAULT	Das Betriebssystem bestimmt die Ausrichtung der Bitmap.
DIBOR_BOTTOMUP	Die Zeilen der Bitmap beginnen am unteren Ende des Bilds und setzen sich nach oben fort.
DIBOR_TOPDOWN	Die Zeilen der Bitmap beginnen am oberen Ende des Bilds und setzen sich nach unten fort.

  Die Funktion Detach hängt das aktuelle Bild von einer Instanz der Klasse CImage ab. Diese Funktion übernimmt keine Parameter und gibt ein Handle auf das Bild zurück, wenn eines geladen war.

- Die Funktionen Create und Destroy

  Um ein eigenes Bild zu erzeugen, verwenden Sie die Funktion Create. Sie besitzt folgende Syntax:

```
BOOL Create(int nWidth, int nHeight, int nBPP, DWORD dwFlags)
```

  Der erste Parameter, nWidth, gibt die Breite (in Pixeln) des zu erzeugenden Bilds an. Der zweite Parameter, nHeight, gibt die Höhe des Bilds an. Der dritte Parameter, nBPP, gibt die Anzahl von Bits pro Pixel im Bild an (Bits Per Pixel - BPP) . Die Anzahl von Bits pro Pixel legt die Anzahl verschiedener Farben fest, die in einem Bild verwendet werden können; je mehr Bits pro Pixel, desto realistischer können die Farben sein. Die normalen Werte für diesen Parameter sind 1 (für ein einfarbiges Bild), 4 (16 Farben), 8 (256 Farben), 16 (65536 Farben), 24 (über 16.7 Mio. Farben) oder 32 (mehr als 4.2 Mrd. Farben), die beiden letzten Stufen werden auch true color oder Echtfarbenbilder genannt.

  Der vierte Parameter, dwFlags, ist ein Flagwert, der angibt, ob das Bild einen so genannten Alpha-Kanal besitzt. Derzeit kann dieses Flag nur die Werte 0 oder createAlphaChannel annehmen. Alpha-Kanal bedeutet, dass das vierte Byte in einem 32-BPP-Bild verwendet wird, um einen Transparenzfaktor festzulegen. So kann der Hintergrund durch das Bild hindurch gesehen werden, wenn der Transparenzfaktor hoch genug ist. Dieses Flag kann nur bei 32-BPP-Bildern verwendet werden. Dieses Verhalten wird nur bei Windows 98, Windows 2000 und Windows XP unterstützt.

  Die Funktion Destroy übernimmt keine Parameter und gibt kein Ergebnis zurück. Sie zerstört das aktuelle Bild in der CImage-Klasse und gibt den vom Bild verwendeten Speicher und die Ressourcen frei.

- Die Funktionen GetHeight, GetWidth und GetBPP

  Wenn Sie die Größe des Bilds wissen möchten, verwenden Sie die Funktionen GetHeight und GetWidth. Keine dieser Funktionen übernimmt Parameter, beide geben einen Integer-Wert zurück. Eine weitere ähnliche Funktion ist GetBPP, mit der man den Bits-pro-Pixel-Wert des Bilds (auch als Farbtiefe bezeichnet) abfragen kann.

- Die Funktionen BitBlt, StretchBlt und Draw

  Wenn Sie bereit sind, das Bild anzuzeigen, müssen Sie es in den Ausgabe-Gerätekontext kopieren. Es gibt mehrere Funktionen, die Sie dafür verwenden können. Die Funktionen BitBlt und StretchBlt arbeiten im Großen und Ganzen genauso wie die CDC-Versionen - sie übernehmen sogar die gleichen Parameter. Die Funktion Draw entspricht im Prinzip der Funktion StretchBlt. Sie übernimmt die gleichen Parameter wie die CDC-Funktion StretchBlt, allerdings ohne den letzten Parameter (der angibt, wie der Kopiervorgang durchzuführen ist). Der Unterschied liegt in der Verarbeitung von Bildern, die einen transparenten oder Alpha-Kanal enthalten (StretchBlt kopiert die Transparenz nicht korrekt). Die Funktion Draw hat folgende Syntax:

```
Draw(HDC hDestDC,
     int xDest, int yDest, int nDestWidth, int nDestHeight,
     int xSrc, int ySrc, int nSrcWidth, int SrcHeight);
```

- Die Funktionen Load und Save

  Wenn Sie ein Bild in eine Instanz der Klasse CImage laden möchten, können Sie die Funktion Load verwenden, die den Dateinamen als einzigen Parameter übernimmt. Sie stellt das Format anhand der Dateinamenserweiterung fest und liest die Datei entsprechend ein. Die Funktion Load hat folgende Syntax:

```
Load(LPCTSTR strFileName);
```

  Wenn Sie ein Bild speichern wollen, verwenden Sie die Funktion Save. Sie hat folgende Syntax:

```
Save(LPCTSTR strFileName, REFGUID guidFileType);
```

  Die Funktion Save übernimmt zwei Parameter: den Dateinamen (strFileName) und das Format, in dem das Bild gespeichert werden soll (guidFileType). Das Bildformat kann folgende Werte annehmen:

Flag	Beschreibung
GUID_NULL	Das Bildformat wird anhand der Dateinamenerweiterung festgestellt.
GUID_BMPFile	Das Bild wird als unkomprimierte Bitmap gespeichert.
GUID_PNGFile	Das Bild wird als PNG gespeichert.
GUID_JPEGFile	Das Bild wird als JPEG gespeichert.
GUID_GIFFile	Das Bild wird als GIF gespeichert.

Mit der reichhaltigen in den CImage-Funktionen Save und Load verkapselten Funktionalität könnte man leicht eine Utility zum Konvertieren von Bildformaten schreiben, die aus zwei Hauptcode-Zeilen besteht:

```
// Bild einlesen
m_Image.Load(strInFile);
// Bild als GIF schreiben
m_Image.Save(strOutFile, GUID_GIFFile);
```

Abbildungsmodi und Koordinatensysteme

Wenn Sie die Grafikausgabe in ein Fenster vorbereiten, können Sie die Skalierung und den Zielbereich in weiten Grenzen festlegen. Diese Faktoren lassen sich über den Abbildungsmodus und den Zeichenbereich spezifizieren.

Mit dem Abbildungsmodus bestimmen Sie, wie die spezifizierten Koordinaten in Positionen auf dem Bildschirm zu übersetzen sind. Für die einzelnen Abbildungsmodi sind dabei verschiedene Maßeinheiten definiert. Die Abbildungsmodi legen Sie mit der Gerätekontextfunktion SetMapMode fest:

```
dc->SetMapMode(MM_ANSIOTROPIC);
```

Die verfügbaren Abbildungsmodi sind in Tabelle 9.3 aufgeführt.

Modus	Beschreibung
MM_ANSIOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit frei skalierbaren Achsen abgebildet.
MM_HIENGLISH	Eine logische Einheit wird auf 0,001 Zoll abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_HIMETRIC	Eine logische Einheit wird auf 0,01 Millimeter abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_ISOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit gleich skalierten Achsen abgebildet.
MM_LOENGLISH	Eine logische Einheit wird auf 0,01 Zoll abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_LOMETRIC	Eine logische Einheit wird auf 0,1 Millimeter abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_TEXT	Eine logische Einheit wird auf 1 Pixel abgebildet. Positive x gehen nach rechts, positive y nach unten.
MM_TWIPS	Eine logische Einheit wird auf 1/20 Punkt (etwa 1/1440 Zoll) abgebildet. Positive x laufen nach rechts, positive y nach oben.

Tabelle 9.3: Abbildungsmodi

Wenn Sie die Abbildungsmodi MM_ANSIOTROPIC oder MM_ISOTROPIC verwenden, können Sie den Zielbereich für die Grafik entweder mit der Funktion SetWindowExt oder mit SetViewportExt festlegen.

9.2 Eine Grafikanwendung erstellen

Um einen Eindruck von den genannten Möglichkeiten zu bekommen, erstellen Sie heute eine Beispielanwendung, die einen Großteil des bisherigen Stoffes in die Praxis umsetzt. Die Anwendung verfügt über zwei unabhängige Fenster. Das erste enthält eine Anzahl von Optionen, über die man die Figur, das Werkzeug und die Farbe zur Anzeige auswählen kann. Das andere Fenster dient als Leinwand, auf der alle ausgewählten Optionen gezeichnet werden. Der Benutzer kann wählen, ob er Linien, Quadrate, Kreise oder eine Bitmap in das zweite Fenster zeichnen will. Außerdem kann er die Farbe spezifizieren und wählen, ob der Stift oder der Pinsel für die Kreise und Quadrate anzuzeigen ist (siehe Abbildung 9.3).

Um diese Anwendung zu erstellen, werden Sie im Großen und Ganzen diesen Schritten folgen:

1. Das Anwendungsgerüst erstellen

2. Das Hauptdialogfeld erstellen

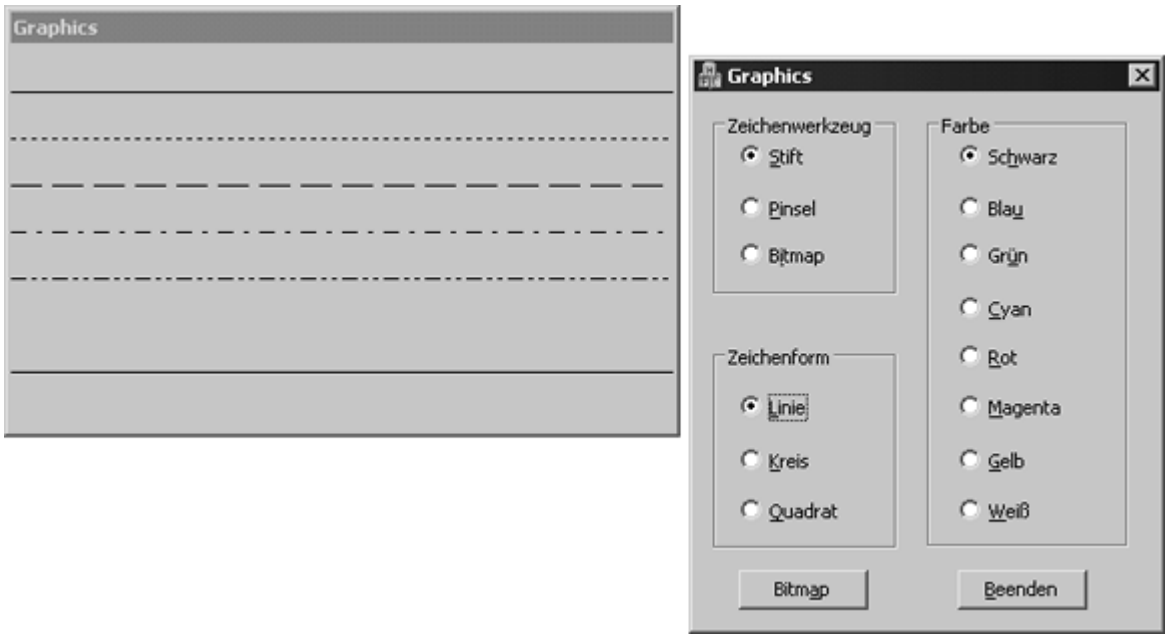


Abbildung 9.3: Die Grafikanwendung in Betrieb

3. Ein zweites nicht modales Dialogfeld einfügen, in dem alle Zeichnungen ausgeführt werden
4. Funktionalität für das Zeichnen von Linien im zweiten Dialogfeld einfügen
5. Funktionalität für das Zeichnen von Figuren in dieses einfügen
6. Funktionalität für das Laden und die Anzeige von Bilder in dieses einfügen

Das Anwendungsgerüst erstellen

Wie Sie bisher gelernt haben, erstellen Sie im ersten Schritt das Gerüst einer Anwendung mit der grundlegenden Funktionalität. Es zeigt das erste Dialogfeld der Anwendung an und verfügt über den gesamten Code, der erforderlich ist, um die Anwendung zu starten und zu beenden. Für die heutige Beispielanwendung beginnen Sie mit dem Gerüst einer dialogfeldbasierenden Anwendung. Dazu folgen Sie diesen Schritten:

1. Legen Sie ein neues MFC-Anwendung-Visual-C++-Projekt an. Geben Sie dem Projekt den Namen Graphics.

2. Verwenden Sie die gleichen Standardeinstellungen wie für die bisherigen Projekte.

Das Hauptdialogfeld entwerfen

Nachdem Sie sich durch den MFC-Anwendungs-Assistenten gearbeitet haben, können Sie an den Entwurf des Hauptdialogfelds gehen. Dieses Fenster enthält drei Gruppen von Optionsfeldern: zur Auswahl des Zeichenwerkzeugs, zur Festlegung der Zeichenfigur und zur Auswahl der Farbe. Neben diesen Gruppen von Optionsfeldern platzieren Sie noch zwei Schaltflächen im Fenster: Über die eine Schaltfläche rufen Sie das Dialogfeld Öffnen auf, um eine anzuzeigende Bitmap auszuwählen, und mit der zweiten Schaltfläche schließen Sie die Anwendung.

Die Steuerelemente ordnen Sie gemäß Abbildung 9.4 an. Tabelle 9.4 listet die zugehörigen Eigenschaften der Steuerelemente auf.

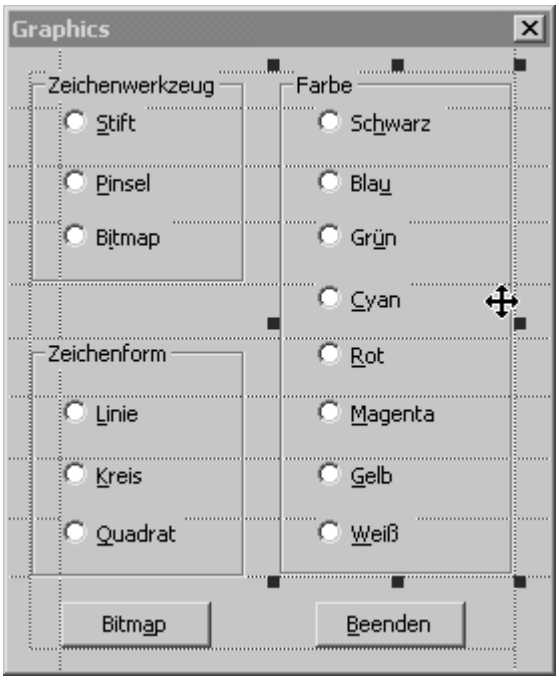


Abbildung 9.4: Das Layout des Hauptdialogfelds

Objekt	**Eigenschaft**	**Einstellung**
Gruppenfeld	Beschriftung	Werkzeug
Optionsfeld	ID Beschriftung Gruppe	IDC_RTPEN &Stift True
Optionsfeld	ID Beschriftung	IDC_RTBRUSH &Pinsel
Optionsfeld	ID Beschriftung	IDC_RTBITMAP Bit&map
Gruppenfeld	Beschriftung	Figur
Optionsfeld	ID Beschriftung Gruppe	IDC_RSLINE &Linie True
Optionsfeld	ID Beschriftung	IDC_RSCIRCLE &Kreis
Optionsfeld	ID Beschriftung	IDC_RSSQUARE &Quadrat
Gruppenfeld	Beschriftung	Farbe
Optionsfeld	ID Beschriftung	IDC_RCBLACK Schwar&z

	Gruppe	True
Optionsfeld	ID Beschriftung	IDC_RCBLUE Bla&u
Optionsfeld	ID Beschriftung	IDC_RCGREEN Gr&ün
Optionsfeld	ID Beschriftung	IDC_RCCYAN &Cyan
Optionsfeld	ID Beschriftung	IDC_RCRED &Rot
Optionsfeld	ID Beschriftung	IDC_RCMAGENTA M&agenta
Optionsfeld	ID Beschriftung	IDC_RCYELLOW &Gelb
Optionsfeld	ID Beschriftung	IDC_RCWHITE &Weiß
Schaltfläche	ID Beschriftung	IDC_BBITMAP B&itmap
Schaltfläche	ID Beschriftung	IDC_BEXIT &Beenden

Tabelle 9.4: Eigenschaftseinstellungen der Steuerelemente

Nachdem Sie das Hauptdialogfeld entworfen haben, weisen Sie jeder Gruppe der Optionsfelder jeweils eine Variable zu. Dazu weisen Sie dem ersten Optionsfeld jeder Gruppe eine Integer-Variable zu. Alle darauf folgenden Optionsfelder in jeder Gruppe bekommen dieselbe Variable zugewiesen und erhalten aufeinander folgende Werte in der Reihenfolge der Objekt-IDs. Aus diesem Grund ist es wichtig, alle Optionsfelder der einzelnen Gruppen in der Reihenfolge zu erstellen, in der die Werte aufeinander folgen sollen. Wenn Sie die Optionsfelder nicht in der richtigen Reihenfolge hinzugefügt haben, können Sie das Problem durch Setzen der Tabulator-Reihenfolge für das Dialogfeld beheben.

Um die erforderlichen Variablen an die Gruppen der Optionsfelder der Beispielanwendung zuzuweisen, fügen Sie die folgenden Variablen für die Objekte des Dialogfelds hinzu, indem Sie mit der rechten Maustaste auf das erste Optionsfeld jeder Gruppe klicken und Variable hinzufügen wählen.

Objekt	**Name**	**Kategorie**	**Typ**	**Zugriff**
IDC_RTPEN	m_iTool	Value	int	public
IDC_RSLINE	m_iShape	Value	int	public
IDC_RCBLACK	m_iColor	Value	Int	public

Tabelle 9.5: Variablen für die Optionsfelder

Nachdem den Optionsfeldern die Variablen zugewiesen sind, fügen Sie eine Behandlungsroutine für die Schaltfläche Beenden hinzu. Im Code für die Schaltfläche rufen Sie wie gewohnt die Funktion OnOK auf. Jetzt können Sie Ihre Anwendung kompilieren und ausführen. Überzeugen Sie sich davon, dass Sie alle Gruppen der Optionsfelder korrekt definiert haben, dass Sie nur jeweils ein Optionsfeld in einer Gruppe auswählen können und dass Sie eine Option in der einen Gruppe wählen können, ohne eine der anderen Gruppen zu beeinflussen.

Das zweite Dialogfeld hinzufügen

Nachdem Sie das Hauptdialogfeld erstellt haben, fügen Sie das zweite Fenster hinzu, auf dem Sie in der Art einer Leinwand Ihre Grafiken zeichnen. Es handelt sich um ein nicht modales Dialogfeld, das die ganze Zeit, während die Anwendung läuft, geöffnet bleibt. In dieses Dialogfeld nehmen Sie keine Steuerelemente auf und stellen damit eine vollkommen leere Leinwand zum Zeichnen bereit.

Um das zweite Dialogfeld zu erstellen, folgen Sie diesen Schritten:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte Ressourcen-Ansicht. Klicken Sie im Ressourcenbaum mit der rechten Maustaste auf den Ordner Dialog. Wählen Sie aus dem Kontextmenü den Befehl Dialog einfügen.

2. Wenn das neue Dialogfeld im Dialog-Editor geöffnet wird, entfernen Sie alle Steuerelemente aus dem Fenster.
3. Wählen Sie den Dialog aus und gehen Sie ins Eigenschaftenfenster. Setzen Sie die Option Systemmenü auf FALSE, damit der Benutzer das Dialogfeld nicht schließen kann, ohne die Anwendung zu verlassen.
4. Geben Sie dem Dialogfeld eine Objekt-ID, die seine Funktion beschreibt. In diesem Fall geben Sie IDD_PAINT_DLG an.
5. Klicken Sie mit der rechten Maustaste auf das Dialogfeld und wählen Sie Klasse hinzufügen.
6. Geben Sie als Namen für die neue Klasse CPaintDlg ein und vergewissern Sie sich, dass die Basisklasse auf CDialog gesetzt ist.
7. Klicken Sie auf Fertig stellen, um die neue Klasse zu erstellen.

Das neue Dialogfeld muss markiert sein, wenn Sie versuchen, eine neue Klasse hinzuzufügen. Wenn das Dialogfeld nicht markiert ist und Sie zu einem anderen Objekt oder sogar zu Code in Ihrer Anwendung gewechselt haben, weiß der Klassen-Assistent nicht, dass Sie eine Klasse für das zweite Dialogfeld in Ihrer Anwendung benötigen.

Nachdem Sie nun das zweite Dialogfeld definiert haben, fügen Sie zum ersten Dialogfeld Code hinzu, um das zweite Dialogfeld zu öffnen. Das erreichen Sie mit zwei Codezeilen in der Funktion OnInitDialog in der Klasse des ersten Fensters:

- Erstellen Sie das Dialogfeld mit der Methode Create der Klasse CDialog. Diese Funktion übernimmt zwei Argumente: die Objekt-ID des Dialogfelds und einen Zeiger auf das übergeordnete Fenster, bei dem es sich um das Hauptdialogfeld handelt.
- Verwenden Sie die Funktion ShowWindow, der Sie den Wert SW_SHOW als einziges Argument übergeben, um das zweite Dialogfeld neben dem ersten anzuzeigen.

Diese Funktionalität fügen Sie mit den folgenden Schritten hinzu:

1. Wählen Sie zuerst die Klassenansicht aus. Erweitern Sie den Baum, bis die Klasse CGraphicsDlg zu sehen ist.

2. Klicken Sie mit der rechten Maustaste auf die Klassen CGraphicsDlg und wählen Sie Hinzufügen / Variable hinzufügen.
3. Geben Sie den Variablentyp als CPaintDlg, den Namen als m_dlgPaint und die Zugriffsart als private an. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.

4. Erweitern Sie den Knoten CGraphicsDlg, um die Methoden zu sehen. Doppelklicken Sie auf die Methode OnInitDialog. Fügen Sie einige Zeilen hinzu, um die Variablen zu initialisieren. Die Funktion OnInitDialog sollte dann Listing 9.1 entsprechen.

Listing 9.1: Die Funktion OnInitDialog

```
 1: BOOL CGraphicsDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
 5: . . .
 6:
 7:   // Symbol für dieses Dialogfeld festlegen. Wird
 8:   // automatisch erledigt
 9:   // wenn das Hauptfenster der Anwendung kein Dialogfeld ist
10:   SetIcon(m_hIcon, TRUE);         // Großes Symbol verwenden
11:   SetIcon(m_hIcon, FALSE);        // Kleines Symbol verwenden
12:
13:   // TODO: Hier zusätzliche Initialisierung einfügen
14:   // Variablen initialisieren und Dialogfeld aktualisieren
15:   m_iColor = 0;
16:   m_iShape = 0;
17:   m_iTool = 0;
18:   UpdateData(FALSE);
19:
20:   // Das zweite Dialogfeld erzeugen
21:   m_dlgPaint.Create(IDD_PAINT_DLG, this);
22:   // Zweites Dialogfeld anzeigen
23:   m_dlgPaint.ShowWindow(SW_SHOW);
24:
25:   return TRUE;  // Geben Sie TRUE zurück, außer ein
26:                 // Steuerelement soll den Fokus erhalten
27: }
```

Bevor Sie die Anwendung kompilieren und ausführen können, müssen Sie die Header- Datei für die zweite Dialogfeldklasse in den Quellcode des ersten Dialogfelds einbinden. Um die Header-Datei in das erste Dialogfeld einzubinden, folgen Sie diesen Schritten:

1. Scrollen Sie zum Beginn des Quellcodes für das erste Dialogfeld und fügen eine #include-Anweisung wie in Listing 9.2 hinzu.

Listing 9.2: Die #include-Anweisung des Hauptdialogs

```
1: // GraphicsDlg.cpp : Implementierungsdatei
2: //
3:
4: #include "stdafx.h"
5: #include "Graphics.h"
6: #include "PaintDlg.h"
7: #include "GraphicsDlg.h"
```

2. Umgekehrt müssen Sie die Header-Datei für das Hauptdialogfeld in den Quellcode für das zweite Dialogfeld einbinden. Bearbeiten Sie die Datei PaintDlg.cpp, sodass die #include-Anweisungen am oberen Ende der Datei denen in Listing 9.2 entsprechen. Da auch in der Datei Graphics.cpp die Datei GraphicsDlg.h eingebunden wird müssen Sie auch in dieser cpp-Datei die Datei PaintDlg.h einbinden.

Wenn Sie Ihre Anwendung kompilieren und ausführen, sollte das zweite Dialogfeld zusammen mit dem ersten geöffnet werden. Schließt man das erste Dialogfeld und beendet damit die Anwendung, wird das zweite Dialogfeld ebenfalls geschlossen, selbst wenn Sie bisher keinerlei Code hinzugefügt haben, der das bewirkt. Das zweite Dialogfeld ist ein untergeordnetes Fenster zum ersten Dialogfeld. Wenn Sie das zweite Dialogfeld erstellen, übergeben Sie einen Zeiger auf das erste Dialogfeld als übergeordnetes Fenster für das zweite Fenster (der Parameter »this«). Damit wird eine Beziehung zwischen übergeordnetem und untergeordnetem

Fenster eingerichtet. Wird das übergeordnete Fenster geschlossen, schließt sich auch das untergeordnete. Es handelt sich hier um die gleiche Beziehung, die das erste Dialogfeld mit allen darauf platzierten Steuerelementen hat. Jedes Steuerelement ist ein untergeordnetes Fenster des Dialogfelds. Praktisch haben Sie das zweite Dialogfeld zu einem weiteren Steuerelement des ersten Dialogfelds gemacht.

Grafikfunktionen realisieren

Da alle Variablen für die Optionsfelder als öffentlich (public) deklariert sind, kann sie das zweite Dialogfeld sehen und referenzieren. Die gesamte Funktionalität zum Zeichnen lässt sich in der zweiten Dialogfeldklasse unterbringen. Allerdings müssen Sie einen Teil der Funktionen im ersten Dialogfeld realisieren, um die Variablen zu synchronisieren und das zweite Dialogfeld anzuweisen, die Grafiken zu zeichnen. Das Ganze ist aber einfacher, als Sie vielleicht annehmen.

Sobald ein Fenster neu zu zeichnen ist (etwa weil es hinter einem anderen Fenster verborgen war und nun in den Vordergrund kommt, oder weil es minimiert war oder weil es außerhalb des sichtbaren Bereichs gelegen hat), löst das Betriebssystem die Funktion OnPaint des Fensters aus. Die gesamte Funktionalität zum Zeichnen der Grafiken können Sie in dieser Funktion unterbringen und die angezeigten Grafiken dauerhaft machen.

Es ist nun klar, wo der Code für die Anzeige der Grafiken hingehört. Wie kann man aber das zweite Dialogfeld veranlassen, seine OnPaint-Funktion aufzurufen, wenn der Benutzer eine der Auswahlen im ersten Dialogfeld verändert? Nun, man kann das zweite Dialogfeld verbergen und dann wieder anzeigen, aber das dürfte dem Benutzer etwas seltsam vorkommen. Eigentlich »überzeugt« eine einzige Funktion das zweite Fenster, dass es sein gesamtes Dialogfeld neu zu zeichnen hat. Diese Funktion, Invalidate, erfordert keine Argumente und ist eine Elementfunktion der Klasse CWnd, sodass man sie für jedes Fenster oder Steuerelement einsetzen kann. Die Funktion Invalidate teilt dem Fenster - und dem Betriebssystem - mit, dass der Anzeigebereich des Fensters nicht mehr gültig und neu zu zeichnen ist. Man kann die Funktion OnPaint im zweiten Dialogfeld nach Belieben aufrufen, ohne dass man auf irgendwelche ausgefallenen Tricks oder Hacks zurückgreifen müsste.

Wie Sie mittlerweile wissen, können alle Optionsfelder die gleiche Funktionalität für ihre Klickereignisse verwenden. Somit genügt eine einzige Behandlungsroutine für das Klickereignis aller Optionsfelder. In dieser Behandlungsfunktion synchronisieren Sie die Klassenvariablen mit den Steuerelementen des Dialogfelds durch Aufruf der Funktion UpdateData und weisen dann das zweite Dialogfeld mithilfe seiner Funktion Invalidate an, sich selbst neu zu zeichnen. Sie können eine Behandlungsroutine schreiben, die beides tut:

1. Öffnen Sie das erste Dialogfeld. Markieren Sie das erste Optionsfeld im ersten Dialogfeld.

2. Fügen Sie eine Behandlungsroutine für das Ereignis BN_CLICKED ein. Übernehmen Sie nicht den voreingestellten Namen für diese Funktion, sondern geben Sie den Namen OnRSelection ein.
3. Öffenen Sie die Klassenansicht und dort die Klasse CGraphicsDlg. Im Ordner Zuordnung selektieren Sie MESSAGE. Selektieren Sie aus dem Kontextmenü Gehe zu Definition (siehe Abbildung 9.5).

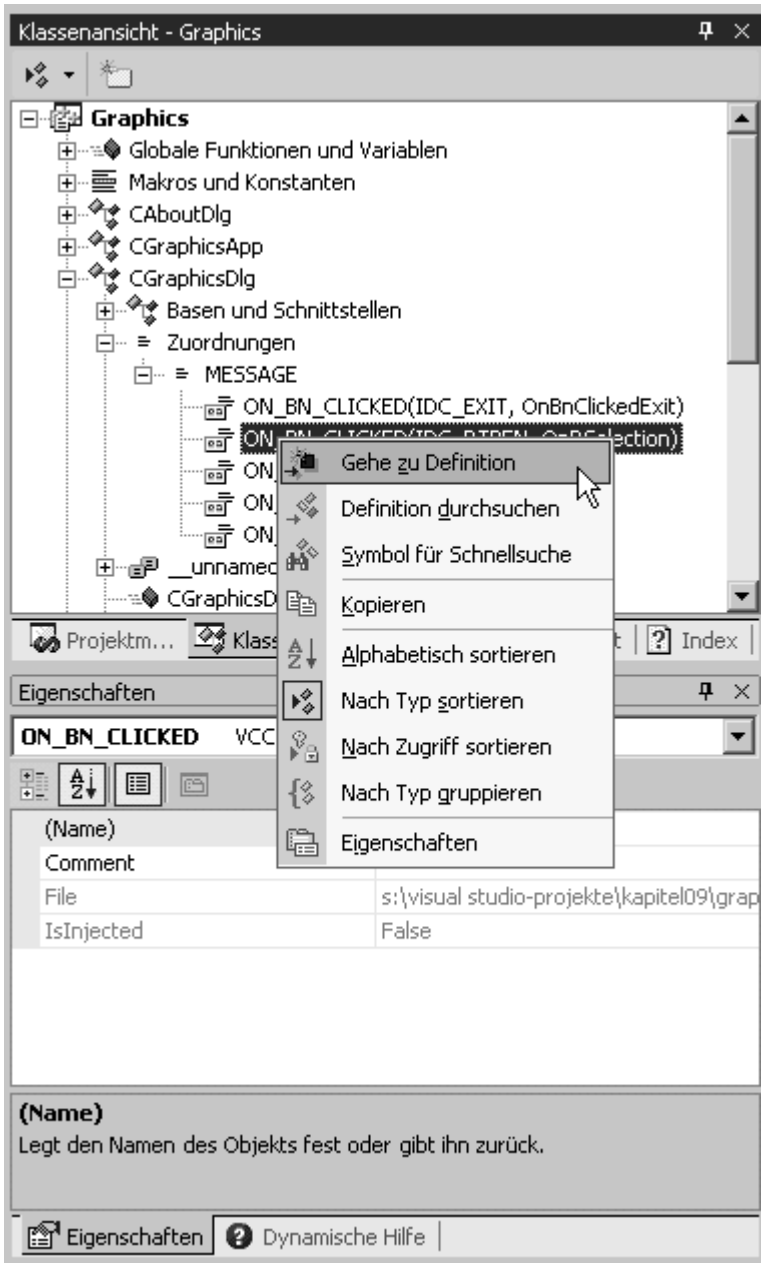


Abbildung 9.5: Zugriff auf die MESSAGE-MAP

4. Es wird die Datei GraphicsDlg.cpp geöffnet und Sie sehen den Abschnitt für die Nachrichten der Klasse CGraphicsDlg. Er sollte wie folgt aussehen:

```
BEGIN_MESSAGE_MAP(CGraphicsDlg, CDialog)
  ON_WM_SYSCOMMAND()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
  //}}AFX_MSG_MAP
  ON_BN_CLICKED(IDC_EXIT, OnBnClickedExit)
  ON_BN_CLICKED(IDC_RTPEN, OnRSelection)
END_MESSAGE_MAP()
```

5. Kopieren Sie das Makro ON_BN_CLICKED mit der Funktion OnRSelection und fügen Sie es noch dreizehnmal in die Nachrichtenzuordnungstabelle ein. Bearbeiten Sie diese neuen Kopien des Makros und ersetzen Sie die ID für das erste Optionsfeld durch die IDs der anderen 13 Optionsfelder. Die endgültige Nachrichtenzuordnungstabelle sollte wie Listing 9.3 aussehen.

Listing 9.3: Die modifizierte Nachrichtenzuordnungstabelle

```
 1: BEGIN_MESSAGE_MAP(CGraphicsDlg, CDialog)
 2:   ON_WM_SYSCOMMAND()
 3:   ON_WM_PAINT()
 4:   ON_WM_QUERYDRAGICON()
 5:   //}}AFX_MSG_MAP
 6:   ON_BN_CLICKED(IDC_EXIT, OnBnClickedExit)
 7:   ON_BN_CLICKED(IDC_RTPEN, OnRSelection)
 8:   ON_BN_CLICKED(IDC_RTBRUSH, OnRSelection)
 9:   ON_BN_CLICKED(IDC_RTBITMAP, OnRSelection)
10:   ON_BN_CLICKED(IDC_RSLINE, OnRSelection)
11:   ON_BN_CLICKED(IDC_RSSQUARE, OnRSelection)
12:   ON_BN_CLICKED(IDC_RSCIRCLE, OnRSelection)
13:   ON_BN_CLICKED(IDC_RCBLACK, OnRSelection)
14:   ON_BN_CLICKED(IDC_RCBLUE, OnRSelection)
15:   ON_BN_CLICKED(IDC_RCGREEN, OnRSelection)
16:   ON_BN_CLICKED(IDC_RCCYAN, OnRSelection)
17:   ON_BN_CLICKED(IDC_RCRED, OnRSelection)
18:   ON_BN_CLICKED(IDC_RCMAGENTA, OnRSelection)
19:   ON_BN_CLICKED(IDC_RCYELLOW, OnRSelection)
20:   ON_BN_CLICKED(IDC_RCWHITE, OnRSelection)
21: END_MESSAGE_MAP()
```

6. Bearbeiten Sie die Funktion OnRSelection und fügen Sie den Code aus Listing 9.4 ein.

Listing 9.4: Die Funktion OnRSelection

```
1: void CGraphicsDlg::OnRSelection()
2: {
3:   // TODO: Geben Sie hier Ihren Kontrollbehandlungscode für die
4:   // Benachrichtigung ein.
5:   // Daten synchronisieren
6:   UpdateData(TRUE);
7:   // Zweites Dialogfeld neu zeichnen
8:   m_dlgPaint.Invalidate();
9: }
```

Linien zeichnen

Wenn Sie die Anwendung jetzt kompilieren und ausführen, zeichnet sich das zweite Dialogfeld neu, wenn Sie ein anderes Optionsfeld im Hauptdialogfeld wählen. Davon bemerken Sie allerdings gar nichts. Sie haben zwar das Neuzeichnen initiiert, aber dem zweiten Dialogfeld noch nicht mitgeteilt, was zu zeichnen ist. Das erledigen wir nun im nächsten Schritt für die Beispielanwendung.

Am einfachsten lassen sich Linien in verschiedenen Stilen im zweiten Dialogfeld zeichnen, weil Sie bereits etwas Erfahrung damit haben. Für die einzelnen Stiftstile ist jeweils ein Stift zu erzeugen, der die momentan ausgewählte Farbe verwendet. Nachdem Sie alle Stifte erstellt haben, gehen Sie in einer Schleife durch die einzelnen Stifte, selektieren sie nacheinander und zeichnen mit dem jeweiligen Stift eine Linie über das Dialogfeld. Bevor Sie diese Schleife starten können, müssen Sie ein paar Berechnungen anstellen, um die Lage der Linien - die Anfangs- und Endpunkte - im Dialogfeld zu bestimmen.

Zunächst erstellen Sie eine Farbtabelle, die je einen Eintrag für die Optionen des Gruppenfelds Farben im ersten Dialogfeld hat. Dazu folgen Sie diesen Schritten:

1. Fügen Sie dem zweiten Dialogfeld (CPaintDlg) eine Member-Variable hinzu. Legen Sie den Variablentyp als COLORREF, den Namen als m_crColors und den Zugriff als public fest.

2. Öffnen Sie die Header-Datei CPaintDlg.h. Suchen Sie die Definition der gerade erzeugten Variablen und ändern Sie sie wie in Listing 9.5. Sie können auch in der Klassenansicht von CPaintDlg aus dem Kontextmenü Gehe zu Definition auswählen.

Listing 9.5: Die Variablendeklaration für die Farbtabelle

```
1: public:
2:   static const COLORREF m_crColors[8];
3: };
```

3. Öffnen Sie den Quellcode der Datei für die zweite Dialogfeldklasse (PaintDlg.cpp) und fügen Sie die Farbtabelle aus Listing 9.6 am Beginn der Datei vor dem Konstruktor und Destruktor der Klasse ein.

Listing 9.6: Die Variableninitialisierung für die Farbtabelle

```
 1: #include "GraphicsDlg.h"
 2:
 3: const COLORREF CPaintDlg::m_crColors[8] = {
 4:   RGB(   0,   0,   0),    // Schwarz
 5:   RGB(   0,   0, 255),    // Blau
 6:   RGB(   0, 255,   0),    // Grün
 7:   RGB(   0, 255, 255),    // Cyan
 8:   RGB( 255,   0,   0),    // Rot
 9:   RGB( 255,   0, 255),    // Magenta
10:   RGB( 255, 255,   0),    // Gelb
11:   RGB( 255, 255, 255)     // Weiß
12: };
13:
14: // Dialogfeld CpaintDlg
```

Die Farbtabelle befindet sich nun am richtigen Platz und Sie können eine neue Funktion hinzufügen, um die Linien zu zeichnen. Um die Funktion OnPaint nicht übermäßig kompliziert und unüberschaubar zu machen, ist es sinnvoller, nur einen kleinen Teil des Codes in dieser Funktion unterzubringen. Mit diesem Code ermittelt man lediglich, was im zweiten Dialogfeld zu zeichnen ist und ruft dann spezialisierte Funktionen auf, die die verschiedenartigen Figuren zeichnen. In diesem Sinne fügen Sie eine neue Member- Funktion für die zweite Dialogfeldklasse zum Zeichnen von Linien hinzu:

1. Wählen Sie in der Klassenansicht die Klasse CPaintDlg aus und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.

2. Legen Sie den Typ der Funktion als void und den Namen als DrawLine fest. Fügen Sie den ersten Parameter vom Typ CPaintDC* mit dem Namen pdc ein. Klicken Sie auf die Schaltfläche Hinzufügen, um den Parameter hinzuzufügen. Fügen Sie den zweiten Parameter vom Typ int mit dem Namen iColor ein. Klicken Sie auf die Schaltfläche Hinzufügen, um den zweiten Parameter hinzuzufügen. Legen Sie den Zugriff als private fest. Beenden Sie den Assistenten mit Fertig stellen.
3. Schreiben Sie den Code aus Listing 9.7 in diese Funktion.

Listing 9.7: Die Funktion DrawLine

```
 1: void CPaintDlg::DrawLine(CPaintDC *pdc, int iColor)
 2: {
 3:   // Stifte deklarieren und erzeugen
 4:   CPen pnSolidPen (PS_SOLID, 1, m_crColors[iColor]);
 5:   CPen pnDotPen (PS_DOT, 1, m_crColors[iColor]);
 6:   CPen pnDashPen (PS_DASH, 1, m_crColors[iColor]);
 7:   CPen pnDashDotPen (PS_DASHDOT, 1, m_crColors[iColor]);
 8:   CPen pnDashDotDotPen (PS_DASHDOTDOT, 1, m_crColors[iColor]);
 9:   CPen pnNullPen (PS_NULL, 1, m_crColors[iColor]);
10:   CPen pnInsidePen (PS_INSIDEFRAME, 1, m_crColors[iColor]);
11:
12:   // Zeichenbereich ermitteln
13:   CRect rRect;
14:   GetClientRect(rRect);
15:   rRect.NormalizeRect();
16:
17:   // Abstand zwischen den Linien berechnen
18:   CPoint ptStart;
19:   CPoint ptEnd;
```

```
20:   int iDist = rRect.Height() / 8;
21:   CPen* pOldPen = NULL;
22:   // Startpunkte festlegen
23:   ptStart.y = rRect.top;
24:   ptStart.x = rRect.left;
25:   ptEnd.y = ptStart.y;
26:   ptEnd.x = rRect.right;
27:   int i;
28:   // Schleife durch die verschiedenen Stifte
29:   for (i = 0; i < 7; i++)
30:   {
31:     // Bei welchem Stift sind wir?
32:     switch (i)
33:     {
34:       case 0:    // durchgehend
35:         pOldPen = pdc->SelectObject(&pnSolidPen);
36:         break;
37:       case 1:    // Punkte
38:         pdc->SelectObject(&pnDotPen);
39:         break;
40:       case 2:    // Striche
41:         pdc->SelectObject(&pnDashPen);
42:         break;
43:       case 3:    // Strichpunkt
44:         pdc->SelectObject(&pnDashDotPen);
45:         break;
46:       case 4:    // Strich Punkt Punkt
47:         pdc->SelectObject(&pnDashDotDotPen);
48:         break;
49:       case 5:    // Unsichtbar
50:         pdc->SelectObject(&pnNullPen);
51:         break;
52:       case 6:    // Innenseite
53:         pdc->SelectObject(&pnInsidePen);
54:         break;
55:     }
56:     // Nach unten zur nächsten Position
57:     ptStart.y = ptStart.y + iDist;
58:     ptEnd.y = ptStart.y;
59:     // Linie zeichnen
60:     pdc->MoveTo(ptStart);
61:     pdc->LineTo(ptEnd);
62:   }
63:   // Originalstift selektieren
64:   pdc->SelectObject(pOldPen);
65: }
```

Der Code in Listing 9.7 deklariert zuerst mehrere Stiftvariablen, eine für jeden in Abbildung 9.1 gezeigten Linienstil. Als Nächstes stellt er die Größe des Zeichenbereichs fest, die zur Berechnung des Abstands zwischen den zu zeichnenden Linien dient. Nachdem die Startposition ermittelt ist, beginnt der Code mit einer Schleife durch alle Stiftstile und wählt bei jedem Durchgang den nächsten Stift als aktuellen Stift. Beachten Sie, dass der vorherige Stift nur beim ersten Schleifendurchgang gespeichert wird. Nur diesen Stift benötigen Sie, um den aktuellen Stift nach dem Beenden der Schleife zurückzusetzen. Nachdem die switch-Anweisung verlassen wurde, fährt der Code weiter unten beim Startpunkt für die nächste zu zeichnende Linie fort und zeichnet die Linie mithilfe der Methoden MoveTo und LineTo. Wenn die Schleife beendet ist, wird der ursprüngliche Stift wieder als aktueller Stift ausgewählt.

Jetzt müssen Sie noch die Funktion OnPaint bearbeiten, damit sie die Funktion DrawLine im Bedarfsfall aufruft. Fügen Sie diese Funktion als Behandlungsfunktion für die Nachricht WM_PAINT hinzu. Es fällt auf, dass der generierte Code für diese Funktion eine CPaintDC-Variable statt der normalen CDC-Klasse erstellt hat. Die Klasse CPaintDC ist von der Gerätekontextklasse CDC abgeleitet. Sie ruft automatisch die API-Funktionen BeginPaint und EndPaint auf, die alle Windows-Anwendungen vor dem Zeichnen während der Verarbeitung der WM_PAINT-Nachricht aufrufen müssen. Man kann sie genau wie ein normales Gerätekontext-Objekt behandeln und auch die gleichen Funktionen aufrufen.

In der Funktion OnPaint ist ein Zeiger auf das übergeordnete Fenster zu ermitteln, damit man die Werte der Variablen überprüfen kann, die mit den Gruppen von Optionsfeldern für die Festlegung von Farbe, Werkzeug und der im zweiten Dialogfeld zu zeichnenden Figur verbunden sind. Aus diesen Informationen leiten Sie ab, ob die Funktion DrawLine oder eine andere Funktion, die Sie noch nicht geschrieben haben, aufzurufen ist. Um diese Funktionalität in die Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie eine Behandlungsroutine für die Nachricht WM_PAINT in die zweite Dialogfeldklasse (CPaintDlg) ein.

2. Schreiben den Code aus Listing 9.8 in diese in Ihrer Klasse erstellte Funktion.

Listing 9.8: Die Funktion OnPaint

```
 1: void CPaintDlg::OnPaint()
 2: {
 3:   CPaintDC dc(this); // device context for painting
 4:
 5:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein
 6:   // CDialog::OnPaint() soll zum Zeichnen von Meldungen nicht
 7:   // aufgerufen werden
 8:   // Zeiger auf übergeordnetes Fenster ermitteln
 9:   CGraphicsDlg *pWnd = (CGraphicsDlg*)GetParent();
10:   // Ist der Zeiger gültig?
11:   if (pWnd)
12:   {
13:     // Bitmap als Werkzeug gewählt?
14:     if (pWnd->m_iTool == 2)
15:     {
16:     }
17:     else    // Nein, Figur zeichnen
18:     {
19:       // Wird Linie gezeichnet?
20:       if (pWnd->m_iShape == 0)
21:         DrawLine(&dc, pWnd->m_iColor);
22:     }
23:   }
24: }
```

Die Funktion in Listing 9.8 fragt zuerst einen Zeiger auf das übergeordnete Fenster, das Hauptdialogfeld, ab. Danach stellt der Code fest, welche Figur zu zeichnen ist, indem er zuerst das gewählte Werkzeug überprüft (um sicherzugehen, dass der Benutzer nicht das Bitmap-Bild anzeigen will, das später hinzugefügt wird) und dann ermittelt, welche Form ausgewählt ist. Nachdem Sie festgestellt haben, dass Linien zu zeichnen sind, ruft der Code die Funktion DrawLine auf und übergibt ihr den Gerätekontext und die aktuell ausgewählte Farbe.

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie Linien über das zweite Dialogfeld zeichnen können, wie es Abbildung 9.6 demonstriert.

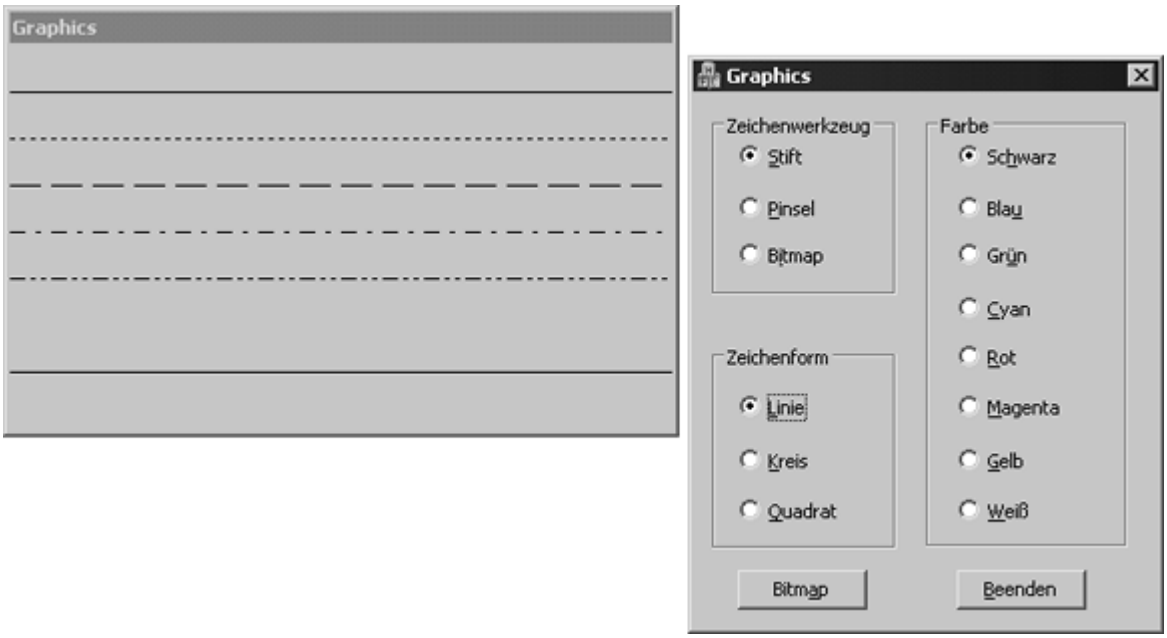


Abbildung 9.6: Linien im zweiten Dialogfeld zeichnen

C++-Exkurs: Die Schlüsselworte static und const

Das Schlüsselwort static wird verwendet, um anzuzeigen, dass Variablen eine statische Lebensdauer haben (die Lebensdauer der Anwendung). Wird eine Member-Variable einer Klasse als static deklariert, teilen sich alle Instanzen dieser Klasse dieselbe Instanz der Variable. Es gibt also nicht so viele Instanzen der Variable wie es Instanzen der Klasse gibt. Als static markierte Variablen werden beim Start der Anwendung erzeugt und beim Beenden der Anwendung zerstört.

Das Schlüsselwort const wird verwendet, um anzuzeigen, dass Variablen nicht verändert werden können. Wenn eine Variable mit dem Schlüsselwort const versehen ist, wird der Wert, mit dem die Variable initialisiert wird, der einzige sein, den sie jemals annehmen kann. Ein Versuch, den Wert der Variablen zu ändern, löst einen Compilerfehler aus. Wird das Schlüsselwort const bei einem Funktionsparameter verwendet, verhindert es die Änderung des Parameters innerhalb der Funktion.

Kreise und Quadrate zeichnen

Die grundlegenden Strukturen sind nun realisiert und Sie wissen, wie man die Ausgaben im zweiten Dialogfeld nach Belieben ändern kann. Jetzt können Sie den Code in das zweite Dialogfeld aufnehmen, um die Kreise und Quadrate zu zeichnen. Für diese Figuren kommen die Gerätekontextfunktionen Ellipse und Rectangle zum Einsatz. Diese Funktionen verwenden den momentan ausgewählten Stift und den entsprechenden Pinsel, um die Figuren an der angegebenen Position zu zeichnen. Bei beiden Funktionen kann man ein CRect-Objekt übergeben, um das Rechteck zu definieren, in dem die spezifizierte Figur darzustellen ist. Die Funktion Rectangle füllt den gesamten angegebenen Bereich, während die Funktion Ellipse einen Kreis oder eine Ellipse zeichnet, wobei die Mittelpunkte der Rechteckseiten vom Umfang der Ellipse berührt werden. Da diese Funktionen sowohl den Stift als auch den Pinsel verwenden, müssen Sie einen unsichtbaren Stift bzw. einen unsichtbaren Pinsel erzeugen und selektieren, um dem Benutzer zu ermöglichen, entweder den Stift oder den Pinsel zu wählen. Beim Stift können Sie zu diesem Zweck auf den Null-Stift zurückgreifen, während Sie für den Pinsel einen durchgehenden Pinsel in der Hintergrundfarbe des Fensters (Hellgrau) erzeugen müssen.

Die Positionen für die Figuren berechnen Sie nach einem anderen Verfahren als für die Linien. Bei den Linien konnte man einfach die Höhe des Fensters durch 8 teilen und dann eine Linie in jedem Abschnitt vom linken Rand zum rechten ziehen. Bei Ellipsen und Rechtecken teilen Sie das Dialogfenster in acht gleich große Rechtecke. Am einfachsten lässt sich das erreichen, indem man zwei Reihen mit je vier Figuren erzeugt. Zwischen jeder Figur bleibt etwas Platz, damit der Benutzer die verschiedenen Stifte für die Umrisse der einzelnen Figuren erkennen kann.

Um diese Funktionalität in die Beispielanwendung einzubauen, tun Sie Folgendes:

1. Fügen Sie für die zweite Dialogfeldklasse eine neue Funktion hinzu.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen als DrawRegion fest und fügen Sie vier Parameter hinzu. Der erste Parameter sollte vom Typ CPaintDC * sein und den Namen pdc erhalten; die restlichen Parameter sind vom Typ int und tragen die Namen iColor, iTool und iShape. Legen Sie den Zugriff als private fest.
3. In die Funktion schreiben Sie den Code aus Listing 9.9.

Listing 9.9: Die Funktion DrawRegion

```
 1: void CPaintDlg::DrawRegion(CPaintDC *pdc,
 2:                            int iColor, int iTool, int iShape)
 3: {
 4:   // Stifte deklarieren und erzeugen
 5:   CPen pnSolidPen (PS_SOLID, 1, m_crColors[iColor]);
 6:   CPen pnDotPen (PS_DOT, 1, m_crColors[iColor]);
 7:   CPen pnDashPen (PS_DASH, 1, m_crColors[iColor]);
 8:   CPen pnDashDotPen (PS_DASHDOT, 1, m_crColors[iColor]);
 9:   CPen pnDashDotDotPen (PS_DASHDOTDOT, 1, m_crColors[iColor]);
10:   CPen pnNullPen (PS_NULL, 1, m_crColors[iColor]);
11:   CPen pnInsidePen (PS_INSIDEFRAME, 1, m_crColors[iColor]);
12:
13:   // Pinsel deklarieren und erzeugen
14:   CBrush brSolidBrush(m_crColors[iColor]);
15:   CBrush brBDiagBrush(HS_BDIAGONAL, m_crColors[iColor]);
16:   CBrush brCrossBrush(HS_CROSS, m_crColors[iColor]);
17:   CBrush brDiagCrossBrush(HS_DIAGCROSS, m_crColors[iColor]);
18:   CBrush brFDiagBrush(HS_FDIAGONAL, m_crColors[iColor]);
19:   CBrush brHorizBrush(HS_HORIZONTAL, m_crColors[iColor]);
20:   CBrush brVertBrush(HS_VERTICAL, m_crColors[iColor]);
21:   CBrush brNullBrush(RGB(208, 208, 208));
22:
23:   // Größe der Zeichenbereiche berechnen
24:   CRect rRect;
25:   GetClientRect(rRect);
26:   rRect.NormalizeRect();
27:   int iVert = rRect.Height() / 2;
28:   int iHeight = iVert - 10;
29:   int iHorz = rRect.Width() / 4;
30:   int iWidth = iHorz - 10;
31:   CRect rDrawRect;
32:   CPen *pOldPen = NULL;
33:   CBrush *pOldBrush = NULL;
34:   // Schleife durch alle Pinsel und Stifte
35:   for (int i = 0; i < 7; i++)
36:   {
37:     if (i == 0)
38:     {
39:       // Position für diese Figur bestimmen
40:       // Die erste Reihe beginnen
41:       rDrawRect.top = rRect.top + 5;
42:       rDrawRect.left = rRect.left + 5;
43:       rDrawRect.bottom = rDrawRect.top + iHeight;
44:       rDrawRect.right = rDrawRect.left + iWidth;
45:     }
46:     else if (i == 4)
47:     {
48:       // Position für diese Figur bestimmen
49:       // Die zweite Reihe beginnen
50:       rDrawRect.top = rDrawRect.top + iVert;
51:       rDrawRect.left = rRect.left + 5;
52:       rDrawRect.bottom = rDrawRect.top + iHeight;
53:       rDrawRect.right = rDrawRect.left + iWidth;
54:     }
```

```
 55:     else
 56:     {
 57:       // Position für diese Figur bestimmen
 58:       rDrawRect.left = rDrawRect.left + iHorz;
 59:       rDrawRect.right = rDrawRect.left + iWidth;
 60:     }
 61:     switch (i)
 62:     {
 63:       case 0:   // Durchgehend
 64:         // Passenden Stift und Pinsel auswählen
 65:         pOldPen = pdc->SelectObject(&pnSolidPen);
 66:         pOldBrush = pdc->SelectObject(&brSolidBrush);
 67:         break;
 68:       case 1:   // Punkt - Diagonal von rechts oben nach
 69:                 //                  links unten
 70:         // Passenden Stift und Pinsel auswählen
 71:        pdc->SelectObject(&pnDotPen);
 72:         pdc->SelectObject(&brBDiagBrush);
 73:         break;
 74:       case 2:   // Strich - Pinsel kreuzweise
 75:         // Passenden Stift und Pinsel auswählen
 76:         pdc->SelectObject(&pnDashPen);
 77:         pdc->SelectObject(&brCrossBrush);
 78:         break;
 79:       case 3:   // Strichpunkt - Diagonal kreuzweise
 80:         // Passenden Stift und Pinsel auswählen
 81:         pdc->SelectObject(&pnDashDotPen);
 82:         pdc->SelectObject(&brDiagCrossBrush);
 83:         break;
 84:       case 4:  // Strich Punkt Punkt -
 85:                // Diagonal von links oben nach
 86:                // rechts unten
 87:         // Passenden Stift und Pinsel auswählen
 88:         pdc->SelectObject(&pnDashDotDotPen);
 89:        pdc->SelectObject(&brFDiagBrush);
 90:         break;
 91:       case 5:   // Null - Horizontal
 92:         // Passenden Stift und Pinsel auswählen
 93:         pdc->SelectObject(&pnNullPen);
 94:         pdc->SelectObject(&brHorizBrush);
 95:         break;
 96:       case 6:   // Innenseite - Vertikal
 97:         // Passenden Stift und Pinsel auswählen
 98:         pdc->SelectObject(&pnInsidePen);
 99:         pdc->SelectObject(&brVertBrush);
100:         break;
101:       }
102:       // Welches Werkzeug wird verwendet?
103:       if (iTool == 0)
104:         pdc->SelectObject(brNullBrush);
105:       else
106:         pdc->SelectObject(pnNullPen);
107:       // Welche Figur wird gezeichnet?
108:       if (iShape == 1)
109:         pdc->Ellipse(rDrawRect);
110:       else
111:         pdc->Rectangle(rDrawRect);
112:     }
113:     // Auf ursprünglichen Stift und Pinsel zurücksetzen
114:     pdc->SelectObject(pOldBrush);
115:     pdc->SelectObject(pOldPen);
116: }
```

Listing 9.9 folgt im Grunde der Logik in der Funktion DrawLine. Zuerst deklariert es einen Satz Stifte, einen für

jeden Linienstil, und dann einen Satz Pinsel, einen für jeden Pinselstil. Als Nächstes ermittelt der Code die Größe des Zeichenbereichs und der Figuren, die Sie zeichnen können. Dann beginnt der Code mit einer Schleife durch alle Pinsel und Stifte. Die switch-Anweisung wählt den aktuellen Stift und den aktuellen Pinsel und richtet das Rechteck des Zeichenbereichs auf die Position aus, an der die nächste Figur gezeichnet wird. Nach der switch-Anweisung prüft die Funktion, welches Werkzeug der Benutzer verwendet (den Stift oder den Pinsel) und setzt das andere Werkzeug auf den NULL-Stift oder -Pinsel. Zuletzt ermittelt der Code, welche Figur Sie zeichnen wollen und ruft die entsprechende Funktion auf, entweder Ellipse oder Rectangle. Nach Beenden der Schleife werden der ursprüngliche Stift und der ursprüngliche Pinsel wieder ausgewählt, die beim Eintritt in die Funktion selektiert waren.

Nachdem Sie nun die Kreise und Quadrate im zweiten Dialogfeld zeichnen können, müssen Sie noch diese Funktion aufrufen, wenn der Benutzer die Figuren entweder mit einem Stift oder einem Pinsel ausgewählt hat. Dazu fügen Sie die fett gedruckten Zeilen von Listing 9.10 in die Funktion OnPaint ein.

Listing 9.10: Die modifizierte Funktion OnPaint

```
 1: void CPaintDlg::OnPaint()
 2: {
 3:   CPaintDC dc(this); // Gerätekontext zum Zeichnen
 4:
 5:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
 6:
 7:   // Zeiger auf übergeordnetes Fenster ermitteln
 8:   CGraphicsDlg *pWnd = (CGraphicsDlg*)GetParent();
 9:   // Ist der Zeiger gültig?
10:   if (pWnd)
11:   {
12:     // Bitmap als Werkzeug gewählt?
13:     if (pWnd->m_iTool == 2)
14:     {
15:     }
16:     else    // Nein, Figur zeichnen
17:     {
18:       // Wird Linie gezeichnet?
19:       if (m_iShape == 0)
20:         DrawLine(&dc, pWnd->m_iColor);
21:       else    // Ellipse oder Rechteck zeichnen
22:         DrawRegion(&dc, pWnd->m_iColor, pWnd->m_iTool,
23:                    pWnd->m_iShape);
24:     }
25:   }
26:   // CDialog::OnPaint() soll zum Zeichnen von Meldungen nicht
27:   // aufgerufen werden.
28: }
```

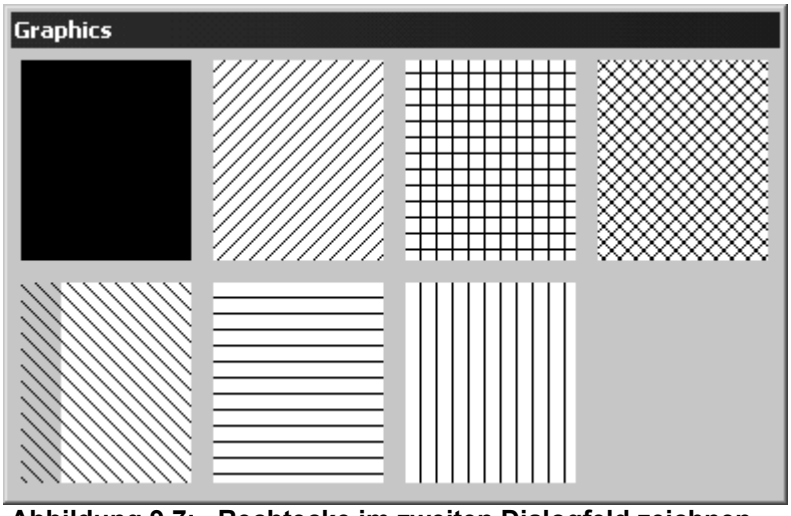


Abbildung 9.7: Rechtecke im zweiten Dialogfeld zeichnen

Jetzt können Sie die Anwendung kompilieren und ausführen und sollten nicht nur Linien, sondern auch

Quadrate und Kreise anzeigen können, wobei Sie auch zwischen der Anzeige der Umrisse und der ausgefüllten Figuren ohne Randlinien wählen können, wie es Abbildung 9.7 darstellt.

Bitmaps laden

Mit der Anwendung lassen sich jetzt verschiedene Grafiken im zweiten Dialogfeld zeichnen. Es ist noch die Funktionalität zu ergänzen, um Bitmaps zu laden und anzuzeigen. Die Bitmaps lassen sich in einfacher Weise zu den Ressourcen in der Anwendung hinzufügen, mit eigenen Objekt-IDs versehen und dann mit der Funktion LoadBitmap und dem Makro MAKEINTRESOURCE in ein Objekt der Klasse CBitmap laden. Allerdings ist das nicht sehr nützlich, wenn man eigene Anwendungen erstellt. Sinnvoll ist es eigentlich nur, wenn man Bitmaps aus Dateien vom Systemlaufwerk laden kann. Die Bitmap laden Sie mit der API-Funktion LoadImage in den Speicher und verbinden dann das geladene Bild mit dem CBitmap-Objekt.

In Ihrer Anwendung können Sie zu diesem Zweck eine Funktion mit der Schaltfläche Bitmap auf dem ersten Dialogfeld verbinden. Die Schaltfläche ruft das Dialogfeld Öffnen auf, in dem der Benutzer eine anzuzeigende Bitmap auswählen kann. Für das Dialogfeld Öffnen erstellt man einen Filter, um aus den verfügbaren Dateien nur die Bitmaps aufzulisten, die sich im zweiten Dialogfeld anzeigen lassen. Nachdem der Benutzer eine Bitmap ausgewählt hat, ermitteln Sie die Namen für die Datei und den Pfad aus dem Dialogfeld und laden die Bitmap mit der Funktion LoadImage. Mit einem gültigen Handle auf die Bitmap, die in den Speicher geladen wurde, löschen Sie die aktuelle Bitmap aus dem CBitmap-Objekt. Wurde eine Bitmap in das CBitmap-Objekt geladen, lösen Sie das CBitmap- Objekt von dem jetzt gelöschten Bild. Nachdem Sie sichergestellt haben, dass im CBitmap- Objekt noch kein Bild geladen ist, verbinden Sie die eben in den Speicher geladene Bitmap mithilfe der Funktion Attach. Machen Sie das zweite Dialogfeld mit der Funktion Invalidate ungültig, sodass eine noch angezeigte alte Bitmap durch die neuere ersetzt wird.

Um diese Funktionalität zu unterstützen, fügen Sie in die Klasse CGraphicsDlg eine Stringvariable für den Namen des Bitmaps und eine CBitmap-Variable für das Bitmap selbst hinzu. Folgen Sie diesen Schritten:

1. Fügen Sie folgende Variablen in die Klasse CGraphicsDlg ein:

Name	Typ	Zugriff
m_strBitmap	Cstring	public
m_bmpBitmap	CBitmap	public

Tabelle 9.6: Variablen für die Bitmapverarbeitung

2. Fügen Sie eine Behandlungsroutine für das Klickereignis der Schaltfläche Bitmap hinzu.
3. In die neue Funktion schreiben Sie den Code aus Listing 9.11.

Listing 9.11: Die Funktion OnBnClickedBbitmap

```
 1: void CGraphicsDlg::OnBnClickedBbitmap()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Filter für Dialogfeld File Open erstellen
 6:   static char BASED_CODE szFilter[]=
 7:                              "Bitmap Files (*.bmp)|*.bmp||";
 8:   // Dialogfeld File Open erzeugen
 9:   CFileDialog dlgFile(TRUE, ".bmp", m_strBitmap,
10:               OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, szFilter);
11:
12:   // Dialogfeld File Open anzeigen und Ergebnis übernehmen
13:   if (dlgFile.DoModal() == IDOK)
14:   {
15:     // Gewählten Dateinamen ermitteln
16:     m_strBitmap = dlgFile.GetPathName();
17:     // Gewählte Bitmap-Datei laden
18:     HBITMAP hBitmap = (HBITMAP) ::LoadImage(
```

```
19:                          AfxGetInstanceHandle(),
20:                          m_strBitmap, IMAGE_BITMAP, 0, 0,
21:                          LR_LOADFROMFILE | LR_CREATEDIBSECTION);
22:
23:     // Ist Handle für das geladene Bild gültig?
24:     if (hBitmap)
25:     {
26:       // Aktuelle Bitmap löschen
27:       if (m_bmpBitmap.DeleteObject())
28:       // War Bitmap vorhanden, lösen
29:       m_bmpBitmap.Detach();
30:       // Aktuell geladene Bitmap mit Bitmap-Objekt verbinden
31:       m_bmpBitmap.Attach(hBitmap);
32:     }
33:
34:     // Zweites Dialogfeld ungültig machen
35:     m_dlgPaint.Invalidate();
36:   }
37: }
```

Der Code in Listing 9.11 erzeugt zuerst mit folgender Zeile einen Filter für das Dialogfeld Öffnen:

```
static char szFilter[] = "Bitmap Files (*.bmp)|*.bmp||";
```

Der Filter für die Dialogfelder zum Öffnen oder Speichern von Dateien besteht aus einer Reihe durch das Pipe-Zeichen (|) getrennter Stringpaare. Der erste String in jedem Paar ist eine Beschreibung des Dateityps, die der Benutzer im Kombinationsfeld sieht, wenn er den oder die zu öffnenden oder speichernden Dateityp (en) auswählt. Der zweite String im Paar ist die Erweiterung, die für diese Auswahl aktiv sein soll. In unserem Fall gibt es nur ein einziges Paar, gefolgt von einem leeren Stringpaar. Das leere Stringpaar muss immer vorhanden sein, unabhängig von der Anzahl der angegebenen Paare. Nach dem Erstellen des Dateityp-Filters erzeugten Sie mit der Klasse CFileDialog das Dialogfeld Öffnen und verwendeten es, um dem Benutzer die Angabe der anzuzeigenden Bilddatei zu ermöglichen. Wenn der Benutzer eine Datei angibt, fragt der Code mit der Methode GetPathName den Dateinamen und den vollständigen Pfad zu der Datei ab.

Wenn Sie den Dateinamen haben, verwenden Sie die Windows-API-Funktion LoadImage, um das Bitmapbild zu laden und ein Handle zum geladenen Bild zurückzugeben. Der erste Parameter der Funktion LoadImage ist das Handle zur Instanz der Anwendung. Dieses Handle können Sie mit der Funktion AfxGetInstanceHandle abfragen. Der zweite Parameter ist der zu ladende Dateiname. Der dritte Parameter ist der Dateityp des zu ladenden Bilds: IMAGE_BITMAP, IMAGE_CURSOR oder IMAGE_ICON. Der vierte Parameter ist die gewünschte Breite, der fünfte die gewünschte Höhe des Bilds. Beide können auf 0 gesetzt werden, um die tatsächliche Bildgröße zu verwenden. Der sechste und letzte Parameter ist ein Flag, das angibt, wie das Bild geladen werden soll. In Tabelle 9.7 sind die verschiedenen Optionen aufgelistet.

Nachdem Sie die neue Bitmap geladen haben, müssen Sie eine eventuell vorher geladene Bitmap entsorgen. Dazu rufen Sie zuerst die Methode DeleteObject auf, die die vorhergehende Bitmap aus dem Speicher löscht. Existierte eine zu löschende Bitmap, gibt DeleteObject TRUE zurück. In diesem Fall müssen Sie als Nächstes mit der Methode Detach das Handle dieser Bitmap vom CBitmap-Objekt lösen. Wenn das vorhergehende Bild entladen und verworfen ist, können Sie mit der Methode Attach, der Sie das Handle zur neu geladenen Bitmap übergeben, das neue Bild an das CBitmap-Objekt anhängen.

Option	**Beschreibung**
LR_DEFAULTCOLOR	Dieses Standardflag gibt nur an, dass das Flag LR_MONOCHROME nicht gesetzt ist.
LR_CREATEIBSECTION	Dieses Flag gilt nur beim Bildtyp IMAGE_BITMAP. Es gibt an, dass eine DIB-Abschnitt-Bitmap zurückzugeben ist, die die Farben in der Bitmap als Farbtabelle verwendet (die in der Bitmap verwendete Farbauswahl). Ohne dieses Flag würde die Bitmap als kompatible Bitmap geladen und die Farben in der Bitmap auf die aktuellen Farben des Anzeigegeräts abgebildet.
LR_DEFAULTSIZE	Dieses Flag gibt an, dass die Systemstandardgrößen für Symbole und Cursor zu verwenden sind. Ist dieses Flag nicht gesetzt und sind die Parameter für Breite und Höhe auf 0 gesetzt, wird die tatsächliche Größe des Symbols oder

	Cursors verwendet.
LR_LOADFROMFILE	Dieses Flag gibt an dass der als zweiter Parameter übergebene Bildname der zu ladende Dateiname ist. Ist dieses Flag nicht gesetzt, wird angenommen, dass es sich bei dem Bildnamen um die Ressourcen-ID für in die Anwendung integrierte Bilder handelt.
LR_LOADMAP3DCOLORS	Dieses Flag bewirkt, dass die Farbtabelle des Bilds nach drei bestimmten Grautönen im Bild durchsucht wird, die durch 3D-Farben des Systemstandards für Schatten, Oberfläche und Licht ersetzt werden.
LR_LOADTRANSPARENT	Dieses Flag bewirkt, dass die Farbe des ersten Pixels im Bild als transparente Farbe interpretiert wird. Alle anderen Pixel im Bild mit derselben Farbe werden als transparent gekennzeichnet.
LR_MONOCHROME	Dieses Flag bewirkt, dass das Bild als Schwarz-Weiß-Bild geladen wird.
LR_SHARED	Wenn das Bild mehrmals in die Anwendung geladen wird, bewirkt dieses Flag, dass die Anwendung bei allen Ladevorgängen ein einziges gemeinsames Handle auf das Bild verwendet. Mit anderen Worten, das Bild wird nur einmal geladen und jede weitere Anforderung, das Bild zu laden, erhält ein Handle auf diese einzelne Instanz des Bilds. Ohne dieses Flag wird bei jeder Anforderung des Bilds eine neue Kopie geladen.
LR_VGACOLOR	Dieses Flag bewirkt, dass das Bild mit echten VGA-Farben geladen wird.

Tabelle 9.7: Optionsflags für LoadImage

Bitmaps anzeigen

Nachdem Sie nun Bitmaps in den Arbeitsspeicher laden können, müssen Sie diese noch anzeigen. Dazu kopieren Sie die Bitmap mithilfe der Funktion GetBitmap aus dem CBitmap-Objekt in eine BITMAP-Struktur. Die Funktion GetBitmap ermittelt die Breite und Höhe der Bitmap. Als Nächstes erzeugen Sie einen neuen Gerätekontext, der mit dem Bildschirmgerätekontext kompatibel ist. Selektieren Sie die Bitmap in den neuen Gerätekontext und kopieren Sie sie dann von diesem zweiten Gerätekontext in den originalen Gerätekontext, wobei Sie während des Kopiervorgangs die Größe mithilfe der Funktion StretchBlt anpassen.

Um diese Funktionalität in der Beispielanwendung zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die zweite Dialogfeldklasse ein.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen als ShowBitmap fest und fügen Sie einen Parameter ein. Der Parameter sollte vom Typ CPaintDC* sein und den Namen pdc erhalten. Legen Sie den Zugriff als private fest.
3. In die Funktion übernehmen Sie den Code aus Listing 9.12.

Beachten Sie, dass der übergebene Fensterzeiger als Zeiger auf ein CWnd-Objekt deklariert ist und nicht als Klassentyp Ihres Hauptdialogfelds. Um ihn als Zeiger auf den Klassentyp des ersten Dialogfelds zu deklarieren, müssen Sie die Klasse für das erste Dialogfeld vor der Klassendeklaration für das zweite Dialogfeld deklarieren. Unterdessen erfordert das erste Dialogfeld, dass die zweite Dialogklasse zuerst deklariert wird. Das hat Einfluss auf die Reihenfolge, in der die Include-Dateien im Quellcode am Beginn jeder Datei eingebunden werden. Man kann nicht beide Klassen vor der jeweils anderen deklarieren, eine muss die erste sein. Obwohl man das Problem umgehen kann, indem man einen Platzhalter für die zweite Klasse vor der Deklaration der ersten Klasse deklariert, ist es in diesem Fall einfacher, den Typ des Zeigers als Zeiger auf die erste Dialogfeldklasse umzuwandeln.

Listing 9.12: Die Funktion ShowBitmap

```
 1: void CPaintDlg::ShowBitmap(CPaintDC* pdc)
 2: {
 3: // Zeiger in Zeiger auf Dialogklasse des Hauptfensters
 4: // umwandeln
 5: CGraphicsDlg *pWnd = (CGraphicsDlg*)GetParent();
 6: BITMAP bm;
 7: CDC dcMem;
 8: CRect rRect;
 9:
10:  // Geladenes Bild holen
11:  pWnd->m_bmpBitmap.GetBitmap(&bm);
12:  // Gerätekontext erzeugen, in den Bitmap geladen wird
13:  dcMem.CreateCompatibleDC(pdc);
14:  // Bitmap in den kompatiblen Gerätekontext selektieren
15:  dcMem.SelectObject(pWnd->m_bmpBitmap);
16:  // Verfügbaren Anzeigebereich ermitteln
17:  GetClientRect(rRect);
18:  rRect.NormalizeRect();
19:  // Bitmap in Dialogfeld kopieren und in Größe anpassen
20:  pdc->StretchBlt(10, 10,
21:                  (rRect.Width() - 20), (rRect.Height()-20),
22:                  &dcMem, 0, 0,
23:                  bm.bmWidth, bm.bmHeight, SRCCOPY);
24: }
```

Die Funktion wandelt zuerst wie in der Funktion OnPaint den Zeiger auf das Hauptdialogfeld in einen Zeiger auf eine CGraphicsDlg-Klasse um. Als Nächstes deklariert sie eine Instanz der Struktur BITMAP und übergibt sie an das CBitmap-Objekt, damit Sie die tatsächliche Bitmap abfragen können (die benötigt wird, um sie an die Gerätekontext- Objekte zu übergeben). Die BITMAP-Struktur enthält Informationen über die Bitmap und einen Zeiger auf das tatsächliche Bild.

Nachdem Sie die Bitmap haben, erzeugen Sie mit der Methode CreateCompatibleDC einen kompatiblen Gerätekontext. Der kompatible Gerätekontext ist eine nützliche Einrichtung, mit deren Hilfe Sie Zeichnungen und Manipulationen in einem Gerätekontext durchführen können, der dem Anzeigegerätekontext entspricht, jedoch mit keinem Ausgabegerät verbunden ist. Das heißt, dass Sie in diesem zweiten Gerätekontext das Bild auf verschiedene Arten im Speicher manipulieren und dann das fertige Bild in den tatsächlichen Gerätekontext kopieren und anzeigen können. Ein wichtiger Vorteil dieser Vorgehensweise ist, dass man ein Bild im Speicher ohne verbundene Anzeigegeräte schneller manipulieren kann als wenn das Bild mit dem Anzeigegerät verbunden ist, da die Anzeigehardware den Prozess der Manipulation bedeutend verlangsamt.

Wenn Sie den kompatiblen Gerätekontext haben, wählen Sie die Bitmap mit der Methode SelectObject in ihn hinein. Nachdem Sie die Bitmap in den kompatiblen Gerätekontext gewählt haben, ermitteln Sie die Abmessungen des Anzeigebereichs und verwenden die Methode StretchBlt, um die Größe des Bilds anzupassen und es in den tatsächlichen Gerätekontext zu kopieren.

Jetzt können Sie die momentan ausgewählte Bitmap im Dialogfeld anzeigen. Es ist noch die Funktion OnPaint im zweiten Dialogfeld anzupassen, um die Funktion ShowBitmap aufzurufen. Ob eine Bitmap spezifiziert wurde, lässt sich anhand des Werts der Variablen m_sBitmap im ersten Dialogfeld testen. Wenn der String leer ist, wird keine Bitmap angezeigt. Ist der String nicht leer, rufen Sie die Funktion ShowBitmap auf. Um dieses letzte Stück der Funktionalität in Ihrer Anwendung zu realisieren, fügen Sie die fett gedruckten Zeilen aus Listing 9.13 in die Funktion OnPaint ein.

Listing 9.13: Die modifizierte Funktion OnPaint

```
 1: void CPaintDlg::OnPaint()
 2: {
 3:   CPaintDC dc(this); // Gerätekontext zum Zeichnen
 4:
 5:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
 6:
 7:   // Zeiger auf übergeordnetes Fenster ermitteln
```

```
 8:    CGraphicsDlg *pWnd = (CGraphicsDlg*)GetParent();
 9:    // Ist der Zeiger gültig?
10:    if (pWnd)
11:    {
12:      // Bitmap als Werkzeug gewählt?
13:      if (pWnd->m_iTool == 2)
14:      {
15:        // Ist eine Bitmap ausgewählt und geladen?
16:        if (pWnd->m_strBitmap.GetLength() >0)
17:          // Bitmap anzeigen
18:          ShowBitmap(&dc);
19:      }
20:      else    // Nein, Figur zeichnen
21:      {
22:        // Wird eine Linie gezeichnet?
23:        if (m_iShape == 0)
24:          DrawLine(&dc, pWnd->m_iColor);
25:        else    // Ellipse oder Rechteck zeichnen
26:          DrawRegion(&dc, pWnd->m_iColor, pWnd->m_iTool,
27:                     pWnd->m_iShape);
28:      }
29:    }
30:    // CDialog::OnPaint() soll zum Zeichnen von Meldungen nicht
31:    // aufgerufen werden.
32: }
```

Jetzt sollten Sie in der Lage sein, eine Bitmap von Ihrem System auszuwählen und im zweiten Dialogfeld anzuzeigen (siehe Abbildung 9.8).

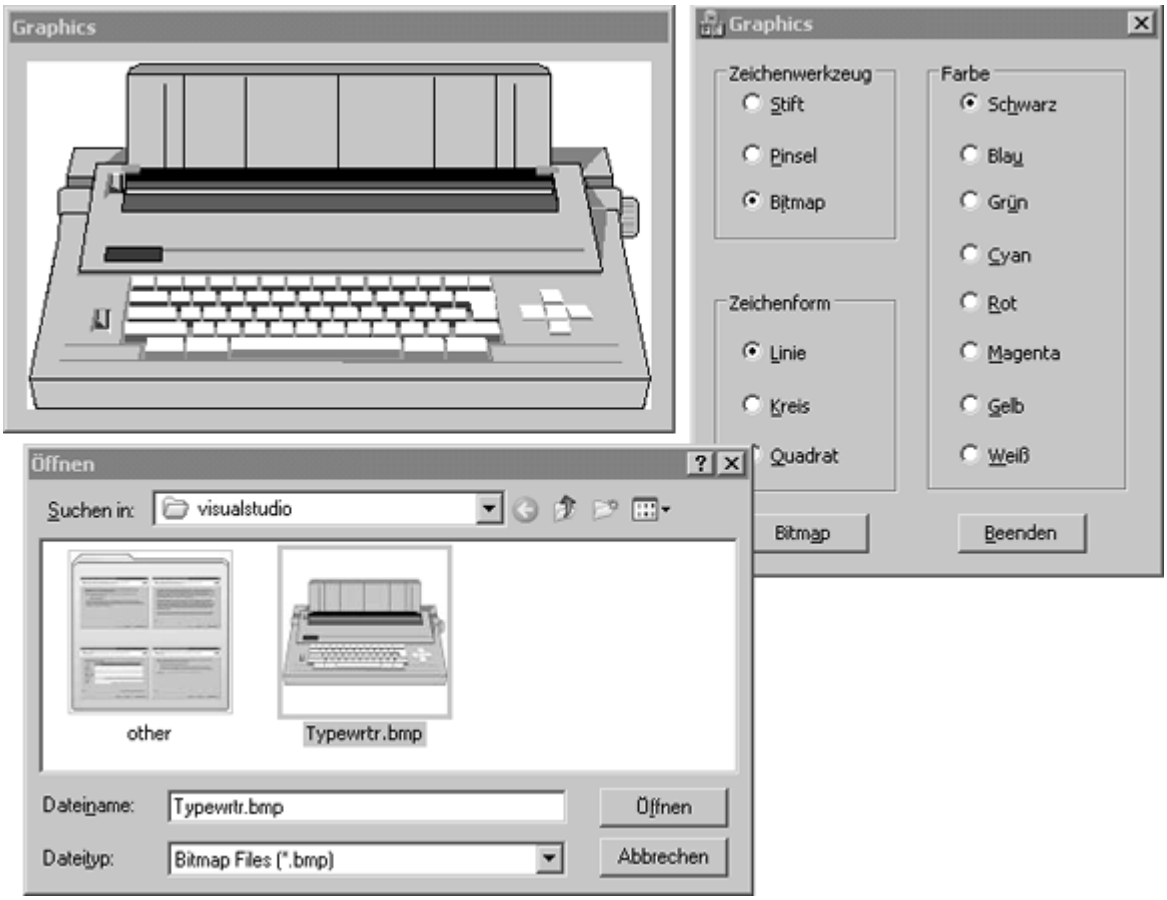


Abbildung 9.8: Eine Bitmap im zweiten Dialogfeld anzeigen, während der Hauptdialog eine neue Bitmap auswählen kann.

C++-Exkurs: Vorwärtsdefinitionen von Klassen

Wenn eine Klasse mit einer anderen interagieren soll, muss die erste Klasse wissen, welche Funktionen und Variablen in der zweiten Klasse bereitgestellt werden. Das erreicht man, indem man die Header-Datei, in der die zweite Klasse deklariert ist, vor dem ersten Verweis auf die zweite Klasse in der Deklaration oder im Code der ersten Klasse deklariert. Wenn eine Klasse eine Instanz einer zweiten Klasse als Member-Variable oder als Parameter in einer ihrer Member-Funktionen hat, muss diese Klasse den Header der zweiten Klasse vor ihrer eigenen Klassendeklaration einbinden. Aber was ist, wenn die zweite Klasse genauso den Header der ersten Klasse vor ihrer Klassendeklaration einbinden muss? Die Deklaration einer der beiden Klassen muss zuerst kommen, was bedeutet, dass die zweite Klasse nicht vor ihrer eigenen Klassendeklaration auf die erste Klasse verweisen kann, es sei denn, man verwendet irgendeine Art von Platzhalter.

Nun, wie der Zufall es will, gibt es einen solchen Platzhalter, der die erste Klasse vor der Deklaration der zweiten Klasse in ihrer eigenen Header-Datei ersetzen kann. Dieser Platzhalter wird als Vorwärtsverweis bezeichnet. So kann die zweite Klasse bis nach ihrer eigenen Deklaration warten, um den Header der ersten Klasse einzubinden. Die beiden Klassen müssen sich nicht einmal in unterschiedlichen Header-Dateien befinden; sie können beide in der gleichen Header-Datei stehen (oder in der gleichen Quelltext-Datei):

```
// Vorwärtsverweis für die zweite Klasse deklarieren
class CClass2;
// Erste Klasse deklarieren
class CClass1
{
  // Zeiger auf die zweite Klasse deklarieren
  CClass2 *pCls2;
};
// Zweite Klasse deklarieren (diesmal richtig)
class CClass2
{
  // Zeiger auf die erste Klasse deklarieren
  CClass1 *pCls1;
};
```

Die Funktionsweise ist im Grunde so: Sie deklarieren, dass Sie eine Klasse haben werden, diese jedoch erst später definieren. Der Compiler gibt sich damit zufrieden und glaubt, dass die Klasse existieren wird, also regt er sich nicht auf, wenn er vor ihrer Deklaration Verweise auf sie im Code findet. Natürlich müssen Sie die vollständige Klassendeklaration einbinden, bevor Sie die Klasse im Code verwenden, wo der Compiler alles über die Methoden und Variablen der Klasse wissen muss, aber Sie kommen so über den Deklarationsteil hinweg bis in den Code, ohne dass sich der Compiler beschwert.

JPG- oder PNG-Bilder laden und anzeigen

Die oben hinzugefügte Funktionalität ist schön und gut, aber wie praktikabel ist es heutzutage, Bitmaps anzuzeigen? Dank der wachsenden Beliebtheit von Web- Bildformaten sieht man Bitmaps nur noch selten in größerem Maßstab verwendet. Heutzutage sieht man gewöhnlich die verschiedenen Web-Bildformate wie JPEG und PNG. Auf Grund dieser Vorherrschaft macht es Sinn, den gerade hinzugefügten Code so zu ändern, dass er statt der Klasse CBitmap die Klasse CImage verwendet.

Mit der Klasse CImage kann die gesamte für das Laden und Anzeigen von Bildern erforderliche Funktionalität mit nur drei Member-Methoden ausgeführt werden:

- Mit der Methode Load können Sie das Bild in die Klasseninstanz laden.
- Mit der Methode Draw können Sie das Bild im zweiten Dialogfeld anzeigen.
- Wenn Sie ein zweites Bild laden wollen, können Sie das vorhergehende Bild mit der Methode Destroy entladen.

Um die Klasse CImage zu verwenden, müssen Sie dem Projekt zuerst die entsprechende Include-Datei hinzufügen. Da es sich bei CImage um eine von MFC und ATL gemeinsam genutzte Klasse handelt, wird sie nicht automatisch mit den MFC-Headern in der Header- Datei stdafx.h eingebunden. Also müssen Sie sie wie folgt zu stdafx.h hinzufügen:

1. Öffnen Sie die Datei stdafx.h.

2. Fügen Sie die fett gedruckte Zeile aus Listing 9.14 ein, um die Header-Datei der entsprechenden gemeinsamen Klasse in das Projekt einzufügen. Setzen Sie diese Zeile an das untere Ende der Header-Datei stdafx.h.

Listing 9.14: Die modifizierten Include-Anweisungen in stdafx.h

```
 1: // stdafx.h : Include-Datei für Standardsystem-Include-Dateien,
 2: // oder häufig verwendete, projektspezifische Include-Dateien,
 3: // die  nur in unregelmäßigen Abständen geändert werden.
 4:
 5: #pragma once
. . .
 8: #include <afxwin.h>    // MFC-Kern- und Standardkomponenten
 9: #include <afxext.h>    // MFC-Erweiterungen
10: #include <afxdisp.h>   // MFC-Automatisierungsklassen
11:
12: #include <afxdtctl.h>  // MFC-Unterstützung für allgemeine
13:                         // Steuerelemente von Internet Explorer 4
14: #ifndef _AFX_NO_AFXCMN_SUPPORT
15: #include <afxcmn.h>    // MFC-Unterstützung für allgemeine
16:                         // Windows-Steuerelemente
17: #endif // _AFX_NO_AFXCMN_SUPPORT
18: #include <atlimage.h>
```

Nachdem Sie die Header-Datei eingebunden haben, müssen Sie die Typdeklaration in der Klasse CGraphicsDlg von CBitmap auf CImage ändern. Dazu führen Sie folgende Änderungen durch:

1. Öffnen Sie die Datei GraphicsDlg.h.

2. Kommentieren Sie die Deklaration der Variablen m_bmpBitmap aus und fügen Sie wie in Listing 9.15 eine neue als CImage-Datentyp definierte Variable ein.

Listing 9.15: Die modifizierten Variablendeklarationen in GraphicsDlg.h

```
1: public:
2:   afx_msg void OnRSelection();
3:   afx_msg void OnBnClickedBexit();
4:   CString m_strBitmap;
5: //  CBitmap m_bmpBitmap;
6:   CImage m_bmpBitmap;
7:   afx_msg void OnBnClickedBbitmap();
```

Als Nächstes müssen Sie die Funktion OnBnClickedBbitmap so abändern, dass sie nicht nur Bitmap- sondern auch JPEG-Dateien lädt. Dazu ändern Sie die Funktion OnBnClickedBbitmap mit dem fett gedruckten Code in Listing 9.16.

Listing 9.16: Die modifizierte Funktion OnBnClickedBbitmap

```
 1: void CGraphicsDlg::OnBnClickedBbitmap(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Einen Filter für das Dialogfeld Öffnen erstellen
 6:   static char szFilter[] =
 7:         "JPEG Files (*.jpg)|*.jpg|Bitmap Files (*.bmp)|*.bmp||";
 8:   // Dialogfeld File Open erzeugen
 9:   CFileDialog dlgFile(TRUE, ".jpg", m_strBitmap,
10:         OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, szFilter);
11:
12:   // Dialogfeld Öffnen anzeigen und Ergebnis abfangen
```

```
13:   if (dlgFile.DoModal() == IDOK)
14:   {
15:     // Ausgewählten Dateinamen ermitteln
16:     m_strBitmap = dlgFile.GetPathName();
17:     // Aktuelles Bild löschen
18:     m_bmpBitmap.Destroy();
19:     // Neues Bild laden
20:     m_bmpBitmap.Load(m_strBitmap);
21:
22:     // Zweites Dialogfenster deaktivieren
23:     m_dlgPaint.Invalidate();
24:   }
25: }
```

Die letzte Änderung findet in der Funktion ShowBitmap statt, sodass diese das Bild mithilfe der CImage-Methode Draw anzeigt. Wie in Listing 9.16 wird die Funktion hierdurch bedeutend vereinfacht, da die Code-Menge verringert wird. Sie müssen nur noch die Abmessungen des Zeichenbereichs ermitteln; danach können Sie den Zeiger auf den in die Funktion übergebenen Gerätekontext verwenden, um das Bild zu zeichnen. Ändern Sie dazu die Funktion ShowBitmap mit den fett gedruckten Zeilen in Listing 9.17.

Listing 9.17: Die modifizierte Funktion ShowBitmap

```
 1: void CPaintDlg::ShowBitmap(CPaintDC *pdc)
 2: {
 3:   // Zeiger in Zeiger auf Hauptdialogklasse umwandeln
 4:   CGraphicsDlg *pWnd = (CGraphicsDlg*)GetParent();
 5:   CDC dcMem;
 6:   CRect rRect;
 7:
 8:   // Verfügbaren Anzeigebereich ermitteln
 9:   GetClientRect(rRect);
10:   rRect.NormalizeRect();
11:   // Die Bitmap kopieren und Größe für Dialogfenster anpassen
12:   pWnd->m_bmpBitmap.Draw(pdc->m_hDC, rRect);
13: }
```

Wenn Sie die Beispielanwendung jetzt kompilieren und starten, können Sie JPEG- Dateien ebenso wie Bitmaps suchen und anzeigen (siehe Abbildung 9.9). Sie können den String, der die im Dialogfeld Öffnen verfügbaren Dateitypen angibt, (in der Funktion OnBnClickedBbitmap) so ändern, dass Ihre Anwendung auch PNG- und GIF-Dateien anzeigen kann.

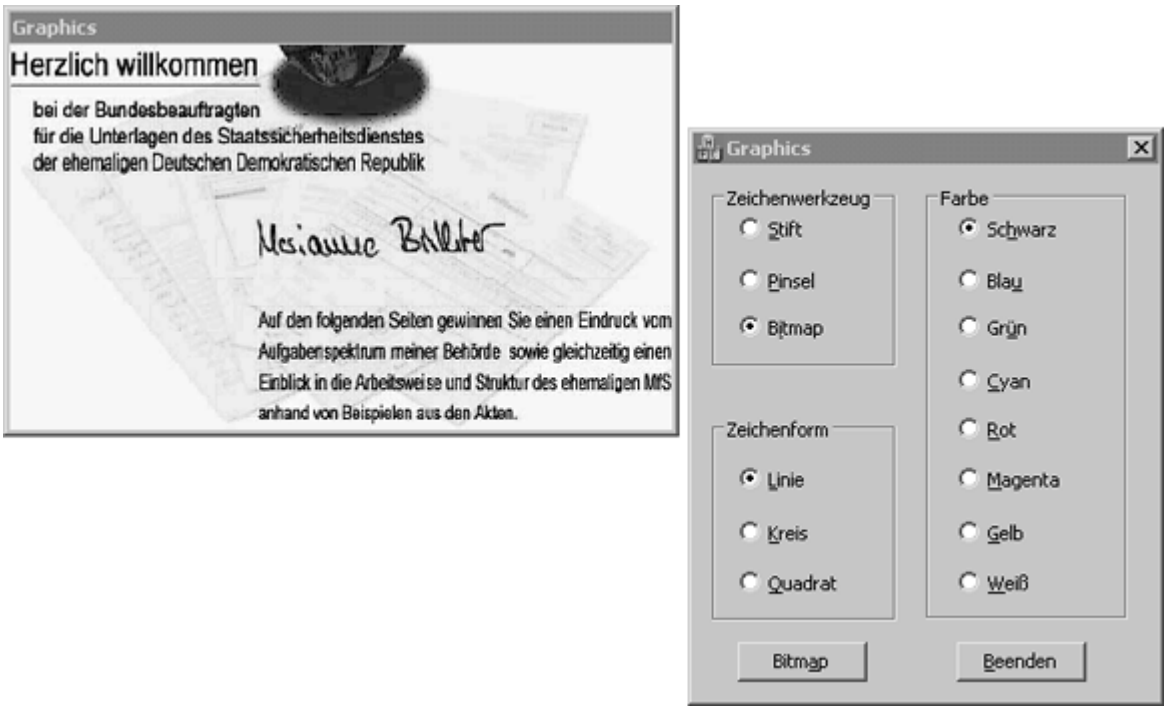


Abbildung 9.9: Anzeige eines JPEG-Bilds im zweiten Dialogfeld

Wenn sich die tatsächliche Größe des angezeigten JPEG-Bildes deutlich von der Größe des Anzeigebereichs unterscheidet, in dem es angezeigt wird, können die Farben ein wenig seltsam aussehen. Die in Windows zum Vergrößern und Verkleinern von Bildern nativ verwendeten Methoden kümmern sich nicht um die Erhaltung feiner Details im Bild. Wenn Sie die Fähigkeit, Bilder in einer anderen als ihrer tatsächlichen Größe anzuzeigen, wirklich benötigen, müssen Sie möglicherweise Ihre eigenen Routinen für die Größenänderungen schreiben, anstatt auf die nativen Windows-Routinen zurückzugreifen.

9.3 Zusammenfassung

Heute haben Sie eine Menge gelernt. Vor allem ging es darum, wie Windows die Gerätekontext-Objekte verwendet, damit Sie Grafiken immer in der gleichen Weise zeichnen können, ohne sich um die konkrete Hardware kümmern zu müssen, die der Benutzer auf seinem Computer installiert hat. Am Beispiel der grundlegenden GDI- Objekte Stift und Pinsel wurde gezeigt, wie man diese Objekte zum Zeichnen von Figuren in Fenstern und Dialogfeldern einsetzt. Weiterhin haben Sie erfahren, wie man Bitmaps von der Festplatte lädt und sie auf dem Bildschirm für den Benutzer anzeigt. Zum Zeichnen von Figuren mit Stiften und Pinseln stehen verschiedene Stile zur Verfügung. Außerdem haben Sie gelernt, wie man Farben für Stifte und Pinsel festlegt, sodass man steuern kann, wie sich Bilder dem Benutzer präsentieren.

9.4 Workshop

Fragen und Antworten

Frage:
Warum muss ich sowohl einen Stift als auch einen Pinsel spezifizieren, wenn ich eigentlich nur den einen oder den anderen verwenden möchte?

Antwort:
Wenn Sie ein Objekt zeichnen, das ausgefüllt ist, zeichnen Sie immer mit beiden Werkzeugen. Der Stift ist für

den Umriss verantwortlich, während der Pinsel den Innenbereich ausfüllt. Man kann nicht nur den einen oder den anderen verwenden, sondern immer nur beide. Wenn man nur den einen oder anderen anzeigen möchte, sind spezielle Schritte zu unternehmen.

Frage:
Warum werden alle Stiftstile durchgängig, wenn ich die Breite des Stifts über 1 erhöhe?

Antwort:
Bei einem breiteren Stift erhöht man auch die Größe der Punkte, die zum Zeichnen verwendet werden. Wie Sie noch aus Lektion 4 wissen, erhalten Sie nur verstreute Punkte, wenn Sie die von der Maus ausgelösten Ereignisse zum Zeichnen von Punkten einzeln auffangen. Sobald Sie die Größe der Punkte für die zu zeichnende Linie erhöhen, werden die Lücken zwischen den Punkten von beiden Seiten aufgefüllt, sodass eine durchgängige Linie entsteht.

Quiz

1. Welche drei Werte fasst man zu einem Farbenwert zusammen?

2. Mit welchem Instrument zeichnet man in ein Fenster, ohne dass man die vom Benutzer eingesetzte Grafikkarte kennen muss?
3. Welche Größe kann man für eine Bitmap verwenden, um einen Pinsel daraus zu erstellen?
4. Welche Nachricht sendet Windows an ein Fenster, um es anzuweisen, sich selbst neu zu zeichnen?
5. Wie kann man erreichen, dass sich ein Fenster selbst neu zeichnet?

Übungen

1. Machen Sie das zweite Dialogfeld in der Größe veränderbar und stellen Sie sicher, dass sich die im Fenster vorhandenen Zeichnungen ebenfalls in der Größe anpassen, wenn man die Größe des Dialogfelds verändert. Hinweis: Verwenden Sie die Eigenschaft Border und das Ereignis WM_SIZE.

2. Nehmen Sie einen Bitmap-Pinsel in die Gruppe der Pinsel auf, um Rechtecke und Ellipsen zu zeichnen.

Tag 10

SDI- und MDI- Anwendungen

Heute erstellen Sie mit Visual C++ eine andere Art von Anwendungen als in den vergangenen Lektionen, nämlich SDI- oder Einzeldokumentanwendungen (SDI - Single Document Interface) und Mehrfachdokumentanwendungen (MDI - Multiple Document Interface). Eine SDI-Anwendung ist eine dokumentbezogene Anwendung, die nur mit einem Dokument zu einem bestimmten Zeitpunkt und nur mit einem Typ von Dokument arbeiten kann. Eine MDI-Anwendung ist ebenfalls eine dokumentbezogene Anwendung, bei welcher der Benutzer an mehreren Dokumenten gleichzeitig arbeiten und zwischen den Fenstern der Anwendung umschalten kann.

Typische Beispiele für SDI-Anwendungen sind der Windows-Editor, WordPad und Paint. Diese Anwendungen können nur eine Art von Aufgabe realisieren und nur an einer Aufgabe zu einem bestimmten Zeitpunkt arbeiten. WordPad verhält sich wie eine SDI- Version von Word, schränkt Sie jedoch auf nur ein einziges Dokument zu einem bestimmten Zeitpunkt ein. Ab Word 2000 wurde die Anwendung Word zu einer Kombination aus SDI- und MDI-Anwendung, die wie eine MDI-Anwendung aussieht und eine untergeordnete MDI-Fenster-Schaltfläche zum Schließen unterhalb der Haupt- Titelleiste besitzt, wenn nur ein Dokument geöffnet ist. Wenn jedoch mehrere Dokumente geöffnet sind, erscheint Word wie mehrere SDI-Anwendungen und verhält sich auch so.

Heute lernen Sie unter anderem, wie ...

- Visual C++ die Dokument-/View-Architektur für das Erstellen von SDI-Anwendungen verwendet,
- man das Gerüst einer SDI- oder MDI-Anwendung erstellt,
- man die Daten von der visuellen Repräsentation der Daten trennt,
- man die Daten in eigenen C++-Klassen abkapselt,
- man die Interaktion zwischen den Daten und den Menüs realisiert,
- man Nachrichten von mehreren Menübefehlen mithilfe einer anderen Methode als bisher an eine einzige Behandlungsfunktion sendet.

10.1 Die Dokument-/View-Architektur

Bei einer SDI-Anwendung werden mehr Klassen als bei einer dialogfeldbasierten Anwendung erstellt. Die Klassen realisieren jeweils bestimmte Funktionsweisen einer SDI- Anwendung. Wenn man einmal von der Dialogfeldklasse About absieht, besteht eine SDI- Anwendung aus vier speziellen Klassen:

- der von CWinApp abgeleiteten Klasse,
- der von CFrameWnd abgeleiteten Klasse,
- der von CDocument abgeleiteten Klasse,
- der von CView abgeleiteten Klasse.

Die Klasse CWinApp erzeugt alle anderen Komponenten in der Anwendung. Diese Klasse empfängt alle Ereignisnachrichten und übergibt dann die Nachrichten an die Klassen CFrameWnd und CView.

Die Klasse CFrameWnd bildet den Fensterrahmen. Sie nimmt die Menüs, Symbolleisten, Bildlaufleisten und alle anderen sichtbaren Objekte auf, die mit dem Rahmen verbunden sind. Die Klasse bestimmt, wie viel vom Dokument zu einem bestimmten Zeitpunkt zu sehen ist. Bei SDI-Anwendungen brauchen Sie an den beiden ersten Klassen nur wenige (oder überhaupt keine) Programmänderungen vorzunehmen.

Die Klasse CDocument nimmt Ihr Dokument auf. In dieser Klasse erstellen Sie die erforderlichen Datenstrukturen, um die Daten des Dokuments zu speichern und zu verwalten. Die Klasse empfängt die Eingaben von der Klasse CView und übergibt Anzeigeinformationen an die Klasse CView. Außerdem ist sie für das Speichern und Abrufen der Dokumentdaten in bzw. aus Dateien verantwortlich.

Die Klasse CView ist die Klasse, die die visuelle Repräsentation (View - Ansicht) des Dokuments für den Benutzer anzeigt. Die Klasse reicht Eingabeinformationen an die Klasse CDocument weiter und empfängt Anzeigeinformationen aus der Klasse CDocument. Das meiste, was Sie in dieser Klasse realisieren, bezieht sich auf die Ausgabe des Dokuments für den Benutzer und die Behandlung der Benutzereingaben. Die Klasse CView hat mehrere abgeleitete Klassen, die man als Basis für die Ansichtsklasse verwenden kann. Diese abgeleiteten Klassen sind in Tabelle 10.1 aufgeführt.

Klasse	**Beschreibung**
CEditView	Liefert die Funktionalität eines Eingabefelds. Mit dieser Klasse lassen sich einfache Text-Editoren implementieren.
CFormView	Die Basisklasse für Ansichten, die Steuerelemente enthalten. Mit dieser Klasse lassen sich formularbasierte Dokumente in Anwendungen bereitstellen.
CHtmlView	Liefert die Funktionalität eines Webbrowsers. Diese Ansicht behandelt direkt die URL-Navigation, Hyperlinks usw. verwaltet eine Verlaufsliste für das Durchsuchen in Rückwärts- und Vorwärtsrichtung.
CListView	Stellt die Funktionalität von Listen in der Dokument-/View-Architektur bereit
CRichEditView	Realisiert Zeichen- und Absatzformatierungen. Mit dieser Klasse lassen sich Textverarbeitungen implementieren.
CScrollView	Stellt Bildlauffähigkeiten für eine CView-Klasse bereit
CTreeView	Stellt die Funktionalität zur Steuerung von Bäumen in der Dokument-/View-Architektur bereit

Tabelle 10.1: Von CView abgeleitete Klassen

Alle vier Klassen arbeiten zusammen, um die vollständige Funktionalität einer SDI- Anwendung bereitzustellen, wie es Abbildung 10.1 zeigt. Auf der Grundlage dieser Architektur lassen sich relativ einfach leistungsfähige dokumentbezogene Anwendungen aufbauen.

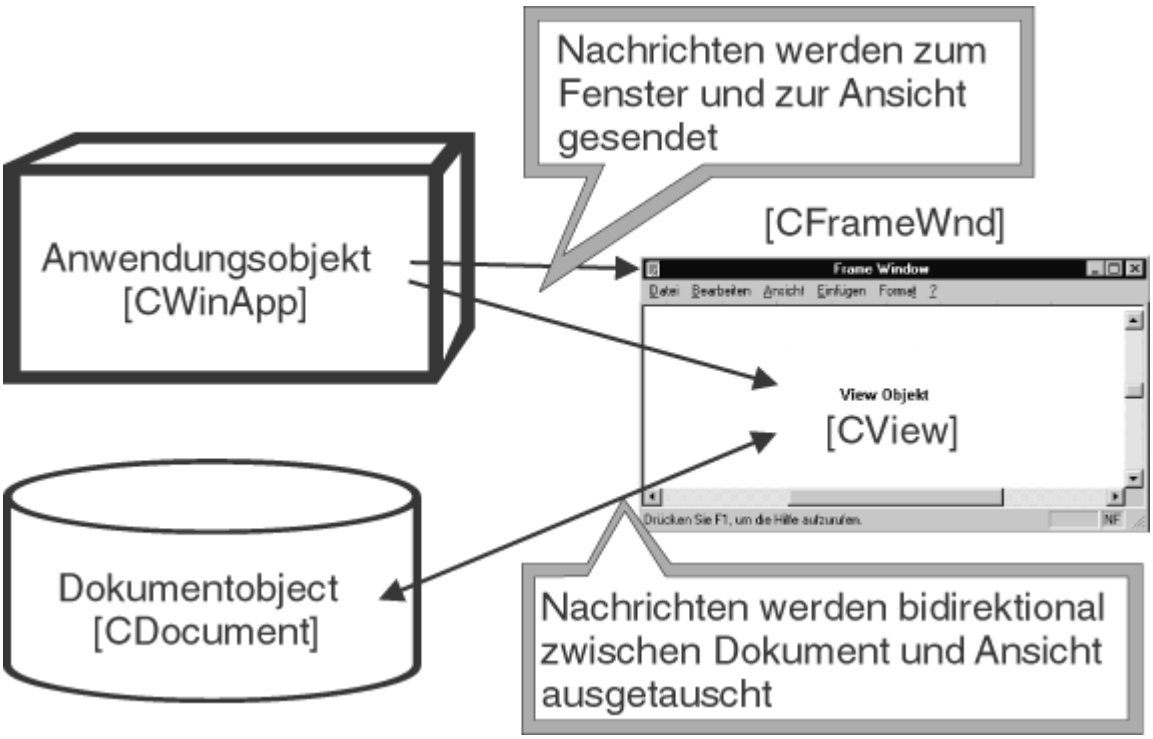


Abbildung 10.1: Die Dokument-/View-Architektur

Lassen Sie sich vom Begriff Dokument nicht täuschen. Es bedeutet nicht, dass Sie nur Anwendungen wie Textverarbeitungen und Tabellenkalkulationen erstellen können. Im weiteren Sinne bezeichnet man mit Dokument die Daten, die Ihre Anwendung verarbeitet, während sich die Ansicht auf die visuelle Repräsentation der Daten bezieht. Beispielsweise könnte man die Anwendung Solitär als Dokument-/View-Anwendung erstellen, wobei das Dokument die Karten und deren Lage im Spielfeld darstellt. In diesem Fall handelt es sich bei der Ansicht um die Anzeige der Karten, wobei jede Karte dort erscheint, wo es das Dokument festgelegt hat.

MFC-Exkurs: Die Klasse CWinApp

Die Klasse CWinApp ist die Basisklasse für die Funktionalität beim Starten einer Anwendung. Sie wurde am ersten Tag, »Erste Schritte mit Visual C++«, erwähnt und Sie haben sie bisher in jeder Ihrer Anwendungen gesehen. Sie können eine Reihe globaler Funktionen verwenden, um mit dem CWinApp-Objekt zu interagieren, und die Klasse CWinApp enthält mehrere Funktionen, die möglicherweise die in Ihrer Anwendung benötigte Funktionalität bereitstellen.

Globale CWinApp-bezogene Funktionen

Für die Interaktion mit dem Anwendungsobjekt können vier globale Funktionen verwendet werden:

- AfxGetApp haben Sie in den Beispielen der vergangenen Tage bereits gesehen und verwendet. Diese Funktion gibt einen Zeiger auf das Anwendungsobjekt der CWinApp- Klasse zurück. In einer MFC-Anwendung können Sie nicht direkt auf die Klasse CWinApp zugreifen. Sie müssen die Funktion AfxGetApp verwenden, um auf die benötigte CWinApp-Funktionalität zuzugreifen.
- AfxGetInstanceHandle gibt den Handle der Anwendung zurück. Dieser Handle kann in Situationen verwendet werden, in denen Sie den Anwendungs-Handle an eine API- Funktion übergeben müssen, von der er benötigt wird.
- AfxGetResourceHandle gibt den Instanz-Handle für die Anwendungsressourcen zurück.
- AfxGetAppName gibt einen nullterminierten String zurück, der den Namen der Anwendung enthält.

Keine dieser Funktionen übernimmt Parameter; sie können von jedem Ort in Ihrer Anwendung aufgerufen werden.

Funktionen für den Zugriff auf das Anwendungsprofil

Oft müssen Sie Konfigurationsinformationen über Ihre Anwendung speichern, um sie beim nächsten Start der Anwendung wiederherstellen zu können. Manchmal werden die Informationen unter Windows 95/98/ME in .ini-Dateien gespeichert, während sie sich unter Windows NT/2000 fast immer in der Registry befinden.

Die Klasse CWinApp enthält die zum Schreiben und Lesen von Informationen an diesen Speicherorten für Konfigurationen benötigte Funktionalität. Diese Funktionen können zur Wartung der Konfigurationseinstellungen in der Registry verwendet werden.

Verwenden Sie die Funktionen GetProfileString und GetProfileInt, um Konfigurationsinformationen zu lesen. Diese besitzen folgende Syntax:

```
CString GetProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry,
    LPCTSTR lpszDefault = NULL);
UINT GetProfileInt(LPCTSTR lpszSection, LPCTSTR lpszEntry,
    int iDefault);
```

Der erste Parameter beider Funktionen, lpszSection, ist der Konfigurationsabschnitt oder Unterschlüssel für die mit diesen Funktionen zu lesende Konfigurationseinstellung. Der zweite Parameter, lpszEntry, ist der Name des aus den Konfigurationseinstellungen zu lesenden Eintrags. Dieser Eintragsname muss innerhalb des Abschnitts eindeutig sein. Der dritte Parameter, lpszDefault beziehungsweise iDefault, ist der

Standardwert, den diese Funktionen zurückgeben, wenn die Konfigurationseinstellung nicht in der Registry bzw. in der .ini-Datei gefunden wird. Wird an die Funktion GetProfileString der dritte Parameter nicht übergeben, gibt sie einen leeren String zurück.

Um Konfigurationsinformationen zu schreiben, verwenden Sie die Funktionen WriteProfileString und WriteProfileInt. Diese beiden Funktionen schreiben die Konfigurationseinstellungen in die Registry bzw. in die .ini-Datei. Ihre Syntax lautet wie folgt:

```
BOOL WriteProfileString(LPCTSTR lpszSection, LPCTSTR lpszEntry,
    LPCTSTR lpszValue);
BOOL WriteProfileInt(LPCTSTR lpszSection, LPCTSTR lpszEntry,
    int iValue);
```

Die ersten beiden Parameter, lpszSection und lpszEntry, sind die gleichen wie in den Lesefunktionen. Der erste Parameter beider Funktionen, lpszSection, ist die Konfigurationseinstellung oder der Unterschlüssel für die von diesen Funktionen zu schreibende Konfigurationseinstellung. Der zweite Parameter, lpszEntry, ist der Name des Eintrags, der in die Konfigurationseinstellungen geschrieben werden soll. Der dritte Parameter, lpszValue und iValue, ist der in die Registry zu schreibende Wert.

Bevor Sie damit anfangen, in der Registry zu lesen und zu schreiben, müssen Sie beim Start Ihrer Anwendung ein Detail beachten. Im Anwendungsobjekt, der von CWinApp abgeleiteten Klasse (gewöhnlich als C<Ihr Anwendungsname>App bezeichnet), müssen Sie die Funktion SetRegistryKey in der Funktion InitInstance suchen. Ändern Sie den an SetRegistryKey übergebenen String auf den Namen, unter dem Ihr Anwendungsabschnitt abgelegt werden soll. Die Voreinstellung für diesen Abschnitt sieht wie Listing 10.1 aus.

Listing 10.1: Eine typische InitInstance-Funktion

```
 1: BOOL CSDISquigApp::InitInstance()
 2: {
 3:    CWinApp::InitInstance();
   ...
 4:    // Standardinitialisierung
 5:    // Wenn Sie diese Features nicht verwenden und die Größe
 6:    // der ausführbaren Datei verringern möchten, entfernen Sie
 7:    // die nicht erforderlichen Initialisierungsroutinen.
 8:    // Ändern Sie den Registrierungsschlüssel unter dem Ihre
 9:    // Einstellungen gespeichert sind.
10:    // TODO: Ändern Sie diese Zeichenfolge entsprechend,
11:    // z.B. zum Namen Ihrer Firma oder Organisation.
12:    SetRegistryKey(_T("Vom lokalen Anwendungs-Assistenten
       generierte Anwendungen"));
13:    LoadStdProfileSettings(4);  // Standard INI-Dateioptionen
15:                                // laden (einschließlich MRU)
   ...
```

Gewöhnlich ändern Sie den an die Funktion SetRegistryKey übergebenen String von "Vom lokalen Anwendungs-Assistenten generierte Anwendungen" in den Namen Ihrer Firma um. Der Schlüssel wird in der Registry unterhalb HKEY_CURRENT_USER/Software/ angelegt. Dieser Schlüssel enthält einen Unterschlüssel, den Namen der Anwendung, gefolgt von allen in den vorhin besprochenen Schreib- und Lesefunktionen verwendeten Abschnittsschlüsseln. Wenn Sie den String beispielsweise wie folgt auf "MeineFirma" abgeändert haben:

```
SetRegistryKey(_T("MeineFirma"));
```

und Ihre Anwendung den Namen MeineApplikation trägt, werden alle Konfigurationseinstellungen für Ihre Anwendung unter folgendem Registry-Schlüssel untergebracht:

```
HKEY_CURRENT_USER/Software/MeineFirma/MeineApplikation/
```

Symbol- und Cursorfunktionen

Neben den Konfigurationsfunktionen könnten Sie noch einen weiteren Satz Funktionen der Klasse CWinApp benötigen: die Funktionen für das Laden von Cursor und Symbolen. Sie haben am vierten Tag, »Maus und Tastatur«, die Funktion LoadStandardCursor verwendet. Die Funktionen LoadCursor und LoadIcon sind im Grunde nicht anders. Diese beiden Funktionen besitzen folgende Syntax:

```
HCURSOR LoadCursor(UINT nIDResource);
HICON LoadIcon(UINT nIDResource);
```

In beiden Funktionen übergeben Sie die ID der Symbol- oder Cursor-Ressource, die Sie in Ihre Anwendung hinein kompiliert haben (alle, die Sie als Teil Ihrer Anwendung erstellt haben, sind in die Anwendung einkompiliert). Beide Funktionen geben ein Handle auf den geladenen Cursor oder das geladene Symbol zurück.

MFC-Exkurs: Die Klasse CFrameWnd

Die Klasse CFrameWnd ist die Basisklasse, von der sich der Hauptfensterrahmen in SDI- Anwendungen ableitet. Sie stellt der Anwendung zwar eine beachtliche Menge wichtiger Funktionalität zur Verfügung, doch Sie werden diese Klasse bei der Erstellung der meisten Anwendungen eher weniger verwenden.

Die Funktionalität, die Sie in dieser Klasse am wahrscheinlichsten verwenden werden, leitet sich eigentlich aus Basisklassen ab. Es handelt sich dabei um GetWindowPlacement und SetWindowPlacement. Diese Funktionen können verwendet werden, um die Position und Größe eines Fensters zum Speichern in der Konfiguration der Anwendung abzufragen und um den Zustand des Fensters beim nächsten Start der Anwendung wiederherzustellen. Beide Funktionen übernehmen einen Zeiger auf eine WINDOWPLACEMENT-Struktur, die wie folgt definiert ist:

```
typedef struct tagWINDOWPLACEMENT {      /* wndpl */
    UINT  length;
    UINT  flags;
    UINT  showCmd;
    POINT ptMinPosition;
    POINT ptMaxPosition;
    RECT  rcNormalPosition;
} WINDOWPLACEMENT;
```

Das erste Element dieser Struktur, length, ist die Länge der Struktur selbst in Bytes. Das zweite Element, flags, enthält Informationen über die Optionen für das Minimieren und Maximieren des Fensters (siehe Tabelle 10.2). Das dritte Element, showCmd, legt den Zustand des Fensters fest. In Tabelle 10.3 sind die möglichen Werte dieses Elements aufgelistet. Das vierte Element, ptMinPosition, ist die Position der linken oberen Ecke des minimierten Fensters. Das fünfte Element, ptMaxPosition, ist die Position der linken oberen Ecke des maximierten Fensters. Das letzte Element, rcNormalPosition, gibt die Position und Größe des Fensters im normalen Modus an. Sie können wie folgt den aktuellen Zustand des Fensters abfragen:

```
WINDOWPLACEMENT wndpl;
wndpl.length = sizeof(WINDOWPLACEMENT);
// ermittelt aktuelle Position und Minimiert/Maximiert-Status
BOOL bRet = GetWindowPlacement(&wndpl);
```

Dann können Sie den Zustand des Fensters wie folgt wiederherstellen:

```
// setzt Position und Minimiert/Maximiert-Status des Fensters
BOOL bRet = SetWindowPlacement(&wndpl);
```

Indem Sie die Daten der WINDOWPLACEMENT-Struktur in der Registry speichern und beim nächsten Start der Anwendung wiederherstellen, können Sie die Größe und Position der Anwendungsfenster erhalten.

Option	Beschreibung
WPF_SETMINPOSITION	Gibt an, dass die X- und Y-Positionen des Fensters für Minimierung gesetzt werden können

WPF_RESTORETOMAXIMIZED	Gibt an, dass das Fenster im maximierten Zustand wiederhergestellt wird, unabhängig von seinem aktuellen Zustand
WPF_ASYNCWINDOWPLACEMENT	Gibt an, dass bei mehreren laufenden Threads die Nachricht an den Thread übergeben wird, der das Fenster besitzt. Andere Threads werden daran gehindert, diesen Thread am Setzen des Fensterzustands zu hindern. Diese Option ist nur unter Windows 2000 und später verfügbar.

Tabelle 10.2: Flag-Optionen für WINDOWPLACEMENT

Befehl	Beschreibung
SW_HIDE	Verbirgt das Fenster
SW_MAXIMIZE	Maximiert das Fenster
SW_MINIMIZE	Minimiert das Fenster
SW_RESTORE	Aktiviert und zeigt das Fenster an seiner normalen Position (entspricht SW_SHOWNORMAL)
SW_SHOW	Aktiviert und zeigt das Fenster in seiner aktuellen Größe und Position
SW_SHOWMAXIMIZED	Aktiviert und maximiert das Fenster
SW_SHOWMINIMIZED	Aktiviert und minimiert das Fenster
SW_SHOWMINNOACTIVE	Zeigt das Fenster im minimierten Zustand an. Das Fenster wird nicht aktiviert.

Tabelle 10.3: Die cmdShow-Optionen

Befehl	Beschreibung
SW_SHOWNA	Zeigt das Fenster in seiner aktuellen Größe und Position an. Das Fenster wird nicht aktiviert.
SW_SHOWNOACTIVATE	Zeigt das Fenster in seiner vorherigen Größe und Position an. Das Fenster wird nicht aktiviert.
SW_SHOWNORMAL	Aktiviert das Fenster und zeigt es in seiner normalen Größe und Position (nicht minimiert oder maximiert)

Tabelle 10.3: Die cmdShow-Optionen (Forts.)

C++-Exkurs: Die Funktion sizeof

Die Funktion sizeof kann aufgerufen werden, um die Größe einer Struktur, einer Klasse oder eines Datentyps in Bytes zu ermitteln. Gewöhnlich wird sie in Situationen gebraucht, in denen man die Größe einer Struktur oder eines Datentyps wissen muss, wie bei Speicherzuweisungen oder um Größenparameter oder Strukturelemente zu füllen. Der Vorteil dieser Funktion ist, dass Sie sich selbst keine Gedanken um die Größe einer Struktur machen müssen. Das ist wichtig, wenn eine Struktur möglicherweise ihre Größe ändert. Beispielsweise haben einige der Standard-Strukturen von Windows unterschiedliche Größen, je nachdem, auf welcher Version von Windows sie laufen. Wenn Sie die Funktion sizeof verwenden, müssen Sie sich nicht mit Details wie der Windows-Version aufhalten und wie sie die Größe einzelner Strukturen beeinflusst.

MFC-Exkurs: Die Klasse CDocument

Die Klasse CDocument ist die Basisklasse für die Daten in Ihrem Dokument. Das Dokument besteht aus den in Ihrer Anwendung verwendeten Daten, unabhängig von der Form, die diese Daten annehmen. Sie sind

natürlich nicht darauf beschränkt, Textverarbeitungen oder Tabellenkalkulationen zu erstellen. Es ist eher ein Mittel, um die Daten von der Anzeige zu unterscheiden. So wird erwartet, dass Sie den größten Teil der Funktionalität in Ihrem Abkömmling der Dokumentklasse erstellen und aus der Klasse CDocument nur unterstützende Funktionalität benötigen.

Die meisten Funktionen, mit denen Sie in der Klasse CDocument arbeiten werden (siehe Tabelle 10.4), sind die Funktionen, die Sie für Ihre eigenen Zwecke überschreiben.

Funktion	Beschreibung
DeleteContents	Wird aufgerufen, um den aktuellen Inhalt des Dokuments zu löschen
OnCloseDocument	Wird beim Schließen des Dokuments aufgerufen
OnNewDocument	Wird bei der Erstellung eines neuen Dokuments aufgerufen, um dieses zu initialisieren
OnOpenDocument	Wird beim Öffnen eines existierenden Dokuments aufgerufen
OnSaveDocument	Wird beim Speichern des geöffneten Dokuments aufgerufen

Tabelle 10.4: Die überschreibbaren CDocument-Funktionen

Wenn die Dokumentdaten geändert werden, sollten Sie die Funktion SetModifiedFlag aufrufen, um die Daten als »schmutzig« (dirty) zu kennzeichnen. Das löst die verkapselte Funktionalität aus, die den Benutzer fragt, ob er das Dokument vor dem Schließen speichern möchte.

MFC-Exkurs: Die CView-Klassen

Die Klasse CView ist mit der Klasse CDocument insofern verknüpft, als sie implementiert, wie die Daten in der CDocument-Klasse angezeigt werden. Sie benötigen für jede Instanz der Klasse CDocument in Ihrer laufenden Anwendung mehrere Instanzen der Klasse CView. Wie bei CDocument fügen Sie den größten Teil der Funktionalität in der von CView abgeleiteten Klasse in Ihrer Anwendung ein. Die Funktionalität wird meist von Maus- oder Tastaturereignissen oder von den überschreibbaren Funktionen in der Klasse CView ausgelöst (in Tabelle 10.5 sind diese überschreibbaren Funktionen aufgelistet). Mehrere Funktionen besitzen bereits Funktionsrahmen in der vom Anwendungs-Assistent für Ihre Anwendung erzeugten Ansichtsklasse.

Funktion	Beschreibung
OnDraw	Wird aufgerufen, wenn die Daten im Dokument ins Fenster gezeichnet werden sollen
OnBeginPrinting	Wird bei Druckbeginn aufgerufen. Hier führen Sie alle speziellen Aktionen aus, die vor dem Drucken erledigt werden müssen.
OnEndPrinting	Wird bei Druckende aufgerufen. Hier führen Sie alle nach dem Drucken notwendigen Aufräumarbeiten aus.

Tabelle 10.5: Die überschreibbaren CView-Funktionen

Funktion	Beschreibung
OnPrint	Wird beim Drucken oder beim Betrachten der Druckvorschau des aktuellen Dokuments aufgerufen
OnEndPrintPreview	Wird beim Verlassen der Druckvorschau aufgerufen
OnPreparePrinting	Wird vor dem Drucken aufgerufen. Hier initialisieren Sie den Druckdialog, wenn Ihr Dokument eine besondere Druckkonfiguration benötigt.
OnUpdate	Wird aufgerufen, um der View mitzuteilen, dass der Inhalt des Dokuments aktualisiert wurde

Tabelle 10.5: Die überschreibbaren CView-Funktionen (Forts.)

10.2 Was sind MDI-Anwendungen?

In Bezug auf die Programmierung mit Visual C++ gibt es kaum einen Unterschied zwischen SDI- und MDI-Anwendungen. Erst wenn man tiefer in die beiden Anwendungsstile einsteigt, treten die Unterschiede zutage. Eine SDI-Anwendung erlaubt dem Benutzer lediglich, an einem Dokument zu einem bestimmten Zeitpunkt zu arbeiten und lässt dabei nur einen einzigen Dokumenttyp zu. MDI-Anwendungen erlauben dagegen dem Benutzer nicht nur, an mehreren Dokumenten gleichzeitig zu arbeiten, sondern bieten auch die Möglichkeit, mehrere Arten von Dokumenten zu behandeln.

Eine MDI-Anwendung ist in einer Art Fenster-in-Fenster aufgebaut, wobei ein Rahmenfenster ein oder mehrere untergeordnete Fenster umschließt. Diesen Anwendungsstil findet man in vielen bekannten Softwarepaketen wie zum Beispiel in älteren Versionen von Word und Excel.

Die Architektur einer MDI-Anwendung entspricht weitgehend der einer SDI-Anwendung. Praktisch besteht der einzige Unterschied bei einer einfachen MDI-Anwendung nur darin, dass der Anwendungs-Assistent neben den üblichen Klassen noch eine zweite Rahmenklasse erzeugt, wie es Abbildung 10.2 zeigt. Der Abbildung können Sie auch entnehmen, dass die Herangehensweise bei der Entwicklung von MDI-Anwendungen der für SDI-Anwendungen sehr ähnlich ist.

Für eine MDI-Anwendung erstellen Sie lediglich eine Klasse mehr als bei einer SDI- Anwendung. Zu diesen Klassen gehören die jeweils von:

- CWinApp
- CMDIFrameWnd
- CMDIChildWnd
- CDocument
- CView

abgeleitete Klasse.

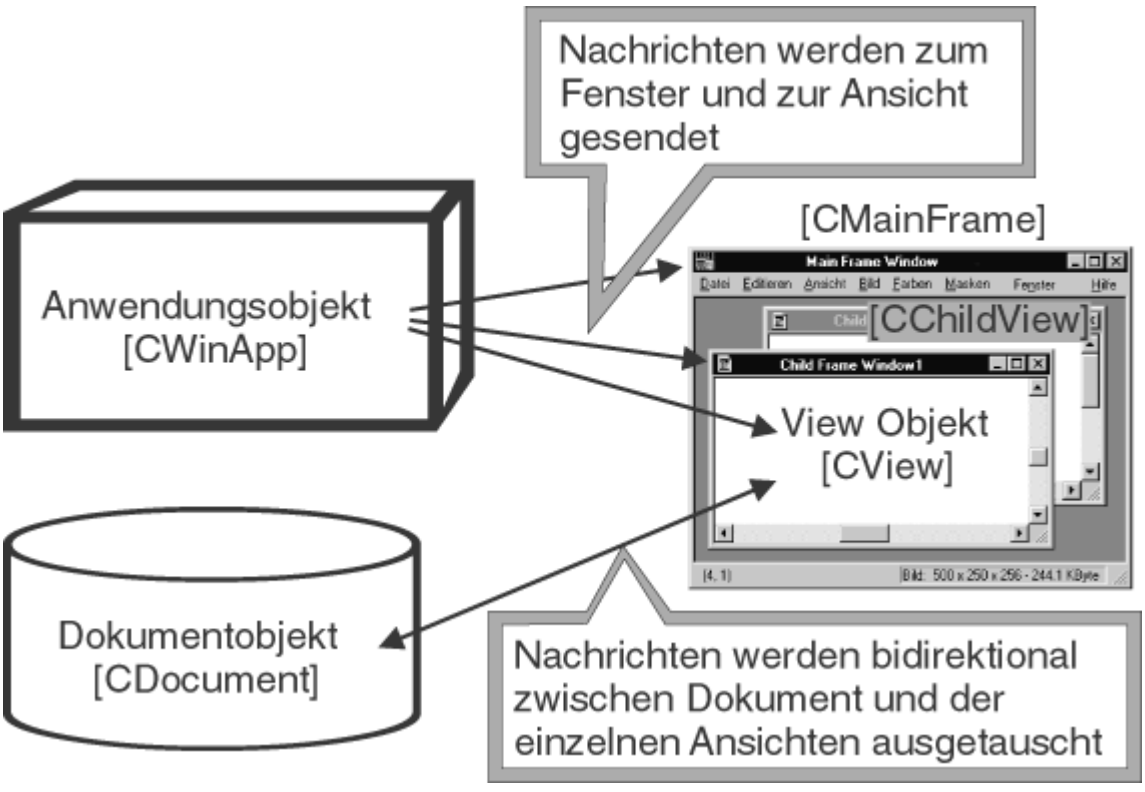


Abbildung 10.2: Die Dokument-/View-Architektur einer MDI-Anwendung

Die beiden MDI-spezifischen Klassen CMDIFrameWnd (die Klasse CMainFrame in Ihrem Projekt) und CMDIChildWnd (die Klasse CChildFrame in Ihrem Projekt) sind die beiden einzigen Klassen, die sich von den

SDI-Anwendungen unterscheiden.

- Die von CMDIFrameWnd abgeleitete Klasse CMainFrame stellt den Hauptrahmen der Anwendung dar und liefert einen abgeschlossenen Bereich auf dem Desktop, in dem die gesamte Interaktion der Anwendung stattfindet. Mit dem Rahmenfenster sind das Menü und die Symbolleisten verbunden.
- Die von CMDIChildWnd abgeleitete Klasse CChildFrame bildet den Rahmen, der die CView-Klassen aufnimmt. Dieser Rahmen leitet die Nachrichten und Ereignisse an die Ansichtsklasse zur Verarbeitung oder Anzeige weiter.

Gewissermaßen ist die Funktionalität der Rahmenklasse einer SDI-Anwendung in die beiden MDI-spezifischen Klassen einer MDI-Anwendung aufgeteilt worden. Gleichzeitig gibt es eine zusätzliche Unterstützung für die Ausführung mehrerer untergeordneter Rahmen in deren eigenen Dokument-/View-Klasseninstanzen.

MFC-Exkurs: Die Klassen CMDIFrameWnd und CMDIChildWnd

Die Klasse CMDIFrameWnd ist das äußere Rahmenfenster einer MDI-Anwendung. Diese Klasse ist ein Abkömmling der Klasse CFrameWnd und enthält daher die gleiche Funktionalität wie das SDI-Rahmenfenster. Eine Member-Funktion, die Sie in dieser Klasse verwenden werden, ist MDIGetActive, das einen Zeiger auf das momentan aktive untergeordnete Fenster zurückgibt. Die meisten anderen Funktionen, die mit dem untergeordneten Fenster arbeiten, sind in dem im MDI-Anwendungsrahmen vom Anwendungs-Assistent erstellten Fenstermenü enthalten.

Die Klasse CMDIChildWnd ist das innere Rahmenfenster einer MDI-Anwendung. Wie die Klasse CMDIFrameWnd ist auch sie ein Abkömmling der Klasse CFrameWnd. Die meisten spezialisierten Funktionen in dieser Klasse haben mit dem Zustand des untergeordneten Fensters im übergeordneten Rahmen zu tun: MDIDestroy, MDIActivate, MDIMaximize und MDIRestore. Keine dieser Funktionen übernimmt Parameter oder gibt Ergebnisse zurück. Die einzige weitere Member-Funktion, GetMDIFrame, gibt einen Zeiger auf das übergeordnete CMDIFrameWnd-Fenster zurück.

10.3 Eine SDI-Anwendung erstellen

Um eine Vorstellung von der Arbeitsweise der Dokument-/View-Architektur in einer realen Anwendung zu bekommen, erstellen Sie heute eine neue Version der Zeichenanwendung aus Lektion 4. In der neuen Version bleibt die Zeichnung des Benutzers dauerhaft erhalten. Das bedeutet, dass die Zeichnung nicht gelöscht wird, wenn ein anderes Fenster über der Anwendung liegt. Außerdem kann man jetzt die Zeichnungen speichern und wiederherstellen.

Das Anwendungsgerüst erstellen

Das Gerüst für die heutige Anwendung erstellen Sie in folgenden Schritten:

1. Legen Sie ein neues MFC-Anwendung Visual- C++-Projekt namens SDISquig an.

2. Im Bereich Anwendungstyp wählen Sie die Option Einfaches Dokument (siehe Abbildung 10.3).
3. Im Bereich Zeichenfolgen für Dokumentvorlagen geben Sie wie in Abbildung 10.4 eine Dateinamenserweiterung für die Dateien an, die Ihre Anwendung erzeugen wird (beispielsweise squiq). Da die meisten Kombinationen von drei Buchstaben bereits von einer (oder mehreren) Anwendungen verwendet werden sollten Sie versuchen eine möglichst »sprechende« Buchstabenkombination von mehr als drei Buchstaben zu verwenden.
4. Im Bereich Erstellte Klassen können Sie die Basisklasse auswählen, auf der Ihre Ansichtsklasse basieren soll (siehe Abbildung 10.5). Behalten Sie die Einstellung CView und klicken Sie auf Fertig stellen. Der MFC-Anwendungs-Assistent erzeugt das Anwendungsgerüst.

Abbildung 10.3: Den Anwendungstyp festlegen

Abbildung 10.4: Die Dateinamenserweiterung für Dokumente festlegen

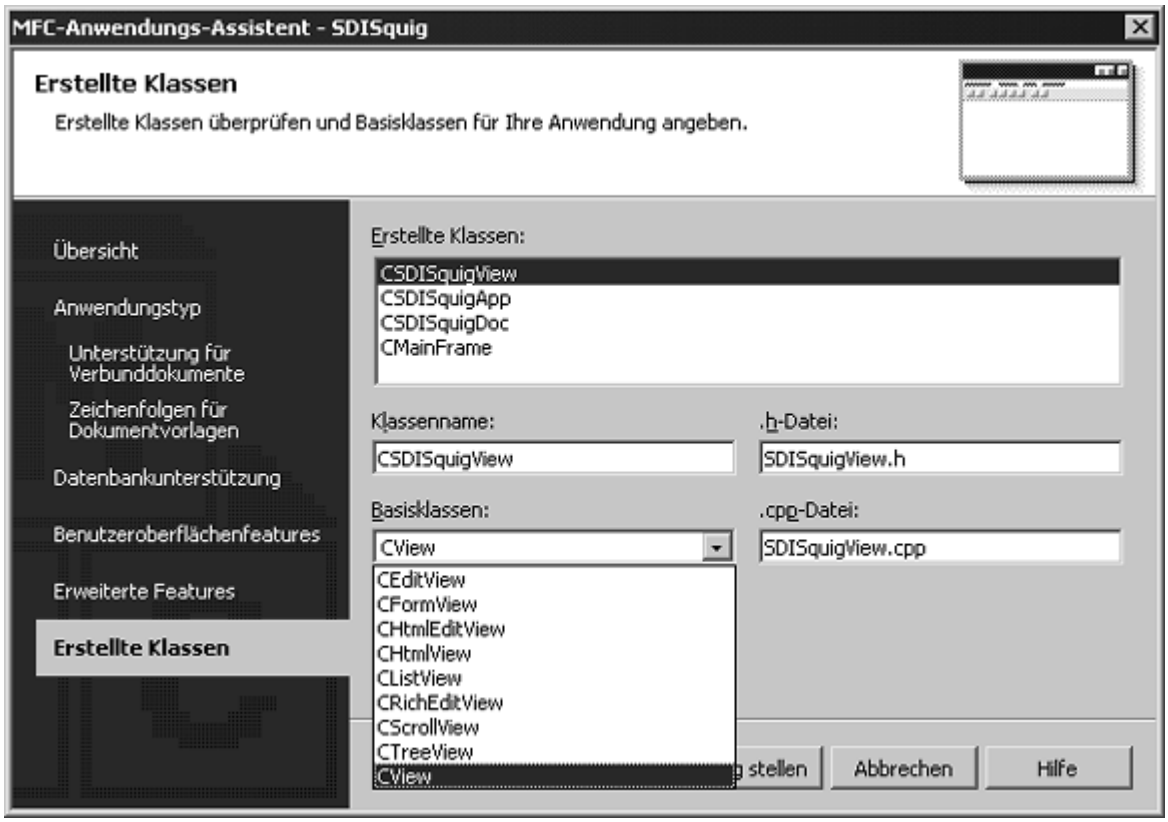


Abbildung 10.5: Die zu verwendende Ansichtsklasse festlegen

Eine Linienklasse erzeugen

Als Erstes beschäftigen wir uns mit der Frage, wie man die Daten in der Dokumentklasse darstellt. Bei jeder Zeichenanwendung hat man es mit einer Reihe von Linien zu tun. Jede Linie besteht aus einem Anfangs- und einem Endpunkt. Man könnte nun annehmen, dass man für die Datendarstellung eine Folge von Punkten verwendet. In diesem Fall müssen Sie auch spezielle Anpassungen über die Folge von Linien zwischen dem Endpunkt und dem nächsten Anfang vornehmen. Es ist sinnvoller, die Zeichnung als Folge von Linien zu repräsentieren. Damit kann man jede einzelne Linie, die im Fenster gezeichnet wird, speichern, ohne sich darum kümmern zu müssen, wo eine Gruppe von zusammenhängenden Punkten endet und die nächste beginnt.

Leider stellt die MFC-Bibliothek keine Klasse für Linienobjekte bereit, obwohl es eine Klasse für Punktobjekte (CPoint) gibt. Es bleibt Ihnen nichts weiter übrig, als Ihre eigene Linienklasse in folgenden Schritten zu erzeugen:

1. Markieren Sie in der Klassenansicht das oberste Objekt im Baum (SDISquig). Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Klasse hinzufügen aus dem Kontextmenü.

2. Im Dialogfeld Klasse hinzufügen wählen Sie als Klassentyp Allgemeine Klasse (siehe Abbildung 10.6). Klicken Sie auf Öffnen, um mit dem nächsten Schritt der Klassenerstellung fortzufahren.

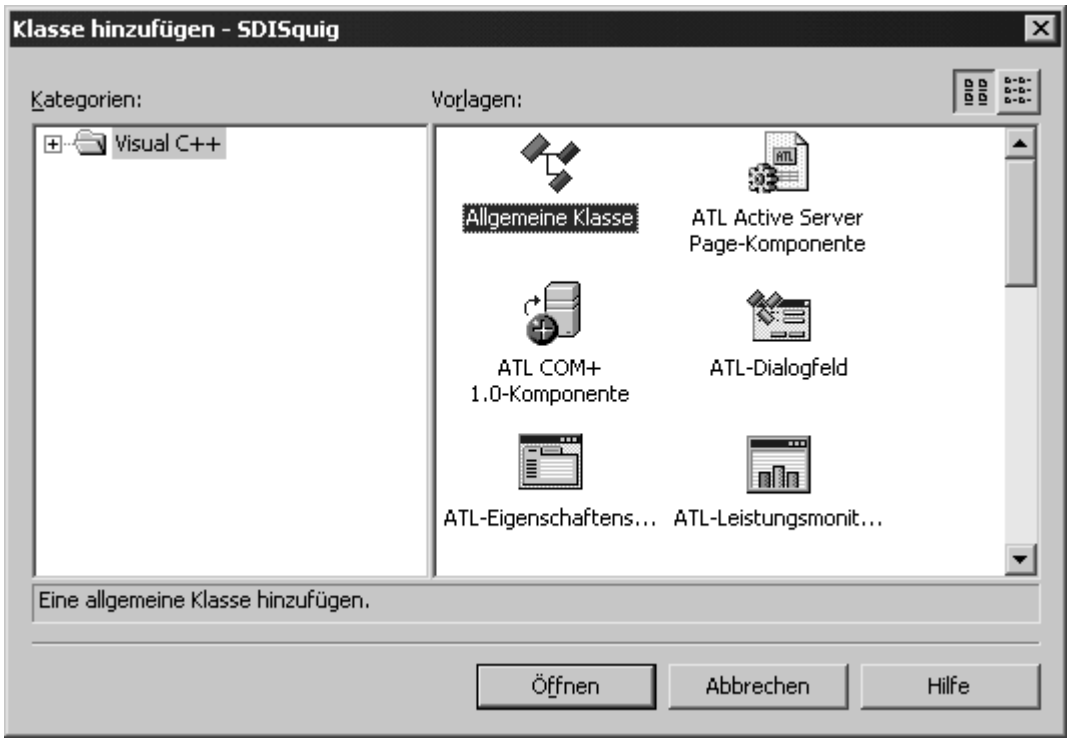


Abbildung 10.6: Das Dialogfeld Klasse hinzufügen

3. Geben Sie CLine als Namen für die Klasse und CObject als Basisklasse an. Behalten Sie für den Zugriff die Einstellung public (siehe Abbildung 10.7).

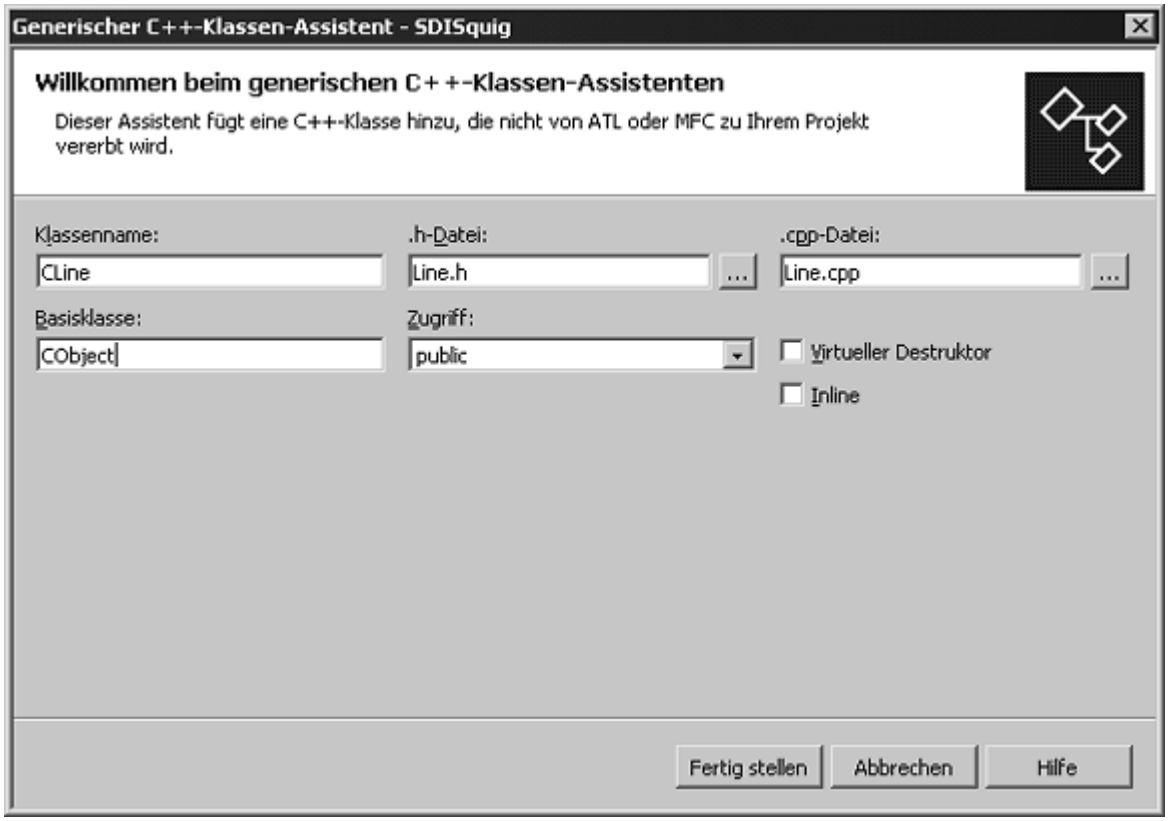


Abbildung 10.7: Der Generische C++-Klassen-Assistent

4. Klicken Sie auf Fertig stellen, um die Klasse CLine in Ihr Projekt einzufügen.

Die Klasse CLine konstruieren

Momentan muss Ihre CLine-Klasse nur zwei Datenelemente aufnehmen und zwar die beiden Endpunkte der Linie, die sie repräsentiert. Fügen Sie diese beiden Datenelemente hinzu und nehmen Sie auch einen Klassenkonstruktor auf, der beide Werte setzt, wenn die Instanz der Klasse erzeugt wird. Führen Sie dazu folgende Schritte aus:

1. Markieren Sie in der Klassenansicht die Klasse CLine.

2. Klicken Sie mit der rechten Maustaste auf die Klasse CLine und wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.
3. Geben Sie CPoint als Variablentyp und m_ptFrom als Variablenname ein. Als Zugriff wählen Sie die Option private. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.
4. Wiederholen Sie die Schritte 2 und 3, nennen Sie die Variable diesmal m_ptTo.
5. Klicken Sie mit der rechten Maustaste auf die Klasse CLine und wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.
6. Geben Sie als Funktionstyp void und als Funktionsnamen CLine an. Fügen Sie zwei Parameter ein. Beide Parameter sind vom Typ CPoint, ihre Namen sind ptFrom und ptTo. Klicken Sie auf Fertig stellen, um die Funktion hinzuzufügen.
7. In die neue Funktion übernehmen Sie den Code aus Listing 10.2.

Listing 10.2: Der Konstruktor von CLine

```
1: CLine::CLine(CPoint ptFrom,  CPoint ptTo)
2: {
3:    // Anfangs- und Endpunkte initialisieren
4:    m_ptFrom = ptFrom;
5:    m_ptTo = ptTo;
6: }
```

Dieser Objektkonstruktor initialisiert die Anfangs- und Endpunkte mit den Punkten, die man an den Konstruktor übergibt.

In den anderen Funktionen, die Sie hinzugefügt haben, stand der Rückgabetyp vor dem Funktionsnamen. In diesem Fall wurde der Rückgabetyp void nicht in die Zeile mit der Konstruktordefinition aufgenommen. Der Rückgabetyp void wurde nicht eigens aufgenommen, weil es sich um einen Klassenkonstruktor handelt. Bei Klassenkonstruktoren und -destruktoren wird niemals ein Rückgabetyp angegeben.

Die Klasse CLine zeichnen

Nach den Regeln des objektorientierten Entwurfs sollte Ihre Klasse CLine in der Lage sein, sich selbst zu zeichnen. Wenn die Ansichtsklasse die Linie anzeigen soll, kann sie dann mit einer Nachricht das Linienobjekt anweisen, sich selbst neu zu zeichnen. Diese Funktionalität realisieren Sie in folgenden Schritten:

1. Wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü, um eine neue Funktion in die Klasse CLine einzufügen.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen als Draw fest. Fügen Sie einen Parameter von Typ CDC* mit dem Namen pDC ein. Legen Sie den Zugriff als public fest.
3. Schreiben Sie den Code aus Listing 10.3 in die gerade erstellte Funktion Draw.

Listing 10.3: Die Funktion Draw der Klasse CLine.

```
1: void CLine::Draw(CDC* pDC)
2: {
3:    // Linie zeichnen
```

```
4:    pDC->MoveTo(m_ptFrom);
5:    pDC->LineTo(m_ptTo);
6: }
```

Diese Funktion ist fast identisch mit der, die Sie an Tag 4 erstellt haben. Es handelt sich um eine einfache Funktion, die eine Bewegung zum ersten Punkt im Gerätekontext ausführt und dann eine Linie zum zweiten Punkt im Gerätekontext zeichnet.

Die Funktionalität des Dokuments implementieren

Nachdem Sie nun über ein Objekt verfügen, mit dem sich die vom Benutzer ausgeführten Zeichnungen darstellen lassen, können Sie diese CLine-Objekte im Dokumentobjekt in einem einfachen dynamischen Array speichern. Für dieses Array fügen Sie eine CObArray- Elementvariable in die Dokumentklasse ein.

Die Klasse CObArray ist eine Objekt-Array-Klasse, die sich selbst dynamisch in der Größe an die Anzahl der eingefügten Elemente anpasst. Die Klasse kann beliebige Objekte aufnehmen, die von der Klasse CObject abgeleitet sind. Die Anzahl der Objekte ist lediglich durch die Größe des Speichers begrenzt.

Fügen Sie das CObArray in CSDISquidDoc mit dem Assistenten zum Hinzufügen von Member-Variablen ein und benennen Sie das Objekt mit m_oaLines. Legen Sie den Zugriff als private fest.

MFC-Exkurs: Die Array-Klassen

Die MFC-Bibliothek enthält mehrere Array-Klassen für verschiedene Datentypen. Im Großen und Ganzen gleichen sich die Funktionsweisen dieser Klassen; sie besitzen die gleiche Funktionalität. Der Hauptunterschied zwischen ihnen ist der Datentyp, den sie jeweils enthalten können. In Tabelle 10.6 sind die Array-Klassen aufgelistet.

Klasse	Beschreibung
CArray	Eine Vorlagenklasse für die Erstellung eigener Array-Klassen
CByteArray	Ein Array von Byte-Datentypen
CDWordArray	Ein Array von DWORD-Datentypen
CObArray	Ein Array mit von CObject abgeleiteten Datentypen
CPtrArray	Ein Array von Zeigern (auf ein beliebiges Objekt oder einen beliebigen Datentyp)
CStringArray	Ein Array von CString-Objekten
CUIntArray	Ein Array von UINT-Datentypen (vorzeichenlose Zahlen vom Typ Integer)
CWordArray	Ein Array von WORD-Datentypen

Tabelle 10.6: Die Array-Klassen

Array-Grenzen

Sie können mehrere Funktionen verwenden, um festzustellen, wie viele Objekte sich momentan in einem Array befinden. Die am nächsten liegende Funktion ist GetCount, das die Anzahl der Elemente im Array zurückgibt. Dabei handelt es sich um eine neue Funktion in den Array-Klassen. Vor der neuesten Version von MFC fragte man die Anzahl der Elemente in einem Array mit GetSize ab. Sie können GetSize, dessen Funktionalität der von GetCount entspricht, noch immer verwenden.

Eine dritte Funktion, mit der man die Anzahl der Elemente in einem Array ermitteln kann, ist GetUpperBound. Diese Funktion unterscheidet sich geringfügig von GetCount und GetSize, da sie die Nummer der Endposition im Array zurückgibt und nicht die Anzahl der Elemente. Enthält das Feld keine Elemente, so liefert GetUpperBound den Wert -1 zurück.

Denken Sie daran, dass man in C/C++ immer bei 0 anfängt zu zählen, nicht bei 1. Wenn Sie also eine

Schleife erzeugen wollen, die jedes Element des Arrays bearbeitet, können Sie die Funktion GetCount folgendermaßen verwenden:

```
int iMax = myArray.GetCount();
for (int i = 0; i < iMax; i++)
{
  // Hier etwas tun
}
```

Die gleiche Schleife mit der Funktion GetUpper würde dagegen so aussehen:

```
int iMax = myArray.GetUpperBound();
for (int i = 0; i <= iMax; i++)
{
  // Hier etwas tun
}
```

Beachten Sie, dass Sie bei der zweiten Version die Schleife durchlaufen, solange i kleiner als oder gleich iMax ist, während Sie die Schleife bei der ersten Version nur durchlaufen, solange i kleiner als iMax ist. Der Grund dafür ist, dass die Funktion GetUpperBound die Array-Position des letzten Elements zurückgibt. Diese trägt die Nummer, die um eins kleiner ist als die Anzahl der Elemente im Array, da die Zählung der Array-Positionen bei 0 beginnt.

Die Funktion SetSize kontrolliert die Größe und die Methode der Vergrößerung des Arrays. Sie hat folgende Syntax:

```
void SetSize(int iNewSize, int iGrowBy = -1 );
```

Der erste Parameter der Funktion SetSize, iNewSize, ist die Anzahl der Elemente, die das Array zu diesem Zeitpunkt enthalten soll. Enthält das Array mehr Elemente, wird es abgeschnitten, sodass die überzähligen Elemente verloren gehen. Der zweite Parameter, iGrowBy, gibt an, wie viele Elemente gleichzeitig hinzugefügt werden sollen, wenn das Array mehr Elemente erhält, als es derzeit halten kann. Diese Funktion beschränkt das Array nicht auf die angegebene Elementzahl (falls Sie den Parameter iGrowBy nicht mit 0 angeben), sondern legt fest, wie neue Elemente bei der Vergrößerung des Arrays zusätzlich im Speicher alloziert werden. Wird der Wert weggelassen oder auf -1 gesetzt, so verwendet die Klasse eine optimierte Methode, um möglichst die Fragmentierung (Zerstückelung) des Speichers zu minimieren.

Array-Elementklassen

Man kann auf zwei verschiedene Arten auf Elemente in einem Array zugreifen. Sie können die Funktion GetAt verwenden, der Sie die Positionsnummer des Elements übergeben und die Ihnen das Objekt an dieser Position zurückliefert oder Sie greifen wie bei einem gewöhnlichen C/C++-Array auf die Elemente zu, indem Sie dem Variablennamen des Arrays die Position in eckigen Klammern ([]) anhängen. Kurz gesagt, die folgenden beiden Methoden liefern das gleiche Ergebnis:

```
pObject = myArray.GetAt(5);
pObject = myArray[5];
```

Um ein Element in einem Array zu setzen, können Sie die Funktion SetAt verwenden. SetAt hat folgende Syntax:

```
void SetAt(int iIndex, datatype element);
```

Der erste Parameter, iIndex, gibt die Position im Array an, die Sie setzen möchten. Der zweite Parameter, element, ist der an der angegebenen Position zu setzende Wert. Der Datentyp des zweiten Parameters sollte der verwendeten Array-Klasse entsprechen. Die Funktion SetAt vergrößert das Array nicht, um zusätzliche Elemente einfügen zu können. Die an die Funktion übergebene Position iIndex muss bereits im Array existieren.

Elemente hinzufügen und entfernen

Wenn Sie einem Array neue Elemente hinzufügen wollen, können Sie einige Funktionen verwenden. Die erste dieser Funktionen ist Add. Die Funktion Add übernimmt das hinzuzufügende Element als einzigen Parameter und hängt dieses Element am Ende des Arrays an. Die zweite Möglichkeit, neue Elemente hinzuzufügen, ist die Funktion InsertAt. Während Add neue Elemente am Ende des Arrays hinzufügt, fügt die Funktion InsertAt neue Elemente mitten im Array ein. Diese Funktion hat folgende Syntax:

```
void InsertAt(int iIndex, datatype element);
```

Der erste Parameter, iIndex, ist die Position im Array, an der das neue Element hinzugefügt werden soll. Der zweite Parameter, element, ist das hinzuzufügende Element. Bei beiden Funktionen wird das Array vergrößert, wenn es momentan nicht genug Positionen hat, um die neu hinzugefügten Elemente aufzunehmen.

Wenn Sie Elemente aus einem Array entfernen möchten, können Sie die Funktion RemoveAt mit folgender Syntax verwenden:

```
void RemoveAt(int iIndex, int iCount = 1);
```

Der erste Parameter dieser Funktion, iIndex, ist die Position des Elements, das aus dem Array entfernt werden soll. Der zweite Parameter, iCount, gibt an, wie viele Elemente aus dem Array entfernt werden sollen. Der zweite Parameter ist optional; wenn Sie ihn nicht übergeben, ist er standardmäßig 1.

Wenn Sie alle Elemente aus einem Array entfernen wollen, können Sie die Funktion RemoveAll verwenden, die alle Objekte aus dem Array entfernt und die Anzahl der Elemente im Array auf 0 setzt.

Wenn Sie ein CObArray oder ein CPtrArray haben, wird das Element nicht zerstört, sondern nur der Zeiger darauf aus dem Array entfernt. Bei den Funktionen RemoveAt und RemoveAll müssen Sie die eigentlichen Objekte daher löschen, bevor Sie sie aus dem Array entfernen.

Arrays kopieren

Wenn Sie ein Array in ein anderes kopieren oder an ein anderes anhängen wollen, verwenden Sie die Funktion Copy bzw. Append. Beide Funktionen übernehmen das zu kopierende oder anzuhängende Array als einziges Argument und kopieren es in das aktuelle Array bzw. hängen es am Ende desselben an.

Bei den Klassen CObArray und CPtrArray kopieren diese Funktionen nur die Zeiger im Array, nicht die eigentlichen Objekte. Daher wirken sich Änderungen an den Werten von Objekten in einem der beiden Arrays auch auf die Werte der gleichen Objekte im zweiten Array aus. Genauso verliert der Zeiger auf ein Objekt im einen Array seine Gültigkeit, wenn dieses Objekt im anderen Array gelöscht wird.

Linien hinzufügen

Als Erstes müssen Sie der Dokumentklasse beibringen, wie man neue Linien hinzufügt. Dazu sind lediglich die Anfangs- und Endpunkte zu holen, ein neues Linienobjekt anzulegen und dann das Objekt in das Objektarray einzufügen. Um diese Funktion zu implementieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Klasse CSDISquigDoc ein.

2. Legen Sie den Rückgabetyp der Funktion als CLine* und den Funktionsnamen als AddLine fest. Die beiden Parameter sollten mit dem Variablentyp CPoint und den Namen ptFrom und ptTo angegeben werden. Legen Sie den Zugriffsstatus als public fest.
3. Schreiben Sie den Code aus Listing 10.4 in die Funktion.

Listing 10.4: Die Funktion AddLine der Klasse CSDISquigDoc

```
 1: CLine* CSDISquigDoc::AddLine(CPoint ptFrom, CPoint ptTo)
 2: {
 3:   CLine* pLine = NULL;
 4:
 5:   try
 6:   {
 7:     // Ein neues CLine-Objekt erzeugen
 8:     pLine = new CLine(ptFrom, ptTo);
 9:     // Die neue Linie in das Objektarray einfügen
10:     m_oaLines.Add(pLine);
11:     // Dokument als bearbeitet markieren
12:     SetModifiedFlag();
13:   }
14:   // Ist Speicherausnahme aufgetreten?
15:   catch (CMemoryException* perr)
16:   {
17:     // Meldung für Benutzer, schlechte Neuigkeiten mitteilen
18:     AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK);
19:     // Wurde Linienobjekt erzeugt?
20:     if (pLine)
21:     {
22:       // Objekt löschen
23:       delete pLine;
24:       pLine = NULL;
25:     }
26:     // Ausnahmeobjekt löschen
27:     perr->Delete();
28:   }
29:   return pLine;
30: }
```

Auf den ersten Blick handelt es sich um eine unkomplizierte Funktion. Als Erstes legt sie eine neue Instanz von CLine an und übergibt an den Konstruktor die Anfangs- und Endpunkte als Argumente. Unmittelbar danach allerdings erscheint die folgende interessante Konstruktion, die den Programmablauf steuert:

```
try
{
...
}
catch (...)
{
...
}
```

Was soll das bedeuten? Diese Konstruktion ist ein Beispiel der strukturierten Ausnahmebehandlung. Bestimmte Code-Abschnitte könnten auf Grund eines unvorhergesehenen Umstandes scheitern, beispielsweise wegen Speichermangel oder unzureichendem Platz auf dem Datenträger. Den problematischen Code-Abschnitt können Sie in einen try-Block einschließen. Diesem sollten sich immer ein oder mehrere catch-Blöcke anschließen. Wenn ein Problem während der Code-Ausführung im try- Block auftritt, springt das Programm unmittelbar zu den catch-Blöcken. Jeder catch-Block legt fest, welchen Ausnahmetyp er behandelt (im speziellen Fall der Funktion AddLine behandelt er lediglich Ausnahmen infolge Speichermangels). Gibt es für das aufgetretene Problem einen entsprechenden catch-Block, wird dieser ausgeführt und bietet der Anwendung die Möglichkeit, Gegenmaßnahmen einzuleiten. Wenn kein catch-Block für den aufgetretenen Problemtyp vorhanden ist, springt das Programm zu einer Standardbehandlungsroutine für Ausnahmen - die wahrscheinlich Ihre Anwendung herunterfährt.

Innerhalb des try-Blocks fügen Sie die neue CLine-Instanz in das Array der Linienobjekte ein. Als Nächstes rufen Sie die Funktion SetModifiedFlag auf, um das Dokument als bearbeitet oder bearbeitet und noch nicht gespeichert zu markieren. Wenn der Benutzer die Anwendung schließt oder ein anderes Dokument öffnet, ohne zuerst die aktuelle Zeichnung zu speichern, fordert ihn die Anwendung auf, die aktuelle Zeichnung zu speichern (mit dem bekannten Meldungsfeld Ja, Nein, Abbrechen).

Im catch-Block informieren Sie den Benutzer, dass das System keinen Speicher mehr zur Verfügung hat, und

löschen dann das CLine-Objekt und das Ausnahmeobjekt. Sie sollten immer die Delete-Methode des Ausnahmeobjekts aufrufen, um die Ausnahme zu löschen.

Schließlich gibt die Funktion das CLine-Objekt an die aufrufende Routine zurück. Damit kann das Ansichtsobjekt das Linienobjekt veranlassen, sich selbst zu zeichnen.

Bevor Sie Ihre Anwendung kompilieren können, müssen Sie die Header-Datei für die Klasse CLine in der Quellcode-Datei für die Dokument- und Ansichtklassen einbinden. Das ist in jeder Quellcode-Datei notwendig, die den Dokumentklassen-Header einbindet, also macht es mehr Sinn, die Include-Anweisung im Dokumentklassen-Header hinzuzufügen. Um das bei Ihrer Anwendung zu tun, bearbeiten Sie die Datei SDISquigDoc.h und fügen Sie wie in Listing 10.5 gezeigt die Include-Anweisung für Line.h nahe dem oberen Ende der Datei ein.

Listing 10.5: Die Include-Anweisungen vom CSDISquigDoc

```
1: // SDISquigDoc.h : Schnittstelle der Klasse CSDISquigDoc
2: //
3:
4: #pragma once
5: #include "afxcoll.h"
6: #include "line.h"
7:
8: class CSDISquigDoc : public CDocument
```

Die Linienanzahl ermitteln

Als nächstes Element nehmen Sie in die Dokumentklasse eine Funktion auf, welche die Anzahl der Linien im Dokument zurückgibt. Diese Funktionalität ist erforderlich, da das Ansichtsobjekt in einer Schleife das Linienarray durchgehen muss, um jedes Linienobjekt anzuweisen, sich selbst zu zeichnen. Das Ansichtsobjekt muss die Gesamtzahl der Linien im Dokument bestimmen und jede gewünschte Linie aus dem Dokument zurückgeben können.

Um die Anzahl der Linien im Dokument zurückzugeben, ermittelt man die Anzahl der Linien im Objektarray, sodass man einfach den Rückgabewert der Methode GetSize der Klasse CObArray zurückgeben kann. Um diese Funktion zu implementieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Klasse CSDISquigDoc ein.

2. Legen Sie den Funktionstyp als int und den Funktionsnamen als GetLineCount mit Zugriffsstatus public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 10.6.

Listing 10.6: Die Funktion GetLineCount der Klasse CSDISquigDoc

```
1: int CSDISquigDoc::GetLineCount()
2: {
3:    // Größe des Arrays zurückgeben
4:    return (int)m_oaLines.GetCount();
5: }
```

Sie wandeln den Rückgabewert der Funktion GetCount in einen Integer um. Der Grund dafür ist, dass GetCount einen Wert mit dem Datentyp INT_PTR zurückliefert. Der Datentyp INT_PTR ist eine spezielle Instanz des Integer-Datentyps. Es handelt sich bei ihm um einen besonderen Integer, der garantiert die gleiche Größe wie ein Zeiger hat, unabhängig von der Zeigergröße und dem Adressraum eines Systems. In Fällen, in denen Sie Zeiger in Integer-Werte umwandeln, ist das der empfehlenswerteste Integer-Datentyp. Da der Compiler denkt, es sei ein

anderer Datentyp als der Standarddatentyp int, müssen wir ihn in ein int umwandeln, damit keine Warnung erzeugt wird.

Eine bestimmte Linie abrufen

Schließlich brauchen Sie noch eine Funktion, die eine bestimmte Linie aus dem Dokument liefert. Dazu gibt man einfach das Objekt an der angegebenen Position im Objektarray zurück. Um diese Funktion zu implementieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Klasse CSDISquigDoc ein.

2. Legen Sie den Funktionstyp als CLine* und den Funktionsnamen als GetLine fest. Fügen Sie einen einzelnen Parameter vom Typ int mit dem Namen iIndex ein. Legen Sie den Zugriff mit public fest.
3. In die Funktion schreiben Sie den Code aus Listing 10.7.

Listing 10.7: Die Funktion GetLine der Klasse CSDISquigDoc

```
1: CLine * CSDISquigDoc::GetLine(int iIndex)
2: {
3:    // Zeiger auf Linienobjekt für die angegebene
4:    // Position im Objektarray zurückgeben
5:    return (CLine*)m_oaLines[iIndex];
6: }
```

Das zurückzugebende Objekt musste in den Typ eines CLine-Objekts umgewandelt werden. Da die Klasse CObArray ein Array von CObjects ist, stellt jedes vom Array gelieferte Element eine Instanz von CObject und nicht eine Instanz eines CLine-Objekts dar.

Die Zeichnung anzeigen

Die Dokumentklasse verfügt nun über die Fähigkeit, die Zeichnung zu speichern. Als Nächstes sind im Ansichtsobjekt die Benutzereingaben zu lesen und dann anzuzeigen. Die Mausereignisse zur Übernahme der Benutzereingaben sind fast identisch zu denen, die Sie an Tag 4 erstellt haben. Der zweite Teil der Funktionalität betrifft die Ausgabe der Zeichnung. Dazu erweitern Sie eine bereits vorhandene Funktion in der Ansichtsobjektklasse.

Bevor Sie diese Funktionen hinzufügen, brauchen Sie noch eine Member-Variable in der Klasse CSDISquigView, um die vorherige Position des Mauszeigers festzuhalten (genau wie an Tag 4):

1. Fügen Sie über die Klassenansicht eine Member-Variable zu der Klasse CSDISquigView hinzu.

2. Legen Sie den Funktionstyp als CPoint, den Funktionsnamen als m_ptPrevPos und den Zugriff als private fest.

Die Mausereignisse hinzufügen

Die Benutzereingaben fangen Sie mit Mausereignissen ab:

1. Markieren Sie in der Klassenansicht die Klasse CSDISquigView.

2. Fügen Sie über das Eigenschaftenfenster für die Ereignisse WM_LBUTTONDOWN, WM_LBUTTONUP und WM_MOUSEMOVE Funktionen ein.
3. Bearbeiten Sie die Funktionen wie in Listing 10.8.

Listing 10.8: Die Mausfunktionen der Klasse CSDISquigDoc

```
 1: void CSDISquigView::OnLButtonDown(UINT nFlags, CPoint point)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein
 4:   // und/oder benutzen Sie den Standard.
 5:   // Maus abfangen, sodass keine andere Anwendung sie beim
 6:   // Verlassen des Fensterbereichs einfangen kann
 7:   SetCapture();
 8:   // Punkte sichern
 9:   m_ptPrevPos = point;
10:
11:   CView::OnLButtonDown(nFlags, point);
12: }
13:
14: void CSDISquigView::OnLButtonUp(UINT nFlags, CPoint point)
15: {
16:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein
17:   // und/oder benutzen Sie den Standard.
18:   // Wurde Maus abgefangen?
19:   if (GetCapture() == this)
20:     // Ja, dann Maus für die Verwendung durch andere Anwendungen
21:     // freigeben
22:     ReleaseCapture();
23:
24:   CView::OnLButtonUp(nFlags, point);
25: }
26:
27: void CSDISquigView::OnMouseMove(UINT nFlags, CPoint point)
28: {
29:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein
30:   // und/oder benutzen Sie den Standard.
31:   // Ist die linke Maustaste gedrückt?
32:   if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
33:   {
34:     // Wurde Maus abgefangen?
35:     if (GetCapture() == this)
36:     {
37:       // Gerätekontext ermitteln
38:       CClientDC dc(this);
39:
40:       // Die Linie ins Dokument aufnehmen
41:       CLine *pLine = GetDocument()->AddLine(m_ptPrevPos, point);
42:
43:       // Aktuellen Linienzug zeichnen
44:       pLine->Draw(&dc);
45:
46:       // Aktuellen Punkt als vorherigen Punkt speichern
47:       m_ptPrevPos = point;
48:     }
49:   }
50:   CView::OnMouseMove(nFlags, point);
51: }
```

Als Erstes ruft OnLButtonDown die Funktion SetCapture auf. Diese Funktion »fängt« die Maus und verhindert, selbst wenn die Maus den Fensterbereich dieser Anwendung verlässt, dass andere Anwendungen Mausereignisse erhalten. So kann der Benutzer die Maus während des Zeichnens aus dem Anwendungsfenster heraus- und dann wieder hineinziehen, ohne das

Zeichnen zu unterbrechen. Alle Mausnachrichten werden an die Anwendung gesandt, bis die Maus in der Funktion OnLButtonUp mithilfe der Funktion ReleaseCapture wieder freigegeben wird. Unterdessen können Sie feststellen, ob Ihre Anwendung die Maus abgefangen hat, indem Sie die Funktion GetCapture in einer if-Anweisung platzieren und ihren Rückgabewert mit this vergleichen. Wenn Sie die Maus abgefangen haben, führen Sie den restlichen Code in diesen Funktionen aus, ansonsten nicht.

Die Funktion OnMouseMove erzeugt zuerst den Gerätekontext und realisiert dann mehrere Dinge in einer einzigen Code-Zeile:

```
CLine *pLine = GetDocument()->AddLine(m_ptPrevPos, point);
```

Diese Zeile erzeugt einen neuen Zeiger auf eine Instanz der Klasse CLine. Als Nächstes wird die Funktion GetDocument aufgerufen, die einen Zeiger auf das Dokumentobjekt zurückgibt. Mit diesem Zeiger wird die Funktion AddLine der Dokumentklasse aufgerufen und die vorherigen und aktuellen Punkte werden als Argumente übergeben. Mit dem Rückgabewert der Funktion AddLine wird der Objektzeiger von CLine initialisiert. Der CLine-Zeiger lässt sich nun verwenden, um die Funktion Draw des Linienobjekts aufzurufen.

Ein Zeiger ist die Adresse eines Objekts. Mit Zeigern lassen sich Objekte in einem Programm effizienter übergeben. Wenn man einen Zeiger auf ein Objekt statt das Objekt selbst übergibt, verhält sich das genauso, wie wenn man sagt »Die Fernbedienung liegt auf dem Sofa zwischen dem zweiten und dritten Kissen neben dem herausgefallenen Kleingeld«, statt jemandem die Fernbedienung direkt in die Hand zu drücken. Gibt man die eigentliche Fernbedienung jemand anders, muss man - um im Programmierjargon zu sprechen - eine genaue Kopie der Fernbedienung erstellen und die Kopie an die andere Person aushändigen. Natürlich ist es einfacher, jemandem nur die Stelle zu sagen, wo sich die Fernbedienung finden lässt, anstatt eine genaue Kopie der Fernbedienung zu bauen.

Die Notation -> bedeutet, dass der Zugriff auf die Funktionen oder Eigenschaften eines Objekts über einen Zeiger erfolgt. Mit einem Punkt (.) kennzeichnet man dagegen den Zugriff über das Objekt selbst.

Die Zeichnung darstellen

In der Ansichtsklasse wird die Funktion OnDraw immer dann aufgerufen, wenn das für den Benutzer ausgegebene Bild neu zu zeichnen ist. Es kann zum Beispiel sein, dass ein anderes Fenster vor dem Anwendungsfenster gelegen ist, das Fenster vom minimierten Zustand gerade wiederhergestellt worden ist oder ein neues Dokument aus einer Datei geladen wurde. Warum die Ansicht neu zu zeichnen ist, spielt keine Rolle. Als Anwendungsentwickler müssen Sie sich nur darum kümmern, den Code in die Funktion OnDraw aufzunehmen, um das Dokument wiederzugeben, für dessen Erstellung Ihre Anwendung vorgesehen ist. Tun Sie Folgendes:

1. Gehen Sie zur Funktion OnDraw in der Klasse CSDISquigView und fügen Sie den fett gedruckten Code aus Listing 10.9 ein. Achten Sie darauf, wie im Listing gezeigt, die Kommentarzeichen vor und hinter dem Funktionsparameter zu entfernen.

Listing 10.9: Die Funktion OnDraw der Klasse CSDISquigView

```
1: void CSDISquigView::OnDraw(CDC* pDC)
2: {
3:   CSDISquigDoc* pDoc = GetDocument();
4:   ASSERT_VALID(pDoc);
5:
6:   // TODO: Code zum Zeichnen der systemeigenen Daten hinzufügen
7:   // Anzahl der Linien im Dokument ermitteln
```

```
 8:    int iCount = pDoc->GetLineCount();
 9:
10:    // Gibt es Linien im Dokument?
11:    if (iCount)
12:    {
13:      int iPos;
14:      CLine *pLine = NULL;
15:
16:      // Schleife durch die Linien des Dokuments
17:      for (iPos = 0; iPos < iCount; iPos++)
18:      {
19:        // Zu zeichnenden Linienabschnitt ermitteln
20:        pLine = pDoc->GetLine(iPos);
21:        // Linie zeichnen
22:        pLine->Draw(pDC);
23:      }
24:    }
25: }
```

Die Funktion ermittelt zuerst, wie viele Linien im Dokument zu zeichnen sind. Gibt es keine Linien, ist auch nichts zu tun. Sind im Dokument Linien gespeichert, geht die Funktion in einer for-Schleife durch die Linien, holt jede Linie aus dem Dokument und ruft dann die Funktion Draw des Linienobjekts auf.

Jetzt sollten Sie Ihre Anwendung kompilieren und ausführen können. Es ist nun möglich, Figuren zu zeichnen, wie es Abbildung 10.8 zeigt. Wenn Sie das Fenster minimieren und dann wiederherstellen oder wenn Sie das Fenster einer anderen Anwendung vor das Fenster Ihrer Anwendung legen, sollte Ihre Zeichnung wieder erscheinen, sobald dieses wieder sichtbar ist (im Gegensatz zur Anwendung, die Sie an Tag 4 erstellt haben).

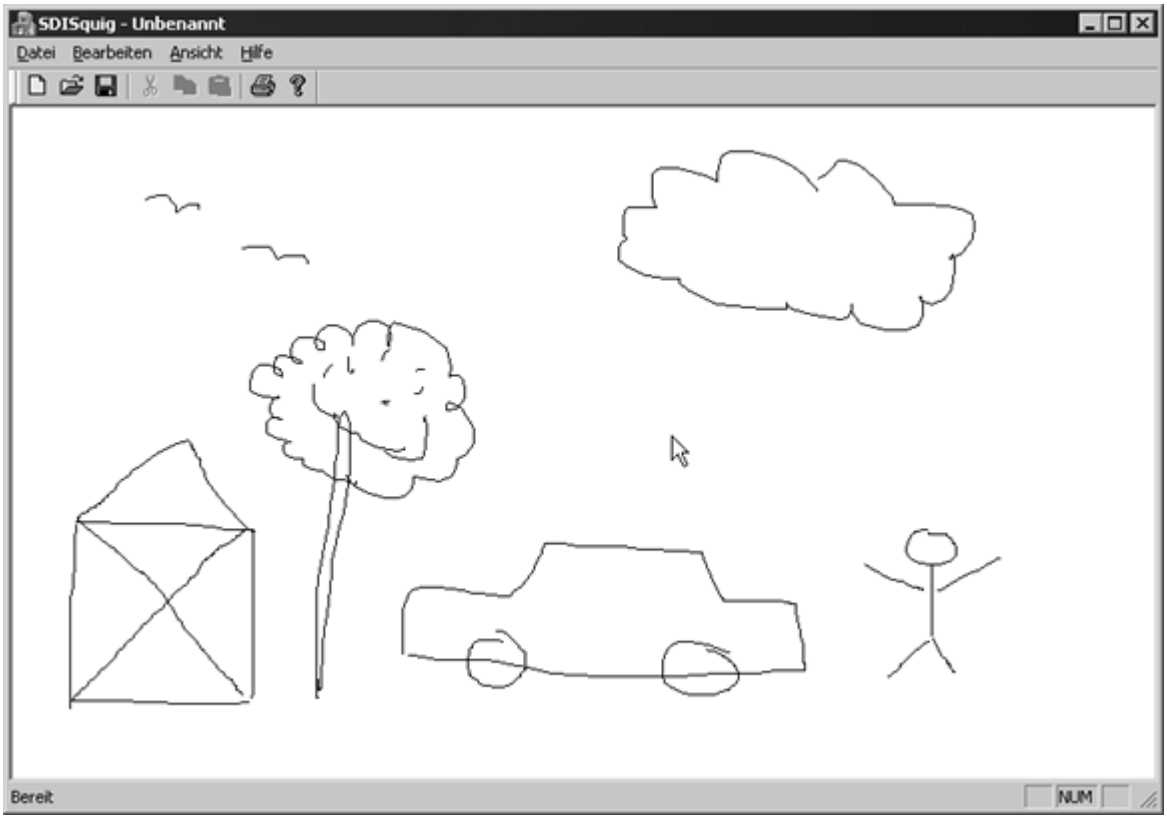


Abbildung 10.8: Mit der Anwendung zeichnen

10.4 Die Zeichnung speichern und laden

Nachdem Ihre Gemälde nun nicht mehr verschwinden, sobald Sie sie aus den Augen lassen, wäre es schön, wenn man sie auch für die Ewigkeit festhalten könnte. Im Menü Datei Ihrer Anwendung sind bereits die Menübefehle Öffnen, Speichern und Speichern unter vorhanden, die aber momentan noch nichts bewirken. Die Menübefehle zum Drucken funktionieren zwar schon, aber die Befehle zum Speichern und Laden einer Zeichnung noch nicht. Nicht einmal beim Menübefehl Neu passiert etwas. Machen wir uns also an die Arbeit.

Die aktuelle Zeichnung löschen

Wenn Sie die Klasse CSDISquigDoc untersuchen, fällt Ihnen sicherlich die Funktion OnNewDocument auf, in der Sie den Code einbauen können, um die aktuelle Zeichnung zu löschen. Weit gefehlt! Diese Funktion ist dafür vorgesehen, alle Klasseneinstellungen zu initialisieren, um mit der Arbeit an einer neuen Zeichnung zu beginnen und nicht um eine vorhandene Zeichnung zu löschen. Stattdessen müssen Sie eine Funktion für die Nachricht DeleteContents hinzufügen. Diese Nachricht ist dafür vorgesehen, den aktuellen Inhalt der Dokumentklasse zu löschen. Sie fügen die Funktion folgendermaßen ein:

1. Markieren Sie in der Klassenansicht die Klasse CSDISquigDoc.

2. Wählen Sie im Eigenschaftenfenster den Modus Überschreibungen aus (siehe Abbildung 10.9).

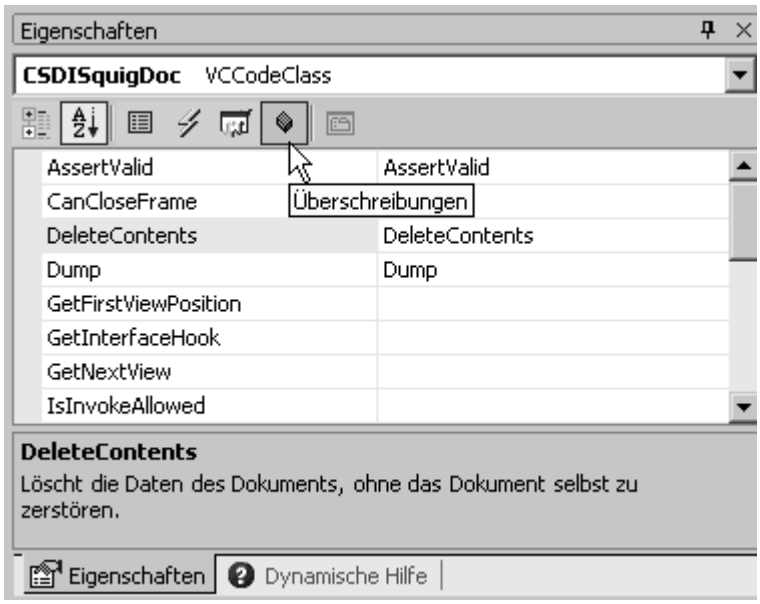


Abbildung 10.9: Überschriebene Basisklassenfunktionen hinzufügen

3. Wählen Sie die Ereignisfunktion DeleteContents aus und fügen Sie eine neue Funktion ein.
4. Nehmen Sie in die neue Funktion den Code aus Listing 10.10 auf.

Listing 10.10: Die Funktion DeleteContents der Klasse CSDISquigDoc

```
 1: void CSDISquigDoc::DeleteContents(void)
 2: {
 3:    // TODO: Fügen Sie hier Ihren spezialisierten Code ein
 4:    // und/oder rufen Sie die Basisklasse auf.
 5:    // Anzahl der Linien im Objektarray ermitteln
 6:    int iCount = (int)m_oaLines.GetCount();
 7:    int iPos;
 8:
 9:    try
10:    {
```

```
11:     // Enthält das Array Objekte?
12:     if (iCount)
13:     {
14:       // Schleife durch Array, dabei jedes Objekt löschen
15:       for (iPos = 0; iPos < iCount; iPos++)
16:         delete (CLine*)m_oaLines.GetAt(iPos);
17:       // Array zurücksetzen
18:       m_oaLines.RemoveAll();
19:     }
20:   }
21:   catch (CMemoryException* perr)
22:   {
23:     // Benutzer über schlechte Neuigkeiten informieren
24:     perr->ReportError();
25:     // Ausnahmeobjekt löschen
26:     perr->Delete();
27:   }
28:   CDocument::DeleteContents();
29: }
```

Die Funktion geht in einer Schleife durch das Objektarray und löscht alle Linienobjekte im Array. Nachdem alle Linien gelöscht sind, setzt die Funktion das Array durch Aufruf der Methode RemoveAll zurück. Wenn Sie die Anwendung kompilieren und ausführen, können Sie den Befehl Datei / Neu wählen, und wenn Sie sich nicht dafür entscheiden, die aktuelle Zeichnung zu speichern, wird der Fensterinhalt gelöscht.

Die Zeichnung speichern und wiederherstellen

Die Funktionalität zum Speichern und Wiederherstellen der Zeichnungen lässt sich relativ einfach implementieren, ist aber vielleicht nicht ohne weiteres verständlich. Momentan soll das nicht weiter stören. An Tag 12 führen Sie sich eine ganze Lektion zu Gemüte, die sich mit dem Speichern und Wiederherstellen von Dateien - der so genannten Serialisierung - beschäftigt. In der Zwischenzeit folgen Sie diesen Schritten, um die Fähigkeit zum Speichern und Laden von Dateien einzufügen:

1. Suchen Sie die Funktion Serialize in der Klasse CSDISquigDoc. Die Funktion sieht folgendermaßen aus:

```
void CSDISquigDoc::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
    // TODO: Hier Code zum Speichern einfügen
  }
  else
  {
    // TODO: Hier Code zum Laden einfügen
  }
}
```

Löschen Sie den gesamten Inhalt der Funktion und nehmen Sie dafür den Code aus Listing 10.11 auf.

Listing 10.11: Die Funktion Serialize der Klasse CSDISquigDoc

```
1: void CSDISquigDoc::Serialize(CArchive& ar)
2: {
3:   // Serialisierung an Objektarray weitergeben
4:   m_oaLines.Serialize(ar);
5: }
```

Die Funktion stützt sich auf die Funktionalität der Klasse CObArray. Dieses Objektarray reicht sein Array von Objekten nach unten weiter und ruft dabei die Funktion Serialize auf jedem Objekt auf. Das bedeutet, dass Sie eine Funktion Serialize in die Klasse CLine aufnehmen müssen:

1. Fügen Sie eine neue Funktion in die Klasse CLine ein.

2. Legen Sie den Funktionstyp als void fest, benennen Sie die Funktion mit Serialize und fügen Sie einen Parameter ein. Der Parametertyp sollte CArchive& und der Name ar sein. Legen Sie den Funktionszugriff als public fest.
3. Fügen Sie den Code aus Listing 10.12 in die Funktion ein.

Listing 10.12: Die Funktion Serialize der Klasse CLine

```
1: void CLine::Serialize(CArchive &ar)
2: {
3:   CObject::Serialize(ar);
4:
5:   if (ar.IsStoring())
6:     ar << m_ptFrom << m_ptTo;
7:   else
8:     ar >> m_ptFrom >> m_ptTo;
9: }
```

Die Funktion weist grundsätzlich den gleichen Programmfluss auf wie die ursprüngliche Version der Serialize-Funktion in der Klasse CSDISquigDoc. Die Funktion greift auf die Funktionalität der I/O-Streams (Input/Output-Streams - Eingabe/Ausgabe-Datenströme) von C++ zurück, um die Inhalte zu speichern und wiederherzustellen.

Wenn Sie die Anwendung zum jetzigen Zeitpunkt kompilieren und ausführen, erwarten Sie sicherlich, dass die Funktionen zum Speichern und Öffnen arbeiten. Leider tun sie das aber (noch) nicht. Wenn Sie die Anwendung ausführen und versuchen, eine Zeichnung zu speichern, weist Sie ein Meldungsfeld darauf hin, dass die Anwendung die Datei nicht speichern konnte (siehe Abbildung 10.10).

Abbildung 10.10: Die Zeichnungen lassen sich momentan noch nicht speichern.

Der Grund für das Scheitern liegt darin, dass man Visual C++ mitteilen muss, dass eine Klasse serialisierbar sein soll. Zu diesem Zweck fügen Sie eine Zeile in die Header-Datei und eine Zeile in die Quellcode-Datei der Klasse CLine ein:

1. Öffnen Sie die Header-Datei der Klasse CLine (Line.h).

2. Schreiben Sie die Zeile DECLARE_SERIAL gemäß Listing 10.13 unmittelbar nach der ersten Zeile der Klassendefinition.

Listing 10.13: Die Header-Datei Line.h für Serialisierung angepasst

```
1: class CLine :
2:   public CObject
3:   {
4:     DECLARE_SERIAL (CLine)
5:     public:
6:     CLine(void);
```

3. Als Nächstes öffnen Sie die Quellcode-Datei von CLine (Line.cpp).
4. Fügen Sie die Zeile IMPLEMENT_SERIAL gemäß Listing 10.14 unmittelbar vor den Konstruktorfunktionen der Klasse ein.

Listing 10.14: Die Quellcode-Datei Line.cpp für Serialisierung angepasst

```
1: #include "StdAfx.h"
2: #include "line.h"
3:
4: IMPLEMENT_SERIAL (CLine, CObject, 1)
5: CLine::CLine(void)
6: : m_ptFrom(0)
7: , m_ptTo(0)
8: {
9: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie Ihr Selbstporträt zeichnen und für die Ewigkeit speichern können (siehe Abbildung 10.11).

Abbildung 10.11: Selbstporträt der Übersetzerin

10.5 Interaktion mit Menüs

Nachdem Sie nun über ein funktionsfähiges Zeichenprogramm verfügen, wäre es schön, wenn der Benutzer die Farbe auswählen könnte, mit der er zeichnen will. Dazu sind Änderungen an der Klasse CLine erforderlich, um die Farbe mit der Linie zu verbinden, sowie an der Klasse CSDISquigDoc, um die momentan ausgewählte Farbe zu verwalten. Schließlich ist noch ein Pulldown-Menü vorzusehen, aus dem sich die gewünschte Farbe auswählen lässt.

Farben in die Klasse CLine aufnehmen

Die Änderungen in der Klasse CLine sind recht unkompliziert. Allgemein gesehen tun Sie Folgendes:

1. Sie fügen eine weitere Member-Variable hinzu, um die Farbe jeder Linie zu speichern.

2. Sie modifizieren den Klassenkonstruktor, um die Liste der Parameter durch die zu übergebende Farbe zu erweitern.
3. Sie modifizieren die Funktion Draw, damit diese die angegebene Farbe verwendet.
4. Sie modifizieren die Funktion Serialize, um die Farbinformation zusammen mit den Punktkoordinaten speichern und wiederherstellen zu können.

Um all das zu tun, führen Sie folgende Schritte aus:

1. Markieren Sie in der Klassenansicht die Klasse CLine. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.

2. Legen Sie den Variablentyp als COLORREF, den Namen als m_crColor und den Zugriff als private fest. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.
3. Klicken Sie mit der rechten Maustaste in der Klassenansicht auf den angepassten Konstruktor von CLine, den Sie erstellt haben. Aus dem Kontextmenü wählen Sie Gehe zu Definition.
4. Fügen Sie COLORREF crColor als drittes Argument in die Konstruktordeklaration ein.
5. Klicken Sie mit der rechten Maustaste in der Baumansicht der Klassenansicht auf den Konstruktor von

CLine. Aus dem Kontextmenü wählen Sie Gehe zu Deklaration.

6. Modifizieren Sie den Konstruktor gemäß Listing 10.15, um das dritte Argument hinzuzufügen und das Element m_crColor auf das neue Argument zu setzen.

Listing 10.15: Der modifizierte Konstruktor von CLine

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor)
2: {
3:    // Anfangs- und Endpunkte initialisieren
4:    m_ptFrom = ptFrom;
5:    m_ptTo = ptTo;
6:    m_crColor = crColor;
7: }
```

7. Gehen Sie nach unten zur Funktion Draw und nehmen Sie die Änderungen gemäß Listing 10.16 vor.

Listing 10.16: Die modifizierte Funktion Draw

```
 1: void, CLine::Draw(CDC * pDC)
 2: {
 3:    // Stift erzeugen
 4:    CPen lpen (PS_SOLID, 1, m_crColor);
 5:
 6:    // Neuen Stift als Zeichenobjekt festlegen
 7:    CPen* pOldPen = pDC->SelectObject(&lpen);
 8:    // Linie zeichnen
 9:    pDC->MoveTo(m_ptFrom);
10:    pDC->LineTo(m_ptTo);
11:    // Auf vorherigen Stift zurücksetzen
12:    pDC->SelectObject(pOldPen);
13: }
```

8. Scrollen Sie weiter nach unten zur Funktion Serialize und ändern Sie die Funktion entsprechend Listing 10.17 ab.

Listing 10.17: Die modifizierte Funktion Serialize

```
1: void  CLine::Serialize(CArchive &ar)
2: {
3:    CObject::Serialize(ar);
4:
5:    if (ar.IsStoring())
6:      ar << m_ptFrom << m_ptTo << (DWORD) m_crColor;
7:    else
8:      ar >> m_ptFrom >> m_ptTo >> (DWORD) m_crColor;
9: }
```

Farbige Dokumente

Die an der Klasse CSDISquigDoc vorzunehmenden Änderungen sind nur wenig umfangreicher als die an der Klasse CLine. Auch hier wieder die allgemeinen Schritte:

1. Fügen Sie eine Member-Variable für die aktuelle Farbe hinzu sowie eine Farbtabelle, um die Farb-IDs in RGB-Werte zu konvertieren.

2. Initialisieren Sie in der Funktion OnNewDocument die Variable für die aktuelle Farbe.
3. Modifizieren Sie die Funktion AddLine, um die aktuelle Farbe in den Konstruktor von CLine einzufügen.
4. Nehmen Sie eine Funktion auf, um die aktuelle Farbe zurückzugeben.

Das ist momentan alles, bis Sie die Behandlungsroutinen für die Menübefehle in Angriff nehmen. Führen Sie im Einzelnen die folgenden Schritte aus:

1. Markieren Sie in der Klassenansicht die Klasse CSDISquigDoc. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.

2. Legen Sie den Variablentyp als UINT, den Namen mit m_nColor und den Zugriff als private fest. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.
3. Öffnen Sie die Header-Datei SDISquigDoc.h.
4. Suchen Sie einen der als public deklarierten Abschnitte und fügen Sie folgende Variablendeklaration ein:

```
// Aktuelle Zeichenfarbe ermitteln
static const COLORREF m_crColors[8];
```

5. Öffnen Sie den Quellcode von CSDISquigDoc (Datei SDISquigDoc.cpp) und fügen Sie die m_crColors-Farbtabelle gemäß Listing 10.18 ein.

Listing 10.18: Die Spezifikation der Farbtabelle

```
END_MESSAGE_MAP()

const COLORREF CSDISquigDoc::m_crColors[8] = {
    RGB(   0,   0,   0),    // Schwarz
    RGB(   0,   0, 255),    // Blau
    RGB(   0, 255,   0),    // Grün
    RGB(   0, 255, 255),    // Cyan
    RGB( 255,   0,   0),    // Rot
    RGB( 255,   0, 255),    // Magenta
    RGB( 255, 255,   0),    // Gelb
    RGB( 255, 255, 255)     // Weiß
};

// CSDISquigDoc construction/destruction
```

6. Scrollen Sie nach unten zur Funktion OnNewDocument und nehmen Sie die Änderungen gemäß Listing 10.19 vor.

Listing 10.19: Die modifizierte Funktion OnNewDocument

```
BOOL CSDISquigDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
    return FALSE;

  // TODO: Hier Code zur Reinitialisierung einfügen
  // (SDI-Dokumente verwenden dieses Dokument)
  // Farbe mit Schwarz initialisieren
  m_nColor = 0;

  return TRUE;
}
```

7. Gehen Sie weiter zur Funktion AddLine und passen Sie die Funktion entsprechend Listing 10.20 an.

Listing 10.20: Die modifizierte Funktion AddLine

```
CLine * CSDISquigDoc::AddLine(CPoint ptFrom, CPoint ptTo)
{
  CLine* pLine = NULL;

  try
  {
    // Ein neues CLine-Objekt erzeugen
    pLine = new CLine(ptFrom, ptTo, m_crColors[m_nColor]);
```

```
 9:     // Die neue Linie in das Objektarray einfügen
10:     m_oaLines.Add(pLine);
11:     // Dokument als bearbeitet markieren
12:     SetModifiedFlag();
13:   }
14:   // Ist Speicherausnahme aufgetreten?
15:   catch (CMemoryException* perr)
16:   {
17:     // Meldung für den Benutzer, schlechte Neuigkeiten mitteilen
18:     AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK);
19:     // Wurde Linienobjekt erzeugt?
20:     if (pLine)
21:     {
22:       // Objekt löschen
23:       delete pLine;
24:       pLine = NULL;
25:     }
26:     // Ausnahmenobjekt löschen
27:     perr->Delete();
28:   }
29:   return pLine;
30: }
```

8. Fügen Sie der Klasse CSDISquigDoc eine neue Member-Funktion hinzu. Legen Sie den Funktionstyp als UINT, den Namen als GetColor und den Zugriff als public fest.
9. In die Funktion GetColor schreiben Sie den Code aus Listing 10.21.

Listing 10.21: Die Funktion GetColor

```
1: UINT CSDISquigDoc::GetColor(void)
2: {
3:   // Aktuelle Farbe zurückgeben
4:   return ID_COLOR_BLACK + m_nColor;
5: }
```

Während Sie in Funktion OnNewDocument die Farbe als Index der Farbtabelle betrachten, führen Sie in GetColor eine Addition des Farbindex mit dem Wert ID_COLOR_BLACK durch. Der Wert ID_COLOR_BLACK ist die kleinste ID des durchnummerierten Farbmenüs, wenn Sie gleich die Menübefehle hinzufügen. Diese Berechnungen verwalten die Variable als Zahl zwischen 0 und 7, während diese Funktionen beim Umgang mit den Menüs wieder Vergleiche mit den eigentlichen Menü-IDs zulassen.

Das Menü modifizieren

Nun kommt der interessantere Teil. In das Hauptmenü fügen Sie ein neues Pulldown- Menü ein. In dieses Menü nehmen Sie Befehle für alle Farben der Farbtabelle auf. Für die Menübefehle müssen Sie noch Behandlungsroutinen erstellen, um den Befehl für die momentan ausgewählte Farbe mit einem Kontrollhäkchen zu versehen. Das Ganze realisieren Sie in folgenden Schritten:

1. Wählen Sie die Ressourcenansicht aus. Erweitern Sie den Baum, sodass der Inhalt des Ordners Menu zu sehen ist. Doppelklicken Sie auf die Menüressource.

2. Klicken Sie auf das leere Menü in der obersten Ebene (am rechten Ende der Menüleiste). Legen Sie den Namen &Farbe fest.
3. Ziehen Sie das Menü in der obersten Ebene (Farbe) nach links und platzieren Sie es vor dem Menüeintrag Ansicht.
4. Fügen Sie dem Menü Farbe Untereinträge hinzu. Erstellen Sie die Untermenüeinträge in der Reihenfolge von Tabelle 10.7 und verwenden Sie die dort angegebenen Beschriftungen.
5. Nachdem Sie die Beschriftungen für die Menüeinträge gesetzt haben, klicken Sie mit der rechten Maustaste und wählen Sie IDs bearbeiten. Die Menü-IDs setzen Sie wie in Tabelle 10.7 angegeben.

Objekt	Eigenschaft	Einstellung

Menübefehl	ID Beschriftung	ID_COLOR_BLACK &Schwarz
Menübefehl	ID Beschriftung	ID_COLOR_BLUE &Blau
Menübefehl	ID Beschriftung	ID_COLOR_GREEN Gr&ün
Menübefehl	ID Beschriftung	ID_COLOR_CYAN &Cyan
Menübefehl	ID Beschriftung	ID_COLOR_RED &Rot
Menübefehl	ID Beschriftung	ID_COLOR_MAGENTA &Magenta
Menübefehl	ID Beschriftung	ID_COLOR_YELLOW &Gelb
Menübefehl	ID Beschriftung	ID_COLOR_WHITE &Weiß

Tabelle 10.7: Eigenschaftseinstellungen des Menüs

6. Klicken Sie mit der rechten Maustaste und deaktivieren Sie IDs bearbeiten. Das fertige Menü sollte dann Abbildung 10.12 entsprechen.

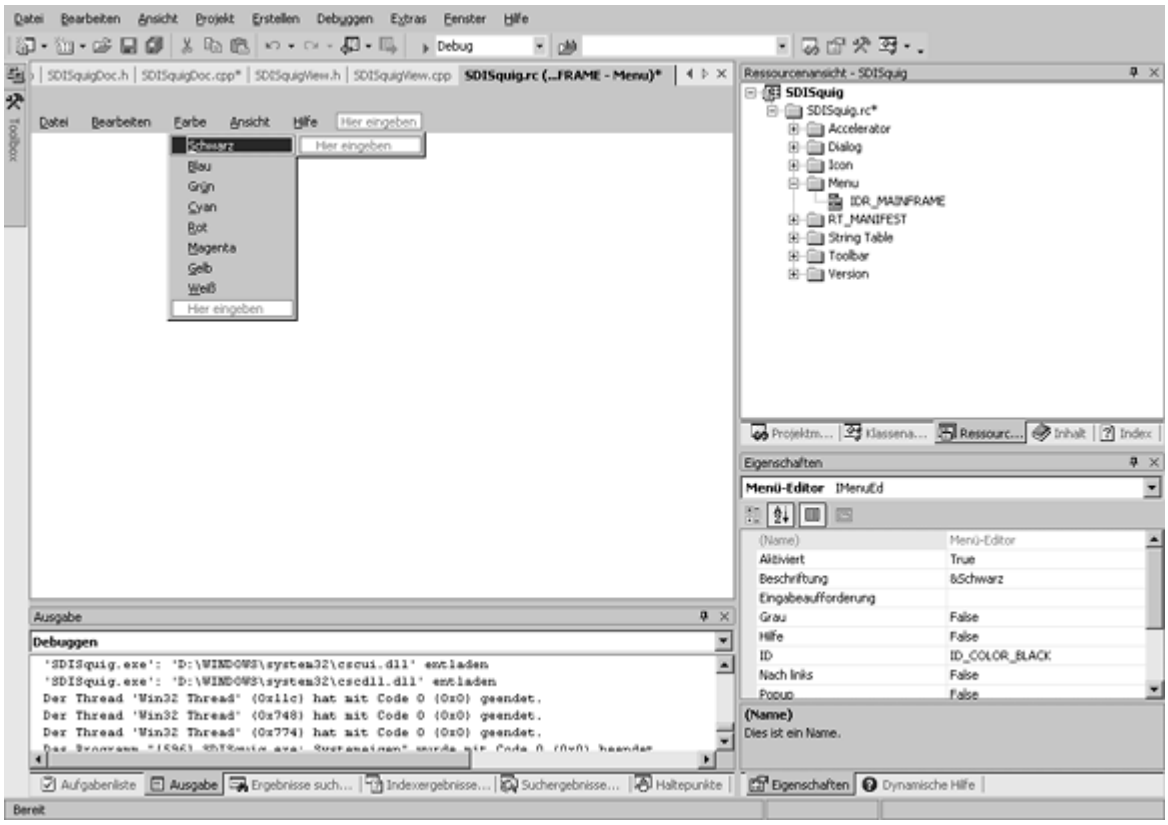


Abbildung 10.12: Das Menü Farbe in der Entwurfsansicht

7. Öffnen Sie die Klassenansicht. Markieren Sie die Klasse CSDISquigDoc.
8. Wählen Sie im Eigenschaftenfenster den Modus Ereignisse aus und scrollen Sie nach unten, bis Sie die Menübefehle finden.
9. Klicken Sie auf das Plus-Zeichen links von den Menübefehlen, um die Menüereignisse zu sehen.
10. Fügen Sie für die beiden Nachrichten COMMAND und UPDATE_COMMAND_UI Funktionen für alle Befehle des Farbmenüs hinzu.
11. Bearbeiten Sie die Menüfunktionen für Schwarz gemäß Listing 10.22.

Listing 10.22: Die Menüfunktionen für Schwarz

```
 1: void CSDISquigDoc::OnColorBlack(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
 4:   // Aktuelle Farbe auf Schwarz setzen
 5:   m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;
 6: }
 7:
 8: void CSDISquigDoc::OnUpdateColorBlack(CCmdUI *pCmdUI)
 9: {
10:   // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-
11:   // UI-Behandlungscode ein.
12:   // Ist Menüpunkt Schwarz mit Kontrollhäkchen zu versehen?
13:   pCmdUI->Enable();
14:   pCmdUI->SetCheck(GetColor() == ID_COLOR_BLACK ? 1 : 0);
15: }
```

12. Bearbeiten Sie die Funktionen des Menüeintrags Blau gemäß Listing 10.23. Die restlichen Menüfunktionen behandeln Sie in der gleichen Weise, wobei Sie für ID_COLOR_BLUE die jeweilige Menü-ID einsetzen.

Listing 10.23: Die Menüfunktionen für Blue

```
1: void  CSDISquigDoc::OnColorBlue(void)
```

```
 2: {
 3:   // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
 4:   // Aktuelle Farbe auf Blau setzen
 5:   m_nColor = ID_COLOR_BLUE - ID_COLOR_BLACK;
 6: }
 7:
 8: void CSDISquigDoc::OnUpdateColorBlue(CCmdUI *pCmdUI)
 9: {
10:   // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-
11:   // UI-Behandlungscode ein.
12:   // Ist Menübefehl Blau mit Kontrollhäkchen zu versehen?
13:   pCmdUI->Enable();
14:   pCmdUI->SetCheck(GetColor() == ID_COLOR_BLUE ? 1 : 0);
15: }
```

In der ersten der beiden Menüfunktionen, der COMMAND-Funktion in Listing 10.23 (OnColorBlue), wird die Variable der aktuellen Farbe auf die neue Farbe gesetzt. Wenn Sie die Menübefehle in der richtigen Reihenfolge hinzugefügt haben, weisen sie fortlaufende ID-Nummern auf, die mit ID_COLOR_ BLACK beginnen. Zieht man den Wert ID_COLOR_BLACK von der Menü-ID ab, erhält man dann die Position in der Farbtabelle für die ausgewählte Farbe. Beispielsweise hat Schwarz die Position 0 in der Farbtabelle: ID_COLOR_BLACK - ID_COLOR_BLACK = 0. Blau steht auf der Position 1 in der Farbtabelle. Da ID_COLOR_BLUE um 1 größer als ID_COLOR_BLACK sein sollte, liefert ID_ COLOR_BLUE - ID_COLOR_BLACK das Ergebnis 1.

Die zweite Funktion, die Funktion UPDATE_COMMAND_UI in Listing 10.23 (OnUpdateColorBlue), erfordert eine nähere Erläuterung. Das Ereignis UPDATE_COMMAND_UI wird für jeden Menübefehl aufgerufen, unmittelbar bevor dieser angezeigt wird. Mit dieser Behandlungsfunktion können Sie in Abhängigkeit von der aktuellen Farbe neben dem betreffenden Menübefehl ein Kontrollhäkchen setzen oder dieses entfernen. Mit diesem Ereignis lassen sich auch Menübefehle aktivieren oder deaktivieren oder bei Bedarf andere Modifikationen realisieren. Der Code in dieser Funktion

```
pCmdUI->SetCheck(GetColor() == ID_COLOR_BLUE ? 1 : 0);
```

bewirkt mehreres. Erstens ist das als einziges Argument übergebene pCmdUI-Objekt ein Zeiger auf ein Menüobjekt. Die Funktion SetCheck kann das Kontrollhäkchen vor den Menübefehl setzen oder entfernen, je nachdem, ob das übergebene Argument 1 oder 0 ist (1 setzt das Kontrollhäkchen, 0 entfernt es). Der Argumentabschnitt für die Funktion SetCheck stellt eine Konstruktion zur Programmsteuerung dar, die Ihnen vielleicht etwas verwirrend vorkommt, wenn Sie noch nicht allzu viel Zeit in die Programmierung mit C/C++ investiert haben. Die erste Hälfte

```
GetColor() == ID_COLOR_BLUE
```

ist eine einfache Boolesche Bedingungsanweisung, die entweder TRUE (wahr) oder FALSE (falsch) liefert. Der sich an diese Bedingungsanweisung anschließende Teil

```
? 1 : 0
```

stellt grundsätzlich eine if...else-Anweisung dar, allerdings in Kurzform. Wenn die Bedingungsanweisung TRUE liefert, dann ist der Wert gleich dem Teil zwischen Fragezeichen (?) und Doppelpunkt (:) (hier also 1), und wenn die Anweisung FALSE ergibt, dann gilt der Wert nach dem Doppelpunkt (:) (hier also 0). Damit lässt sich auf geschickte Weise eine if...else-Steuerstruktur im Argument an eine andere Funktion übergeben.

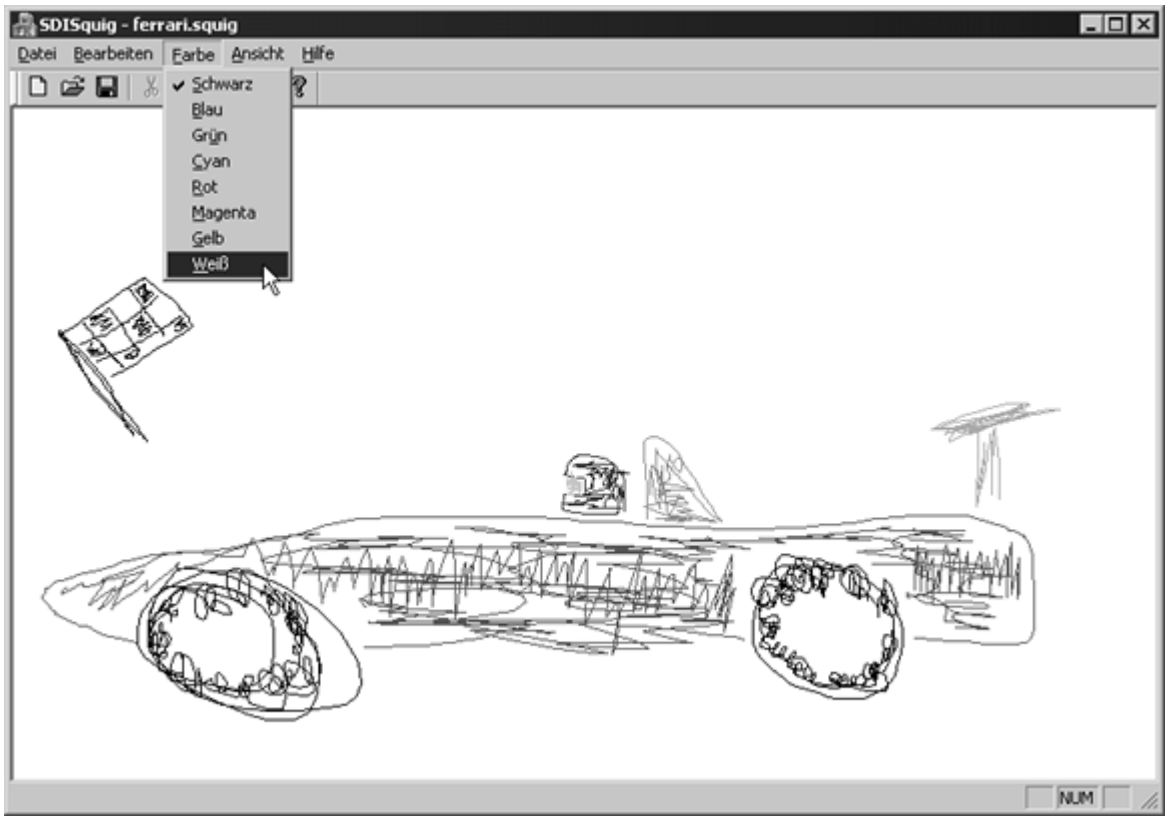


Abbildung 10.13: Die aktuelle Farbe im Menü festlegen

Nachdem Sie die Anwendung kompiliert und gestartet haben, sollten Sie die Zeichenfarbe ändern können. Wenn Sie das Farbmenü öffnen, wird die aktuelle Zeichenfarbe mit einem Kontrollhäkchen versehen sein, wie es Abbildung 10.13 zeigt.

10.6 Zusammenfassung

Was für ein Tag! Diese Lektion war mit Informationen vollgepackt. Zuerst wurde gezeigt, was eine SDI-Anwendung ist und welche bekannten Standardanwendungen in diesem Stil aufgebaut sind. Als Nächstes haben Sie die Dokument-/View-Architektur kennen gelernt, die Visual C++ für SDI-Anwendungen verwendet. Sie haben eine eigene einfache Klasse erstellt, die sich für eine Zeichenanwendung eignet. Dann haben Sie eine Zeichenanwendung erstellt, mit der Sie Bilder zeichnen und verwalten können. Es wurde gezeigt, wie man Dokumente in der Dokument-/View-Architektur speichert und wiederherstellt. Weiterhin wurde die Klasse CObArray behandelt und wie man sie einsetzen kann, um ein dynamisches Objektarray für das Speichern verschiedenartiger Klassen zu erzeugen. Schließlich haben Sie gelernt, wie man Kontrollhäkchen in Menübefehlen von MFC-Anwendungen setzt und entfernt.

10.7 Workshop

Fragen und Antworten

Frage:
Gibt es eine Möglichkeit, die Anzahl der Funktionen für die Nachrichten COMMAND und UPDATE_COMMAND_UI bei Menüs zu verringern?

Antwort:
Ja. Man kann alle Ereignisse des Farbmenüs an dieselbe Funktion senden. In dieser Funktion lässt sich der Wert nID (der als Argument übergeben wird) untersuchen und mit den Menü-IDs vergleichen, um zu ermitteln, welcher Menübefehl die Funktion aufgerufen hat. Letztendlich kann man die COMMAND- Funktion für die Farbmenüs wie folgt formulieren:

```
void CSDISquigDoc::OnColorCommand(UINT nID)
```

```
{
  // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
  // Aktuelle Farbe setzen
  m_nColor = nID - ID_COLOR_BLACK;
}
```

Bei den UPDATE_COMMAND_UI-Funktionen kann man analog vorgehen, allerdings mit einem kleinen Unterschied. Bei diesen Funktionen kann man den Wert pCmdUI->m_nID untersuchen, um zu bestimmen, für welchen Menübefehl die Funktion aufgerufen wurde. Damit sieht die Funktion UPDATE_COMMAND_UI folgendermaßen aus:

```
void CSDISquigDoc::OnUpdateColor(CCmdUI* pCmdUI)
{
  // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-UI-Behandlungscode ei
  // Ist Menübefehl mit Kontrollhäkchen zu versehen?
  pCmdUI->SetCheck(GetColor() == pCmdUI->m_nID ? 1 : 0);
}
```

Der Trick bei diesem Ansatz ist, dass Sie für seine Implementierung nicht die Assistenten verwenden können. Sie müssen die Funktionen zusammen mit den Einträgen in die Nachrichtenzuordungstabelle per Hand einfügen. Wenn Sie die beiden obigen Funktionen hinzufügen, legen Sie den Zugriff als protected fest. Als Nächstes fügen Sie nahe dem oberen Ende der Quellcode-Datei die Nachrichtenzuordungstabelle Einträge hinzu (direkt über der Farbtabelle). Wenn Sie die Nachrichtenzuordungstabelle des heute erstellten Beispiels betrachten, sollte sie ungefähr folgendermaßen aussehen:

```
BEGIN_MESSAGE_MAP(CSDISquigDoc, CDocument)
    ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
    ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
    ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
    ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
    ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
    ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
    ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
    ON_COMMAND(ID_COLOR_RED, OnColorRed)
    ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
    ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
    ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
    ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
    ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
    ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
    ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
END_MESSAGE_MAP()
```

Statt all dieser Einträge für die einzelnen Menüs fügen Sie nur jeweils einen einzigen Eintrag für alle ein:

```
BEGIN_MESSAGE_MAP(CSDISquigDoc, CDocument)
    ON_COMMAND_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE, OnColorCommand)
    ON_UPDATE_COMMAND_UI_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE,
                               OnUpdateColorUI)
END_MESSAGE_MAP()
```

Die normalen Einträge in der Nachrichtenzuordungstabelle haben zwei Argumente: die Nachrichten-ID und den Namen der Behandlungsroutine, die für die Nachricht aufgerufen werden soll. Diese neuen Einträge funktionieren genauso, haben jedoch statt nur einer Ereignis-ID zwei. Die beiden Ereignis-ID- Argumente kennzeichnen die beiden Enden eines Bereichs von Ereignis-IDs, die an die angegebene Funktion übergeben werden sollen. Diese beiden Ereignis-IDs sollten der erste und der letzte Menüeintrag sein, die Sie beim Erstellen des Menüs Color erzeugt haben.

Frage:
Worin unterscheiden sich SDI- und MDI-Anwendungen?

Antwort:
Während SDI-Anwendungen nur eine Aufgabe ausführen können, lassen sich bei MDI-Anwendungen (MDI - Multiple Document Interface) mehrere Dokumente gleichzeitig öffnen. Darüber hinaus müssen in einer MDI-Anwendung nicht alle Dokumente vom gleichen Typ sein.

Frage:
Im Grunde ist es der gleiche Code, ob man nun eine SDI- oder eine MDI-Anwendung erstellt. Warum also sollte ich eine SDI-Anwendung erstellen? Warum erstelle ich nicht alle meine Anwendungen als MDI-Anwendungen?

Antwort:
Das hängt von der Anwendung und ihrem Gebrauch ab. Wahrscheinlich verwenden Sie täglich beide Arten von Anwendungen. Wenn Sie ein Memo schreiben oder an einer Tabellenkalkulation arbeiten, verwenden Sie wahrscheinlich eine MDI-Anwendung. Wenn Sie im WWW surfen, handelt es sich bei Ihrem Browser höchstwahrscheinlich um eine SDI-Anwendung. Ein einfacher Text-Editor wie Notepad (Editor) wäre als MDI-Anwendung wahrscheinlich schwieriger zu bedienen, doch als SDI-Anwendung ist er genau richtig (für die Aufgabe, die er erfüllen soll).

> Bei manchen Anwendungen macht es mehr Sinn, sie als SDI-Anwendungen anstatt als MDI-Anwendungen zu implementieren. Sie müssen durchdenken, wie Ihre Anwendung gebraucht werden wird und dann entscheiden, welches Modell am besten dafür passt.

Frage:
Manche Einträge meines Farbmenüs wechseln auf die falsche Farbe. Wie kann ich die Ursache des Problems feststellen?

Antwort:
Das Problem ist wahrscheinlich, dass die IDs des Farbmenüs nicht in der richtigen Reihenfolge angegeben sind. Sie können das überprüfen, indem Sie mit der rechten Maustaste auf die SDISquig-Ressourcen auf der Registerkarte Ressourcen im Arbeitsbereich klicken. Wählen Sie aus dem Kontextmenü Ressourcensymbole, um eine Liste der IDs in alphabetischer Reihenfolge und der ihnen zugewiesenen Nummern anzuzeigen. Beginnen Sie bei der ID von Schwarz und vergewissern Sie sich, dass die Nummern immer um 1 steigen, ohne dabei Nummern zu überspringen. Achten Sie darauf, die IDs in der Reihenfolge zu überprüfen, in der sie im Menü (und in der Farbtabelle in der Datei SDISquigDoc.cpp) erscheinen, nicht in der alphabetischen Reihenfolge, in der sie in dieser Liste auftauchen. Wenn Sie Fehler finden, müssen Sie Visual C++ beenden und die Datei Resource.h in einem Text-Editor öffnen, um die IDs neu zu nummerieren. Nachdem Sie die Korrekturen durchgeführt haben (achten Sie darauf, doppelte Einträge zu löschen), speichern Sie die Datei, starten Visual C++ neu und rekompilieren Ihre Anwendung. Das Farbmenü sollte jetzt korrekt funktionieren.

Quiz

1. Wofür steht SDI?

2. Welche Funktionalität ist in der Ansichtsklasse realisiert?
3. Welche Funktion wird aufgerufen, um das Dokument neu zu zeichnen, wenn das Fenster hinter einem anderen Fenster verborgen war?
4. Wo bringt man den Code unter, um das aktuelle Dokument zu löschen, bevor man mit einem neuen Dokument beginnt?
5. Welchen Zweck erfüllt die Dokumentklasse?
6. Welche fünf Basisklassen werden in MDI-Anwendungen verwendet?
7. Welche Farbe hat der in Abbildung 10.13 gemalte Wagen?

Übung

Fügen Sie ein weiteres Pulldown-Menü hinzu, um die Breite des Zeichenstifts zu steuern. Verwenden Sie folgende Einstellungen:

Menüeintrag	Breite

Sehr Dünn	1
Dünn	8
Mittel	16
Dick	24
Sehr dick	32

Das zweite Argument des Stiftkonstruktors gibt die Breite an.

Tag 11

Symbolleisten und Statusleisten

Die bisher erstellten SDI- und MDI-Anwendungen verfügen nicht nur über Standardmenüs, sondern weisen auch einfache Symbolleisten auf, die mit den Menüs korrespondieren. Die Symbolleisten sind mit Standardfunktionen (Neu, Öffnen, Speichern, Drucken, Ausschneiden, Kopieren und Einfügen) ausgestattet, die sich in den Symbolleisten der meisten Windows-Anwendungen finden. Die Mehrzahl der Anwendungen beschränkt sich jedoch nicht auf diese vorgegebene Auswahl von Funktionen, sondern passt die Symbolleisten an, um die konkrete Funktionalität der Anwendung widerzuspiegeln.

Neben den Symbolleisten verfügen die SDI- und MDI-Anwendungen auch über eine Statusleiste am unteren Rand des Anwendungsfensters, um eine textuelle Beschreibung der Symbolleisten-Schaltflächen und Menübefehle bereitzustellen. Die Statusleiste verfügt ebenfalls über Standardbereiche, die den Zustand der Feststelltasten für Großbuchstaben (Caps-Lock), Zahlenfeld (Num-Lock) und Bildlauf (Scroll) anzeigen.

Heute lernen Sie vor allem, wie man ...

- eigene Symbolleisten entwirft,
- eine Symbolleiste mit dem Anwendungsgerüst verbindet,
- eine Symbolleiste über einen Menübefehl anzeigt und ausblendet,
- ein Kombinationsfeld in eine Symbolleiste einbindet,
- Beschreibungen der Symbolleistenelemente in der Statusleiste anzeigt,
- eigene Elemente in die Statusleiste einbaut.

11.1 Symbolleisten, Statusleisten und Menüs

Eine der treibenden Kräfte hinter der Entwicklung von grafischen Benutzeroberflächen wie Windows ist das Ziel, die Benutzerfreundlichkeit von Computern zu verbessern. In diesem Zusammenhang haben die Entwickler der Oberflächen festgelegt, dass alle Anwendungen mit einem Standardsatz von Menüs auszustatten sind und zudem die Menüs in einer standardisierten Art und Weise organisiert sein sollen (wir hatten bereits am Tag 7 darüber gesprochen). Der Gestaltung des Betriebssystems Microsoft Windows liegt die gleiche Philosophie zu Grunde, d.h. in den meisten Anwendungen kommt ein standardisierter Satz von Menüs zum Einsatz, die in einer standardisierten Reihenfolge angeordnet sind.

Mit der zunehmenden Verbreitung von Windows ereignete sich etwas Wundersames. Die Anwendungsentwickler stellten fest, dass neue Benutzer trotzdem noch Schwierigkeiten hatten, sich in neue Anwendungen einzuarbeiten, und dass fortgeschrittene Benutzer die Menüs zu umständlich fanden. Daraufhin erfanden die Anwendungsgestalter die Symbolleisten als Lösung für beide Probleme.

> *Eine Symbolleiste ist ein schmales Band, das an einem Fensterrahmen angebunden oder als unverankertes Dialogfeld im Anwendungsrahmen verschiebbar ist. Dieses Band (oder Dialogfeld) verfügt über eine Anzahl kleiner Schaltflächen, die mit Bildern »beschriftet« sind, die man als Alternative zu Menübefehlen verwenden kann. Die Anwendungsentwickler platzieren die am häufigsten benutzten Funktionen ihrer Anwendungen auf diesen Symbolleisten und versuchen, die Bilder auf den Schaltflächen möglichst so zu gestalten, dass der Benutzer die Funktionen ohne weiteres erkennen kann.*

Nachdem sich die fortgeschrittenen Benutzer an die Wirkung der Schaltflächen auf der Symbolleiste gewöhnt

hatten, traten die Symbolleisten ihren Siegeszug an. Allerdings hatten neue Benutzer immer noch Probleme, sich mit dem Zweck der Symbolleisten- Schaltflächen bekanntzumachen. Daraufhin setzten sich die Entwickler wieder ans Reißbrett, um neue Wege auszuknobeln, wie man neuen Benutzern den Umgang mit Symbolleisten-Schaltflächen erleichtern kann.

Heraus kam eine Lösung, bei der in einer Informationsleiste, die bereits viele Anwendungen am unteren Rand des Anwendungsfensters vorgesehen hatten, detaillierte Beschreibungen sowohl zu den Menüeinträgen als auch den Symbolleisten-Schaltflächen angezeigt wurden. Eine weitere Lösung war die Anzeige eines kleinen Hilfe-Popup- Fensters mit einer kurzen Beschreibung der Schaltfläche. Dieses Fenster erscheint, wenn man die Maus ein paar Sekunden über der jeweiligen Schaltfläche ruhen lässt. Die erste Lösung wurde als Statusleiste bekannt, die zweite als QuickInfo (ToolTip). Beide Verfahren sind in den meisten heutigen Windows-Anwendungen gängige Praxis.

Wenn Sie eigene Symbolleisten und Statusleisten in Ihren Anwendungen entwerfen und verwenden möchten, könnte man denken, dass Visual C++ eine Menge Unterstützung für Ihre Anstrengungen bereitstellt und sogar die Implementierung erleichtert. Immerhin haben selbst die Anwendungsentwickler von Microsoft den Vorreiter bei der Entwicklung dieser Elemente gespielt und viele, wenn auch nicht alle, Windows-Anwendungen von Microsoft werden mit dem Visual C++ aus dem eigenen Hause entwickelt. Sie haben Recht mit dieser Annahme und werden heute erfahren, wie Sie eigene Symbolleisten und Statusleisten für Ihre Anwendungen erstellen.

11.2 Eine Symbolleiste entwerfen

Zur Einführung in das Thema Symbolleistenentwurf modifizieren wir die am Tag 10, »SDI- und MDI-Anwendungen«, erstellte SDI-Zeichenanwendung. Hier fügen wir eine Symbolleiste ein, mit der sich die Zeichenfarben auswählen lassen.

> *Bevor Sie mit dem heutigen Beispiel beginnen, sollten Sie sich vergewissern, dass Sie das gestrige Beispiel fertig gestellt haben, einschließlich des Übungscodes am Ende der Lektion. Das heutige Beispiel setzt voraus, dass die gesamte Funktionalität vorhanden ist.*

Wenn Sie lediglich ein paar zusätzliche Symbolleisten-Schaltflächen in die vom MFC- Anwendungs-Assistenten für eine SDI- oder MDI-Anwendung erzeugte Standardsymbolleiste aufnehmen wollen, können Sie die Symbolleiste im Symbolleisten- Designer (erreichbar über die Ressourcenansicht) von Visual C++ bearbeiten. Genau wie im Menü-Designer befindet sich am Ende der Symbolleiste ein leeres Feld, das Sie mit einer weiteren Symbolleisten-Schaltfläche ausgestalten können, wie es Abbildung 11.1 zeigt. Dazu brauchen Sie lediglich die leere Schaltfläche zu markieren und nach rechts zu ziehen, wenn ein Zwischenraum (ein Separator) zwischen ihr und der daneben liegenden Schaltfläche entstehen soll, oder die Schaltfläche an eine andere Position ziehen, wenn Sie die Schaltfläche verschieben möchten. Nachdem Sie die Schaltfläche auf der gewünschten Zielposition platziert haben, zeichnen Sie ein Symbol auf die Schaltfläche, das deren Funktion charakterisiert, wenn man die Schaltfläche anklickt. Schließlich markieren Sie die Schaltfläche in der Symbolleistenansicht (die Ansicht, in der alle Schaltflächen der Symbolleiste zu sehen sind), um auf die Eigenschaften der Schaltfläche zuzugreifen. Geben Sie der Schaltfläche die gleiche ID wie dem Menübefehl, der die betreffende Funktion auslöst. Sobald Sie Ihre Anwendung kompilieren und ausführen, haben Sie eine neue Symbolleisten-Schaltfläche, die den jeweiligen Menübefehl realisiert. Wenn Sie eine Symbolleisten-Schaltfläche entfernen wollen, ziehen Sie diese in der Symbolleistenansicht aus der Symbolleiste heraus.

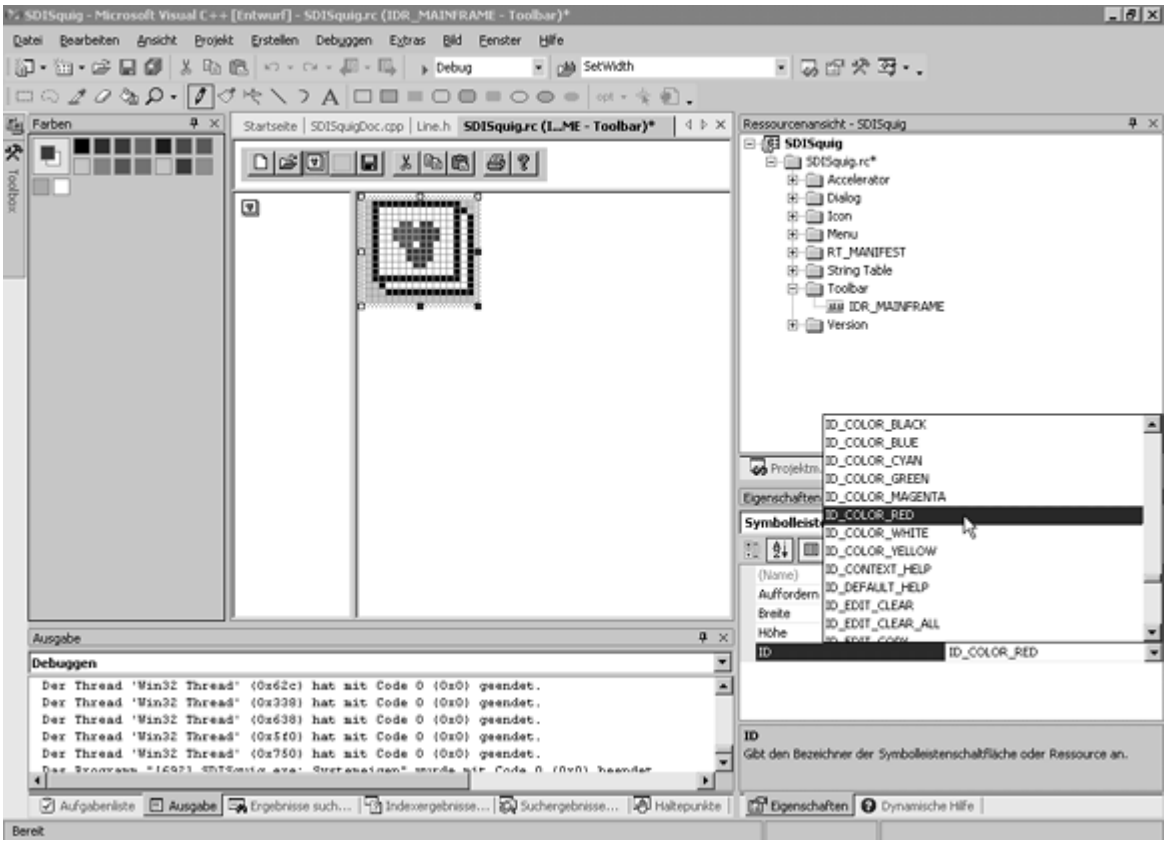


Abbildung 11.1: Der Symbolleisten-Designer

Eine neue Symbolleiste erstellen

Um eine neue Symbolleiste einzufügen, klicken Sie mit der rechten Maustaste auf den Ordner Toolbar und wählen Toolbar einfügen aus dem Kontextmenü. Daraufhin wird eine leere Symbolleiste mit einer einzigen leeren Schaltfläche erzeugt. Sobald Sie ein Symbol auf die jeweils leeren Schaltflächen in der Symbolleiste zeichnen, kommt eine weitere leere Schaltfläche am Ende der Symbolleiste hinzu. Um Farben für die Zeichnung des Symbols verwenden zu können, müssen Sie möglicherweise mit der rechten Maustaste auf die Schaltfläche klicken und aus dem Kontextmenü Fenster »Farben« anzeigen wählen.

Für unsere Zeichenanwendung folgen Sie diesen Schritten:

1. Fügen Sie eine neue Symbolleiste ein, indem Sie in der Ressourcenansicht im Arbeitsbereich den Ordner Toolbar anklicken.

2. Füllen Sie acht Schaltflächen mit den acht verfügbaren Farben des Zeichenprogramms.
3. Nachdem Sie alle Schaltflächen mit Symbolen ausgestattet haben, klicken Sie in der Symbolleistenansicht auf die erste Schaltfläche.
4. Im Feld ID im Eigenschaftenfenster geben Sie die ID des Menübefehls ein, den die Schaltfläche realisieren soll (die ID können Sie auch aus der Dropdown-Liste auswählen).
5. In das Textfeld Auffordern tragen Sie die Beschreibung ein, die in der Statusleiste für diese Schaltfläche erscheinen soll. (Wenn Sie bereits eine Aufforderung für den Menübefehl eingegeben haben, ist in diesem Textfeld schon die Menübeschreibung enthalten.) Am Ende der Beschreibung für die Statusleiste fügen Sie das Zeichen \n und einen kurzen Text für die QuickInfo der Symbolleisten-Schaltfläche an.

In C/C++ steht die Zeichenfolge \n für den Befehl »neue Zeile beginnen«. Im Statuszeilentext für

Symbolleisten-Schaltflächen und Menübefehle dient diese Zeichenfolge dazu, die Beschreibung der Menübefehle in der Statusleiste und die QuickInfos (die erscheinen, wenn man den Mauszeiger ein paar Sekunden über die Schaltfläche bewegt) zu trennen. Die erste Zeile ist für die Beschreibung der Statusleiste vorgesehen, die zweite für den Text in den QuickInfos. Die Beschreibung für QuickInfos kommt nur bei Symbolleisten zum Einsatz, sodass man bei Menübefehlen ohne Symbolleisten-Äquivalente darauf verzichten kann.

Beispielsweise können Sie ID_COLOR_BLACK als ID festlegen und Zeichenfarbe Schwarz\nSchwarz als Statuszeilentext für die schwarze Schaltfläche auf der Symbolleiste, die Sie für Ihr Zeichenprogramm erstellen (siehe Abbildung 11.2).

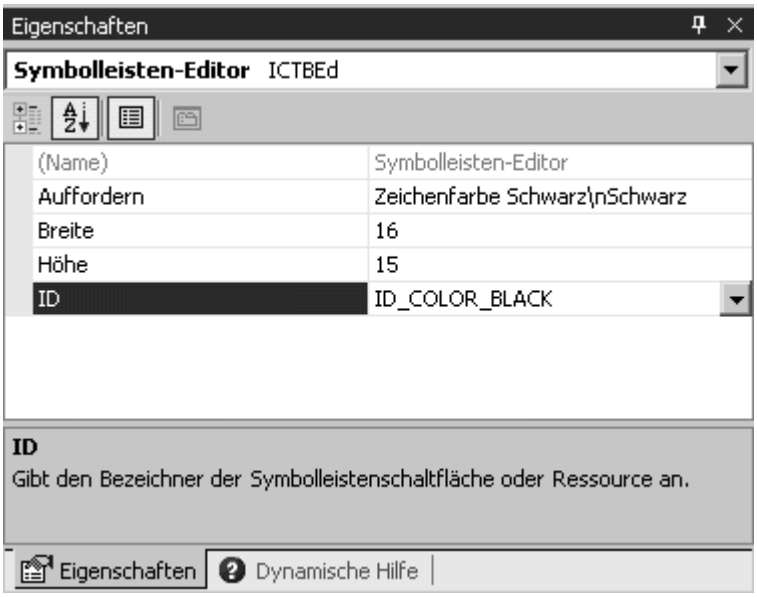


Abbildung 11.2: Die Eigenschaftenansicht für Schaltflächen

Nachdem Sie den Entwurf der Symbolleiste fertig gestellt (die Schaltflächen mit Symbolen ausgestattet und die Eigenschaften für jede Schaltfläche festgelegt) haben, ändern Sie mit folgenden Schritten die ID der Symbolleiste:

1. In der Ressourcenansicht klicken Sie auf die neu hinzugefügte Symbolleiste.

2. Ändern Sie im Eigenschaftenfenster die ID der Symbolleiste in IDR_TBCOLOR.

Die Symbolleiste mit dem Anwendungsgerüst verbinden

In der bisherigen SDI-Anwendung haben Sie keinerlei Funktionalität hinzufügt, die eine Arbeit mit dem Rahmenfenster erfordert hätte. Jetzt, da die Symbolleiste mit dem Rahmen verbunden wird, müssen Sie den Code in diesem Modul hinzufügen und modifizieren. Wenn Sie die Klasse CMainFrame öffnen und zur Funktion OnCreate gehen, sehen Sie, wo die vorhandene Symbolleiste erzeugt und (weiter unten in der Funktion) mit dem Rahmen verbunden wird.

Bevor Sie die Symbolleiste mit dem Anwendungsgerüst verbinden können, müssen Sie zunächst eine Variable der Klasse CMainFrame hinzufügen, um den Wert der neuen Symbolleiste aufzunehmen. Für diese Variable vom Typ CToolBar legen Sie den Zugriffsstatus protected fest.

Um die Symbolleiste Farben in Ihre Zeichenanwendung aufzunehmen, folgen Sie diesen Schritten:

1. Klicken Sie mit der rechten Maustaste in der Klassenansicht auf die Klasse CMainFrame. Wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.

2. Legen Sie den Variablentyp mit CToolBar, den Namen als m_wndColorBar und den Zugriff als protected fest.

Jetzt schreiben Sie noch etwas Code in die Funktion OnCreate der Klasse CMainFrame, um die Symbolleiste hinzuzufügen und sie mit dem Gerüst zu verbinden. Übernehmen Sie die Modifikationen aus Listing 11.1, um die Symbolleiste Farben in Ihre Zeichenanwendung einzubauen.

Listing 11.1: Die modifizierte Funktion CMainFrame.OnCreate

```
 1: int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
 2: {
 3:   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
 4:     return -1;
 5:
 6:   if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
 7:       WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
 8:       CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
 9:       !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
10:   {
11:     TRACE0("Symbolleiste konnte nicht erstellt werden\n");
12     return -1;      // Fehler bei Erstellung
13:   }
14:
15:   // Symbolleiste Farben hinzufügen
16:   int iTBCtlID;
17:   int i;
18:
19:   // Symbolleiste Farben erzeugen
20:   if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
21:       WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
22:       CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
23:       !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
24:   {
25:     TRACE0("Symbolleiste konnte nicht erstellt werden\n");
26:     return -1;    // Fehler bei Erstellung
27:   }
28:   // Schaltfläche Schwarz auf der Symbolleiste suchen
29:   iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
30:   if (iTBCtlID >= 0)
31:   {
32:     // Schleife durch die Schaltflächen, um sie als
33:     // Optionsfelder wirken zu lassen
34:     for (i = iTBCtlID; i < (iTBCtlID + 8); i++)
35:       m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
36:   }
37:   if (!m_wndStatusBar.Create(this) ||
38:       !m_wndStatusBar.SetIndicators(indicators,
39:       sizeof(indicators)/sizeof(UINT)))
40:   {
41:     TRACE0("Symbolleiste konnte nicht erstellt werden\n");
42:     return -1;      // Fehler bei Erstellung
43:   }
44:   // TODO: Löschen Sie diese drei Zeilen, wenn Sie nicht
45:   // möchten, dass die Systemleiste andockbar ist
46:   m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
47:   // Andocken für Symbolleiste Color aktivieren
48:   m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
49:   EnableDocking(CBRS_ALIGN_ANY);
50:   DockControlBar(&m_wndToolBar);
51:
52:   // Symbolleiste Farben andocken
53:   DockControlBar(&m_wndColorBar);
54:   return 0;
55: }
```

Die Symbolleiste erstellen

Der erste hinzugefügte Code-Abschnitt

```
if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
    WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
    CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
    !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
```

enthält zwei separate Funktionen, die für das Erstellen einer Symbolleiste erforderlich sind. Die erste Funktion, CreateEx, erstellt die Symbolleiste selbst, während die zweite Funktion, LoadToolBar, die Symbolleiste lädt, die Sie im Symbolleisten-Designer entworfen haben. Die zweite Funktion, LoadToolBar, erfordert als Argument die ID für die zu erzeugende Symbolleiste.

Der Funktion CreateEx können Sie mehrere Argumente übergeben. Das erste - und einzig erforderliche - Argument ist ein Zeiger auf das übergeordnete Fenster. In diesem Fall (der den Normalfall darstellt) ist dieses Argument ein Zeiger auf das Rahmenfenster, mit dem die Symbolleiste verbunden wird.

Das zweite Argument gibt den Stil der Steuerelemente in der zu erzeugenden Symbolleiste an. Es stehen mehrere Stile zur Auswahl (siehe Tabelle 11.1), von denen einige mit den letzten Versionen des Internet Explorers eingeführt wurden.

Stil	**Beschreibung**
TBSTYLE_ALTDRAG	Erlaubt dem Benutzer, die Symbolleiste bei gedrückter (Alt)-Taste zu verschieben.
TBSTYLE_CUSTOMERASE	Generiert eine NM_CUSTOMDRAW-Nachricht, wenn man die Symbolleiste und den Schaltflächenhintergrund löscht. Damit kann der Programmierer wählen, wann und ob das Löschen des Hintergrunds zu steuern ist.
TBSTYLE_FLAT	Erzeugt eine flache Symbolleiste. Der Schaltflächentext erscheint unter der Bitmap.
TBSTYLE_LIST	Der Schaltflächentext steht rechts neben der Bitmap.
TBSTYLE_REGISTERDROP	Beim Drag&Drop von Objekten auf Symbolleisten-Schaltflächen verwendet
TBSTYLE_TOOLTIPS	Erzeugt ein QuickInfo-Steuerelement, in dem man eine Beschreibung der Symbolleisten-Schaltflächen anzeigen kann
TBSTYLE_TRANSPARENT	Erzeugt eine transparente Symbolleiste
TBSTYLE_WRAPABLE	Erzeugt eine Symbolleiste, die aus mehreren Schaltflächenreihen bestehen kann

Tabelle 11.1: Stile für die Steuerelemente von Symbolleisten

Das dritte Argument gibt den Stil der Symbolleiste selbst an. Dieses Argument ist normalerweise eine Kombination aus Fenster- und Steuerelementstilen. In der Regel verwendet man nur zwei oder drei Fensterstile, während es sich beim Rest der Symbolleistenstile (siehe Tabelle 11.2) um Steuerelementstile handelt.

Stil	**Beschreibung**
WS_CHILD	Die Symbolleiste wird als untergeordnetes Fenster geöffnet.
WS_VISIBLE	Die Symbolleiste ist sichtbar, wenn sie erzeugt wird.
CBRS_ALIGN_TOP	Die Symbolleiste lässt sich am oberen Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_BOTTOM	Die Symbolleiste lässt sich am unteren Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_LEFT	Die Symbolleiste lässt sich an der linken Seite des Anzeigebereichs im Rahmenfenster verankern.

CBRS_ALIGN_RIGHT	Die Symbolleiste lässt sich an der rechten Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_ANY	Die Symbolleiste lässt sich an einer beliebigen Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_BORDER_TOP	Zeichnet einen Rahmen am oberen Rand der Symbolleiste, wenn die Symbolleiste nicht verankert ist
CBRS_BORDER_BOTTOM	Zeichnet einen Rahmen am unteren Rand der Symbolleiste, wenn die Symbolleiste nicht verankert ist
CBRS_BORDER_LEFT	Zeichnet einen Rahmen an den linken Rand der Symbolleiste, wenn die Symbolleiste nicht verankert ist
CBRS_BORDER_RIGHT	Zeichnet einen Rahmen an den rechten Rand der Symbolleiste, wenn die Symbolleiste nicht verankert ist
CBRS_FLOAT_MULTI	Gestattet mehrere unverankerte Symbolleisten in einem einzelnen Minirahmenfenster anzuzeigen
CBRS_TOOLTIPS	Bewirkt die Anzeige von QuickInfos für die Symbolleisten-Schaltflächen
CBRS_FLYBY	Bewirkt die gleichzeitige Aktualisierung des Textes in der Statusleiste für Symbolleisten-Schaltflächen und QuickInfos
CBRS_GRIPPER	Zeichnet eine Griffleiste auf die Symbolleiste

Tabelle 11.2: Stile für Symbolleisten

Das vierte Argument, welches Sie in Ihrem Code nicht bereitstellen, gibt die Größe der Symbolleistenrahmen an. Dieses Argument wird als normale CRect-Klasse übergeben, um die gewünschte Länge und Höhe für die Symbolleiste bereitzustellen. Der Standardwert ist 0 für alle Abmessungen des Rechtecks, sodass die Symbolleiste keine Rahmen aufweist.

Das fünfte und letzte Argument, das Sie ebenfalls nicht in Ihrem Code verwenden, stellt die ID des untergeordneten Symbolleistenfensters bereit. Der Standardwert lautet AFX_IDW_TOOLBAR, man kann aber alle definierten IDs verwenden, die für die Symbolleiste erforderlich sind oder geeignet erscheinen.

Die Stile der Schaltflächen festlegen

An das Erstellen der Symbolleiste schließt sich ein etwas eigentümlicher Code-Abschnitt an:

```
// Schaltfläche Schwarz auf der Symbolleiste suchen
iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
if (iTBCtlID >= 0)
{
  // Schleife durch die Schaltflächen, um sie als Optionsfenter
  // wirken zu lassen
  for (i= iTBCtlID; i < (iTBCtlID + 8); i++)
    m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
}
```

Die erste Zeile in diesem Code-Fragment verwendet die Symbolleistenfunktion CommandToIndex, um die Steuerelementnummer der Schaltfläche ID_COLOR_BLACK zu ermitteln. Wenn Sie die Symbolleiste in der Reihenfolge der Farben, wie sie im Menü erscheinen, erstellt haben, sollte es sich dabei um das erste Steuerelement mit einem Index von 0 handeln. Es empfiehlt sich, den Index aller Symbolleisten-Schaltflächen, die man ändern muss, mit der Funktion CommandToIndex zu ermitteln, einfach deshalb, weil man nicht immer davon ausgehen kann, dass die Schaltfläche an der erwarteten Position sitzt. Die Funktion liefert den Index des angegebenen Symbolleisten-Steuerelements zurück. Diesen Wert verwendet man als Ausgangspunkt, um den Schaltflächenstil der Farbschaltflächen festzulegen.

In der Schleife, in der Sie alle acht Farbschaltflächen der Symbolleiste durchlaufen, steuern Sie mit der Funktion SetButtonStyle das Verhalten der Symbolleisten- Schaltflächen. Das erste Argument an diese Funktion ist der Index der Schaltfläche, die zu ändern ist. Das zweite Argument gibt den Stil für die

betreffende Schaltfläche an. In diesem Fall legen wir fest, dass alle Schaltflächen den Stil TBBS_CHECKGROUP aufweisen. Damit verhalten sich die Schaltflächen wie Optionsfelder, wobei nur eine Schaltfläche in der Gruppe zu einem bestimmten Zeitpunkt ausgewählt sein kann. Die Liste der verfügbaren Schaltflächenstile gibt Tabelle 11.3 wieder.

Stil	Beschreibung
TBBS_AUTOSIZE	Die Breite der Schaltfläche wird nach dem Text auf der Schaltfläche berechnet.
TBBS_BUTTON	Erzeugt eine Standardschaltfläche
TBBS_CHECKBOX	Erzeugt eine Schaltfläche, die sich wie ein Kontrollkästchen verhält und zwischen dem gedrückten und nicht gedrückten Zustand umschaltet
TBBS_CHECKGROUP	Erzeugt eine Schaltfläche, die sich wie ein Optionsfeld verhält. Die Schaltfläche bleibt im gedrückten Zustand, bis eine andere Schaltfläche in der Gruppe betätigt wird. Es handelt sich hier eigentlich um eine Kombination der Schaltflächenstile TBSTYLE_CHECK und TBSTYLE_GROUP.
TBBS_DROPDOWN	Erzeugt eine Schaltfläche in der Art einer DropDown-Liste
TBBS_GROUP	Erstellt eine Schaltfläche, die gedrückt bleibt, bis eine andere Schaltfläche in der Gruppe gedrückt wird
TBBS_NOPREFIX	Der Schaltflächentext weist kein zugeordnetes Präfix auf.
TBBS_SEPARATOR	Erzeugt einen Separator, d.h. eine kleine Lücke zwischen den Schaltflächen zu beiden Seiten

Tabelle 11.3: Stile für Symbolleisten-Schaltflächen

Die Symbolleiste verankern

Der letzte Code-Abschnitt, den Sie in die Funktion OnCreate der Klasse CMainFrame aufgenommen haben, sieht folgendermaßen aus:

```
// Andocken für Symbolleiste Color aktivieren
m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);  // (Vom Anwendungs-Assistenten generierte Zeile)
// Symbolleiste Color andocken
DockControlBar(&m_wndColorBar);
```

In der ersten Zeile steht der Aufruf der Funktion EnableDocking. Diese Funktion erlaubt das Verankern der Symbolleiste mit dem Rahmenfenster. Der an diese Symbolleistenfunktion übergebene Wert muss mit dem Wert übereinstimmen, der im nachfolgenden Aufruf der Funktion EnableDocking für das Rahmenfenster übergeben wird. Die verfügbaren Werte für diese Funktionen sind in Tabelle 11.4 aufgeführt. Mit diesen Funktionen lassen sich die Rahmen der Symbolleiste und das Rahmenfenster für das Andocken aktivieren. Ruft man diese Funktionen nicht auf, kann man die Symbolleiste nicht mit dem Rahmenfenster verankern. Übergibt man eine bestimmte Seite für das Andocken in diesen Funktionen und die Seite entspricht nicht diesem Wert, lässt sich die Symbolleiste ebenfalls nicht am Rahmenfenster verankern.

Stil	Beschreibung
CBRS_ALIGN_TOP	Die Symbolleiste lässt sich am oberen Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_BOTTOM	Die Symbolleiste lässt sich am unteren Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_LEFT	Die Symbolleiste lässt sich an der linken Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_RIGHT	Die Symbolleiste lässt sich an der rechten Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_ANY	Die Symbolleiste lässt sich an einer beliebigen Seite des Anzeigebereichs im

	Rahmenfenster verankern.
CBRS_FLOAT_MULTI	Gestattet mehrere unverankerte Symbolleisten in einem einzelnen Minirahmenfenster
0	Die Symbolleiste lässt sich nicht mit dem Rahmen verankern.

Tabelle 11.4: Seiten, an denen man Symbolleisten verankern kann

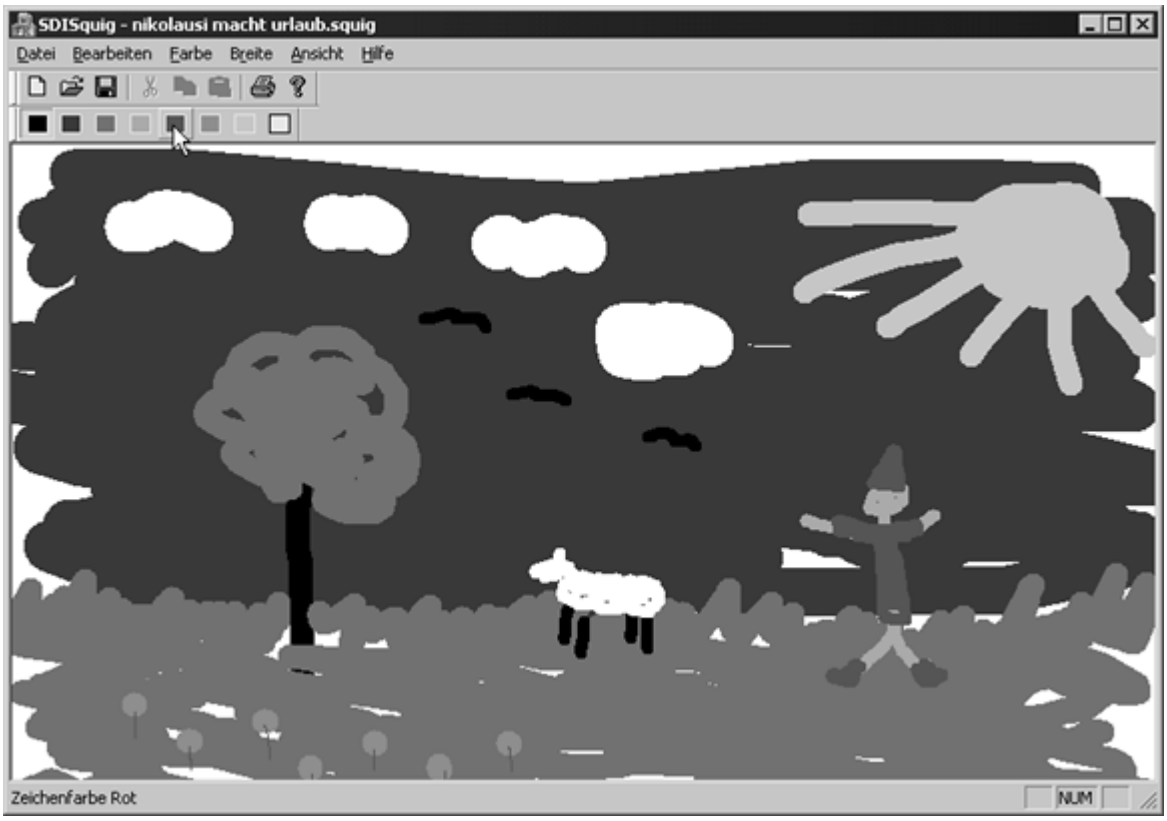


Abbildung 11.3: Die Symbolleiste Color im Zeichenprogramm

Als letzte Funktion wurde die Fensterfunktion DockControlBar hinzugefügt, der die Adresse der Symbolleisten-Variablen übergeben wird. Diese Funktion verankert die Symbolleiste physisch mit dem Rahmenfenster. Da der gesamte Code in der Funktion OnCreate für das Rahmenfenster erscheint, wird die Symbolleiste verankert, bevor der Benutzer das Fenster oder die Symbolleiste zu Gesicht bekommt.

Nachdem Sie nun den gesamten Code in die Funktion OnCreate der Klasse CMainFrame aufgenommen haben, können Sie Ihre Anwendung kompilieren und ausführen. Sie finden eine funktionsfähige Symbolleiste Farben vor, über die sich die Zeichenfarbe auswählen lässt (siehe Abbildung 11.3).

MFC-Exkurs: Die Klasse CToolBar

Die Klasse CToolBar verkapselt die Funktionalität der Symbolleisten-Steuerelemente. Sie benötigen möglicherweise viel von der Funktionalität in dieser Klasse. Wahrscheinlich benötigen Sie auch Funktionalität aus der Vorfahr-Klasse, CControlBar. Die Erstellungsfunktionalität, zu der unter anderem die Funktionen CreateEx, LoadToolBar, CommandToIndex und SetButtonStyle gehören, haben Sie bereits kennen gelernt. Von Zeit zu Zeit müssen Sie möglicherweise weitere Funktionen verwenden.

Informationen über Symbolleisten-Schaltflächen abfragen

Wenn Sie mit den Schaltflächen auf einer Symbolleiste arbeiten, müssen Sie ab und an alle durchprobieren, um diejenige zu finden, die Sie brauchen. Sie können die Funktion GetItemID verwenden, um die Befehls-ID

einer Schaltfläche abzufragen. Dieser Funktion übergeben Sie die Nummer der Schaltfläche auf der Symbolleiste (von links mit der Nummer 0 beginnend) und sie gibt die ID der Schaltfläche zurück.

Wenn Sie die Position und Abmessungen einer Schaltfläche wissen möchten, verwenden Sie die Funktion GetItemRect. Sie übergeben dieser Funktion die Nummer der Schaltfläche auf der Symbolleiste und einen Zeiger auf ein CRect-Objekt, das die Funktion mit der Position und den Abmessungen der Schaltfläche bestückt.

Wenn Sie den Stil einer Schaltfläche wissen möchten, verwenden Sie eine der Funktion SetButtonStyle entsprechende Funktion, GetButtonStyle. Sie übergeben einfach GetButtonStyle die Nummer der zu überprüfenden Schaltfläche und es gibt einen Integer- Wert zurück, der mit den in Tabelle 11.3 aufgelisteten Stilen verglichen werden kann.

Symbolleistentext abfragen und setzen

In Fällen, in denen Sie einer Symbolleisten-Schaltfläche einen Aufforderungstext hinzufügen müssen, können Sie die Funktion SetButtonText verwenden. Wenn Sie den Text einer Schaltfläche abfragen möchten, verwenden Sie die Funktion GetButtonText. Sie übergeben der Funktion GetButtonText einfach die Nummer der Schaltfläche und sie liefert ein CString-Objekt zurück, das den Schaltflächentext enthält.

MFC-Exkurs: Die Klasse CControlBar

Die Klasse CControlBar ist die Basisklasse für alle Steuerleisten-Objektklassen, einschließlich CToolBar, CStatusBar, CControlBar, CReBar und COleResizeBar. CControlBar stellt einen großen Teil der zu Grunde liegenden und gebräuchlichen Funktionalität für all diese Steuerelemente bereit. Gewöhnlich arbeiten Sie nicht direkt mit CControlBar, sondern interagieren mit der abgeleiteten Klasse (wie CToolBar) und rufen die Funktionen der Basisklasse auf, als gehörten sie zur abgeleiteten Klasse.

Wenn Sie den Stil der Steuerleiste abfragen oder setzen möchten, können Sie die Funktion GetBarStyle zum Abfragen und die Funktion SetBarStyle zum Setzen verwenden. Die Funktion GetBarStyle gibt eine DWORD-Variable zurück, die den Stil angibt. Die Funktion SetBarStyle übernimmt einen einzelnen DWORD-Parameter, der den Stil angibt. Mögliche Stile finden Sie in Tabelle 11.2.

Eine nützliche Member-Funktion, mit der man die Anzahl der Elemente auf einer Symbolleiste ermitteln kann, ist GetCount, das einen Integer zurückgibt, der die Anzahl der Schaltflächen und Separatoren auf der Symbolleiste angibt. Eine weitere gute informative Funktion ist IsFloating, das einen Booleschen Wert zurückgibt, der anzeigt, ob die Symbolleiste (nicht) angedockt ist oder frei schwebt.

Wenn Sie einen Zeiger auf das Rahmenfenster benötigen, an dem die Symbolleiste angedockt ist, rufen Sie die Funktion GetDockingFrame auf, die einen Zeiger auf das Rahmenfenster zurückgibt. Die letzte Funktion, EnableDocking, haben Sie bereits kennen gelernt.

Die Sichtbarkeit der Symbolleiste steuern

Nunmehr verfügen Sie über eine Symbolleiste Farben, die Sie mit dem Rahmen Ihres Zeichenprogramms verankern können. Es wäre schön, wenn man die Symbolleiste genauso über das Menü Ansicht anzeigen und ausblenden könnte, wie es mit der Standardsymbolleiste und der Statusleiste möglich ist. Diese Funktionalität lässt sich recht einfach implementieren, aber es funktioniert nicht unbedingt so, wie Sie es vielleicht erwarten.

Als Erstes ist ein Menübefehl hinzuzufügen, um die Sichtbarkeit der Farbpalette umzuschalten:

1. Nehmen Sie im Menü-Designer einen neuen Menübefehl in das Menü Ansicht auf.

2. Legen Sie die Eigenschaften des Menüs wie folgt fest:

Eigenschaft	Einstellung
ID	ID_VIEW_COLORBAR

Beschriftung	&Farbpalette
Eingabeaufforderung	Blendet die Farbpalette ein oder aus\nFarbpalette ein/aus

Tabelle 11.5: Eigenschaften des Menübefehls Farbpalette

Das Menü aktualisieren

Um zu ermitteln, ob die Symbolleiste sichtbar oder ausgeblendet ist, kann man den aktuellen Stil der Symbolleiste lesen und das Stilflag WS_VISIBLE ausmaskieren. Wenn das Flag im aktuellen Symbolleistenstil enthalten ist, ist die Symbolleiste sichtbar. Indem man diese Auswertung in die Funktion SetCheck in der Behandlungsroutine UPDATE_COMMAND_UI schreibt, lässt sich das Kontrollhäkchen neben dem Menüeintrag für die Farbpalette umschalten. Sie können auch die Funktion IsVisible aufrufen, die einen Booleschen Wert zurückgibt, der anzeigt, ob die Symbolleiste sichtbar ist.

Um diese Funktionalität hinzuzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie eine Behandlungsroutine für das Ereignis UPDATE_COMMAND_UI des Menüs ID_VIEW_COLORBAR auf. Achten Sie darauf, diese Behandlungsroutine in die Klasse CMainFrame aufzunehmen. (Bisher führen Sie alle Änderungen in der Rahmenklasse durch.)

2. In die Funktion schreiben Sie den Code von Listing 11.2.

Listing 11.2: Die modifizierte Funktion CMainFrame.OnUpdateViewColorbar

```
1: void CMainFrame::OnUpdateViewColorbar(CCmdUI *pCmdUI)
2: {
3:    // TODO: Fügen Sie hier Ihren
4:    // Befehlsaktualisierungs-UI-Behandlungscode ein.
5:    // Zustand der Farbpalette prüfen
6:    pCmdUI->SetCheck(m_wndColorBar.IsVisible());
7: }
```

Die Sichtbarkeit der Symbolleiste umschalten

Da die Klasse CToolBar von der Klasse CWnd (über die Klasse CControlBar) abgeleitet ist, könnte man meinen, dass man die Funktion ShowWindow auf der Symbolleiste selbst aufrufen kann, um die Symbolleiste anzuzeigen und auszublenden. Das ist zwar möglich, aber der Hintergrund für die Symbolleiste wird nicht zusammen mit der Symbolleiste ausgeblendet. Der Benutzer stellt lediglich fest, dass die Symbolleisten-Schaltflächen erscheinen und verschwinden. (Natürlich ist das die Wirkung, die Sie erzielen möchten, aber Ihre Benutzer werden davon nicht sehr angetan sein.)

Stattdessen greifen Sie auf eine Funktion ShowControlBar für das Rahmenfenster zurück, um die Symbolleiste anzuzeigen und auszublenden. Diese Funktion übernimmt drei Argumente:

- Die Adresse für die Symbolleisten-Variable.
- Ein Boolescher Wert, der angibt, ob die Symbolleiste anzuzeigen ist (TRUE zeigt die Symbolleiste an, FALSE blendet sie aus.)
- Ein Boolescher Wert, der angibt, ob die Symbolleiste verzögert erscheinen soll (TRUE zeigt die Symbolleiste verzögert, FALSE sofort an.)

Wenn eine Symbolleiste ein- oder ausgeschaltet wird, müssen Sie eine weitere Funktion für das Rahmenfenster, RecalcLayout, aufrufen. Diese Funktion bewirkt, dass der Rahmen alle Symbolleisten, Statusleisten und alles innerhalb des Rahmenbereichs neu positioniert. Diese Funktion ist dafür verantwortlich, dass sich die Symbolleiste Farben nach oben oder unten verschiebt, wenn Sie die Standardsymbolleiste ein- und ausschalten.

Diese Funktionalität nehmen Sie mit folgenden Schritten in Ihr Zeichenprogramm auf:

1. Nehmen Sie eine Behandlungsroutine für das Ereignis COMMAND des Menübefehls ID_VIEW_COLORBAR auf. Achten Sie darauf, dass Sie diese Behandlungsfunktion in die Klasse CMainFrame aufnehmen. (Bisher führen Sie alle Änderungen in der Rahmenklasse durch.)

2. In die Funktion schreiben Sie den Code aus Listing 11.3.

Listing 11.3: Die modifizierte Funktion CMainFrame.OnViewColorbar

```
1: void CMainFrame::OnViewColorbar()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Symbolleiste Farben umschalten
5:    ShowControlBar(&m_wndColorBar, !m_wndColorBar.IsVisible(),
6:                                   FALSE);
7:    // Elemente im Rahmenfenster neu anordnen
8:    RecalcLayout();
9: }
```

Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, sollten Sie die Symbolleiste Farben über das Menü Ansicht ein- und ausschalten können.

11.3 Die Symbolleiste mit einem Kombinationsfeld ausstatten

Mittlerweile ist es üblich, dass die Symbolleisten einer Anwendung neben einfachen Schaltflächen auch noch andere Elemente enthalten. Sehen Sie sich zum Beispiel das Visual Studio an. Hier finden Sie ein Kombinationsfeld, mit dem Sie den Build-Modus auswählen können, in dem Sie arbeiten möchten. Wie fügt man nun ein Kombinationsfeld in eine Symbolleiste ein? Der Symbolleisten-Editor stellt es nicht zur Verfügung. Hier finden Sie nur Schaltflächen, auf die Sie Symbole zeichnen können. Mit den Assistenten von Visual C++ lassen sich keine Kombinationsfelder in Symbolleisten aufnehmen. Zu diesem Zweck müssen Sie ein wenig C++-Code schreiben.

Als Einstieg fügen Sie ein Kombinationsfeld in die Symbolleiste Farben ein, die Sie gerade erstellt haben. Mit dem Kombinationsfeld wählt der Benutzer die Breite des Zeichenstifts aus. (Wenn Sie die Unterstützung für unterschiedliche Zeichenbreiten gemäß Übung am Ende von Tag 10 noch nicht realisiert haben, sollten Sie das jetzt nachholen.)

Die Projektressourcen bearbeiten

Um ein Kombinationsfeld in die Symbolleiste aufzunehmen, ist zuerst das zu tun, woran Sie Visual C++ bis vor Kurzem hindern sollte. Sie müssen die Ressourcendatei selbst bearbeiten. Mit den visuellen Werkzeugen des Visual C++ Developer Studios ist das nicht möglich. Wenn Sie versuchen, die Ressourcendatei in der Ressourcenansicht zu öffnen, bearbeiten Sie sie unweigerlich durch die verschiedenen Ressourcen-Editoren und - Designer. Nein, Sie müssen diese Datei in einem anderen Editor bearbeiten.

Gehen Sie in die Projektmappen-Explorer-Ansicht und suchen Sie die .rc-Datei. Markieren Sie diese Datei und klicken Sie mit der rechten Maustaste. Wählen Sie aus dem Menü Öffnen mit.... . Sie erhalten eine Liste der verfügbaren Editoren. Wählen Sie Quellcode-Editor (Text) und klicken Sie auf Öffnen. Wenn Sie die Datei geöffnet haben, scrollen Sie nach unten bis zu den Symbolleisten-Definitionen. (Sie können nach dem Wort »toolbar« suchen.) Dann gehen Sie ans Ende der Symbolleisten-Definition und fügen zwei Separatorlinien am unteren Rand der Symbolleisten-Definition ein.

Um diese Änderungen an Ihrer Zeichenanwendung vorzunehmen, folgen Sie diesen Schritten:

1. Gehen Sie zum Projektmappen-Explorer.

2. Suchen Sie die Datei SDISquig.rc und markieren Sie diese. Klicken Sie mit der rechten Maustaste und wählen Sie Öffnen mit... aus dem Kontextmenü.
3. Wählen Sie Quellcode-Editor (Text) aus dem Dialogfeld Öffnen mit (siehe Abbildung 11.4). Klicken Sie auf Öffnen, um die Ressourcendatei zu bearbeiten. Wenn die Ressourcenansicht oder einer der Ressourcen-Editoren geöffnet ist, werden Sie gefragt, ob Sie ihn schließen möchten. In diesem Fall antworten Sie mit Ja.

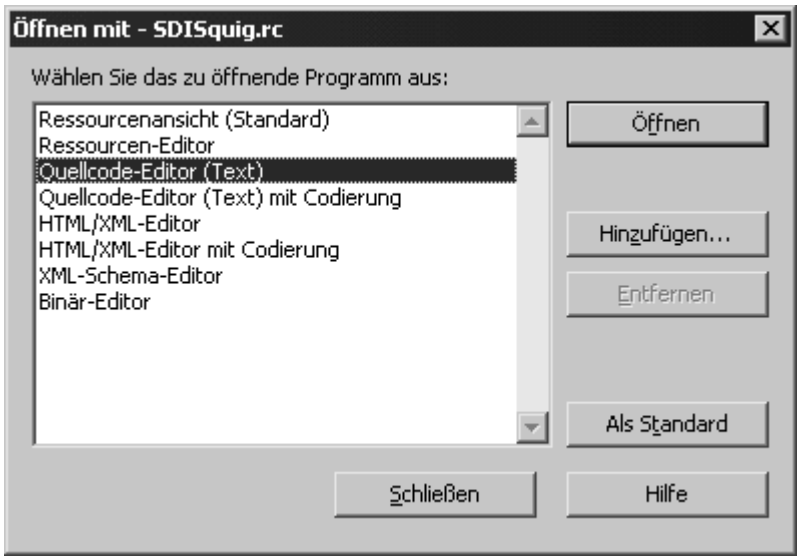


Abbildung 11.4: Einen Editor für die Ressourcendatei des Projekts auswählen

4. Suchen Sie nach dem Symbolleisten-Abschnitt und fügen Sie dann zwei SEPARATOR- Zeilen unmittelbar vor dem Ende des Abschnitts IDR_TBCOLOR ein, wie es Listing 11.4 zeigt.
5. Nachdem Sie die SEPARATOR-Zeilen hinzugefügt haben, speichern Sie die Datei und klicken auf das x rechts über dem Editorbereich, um die Projektressourcendatei zu schließen.

Listing 11.4: Die modifizierte Projektressourcendatei (Toolbar.rc)

```
 1: //////////////////////////////////////////////////////////////////
 2: //
 3: // Toolbar
 4: //
 5:
 6: IDR_MAINFRAME TOOLBAR  16, 15
 7: BEGIN
 8:     BUTTON      ID_FILE_NEW
 9:     BUTTON      ID_FILE_OPEN
10:     BUTTON      ID_FILE_SAVE
11:     SEPARATOR
12:     BUTTON      ID_EDIT_CUT
13:     BUTTON      ID_EDIT_COPY
14:     BUTTON      ID_EDIT_PASTE
15:     SEPARATOR
16:     BUTTON      ID_FILE_PRINT
17:     BUTTON      ID_APP_ABOUT
18:     SEPARATOR
19:     BUTTON      ID_VIEW_COLORBAR
20: END
21:
22: IDR_TBCOLOR TOOLBAR  16, 15
23: BEGIN
24:     BUTTON      ID_COLOR_BLACK
25:     BUTTON      ID_COLOR_BLUE
26:     BUTTON      ID_COLOR_GREEN
27:     BUTTON      ID_COLOR_CYAN
28:     BUTTON      ID_COLOR_RED
29:     BUTTON      ID_COLOR_MAGENTA
30:     BUTTON      ID_COLOR_YELLOW
31:     BUTTON      ID_COLOR_WHITE
32:     SEPARATOR
33:     SEPARATOR
34: END
```

Die beiden SEPARATOR-Zeilen haben Sie in die Symbolleisten-Definition aufgenommen, damit der zweite Separator als Platzhalter für das Kombinationsfeld dienen kann, das Sie in die Symbolleiste einbauen wollen. Es gibt zwei Gründe dafür, dass Sie diese Bearbeitung manuell erledigen müssen und nicht den Symbolleisten-Designer von Visual C++ bemühen können:

- Der Symbolleisten-Designer erlaubt nicht, dass man mehrere Separatoren an das Ende der Symbolleiste anfügt.
- Wenn Sie nichts anderes nach dem Separator an das Ende der Symbolleiste anfügen, nimmt der Symbolleisten-Designer einen Fehler an und entfernt den Separator. Mit anderen Worten: Mit dem Symbolleisten-Designer von Visual C++ lässt sich kein Platzhalter für das Kombinationsfeld in die Symbolleiste einbauen.

Als Nächstes sind die Textzeichenfolgen hinzuzufügen, die Sie in das Kombinationsfeld laden. Um diese Strings aufzunehmen, öffnen Sie die Zeichenfolgentabelle String Table in der Ressourcenansicht des Arbeitsbereichs. Hier finden Sie alle Zeichenketten, die Sie als Statuszeilentext in verschiedenen Eigenschaftsdialogfeldern eingegeben haben. Diese Tabelle enthält eine Anzahl von IDs, die Werte dieser IDs und die Strings, die mit diesen IDs verbunden sind (siehe Abbildung 11.5). Die Strings, die Sie in das Kombinationsfeld eintragen möchten, müssen Sie in die Zeichenfolgentabelle schreiben. Jede Zeile in der Dropdown-Liste muss über eine eindeutige ID und einen Eintrag in der Zeichenfolgentabelle verfügen.

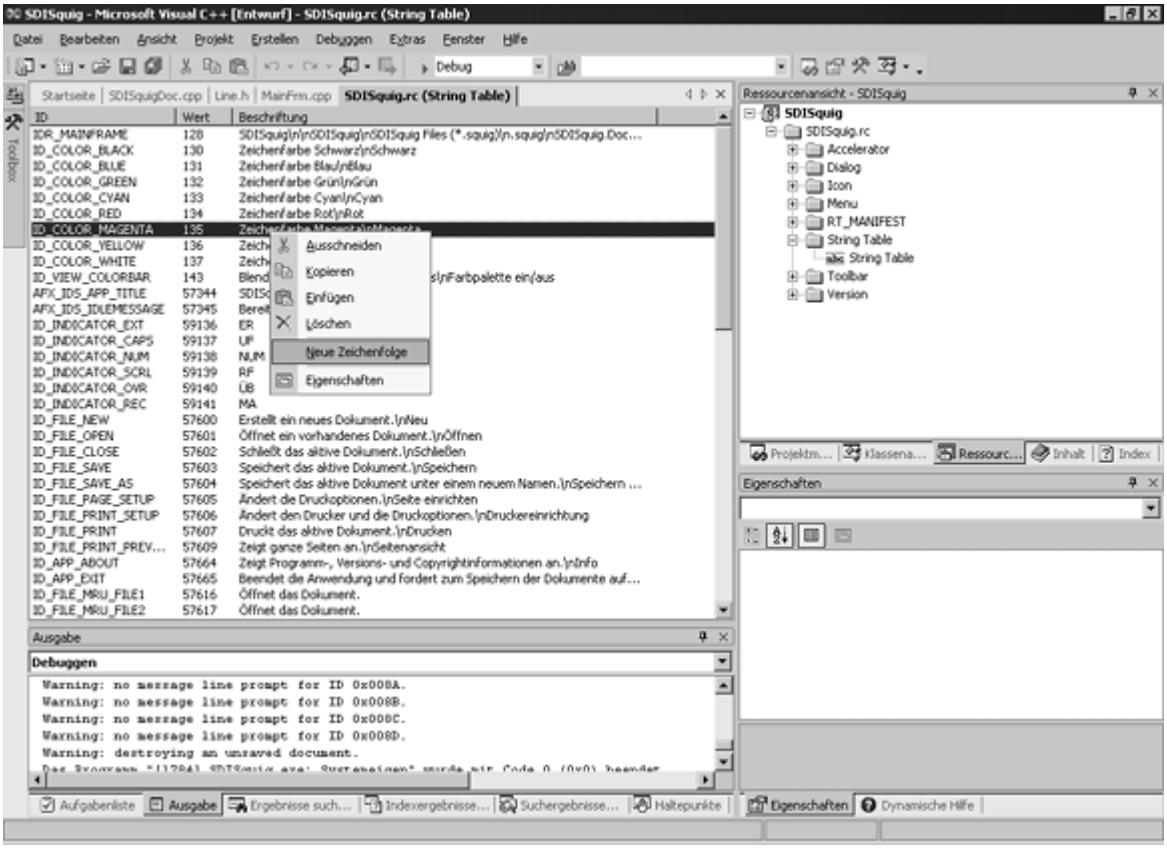


Abbildung 11.5: Der Stringtabellen-Editor

Um die Strings für das Kombinationsfeld hinzuzufügen, das Sie in die Symbolleiste Farben einbauen möchten, gehen Sie nach den folgenden Schritten vor:

1. Fügen Sie einen neuen String ein, indem Sie mit der rechten Maustaste auf die Zeichenfolgentabelle klicken und Neue Zeichenfolge wählen. Sie können auch ans Ende der Liste scrollen und den leeren Eintrag dort auswählen.

2. Im Eigenschaftsdialogfeld des Strings legen Sie die ID für den String fest und geben dann den String so ein, wie er in der Dropdown-Liste erscheinen soll. Für die Strings im Kombinationsfeld Breite, das Sie in die Symbolleiste Farben aufnehmen, tragen Sie folgende Strings ein.

ID	Beschriftung
IDS_WIDTH_VERYTHIN	Sehr dünn
IDS_WIDTH_THIN	Dünn
IDS_WIDTH_MEDIUM	Mittel
IDS_WIDTH_THICK	Dick
IDS_WIDTH_VERYTHICK	Sehr dick

Tabelle 11.6: Strichstärken und Bezeichnungen im Konbinationsfeld

Während Sie die Strings in die Zeichenfolgentabelle einfügen, achten Sie auf die den Strings zugewiesene Werte. Sie müssen sequenziell sein, wobei »Sehr dünn« der kleinste und »Sehr dick« der größte Wert sein soll.

Das Kombinationsfeld für die Symbolleiste erstellen

Bevor Sie das Kombinationsfeld in die Symbolleiste Farben aufnehmen können, müssen Sie eine Variable für das Kombinationsfeld erzeugen. Da sich das Kombinationsfeld nicht über die Editoren hinzufügen lässt, müssen Sie es als Variable in die Klasse CMainFrame aufnehmen.

Um die Kombinationsfeldvariable für die Farbpalette in die Hauptrahmenklasse aufzunehmen, folgen Sie diesen Schritten:

1. Wählen Sie die Klassenansicht aus.

2. Klicken Sie mit der rechten Maustaste auf die Klasse CMainFrame und wählen Sie Hinzufügen / Variable hinzufügen aus dem Kontextmenü.
3. Legen Sie den Variablentyp als CComboBox, den Namen als m_ctlWidth und den Zugriff als protected fest.

Nachdem Sie die Variable für das Kombinationsfeld in die Hauptrahmenklasse aufgenommen haben, sind verschiedene Aufgaben zu erledigen, nachdem die Symbolleiste erzeugt wurde:

- Die Breite und die ID des Platzhalters für das Kombinationsfeld in der Symbolleiste auf die Breite und die ID des Kombinationsfelds setzen
- Die Position des Platzhalters in der Symbolleiste ermitteln und danach Größe und Lage des Kombinationsfelds festlegen
- Das Kombinationsfeld erzeugen. Dabei die Symbolleiste als übergeordnetes Fenster des Kombinationsfelds spezifizieren
- Die Strings in die Dropdown-Liste des Kombinationsfelds laden

Damit diese Abläufe einigermaßen übersichtlich bleiben, empfiehlt es sich, das Erstellen der Symbolleiste Farben in einer eigenen Funktion zu realisieren, die sich aus der Funktion OnCreate der Hauptrahmenklasse aufrufen lässt. Um diese Funktion zu erzeugen, folgen Sie diesen Schritten:

1. Klicken Sie im Arbeitsbereich mit der rechten Maustaste auf die Klasse CMainFrame und wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.

2. Legen Sie den Funktionstyp mit BOOL, den Funktionsnamen als CreateColorBar und den Zugriffsstatus als public fest.
3. In die Funktion schreiben Sie den Code von Listing 11.5.

Listing 11.5: Die Funktion CMainFrame.CreateColorBar

```
 1: BOOL CMainFrame::CreateColorBar(void)
 2: {
 3:   // Farbpalette hinzufügen
 4:   int iTBCtlID;
 5:   int i;
 6:
 7:   // Farbpalette erzeugen
 8:   if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
 9:       WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
10:       CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
11:       !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
12:   {
13:     TRACE0("Symbolleiste konnte nicht erstellt werden\n");
14:       return FALSE;    // Fehler bei Erstellung
15:   }
16:   // Schaltfläche Black auf der Symbolleiste suchen
17:   iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
18:   if (iTBCtlID >= 0)
19:   {
20:     // Schleife durch die Schaltflächen, Verhalten wie
21:     // Optionsfelder setzen
22:     for (i = iTBCtlID; i < (iTBCtlID + 8); i++)
23:       m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
24:   }
25:   // Kombinationsfeld hinzufügen
26:   int nWidth = 100;
27:   int nHeight = 125;
28:
29:   // Platzhalter des Kombinationsfelds konfigurieren
30:   m_wndColorBar.SetButtonInfo(9, IDC_CBWIDTH, TBBS_SEPARATOR,
31:                               nWidth);
32:
33:   // Höhe der Farbpalette ermitteln
34:   CRect rect;
35:   m_wndColorBar.GetItemRect(9, &rect);
36:   rect.bottom = rect.top + nHeight;
37:
38:   // Kombinationsfeld erzeugen
39:   m_ctlWidth.Create(WS_CHILD | WS_VISIBLE | WS_VSCROLL |
40:           CBS_DROPDOWNLIST, rect, &m_wndColorBar, IDC_CBWIDTH);
41:
42:   // Kombinationsfeld füllen
43:   CString strStyle;
44:   if (strStyle.LoadString(IDS_WIDTH_VERYTHIN))
45:     m_ctlWidth.AddString((LPCTSTR)strStyle);
46:   if (strStyle.LoadString(IDS_WIDTH_THIN))
47:     m_ctlWidth.AddString((LPCTSTR)strStyle);
48:   if (strStyle.LoadString(IDS_WIDTH_MEDIUM))
49:     m_ctlWidth.AddString((LPCTSTR)strStyle);
50:   if (strStyle.LoadString(IDS_WIDTH_THICK))
51:     m_ctlWidth.AddString((LPCTSTR)strStyle);
52:   if (strStyle.LoadString(IDS_WIDTH_VERYTHICK))
53:     m_ctlWidth.AddString((LPCTSTR)strStyle);
54:
55:   // Ersten Eintrag im Kombinationsfeld auswählen
56:   m_ctlWidth.SetCurSel(0);
57:
58:   return TRUE;
59: }
```

In Listing 11.5 geben Sie an, dass das Kombinationsfeld mit der Objekt-ID IDC_CBWIDTH zu erstellen ist:

```
m_ctlWidth.Create(WS_CHILD | WS_VISIBLE | WS_VSCROLL |
        CBS_DROPDOWNLIST, rect, &m_wndColorBar, IDC_CBWIDTH);
```

Diese Objekt-ID identifiziert das Kombinationsfeld, wenn es eine Ereignisnachricht an die Anwendung sendet oder wenn man festlegen muss, welcher Listeneintrag im Bearbeitungsfeld anzuzeigen ist. Allerdings existiert diese Objekt-ID nicht in Ihrer Anwendung. Bevor Sie die Anwendung kompilieren können, müssen Sie diese ID zu den Projektressourcen-IDs hinzufügen, wie Sie es am Tag 5, »Timer«, kennen gelernt haben. Um diese ID zu Ihrem Projekt hinzuzufügen, folgen Sie diesen Schritten:

1. Wählen Sie die Ressourcenansicht im Arbeitsbereich.

2. Markieren Sie den Beginn des Ressourcenbaumes, klicken Sie mit der rechten Maustaste darauf und wählen Sie Ressourcensymbole aus dem Kontextmenü.
3. Fügen Sie die Objekt-ID IDC_CBWIDTH hinzu.
4. Vergewissern Sie sich, dass Sie für die neue Objekt-ID einen eindeutigen numerischen Wert vergeben, damit keine Konflikte mit anderen in Ihrer Anwendung verwendeten Objekten auftreten.

Den Platzhalter konfigurieren

Nachdem die Symbolleiste erstellt und alle Schaltflächen konfiguriert sind, konfigurieren Sie als Nächstes den Separator, der als Platzhalter für das zu erzeugende Kombinationsfeld dient. Dazu verwenden Sie die Symbolleistenfunktion SetButtonInfo:

```
m_wndColorBar.SetButtonInfo(9, IDC_CBWIDTH, TBBS_SEPARATOR, nWidth);
```

Diese Funktion übernimmt vier Argumente:

- Der aktuelle Index des Steuerelements in der Symbolleiste - in diesem Fall das zehnte Steuerelement in der Symbolleiste (acht Farbschaltflächen und zwei Separatoren), und denken Sie daran, dass in C/C++ bei 0 angefangen wird zu zählen
- Die neue ID des Symbolleisten-Steuerelements. Dabei handelt es sich um die ID, die in die Nachrichtenwarteschlange gestellt wird, wenn ein Steuerelementereignis auftritt.
- Der Typ des Symbolleisten-Steuerelements
- Das letzte Argument ist der neue Index des Steuerelements in der Symbolleiste, entsprechend der Funktionsdokumentation. Das ist die Position, auf die das Steuerelement verschoben wird. Dieses Argument ist jedoch etwas irreführend. Wenn das Steuerelement ein Separator ist, gibt das Argument die Breite des Steuerelements an und verschiebt es nicht an eine andere Stelle. Da es sich im vorliegenden Fall um einen Separator handelt, legt man mit diesem Argument die Breite des zu erzeugenden Kombinationsfelds fest.

Die Position des Kombinationsfelds in der Symbolleiste ermitteln

Der Separator als Platzhalter für das Kombinationsfeld in der Symbolleiste ist nun konfiguriert. Als Nächstes ermitteln Sie die Position des Platzhalters auf der Symbolleiste, damit sich die Lage des Kombinationsfelds festlegen lässt:

```
m_wndColorBar.GetItemRect(9, &rect);
rect.bottom = rect.top + nHeight;
```

In der ersten Zeile steht der Aufruf der Funktion GetItemRect, um die Position und Größe des Platzhalters für das Kombinationsfeld zu ermitteln. Die nächste Zeile addiert die Höhe der Dropdown-Liste zur Höhe, die das Kombinationsfeld schließlich einnimmt.

Das Kombinationsfeld erstellen

Die Größe des Platzhalters ist nun genau festgelegt und Sie verfügen über Lage und Größe für das Kombinationsfeld. Nun ist es an der Zeit, das Kombinationsfeld zu erzeugen. Dazu dient die Funktion Create des Kombinationsfelds:

```
m_ctlWidth.Create(WS_CHILD | WS_VISIBLE | WS_VSCROLL |
    CBS_DROPDOWNLIST, rect, &m_wndColorBar, IDC_CBWIDTH);
```

Das erste Argument der Funktion Create für das Kombinationsfeld ist der Stil des Kombinationsfelds. Normalerweise kombiniert man verschiedene Stilattribute, um einen Stilwert zu bilden. Tabelle 11.7 listet die Attribute auf, die Sie in diesem Wert verwenden können.

Stil	Beschreibung
WS_CHILD	Kennzeichnet das Kombinationsfeld als untergeordnetes Fenster (erforderlich)
WS_VISIBLE	Macht das Kombinationsfeld sichtbar
WS_DISABLED	Deaktiviert das Kombinationsfeld
WS_VSCROLL	Stattet die DropDown-Liste mit vertikalen Bildlaufleisten aus
WS_HSCROLL	Stattet die DropDown-Liste mit horizontalen Bildlaufleisten aus
WS_GROUP	Gruppiert Steuerelemente
WS_TABSTOP	Schließt das Kombinationsfeld in die Tabulator-Reihenfolge ein
CBS_AUTOHSCROLL	Rollt den Text im Bearbeitungsfeld automatisch nach rechts, wenn der Benutzer ein Zeichen am Ende der Zeile eingibt. Damit kann der Benutzer Text eingeben, der breiter ist, als das Eingabefeld des Kombinationsfelds.
CBS_DROPDOWN	Ähnlich zu CBS_SIMPLE. Allerdings wird die Liste nur dann angezeigt, wenn der Benutzer das Symbol neben dem Eingabefeld wählt.
CBS_DROPDOWNLIST	Ähnlich zu CBS_DROPDOWN. Allerdings wird das Eingabefeld durch ein statisches Textfeld ersetzt, das den momentan in der Liste markierten Eintrag anzeigt.
CBS_HASSTRINGS	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Elemente des Listenfelds bestehen aus Strings.
CBS_OEMCONVERT	Der in das Eingabefeld eingegebene Text wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und wieder zurück nach ANSI konvertiert.
CBS_OWNERDRAWFIXED	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Einträge in der Liste haben alle die gleiche Höhe.
CBS_OWNERDRAWVARIABLE	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Höhe der Listeneinträge ist variabel.
CBS_SIMPLE	Das Listenfeld wird immer angezeigt.
CBS_SORT	Sortiereniert automatisch die Strings im Listenfeld
CBS_DISABLENOSCROLL	Die Liste zeigt eine deaktivierte Bildlaufleiste, wenn nur so viel Einträge in der Liste vorhanden sind, dass sich ein Bildlauf erübrigt.
CBS_NOINTEGRALHEIGHT	Legt fest, dass das Kombinationsfeld genau die angegebene Größe einnimmt

Tabelle 11.7: Stile für Kombinationsfelder

Das zweite Argument beschreibt das Rechteck, das das Kombinationsfeld einnehmen soll. Dieses Argument gibt die Position innerhalb des übergeordneten Fensters - in diesem Fall der Symbolleiste - an, wo das Kombinationsfeld untergebracht ist. Es verschiebt sich mit dem übergeordneten Fenster (der Symbolleiste) und bleibt immer auf der gleichen Relativposition.

Das dritte Argument ist ein Zeiger auf das übergeordnete Fenster. Es handelt sich hierbei um die Adresse der Variablen für die Symbolleiste Farben. Im vierten Argument wird die Objekt-ID für das Kombinationsfeld übergeben.

Das Kombinationsfeld füllen

Als Letztes beim Erstellen des Kombinationsfelds auf der Symbolleiste Farben ist noch die Dropdown-Liste mit den verfügbaren Elementen, aus denen der Benutzer auswählen kann, zu füllen. Das Ganze wird mit

einer Kombination aus zwei Funktionen realisiert:

```
if (strStyle.LoadString(IDS_WIDTH_VERYTHIN))
    m_ctlWidth.AddString((LPCTSTR)strStyle);
```

Die erste Funktion ist die Funktion LoadString der Klasse CString. Diese Funktion übernimmt eine String-ID und lädt den zugehörigen String aus der Zeichenfolgentabelle. Die zweite Funktion ist die Funktion AddString des Kombinationsfelds, die den als Argument übergebenen String der Dropdown-Liste hinzufügt. Indem man diese Kombination von Funktionen für alle Elemente in der Dropdown-Liste aufruft, kann man das Kombinationsfeld aus der Zeichenfolgentabelle der Anwendung füllen.

Die Funktion OnCreate aktualisieren

Nachdem wir den gesamten Code zum Erstellen der Symbolleiste Farben in eine separate Funktion verschoben haben, können wir die Funktion OnCreate auf den neuesten Stand bringen, sodass sie die Funktion CreateColorBar aufruft, mit der die Symbolleiste Farben erzeugt wird. Der entsprechende Code ist in Listing 11.6 wiedergegeben.

Listing 11.6: Die modifizierte CmainFrame.OnCreate-Funktion

```
 1: int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
 2: {
 3:   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
 4:     return -1;
 5:
 6:   if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
 7:       WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
 8:       CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
 9:       !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
10:   {
11:     TRACE0("Failed to create toolbar\ n");
12:     return -1;      // fail to create
13:   }
14:
15:   // Symbolleiste Farben hinzufügen
16:   if (!CreateColorBar())
17:   {
18:     TRACE0("Symbolleiste konnte nicht erstellt werden\n");
19:       return -1;      // Fehler beim Erstellen
20:   }
21:
22:   if (!m_wndStatusBar.Create(this) ||
23:       !m_wndStatusBar.SetIndicators(indicators,
24:       sizeof(indicators)/sizeof(UINT)))
25:   {
26:     TRACE0("Statusleiste konnte nicht erstellt werden\n");
27:       return -1;      // Fehler bei Erstellung
28:   }
29:   // TODO: Löschen Sie diese drei Zeilen, wenn Sie nicht
30:   // möchten, dass die Systemleiste andockbar ist
31:   m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
32:   // Andocken für Symbolleiste Farben aktivieren
33:   m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
34:   EnableDocking(CBRS_ALIGN_ANY);
35:   DockControlBar(&m_wndToolBar);
36:
37:   // Symbolleiste Farben andocken
38:   DockControlBar(&m_wndColorBar);
39:   return 0;
40: }
```

Wenn Sie nun die Anwendung kompilieren und ausführen, sollte sich am Ende der Symbolleiste Farben ein Kombinationsfeld befinden, wie es Abbildung 11.6 zeigt. Allerdings bewirkt das Kombinationsfeld noch keine

Aktionen.

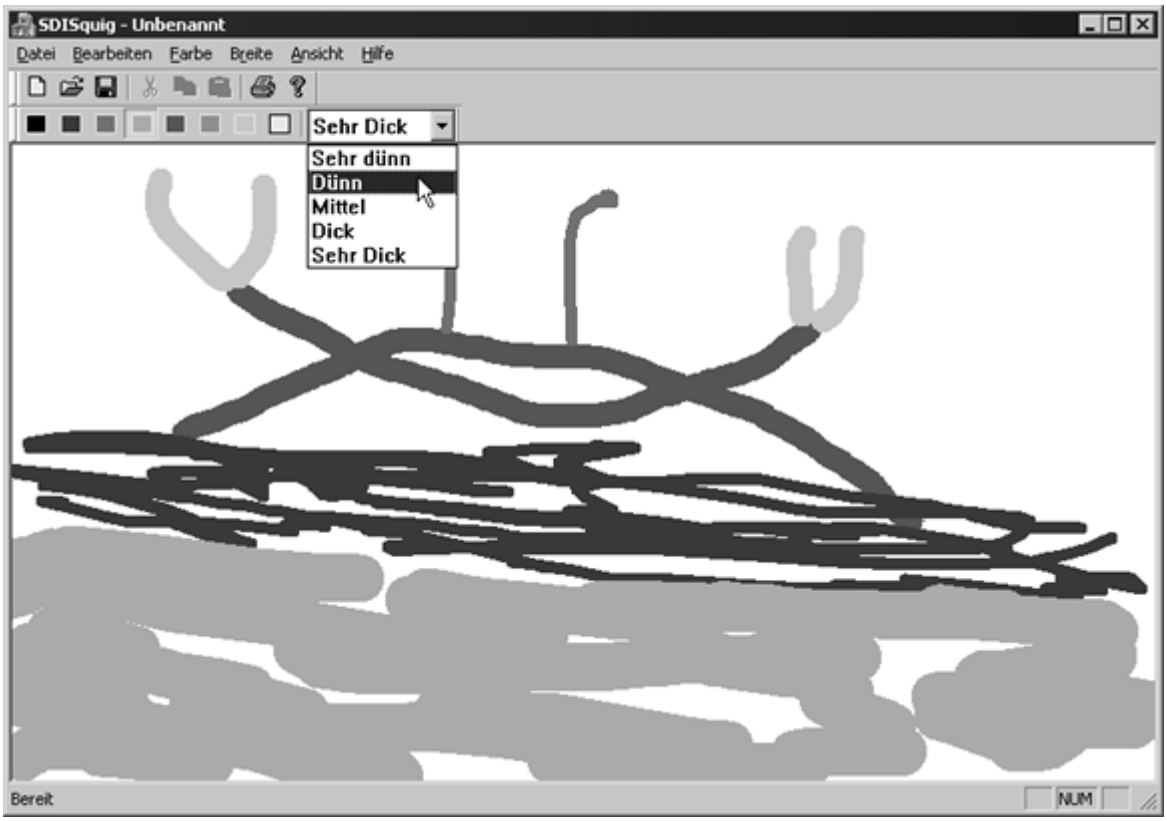


Abbildung 11.6: Die Symbolleiste Farben mit dem Kombinationsfeld Breite

Ereignisse des Kombinationsfelds der Symbolleiste behandeln

Bevor Sie die Funktionalität der Behandlungsroutine für das Kombinationsfeld hinzufügen können, müssen Sie eine kleine, aber wichtige Funktion in die Dokumentklasse einfügen: SetWidth. Um diese Funktion einzufügen, folgen Sie diesen Schritten:

1. Wählen Sie in der Klassenansicht die Klasse CSDISquigDoc, klicken Sie mit der rechten Maustaste darauf und wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.

2. Legen Sie den Rückgabetyp der Funktion als void und den Funktionsnamen als SetWidth fest. Fügen Sie einen Parameter vom Typ unsigned int mit dem Namen nWidth ein.
3. Fügen Sie den Code aus Listing 11.7 in die Funktion ein.

Listing 11.7: Die Funktion SetWidth

```
1: void CSDISquigDoc::SetWidth(unsigned int nWidth)
2: {
3:    // Aktuelle Breite setzen
4:    m_nWidth = nWidth;
5: }
```

Eine Behandlungsroutine für das Kombinationsfeld lässt sich leicht erstellen, auch wenn man das manuell erledigen muss (da der Klassen-Assistent gar nicht weiß, dass das Kombinationsfeld existiert). Es ist ein Eintrag ON_CBN_SELCHANGE in die Nachrichtenzuordnungstabelle aufzunehmen und dann die eigentliche Behandlungsfunktion in die Klasse CMainFrame einzufügen. Folgen Sie zum Anfang diesen Schritten:

1. Nehmen Sie die Behandlungsfunktion auf, indem Sie die Klasse CMainFrame in der Klassen-Ansicht markieren und Hinzufügen / Funktion hinzufügen aus dem Kontextmenü wählen.

2. Geben Sie den Funktionstyp mit afx_msg void, den Funktionsnamen mit OnSelChangeWidth und den

Zugriff mit protected an.

3. Den Code für die Funktion übernehmen Sie aus Listing 11.8. Achten Sie darauf, afx_msg aus der ersten Zeile der Funktionsimplementierung zu löschen (es sollte nur in der Funktionsdeklaration in der Header-Datei stehen).

Listing 11.8: Die Funktion OnSelChangeWidth

```
 1: void CMainFrame::OnSelChangeWidth(void)
 2: {
 3:   // Neue Auswahl des Kombinationsfelds ermitteln
 4:   int iIndex = m_ctlWidth.GetCurSel();
 5:   if (iIndex == CB_ERR)
 6:     return;
 7:
 8:   // Aktives Dokument ermitteln
 9:   CSDISquigDoc* pDoc = (CSDISquigDoc*)GetActiveDocument();
10:   // Handelt es sich um ein gültiges Dokument?
11:   if (pDoc)
12:     // Neue Zeichenbreite setzen
13:     pDoc->SetWidth(iIndex);
14: }
```

Die Funktion in Listing 11.8 ermittelt zuerst die aktuelle Auswahl des Kombinationsfeldes. Da wir die Einträge in der richtigen Reihenfolge eingetragen und beim Erstellen des Kombinationsfelds das Flag CBS_SORT (zum Sortieren der Einträge) nicht angegeben haben, entspricht der Index den Breiten im Dokument. Daher kann man einen Zeiger auf die aktuelle Instanz des Dokuments mit der Funktion GetActiveDocument ermitteln und dann die neue Breite an das Dokument mit der Funktion SetWidth übergeben.

Um die Behandlungsfunktion aufzurufen, wenn der Benutzer die Auswahl im Kombinationsfeld ändert, müssen Sie den entsprechenden Eintrag in die Nachrichtenzuordnungstabelle von CMainFrame einfügen. Gehen Sie im Quellcode von CMainFrame nach oben, bis Sie den Abschnitt der Nachrichtenzuordnungstabelle finden, und fügen Sie hier die fett gedruckte Zeile von Listing 11.9 ein.

Listing 11.9: Die modifizierte Nachrichtenzuordnungstabelle vom CMainFrame

```
 1: // CMainFrame
 2:
 3: IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
 4:
 5: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
 6:     ON_WM_CREATE()
 7:     ON_UPDATE_COMMAND_UI(ID_VIEW_COLORBAR, OnUpdateViewColorbar)
 8:     ON_COMMAND(ID_VIEW_COLORBAR, OnViewColorbar)
 9:     ON_CBN_SELCHANGE(IDC_CBWIDTH, OnSelChangeWidth)
10: END_MESSAGE_MAP()
```

Der eben hinzugefügte Eintrag legt fest, dass die Funktion OnSelChangeWidth aufzurufen ist, wenn Ereignisse mit der Objekt-ID des Kombinationsfelds in der Symbolleiste Farben auf Grund von Änderungen der Auswahl im Kombinationsfeld auftreten. Wenn Sie nun die Anwendung kompilieren und ausführen, sollten Sie die Stiftbreite zum Zeichnen über das Kombinationsfeld der Symbolleiste Farben ändern können.

An diesem Punkt lässt sich Ihre Anwendung nicht ohne Fehler kompilieren. Der Grund dafür ist, dass Sie im Code des Hauptrahmens auf die Dokumentklasse verweisen, im Quellcode des Hauptrahmens jedoch noch nicht die Header-Datei der Dokumentklasse eingebunden haben.

4. Scrollen Sie an den Anfang der Quellcode-Datei MainFrame.cpp.
5. Fügen Sie direkt nach der letzten #include-Anweisung eine weitere #include- Anweisung für die

Header-Datei SDISquigDoc.h ein.

Das Kombinationsfeld der Symbolleiste aktualisieren

Für das Kombinationsfeld ist noch ein Problem zu lösen. Falls der Benutzer einen neuen Wert aus dem Menü und nicht über das Kombinationsfeld auswählt, soll das Kombinationsfeld ebenfalls den neuen Wert widerspiegeln. Die aktuelle Auswahl im Kombinationsfeld lässt sich am effizientesten setzen, wenn das beim Auslösen irgendeines Menübefehls geschieht. Dazu ist eine Funktion in der Hauptrahmenklasse erforderlich, die sich von der Dokumentklasse aufrufen lässt. Die Funktion im Hauptrahmen muss lediglich die aktuelle Auswahl im Kombinationsfeld setzen.

Um diese Funktion im Hauptrahmen zu implementieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Klasse CMainFrame ein.

2. Legen Sie den Funktionstyp mit void und den Namen mit UpdateWidthCB fest und fügen Sie einen Parameter vom Typ int mit dem Namen iIndex ein. Legen Sie den Zugriff mit public fest.
3. In die Funktion schreiben Sie den Code aus Listing 11.10.

Listing 11.10: Die Funktion CMainFrame.UpdateWidthCB

```
1: void CMainFrame::UpdateWidthCB(int iIndex)
2: {
3:   // Die neue Auswahl im Kombinationsfeld setzen
4:   m_ctlWidth.SetCurSel(iIndex);
5: }
```

Die Funktion UpdateWidthCB ruft die Kombinationsfeldfunktion SetCurSel auf, die die aktuelle Auswahl in der Dropdown-Liste des Kombinationsfelds auf den Eintrag mit dem angegebenen Index setzt. Das Eingabefeld des Kombinationsfelds wird mit dem neu ausgewählten Listeneintrag aktualisiert. Wenn man dem Kombinationsfeld einen Index anbietet, der nicht in der Dropdown-Liste existiert, liefert die Funktion einen Fehler zurück.

Auf der Seite des Dokuments muss man die Funktion im Hauptrahmen immer dann aufrufen, wenn die entsprechenden Behandlungsfunktionen für das Menü aufgerufen werden. Da das in verschiedenen Funktionen vorkommen kann, ist es sinnvoll, die erforderliche Funktionalität in einer einzigen Funktion unterzubringen. Diese Funktion benötigt einen Zeiger auf die mit dem Dokument verbundene Ansicht und ermittelt dann - über die Ansicht - einen Zeiger auf den Rahmen. Diesen Zeiger kann man dann im Aufruf der Funktion UpdateWidthCB, die wir eben der Hauptrahmenklasse hinzugefügt haben, verwenden.

Um die Funktion in die Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Markieren Sie die Klasse CSDISquigDoc in der Klassenansicht und wählen Sie Hinzufügen / Funktion hinzufügen aus dem Kontextmenü.

2. Legen Sie als Funktionstyp void und als Funktionsname UpdateColorbar fest und fügen Sie einen Parameter vom Typ int mit dem Namen iIndex ein. Legen Sie als Zugriffsstatus private fest.
3. In die Funktion übernehmen Sie den Code aus Listing 11.11.

Listing 11.11: Die Funktion CSDISquigDoc.UpdateColorbar

```
 1: void CSDISquigDoc::UpdateColorbar(int iIndex)
 2: {
 3:   // Position der ersten Ansicht ermitteln
 4:   POSITION pos = GetFirstViewPosition();
 5:   // Ist die Position gültig?
 6:   if (pos != NULL)
 7:   {
 8:     // Zeiger auf Ansicht in dieser Position holen
 9:     CView* pView = GetNextView(pos);
10:     // Ist der Zeiger auf die Ansicht gültig?
```

```
11:     if (pView)
12:     {
13:       // Zeiger auf Rahmen über Ansicht holen
14:       CMainFrame* pFrame =
15:              (CMainFrame*)pView->GetTopLevelFrame();
16:       // Zeiger auf Rahmen erhalten?
17:       if (pFrame)
18:         // Kombinationsfeld in Symbolleiste Color über
19:         // den Rahmen aktualisieren
20:         pFrame->UpdateWidthCB(iIndex);
21:     }
22:   }
23: }
```

Die Funktion UpdateColorbar verfolgt den Pfad durch die verschiedenen Ansichten, um den Anwendungsrahmen aus der Dokumentklasse zu erhalten. Da in einem Dokument mehrere Ansichten gleichzeitig geöffnet sein können, ermitteln Sie mit GetFirstViewPosition die Position der ersten Ansicht, die mit dem Dokument verbunden ist.

Die nächste Funktion, GetNextView, gibt einen Zeiger auf die Ansicht zurück, die durch die Position spezifiziert wird. Diese Funktion aktualisiert ebenfalls die Positionsvariable mit dem Zeiger auf die nächste Ansicht in der Liste der Ansichten, die mit dem aktuellen Dokument verbunden sind.

Nachdem Sie über einen Zeiger auf die Ansicht verfügen, können Sie die Fensterfunktion GetTopLevelFrame aufrufen, die einen Zeiger auf das Rahmenfenster der Anwendung zurückgibt. Diese Funktion müssen Sie über die Ansicht aufrufen, da das Dokument nicht von der Klasse CWnd abgeleitet ist, während das bei der Ansicht der Fall ist.

Sie besitzen nunmehr einen Zeiger auf das Rahmenfenster und können diesen Zeiger verwenden, um die weiter vorn definierte Funktion aufzurufen und das Kombinationsfeld in der Symbolleiste zu aktualisieren. Wenn Sie jetzt diese neue Funktion aus der Behandlungsroutine des Menübefehls Breite aufrufen, wie es in Listing 11.12 geschieht, wird das Kombinationsfeld in der Symbolleiste Farben automatisch aktualisiert und spiegelt die momentan ausgewählte Stiftbreite zum Zeichnen wider, unabhängig davon, ob die Auswahl über das Menü oder das Kombinationsfeld erfolgt ist.

Listing 11.12: Eine aktualisierte Behandlungsroutine für den Menübefehl Breite

```
1: void CSDISquigDoc::OnWidthVthin()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Neue Breite setzen
5:    m_nWidth = 0;
6:    // Kombinationsfeld in der Symbolleiste Color aktualisieren
7:    UpdateColorbar(m_nWidth);
8: }
```

Wieder lässt sich Ihre Anwendung nicht ohne Fehler kompilieren. Der Grund dafür ist, dass Sie in der Dokumentklasse auf die Hauptrahmenklasse verweisen, die Header-Datei der Hauptrahmenklasse jedoch noch nicht in den Quellcode der Dokumentklasse eingebunden haben.

Scrollen Sie zum Anfang der Quellcode-Datei SDISquigDoc.cpp.

4. Fügen Sie direkt nach der letzten #include-Anweisung eine weitere #include- Anweisung für die Header-Datei MainFrm.h ein.

11.4 Ein neues Element für die Statusleiste

Weiter oben in der heutigen Lektion haben Sie gelernt, wie man Meldungen in der Statusleiste und QuickInfos für Symbolleisten-Schaltflächen und Menübefehle spezifiziert. Wie verhält es sich nun, wenn Sie dem Benutzer in der Statusleiste weitergehende Informationen anbieten möchten? Wie lässt sich analog zu Visual Studio für Visual C++ erreichen, dass man Informationen über die momentan ausgeführten Aktionen, die

gerade im Dokument bearbeitete Stelle oder den aktuellen Zustand der Anwendung anzeigen kann? Diese Angaben gehen über die Anzeigen der Feststelltasten hinaus, die Visual C++ automatisch in der Statusleiste meldet.

Es ist tatsächlich einfach, zusätzliche Ausschnitte in die Statusleiste aufzunehmen oder auch bereits vorhandene Ausschnitte zu entfernen. Um sich mit diesen Vorgängen bekannt zu machen, nehmen Sie einen neuen Ausschnitt in die Statusleiste Ihrer Zeichenanwendung auf, der die momentan gültige Farbe meldet.

Ein neuer Ausschnitt für die Statusleiste

Bevor Sie einen neuen Ausschnitt in die Statusleiste einbinden, müssen Sie einen neuen Eintrag in der Zeichenfolgentabelle der Anwendung für den Statusleisten-Ausschnitt erstellen. Der Eintrag in der Zeichenfolgentabelle realisiert für den Statusleisten- Ausschnitt zwei Aufgaben:

- Er stellt die Objekt-ID für den Statusleisten-Ausschnitt bereit. Diese ID verwenden Sie, um den Ausschnitt in der Statusleiste zu aktualisieren, wenn ein neuer Text in diesem Feld auszugeben ist.
- Die zweite Funktion bezieht sich auf die Größe des Ausschnitts. Um die korrekte Größe des Ausschnitts festzulegen, müssen Sie einen Titel für den Eintrag in der Zeichenfolgentabelle bereitstellen, der mindestens so lang wie der längste String ist, den Sie im Statusleisten-Ausschnitt anzeigen möchten.

Um in die Zeichenfolgentabelle Ihrer Anwendung einen neuen Eintrag einzufügen, folgen Sie diesen Schritten:

1. Verwenden Sie die gleichen Schritte, die Sie im Abschnitt »Die Projektressourcen bearbeiten« ausgeführt haben, um den Text für das auf der Symbolleiste platzierte Kombinationsfeld einzufügen.

2. Legen Sie die String-ID mit ID_INDICATOR_COLOR und den Titel mit SCHWARZ (eine der beiden längsten Zeichenketten, die im Ausschnitt der Statusleiste anzuzeigen sind) fest.

Ein kleiner Teil im ersten Abschnitt des Quellcodes für den Hauptrahmen definiert das Layout der Statusleiste. Diese kleine Tabelle enthält die Objekt-IDs der Statusleisten- Ausschnitte als Tabellenelemente und zwar in der Reihenfolge, in der sie von links nach rechts in der Statusleiste erscheinen. Um den Ausschnitt für die Farbe in die Statusleiste aufzunehmen, fügen Sie die ID des Farbenausschnittes in die Tabellendefinition für die Statusleistenanzeige unmittelbar im Anschluss an die Nachrichtenzuordnungstabelle in der Quellcode-Datei für den Hauptrahmen ein. Schreiben Sie die ID für den Farbenausschnitt in der Tabellendefinition an die Stelle, die der Ausschnitt auf der Statusleiste einnehmen soll, wie es aus der fett gedruckten Zeile von Listing 11.13 hervorgeht.

Listing 11.13: Eine modifizierte Tabellendefinition für Statusleistenanzeigen

```
 1: // CMainFrame
 2:
 3: IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
 4:
 5: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
 6:     ON_WM_CREATE()
 7:     ON_UPDATE_COMMAND_UI(ID_VIEW_COLORBAR, OnUpdateViewColorbar)
 8:     ON_COMMAND(ID_VIEW_COLORBAR, OnViewColorbar)
 9:     ON_CBN_SELCHANGE(IDC_CBWIDTH, OnSelChangeWidth)
10: END_MESSAGE_MAP()
11:
12: static UINT indicators[] =
13: {
14:   ID_SEPARATOR,           // Statusleistenanzeige
15:     ID_INDICATOR_COLOR,
16:     ID_INDICATOR_CAPS,
17:     ID_INDICATOR_NUM,
18:     ID_INDICATOR_SCRL,
19: };
```

Falls Sie eine der Anzeigen für die Feststelltasten aus der Statusleiste entfernen möchten, löschen Sie

einfach den entsprechenden Eintrag in der Tabellendefinition. Weiter unten in der Funktion OnCreate, die die Statusleiste (unmittelbar nach den Symbolleisten) erstellt, sehen Sie auch, wo diese Tabelle zum Einsatz kommt, um die Statusleiste mit dem folgenden Code zu erzeugen:

```
if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
    sizeof(indicators)/sizeof(UINT)))
```

Sobald die Statusleiste erzeugt wurde, folgt ein Aufruf der Funktion SetIndicators auf der Statusleiste, um die Ausschnitte gemäß der Definition in der Tabelle der Statusleistenanzeigen in die Statusleiste aufzunehmen. Die mit den IDs in der Tabelle der Statusleistenanzeigen verbundenen Strings dienen dazu, die Ausschnitte zu initialisieren und deren Größe festzulegen. Wenn Sie jetzt die Anwendung kompilieren und ausführen, sehen Sie den neuen Farbenausschnitt in der Statusleiste. Der Ausschnitt zeigt den Titel aus der Zeichenfolgentabelle an.

Text für einen Statusleisten-Ausschnitt festlegen

Nachdem Sie den Ausschnitt in die Statusleiste aufgenommen haben, können Sie über das Ereignis UPDATE_COMMAND_UI alle Aktualisierungen des Ausschnitts erledigen lassen. Dazu müssen Sie lediglich eine Behandlungsroutine für das Ereignis auf der Objekt-ID des Ausschnitts hinzufügen und dieses Ereignis verwenden, um den Text im Ausschnitt festzulegen. Da die Statusleiste immer sichtbar ist, wird das Ereignis UPDATE_COMMAND_UI für die Ausschnitte in der Statusleiste jedesmal ausgelöst, wenn die Anwendung im Leerlauf arbeitet. Das bedeutet, dass das Ereignis auftritt, nachdem die Anwendung die Abarbeitung aller Tastenbetätigungen und Mausbewegungen abgeschlossen hat. In etwa einer Woche - am Tag 17, »Multitasking« - lernen Sie, wie häufig und wann Tasks, die im Leerlauf einer Anwendung arbeiten, ausgelöst werden.

In der Behandlungsroutine müssen Sie einen String erzeugen, der den Namen der aktuellen Farbe enthält (oder was auch immer für ein Text im Ausschnitt der Statusleiste erscheinen soll). Als Nächstes ist sicherzustellen, dass der Ausschnitt aktiviert ist. Schließlich müssen Sie den Text des Ausschnitts auf den erzeugten String setzen.

Um das Ganze in Ihrer Anwendung zu implementieren, erzeugen Sie eine Behandlungsroutine für die Nachricht UPDATE_COMMAND_UI. Auch in diesem Fall ist der Klassen-Assistent nicht über den Ausschnitt in der Statusleiste informiert, sodass Sie die Behandlungsroutine manuell erzeugen und in die Nachrichtenzuordnungstabelle eintragen müssen. Um die Behandlungsroutine selbst zu erstellen und zur Nachrichtenzuordnungstabelle hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie der Dokumentklasse (CSDISquigDoc) eine neue Member-Funktion mit dem Typ afx_msg void, dem Namen OnUpdateIndicatorColor und einem Parameter vom Typ CCmdUI* mit den Namen pCmdUI hinzu. Legen Sie den Zugriffsstatus als protected fest.

2. In die Funktion nehmen Sie den Code von Listing 11.14 auf.

Listing 11.14: Die Funktion OnUpdateIndicateColor

```
 1: void CSDISquigDoc::OnUpdateIndicatorColor(CCmdUI* pCmdUI)
 2: {
 3:   CString strColor;
 4:
 5:   // Wie heißt die aktuelle Farbe?
 6:   switch (m_nColor)
 7:   {
 8:     case 0:    // Schwarz
 9:       strColor = "SCHWARZ";
10:       break;
11:     case 1:    // Blau
12:       strColor = "BLAU";
13:       break;
14:     case 2:    // Grün
15:       strColor = "GRÜN";
16:       break;
```

```
17:     case 3:    // Cyan
18:       strColor = "CYAN";
19:       break;
20:     case 4:    // Rot
21:       strColor = "ROT";
22:       break;
23:     case 5:    // Magenta
24:       strColor = "MAGENTA";
25:       break;
26:     case 6:    // Gelb
27:       strColor = "GELB";
28:       break;
29:     case 7:    // Weiß
30:       strColor = "WEISS";
31:       break;
32:   }
33:   // Statusleisten-Ausschnitt aktivieren
34:   pCmdUI->Enable(TRUE);
35:   // Text im Statusleisten-Ausschnitt auf aktuelle Farbe setzen
36:   pCmdUI->SetText(strColor);
37: }
```

Die Funktion in Listing 11.14 befolgt genau drei Schritte: Sie erzeugt einen String mit dem Namen der aktuellen Farbe, stellt sicher, dass der Ausschnitt in der Statusleiste aktiviert ist, und setzt den Text im Feld auf den erzeugten String.

Um nun sicherzustellen, dass der Aufruf der neuen Behandlungsroutine wie erwartet erfolgt, müssen Sie einen ON_UPDATE_COMMAND_UI-Eintrag in die Nachrichtenzuordnungstabelle am Beginn des Quellcodes der Dokumentdatei aufnehmen, wie es Listing 11.15 zeigt.

Listing 11.15: Die modifizierte Nachrichtenzuordnungstabelle von CSDISquigDoc

```
 1: // CSDISquigDoc
 2:
 3: IMPLEMENT_DYNCREATE(CSDISquigDoc, CDocument)
 4:
 5: BEGIN_MESSAGE_MAP(CSDISquigDoc, CDocument)
 6:     ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
 7:     ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
 8:     ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
 9:     ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
10:     ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
11:     ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
12:     ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
13:     ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
14:     ON_COMMAND(ID_COLOR_RED, OnColorRed)
15:     ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
16:     ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
17:     ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
18:     ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
19:     ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
20:     ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
21:     ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
22:     ON_COMMAND(ID_WIDTH_VERYTHIN, OnWidthVerythin)
23:     ON_UPDATE_COMMAND_UI(ID_WIDTH_VERYTHIN,
24:                                     OnUpdateWidthVerythin)
25:     ON_COMMAND(ID_WIDTH_THIN, OnWidthThin)
26:     ON_UPDATE_COMMAND_UI(ID_WIDTH_THIN, OnUpdateWidthThin)
27:     ON_COMMAND(ID_WIDTH_MEDIUM, OnWidthMedium)
28:     ON_UPDATE_COMMAND_UI(ID_WIDTH_MEDIUM, OnUpdateWidthMedium)
29:     ON_COMMAND(ID_WIDTH_THICK, OnWidthThick)
30:     ON_UPDATE_COMMAND_UI(ID_WIDTH_THICK, OnUpdateWidthThick)
31:     ON_COMMAND(ID_WIDTH_VERYTHICK, OnWidthVerythick)
```

```
32:     ON_UPDATE_COMMAND_UI(ID_WIDTH_VERYTHICK,
33:                                         OnUpdateWidthVerythick)
34:     ON_UPDATE_COMMAND_UI(ID_INDICATOR_COLOR,
35:                                         OnUpdateIndicatorColor)
36: END_MESSAGE_MAP()
```

Nachdem Sie die Behandlungsroutine und den Eintrag der Nachrichtenzuordnungstabelle hinzugefügt haben, können Sie nun Ihre Anwendung kompilieren und ausführen. In der Statusleiste sollte der Farbenausschnitt automatisch aktualisiert werden, um die momentane Zeichenfarbe anzugeben, wie es Abbildung 11.7 zeigt.

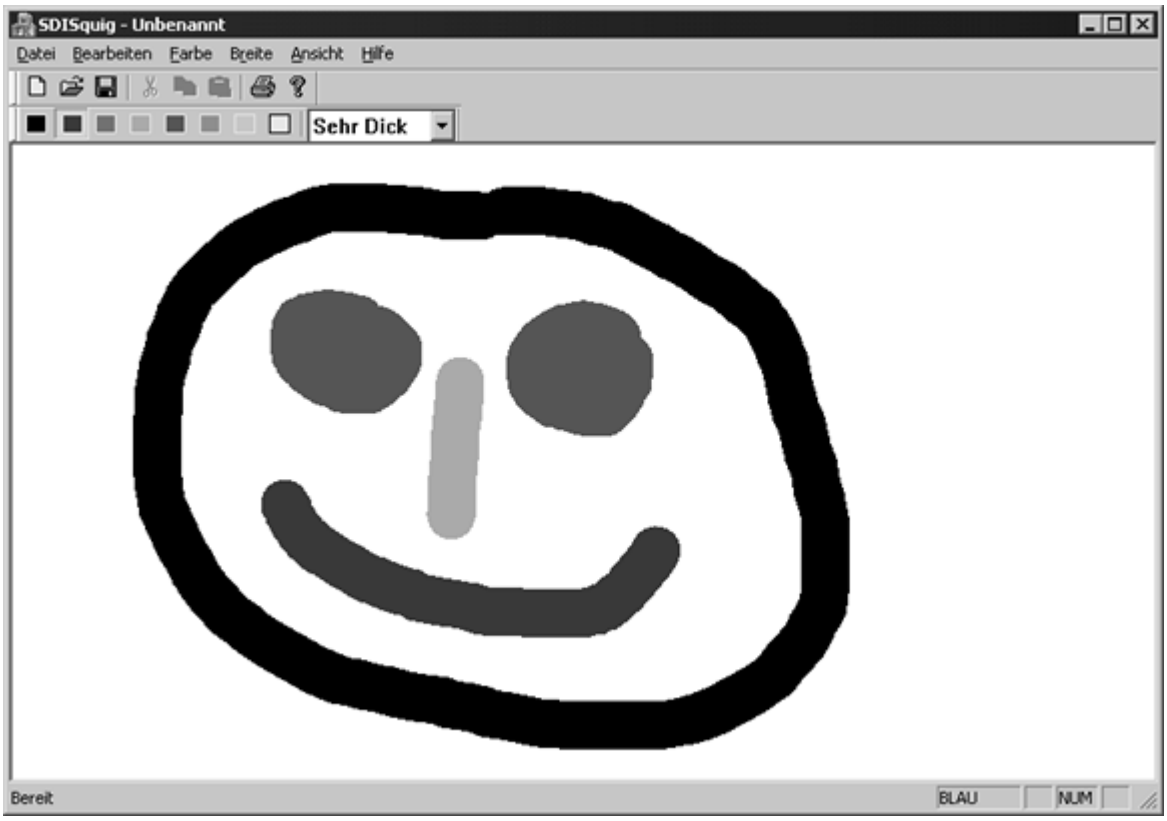


Abbildung 11.7: Die Zeichenanwendung zeigt die aktuelle Farbe in der Statusleiste an.

11.5 Zusammenfassung

Der heutige Lehrstoff war recht umfangreich. (Zeichnet sich hier ein Trend ab?) Sie haben gelernt, wie man eigene Symbolleisten entwirft und erstellt. In diesem Zusammenhang haben Sie auch erfahren, wie man Aufforderungen in der Statusleiste für die Symbolleisten-Schaltflächen und Menüs festlegt sowie QuickInfos anzeigt, wenn man die Maus ein paar Sekunden über einer Symbolleisten-Schaltfläche ruhen lässt. Sie wissen nun, wie man diese Symbolleisten erstellt und sie mit dem Anwendungsgerüst verbindet. Weiterhin haben Sie gelernt, wie man über einen Menübefehl steuert, ob die Symbolleiste sichtbar ist.

Als Nächstes sind wir darauf eingegangen, wie man ein Kombinationsfeld in einer Symbolleiste unterbringt, damit man dem Benutzer der Anwendung die gleiche Bequemlichkeit bieten kann, die er von anderen bekannten Softwarepaketen her gewohnt ist. Gleichzeitig haben Sie erfahren, wie man ein Kombinationsfeld per Code erzeugt, ohne dass man mit dem Dialog-Designer arbeiten muss, und wie man die Dropdown-Liste des Kombinationsfelds mit Texteinträgen füllt. Dann wurde gezeigt, wie man das Kombinationsfeld in die Anwendung einbindet, indem man Behandlungsroutinen für die Ereignisse des Kombinationsfelds erstellt, und wie man das Kombinationsfeld aktualisiert, um die am Anwendungsmenü vorgenommenen Änderungen widerzuspiegeln.

Schließlich hat der heutige Tag gezeigt, wie man eigene Ausschnitte in die Statusleiste aufnimmt und wie man den Ausschnitt aktualisiert, um den momentanen Status der Anwendung wiederzugeben.

11.6 Workshop

Fragen und Antworten

Frage:
Wenn ich ein neues Dokument beginne, bleibt die Anzeige des Kombinationsfeldes zunächst unverändert auf dem alten Wert stehen, obwohl die Applikation nun mit dem sehr dünnen Stift malt.

Antwort:
Sie müssen in der Funktion CSDISquigDoc::OnNewDocument nach dem Zuweisen der neuen Breite an die Variable m_nWidth die Funktion UpdateColorbar aufrufen um die Anzeige des Kombinationsfeldes zu korrigieren.

Frage:
In manchen Anwendungen hat man bei Symbolleisten die Option, Text anzuzeigen, beispielsweise im Internet Explorer. Wie lässt sich Text für meine Symbolleisten- Schaltflächen hinzufügen?

Antwort:
Leider bietet der Symbolleisten-Designer keine Möglichkeit, um Symbolleisten- Schaltflächen einen Text zuzuordnen. Das bedeutet, dass man den Text per Anwendungscode in die Schaltflächen aufnehmen muss, genauso wie man für alle Schaltflächen der Symbolleiste Farben das Verhalten als Optionsfelder realisiert. Mit der Funktion SetButtonText legen Sie den Text auf jeder Symbolleisten- Schaltfläche individuell fest. Diese Funktion übernimmt zwei Argumente: den Index der Schaltfläche und den Text für die Schaltfläche. Wenn Sie wirklich Text auf den Symbolleisten-Schaltflächen unterbringen wollen, müssen Sie außerdem die Symbolleiste in der Größe ändern, um Platz für den anzuzeigenden Text zu schaffen.

Frage:
Ich habe im Symbolleisten-Editor einige Änderungen an der Symbolleiste Farben vorgenommen und erhalte nun jedesmal einen Assertion-Fehler, wenn ich die Anwendung auszuführen versuche. Was ist passiert?

Antwort:
Das Problem besteht darin, dass der Symbolleisten-Designer die Separatoren gefunden hat, die Sie in die Ressourcendatei als Platzhalter für das Kombinationsfeld aufgenommen haben. Der Symbolleisten-Designer nimmt an, dass es sich hier um Fehler handelt, und entfernt die Platzhalter automatisch. Der Fehler tritt auf, weil Sie versuchen, mit einem Steuerelement in der Symbolleiste Farben zu arbeiten, das nicht existiert. Um das Problem zu umgehen, öffnen Sie die Ressourcendatei erneut im Editor und fügen die beiden Separatoren wieder am Ende der Definition der Farben-Symbolleiste an. Kompilieren Sie die Anwendung dann neu.

Frage:
Das Kombinationsfeld in meinen Symbolleisten sieht zu groß aus. Wie kann ich erreichen, dass es sich harmonischer in die Symbolleiste einfügt?

Antwort:
Um das Kombinationsfeld in die Symbolleiste einzupassen, wie es zum Beispiel im Visual Studio von Visual C++ zu sehen ist, sind eine Reihe von Dingen zu erledigen:

1. Verringern Sie den oberen Teil des Kombinationsfelds um 3. Damit entsteht ein schmaler Rand zwischen dem oberen Rand des Kombinationsfeldes und dem Rand der Symbolleiste.
2. Wählen Sie für das Kombinationsfeld eine kleinere Schrift, die sich besser in die Symbolleiste einfügt. Experimentieren Sie etwas mit Schriften und Schriftabständen, bis Sie das Optimum für das Kombinationsfeld in der Symbolleiste gefunden haben.

Frage:
Wie kann ich den Text im ersten Abschnitt der Statusleiste festlegen, ohne Menü- und Symbolleisten-Aufforderungen zu verwenden?

Antwort:
Mit SetWindowText können Sie den Text im ersten Ausschnitt der Statusleiste festlegen. Per Voreinstellung ist der erste Ausschnitt in der Statusleiste ein Separator, der sich automatisch erweitert und dann die Breite der Statusleiste abzüglich der anderen nach rechts ausgerichteten Ausschnitte in der Leiste einnimmt. Ruft man die Funktion SetWindowText auf der Statusleisten-Variablen auf, wird nur der Text im ersten Ausschnitt festgelegt. Wenn Sie den Text in einem der anderen Ausschnitte ändern möchten und das zu anderen Zeiten als in der Behandlungsroutine der Nachricht ON_UPDATE_COMMAND_UI, können Sie mit der Funktion

SetPaneText arbeiten. Es gibt zwei Möglichkeiten, um den Text im Hauptteil der Statusleiste zu setzen. Die erste sieht folgendermaßen aus:

```
CString myString = "Das ist mein String";
m_wndStatusBar.SetWindowText(myString);
```

Antwort:
Die zweite Methode lautet:

```
CString* myString = "Das ist mein String";
m_wndStatusBar.SetPaneText(0, myString);
```

Quiz

1. Wie verbinden Sie eine Symbolleisten-Schaltfläche mit einem Menübefehl, der dieselbe Funktion auslöst?

2. Wie kann man sicherstellen, dass eine Symbolleiste mit dem Rahmenfenster verankert ist?
3. Wie lässt sich die Statusanzeige für die numerische Feststelltaste aus der Statusleiste entfernen?
4. Warum muss man die Ressourcendatei bearbeiten, um ein Kombinationsfeld in eine Symbolleiste aufzunehmen?

Übungen

1. Fügen Sie einen weiteren Ausschnitt in die Statusleiste ein, um die momentan ausgewählte Breite anzuzeigen.

2. Nehmen Sie in die Hauptsymbolleiste eine Schaltfläche auf, mit der man die Symbolleiste Farben ein-/ausschalten kann, wie es Abbildung 11.8 zeigt.

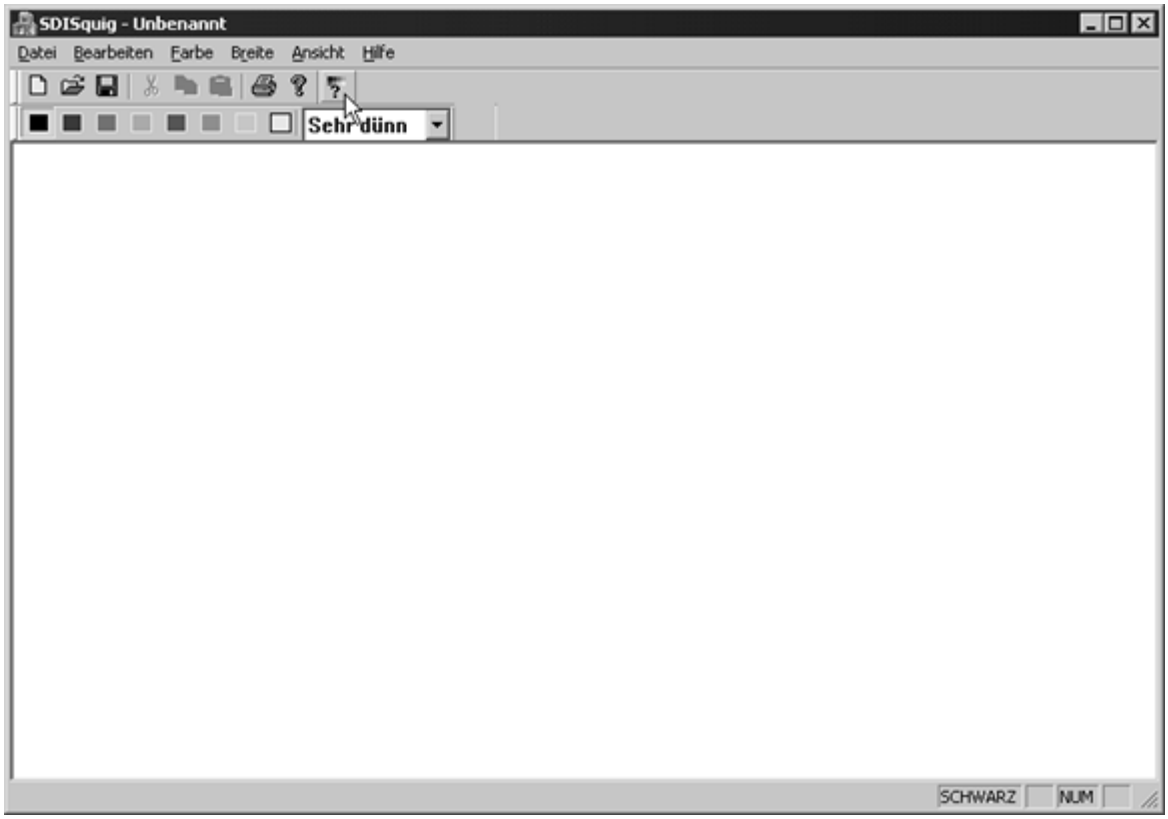


Abbildung 11.8: Der Umschalter für die Symbolleiste Farben

Tag 12

Dateizugriff

Die meisten Anwendungen bieten dem Benutzer die Möglichkeit, die eigene Arbeit zu speichern. Dabei kann es sich um ein Dokument einer Textverarbeitung, ein Tabellenblatt, eine Zeichnung oder eine Menge von Datensätzen handeln. In der heutigen Lektion untersuchen wir, welche Mittel Visual C++ bereitstellt, um diese Funktionalität in einfacher Weise zu implementieren.

Sie werden lernen, wie man ...

- C++-Streams in Visual C++ einsetzt, um Informationen zur Anwendung zu speichern,
- die Anwendungsdaten in Binärdateien speichert,
- die Objekte einer Anwendung »serialisierbar« macht,
- Variablen unterschiedlicher Datentypen in ein und derselben Datei ablegt.

12.1 Serialisierung

Die so genannte Serialisierung zerfällt in zwei Teile. Wenn Anwendungsdaten auf dem Systemlaufwerk in Form einer Datei gespeichert werden, spricht man von Serialisierung. Wird der Anwendungszustand aus der Datei wiederhergestellt, bezeichnet man diesen Vorgang als Deserialisierung. Die Kombination der beiden Teile ergibt die Serialisierung von Anwendungsobjekten in Visual C++.

Die Klassen CArchive und CFile

Die Serialisierung in Visual C++-Anwendungen läuft über die Klasse CArchive. Diese Klasse agiert als Input/Output (I/O)-Stream - auf deutsch Eingabe-/Ausgabe (E/A)-Strom (--) für ein CFile-Objekt, wie es Abbildung 12.1 zeigt. Die C++-Streams gewährleisten einen effizienten Datenfluss in und aus einer Datei, die als Speicher der Anwendungsdaten dient. Die Klasse CArchive kann nicht ohne ein Objekt der Klasse CFile existieren, an die sie gebunden ist.

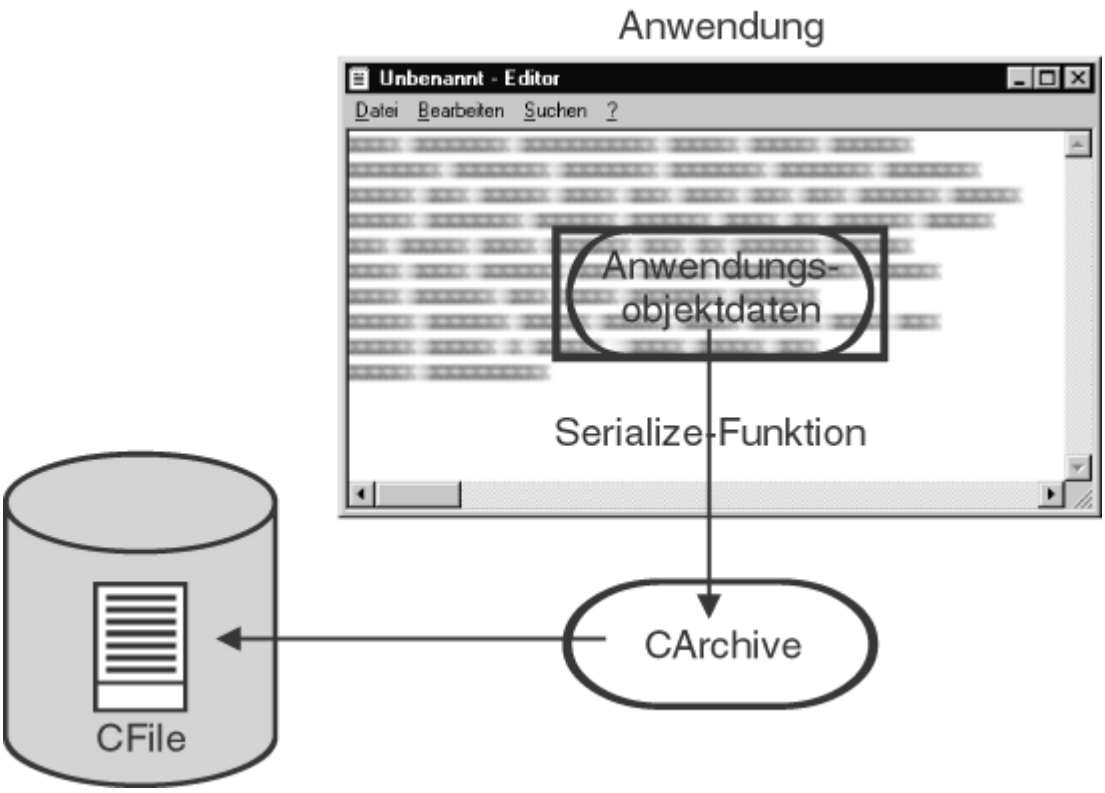


Abbildung 12.1: Die Klasse CArchive speichert Anwendungsdaten in einem CFile-Objekt.

Die Klasse CArchive kann Daten in einer Vielzahl verschiedener Dateitypen speichern, die alle von der Klasse CFile abgeleitet sind. In der Standardeinstellung bindet der AppAssistent die gesamte Funktionalität ein, um normale CFile-Objekte für den Einsatz mit CArchive zu erzeugen und zu öffnen. Wenn Sie mit einem der anderen Dateitypen arbeiten wollen, müssen Sie zusätzlichen Code in Ihre Anwendung einbinden, um die Verwendung abweichender Dateitypen zu ermöglichen.

Die Funktion Serialize

Die Klasse CArchive kommt in der Funktion Serialize auf den Dokument- und Datenobjekten in Visual C++-Anwendungen zum Einsatz. Wenn eine Anwendung eine Datei liest oder schreibt, ruft sie die Funktion Serialize des Dokumentobjekts auf und übergibt das CArchive-Objekt, das das Schreiben in oder das Lesen aus der Datei abwickelt. In der Funktion Serialize ist zunächst zu bestimmen, ob das Archiv zu schreiben oder zu lesen ist. Dazu ruft man die Funktionen IsStoring oder IsLoading der Klasse CArchive auf. Aus dem Rückgabewert dieser Funktionen lässt sich ablesen, ob die Anwendung in/aus I/ O-Streams der Klasse CArchive schreiben bzw. lesen muss. Eine typische Serialize- Funktion in der Dokumentklasse ist in Listing 12.1 wiedergegeben.

Listing 12.1: Eine typische Serialize-Funktion

```
 1: void  CAppDoc::Serialize(CArchive& ar)
 2: {
 3:   // Wird in das Archiv geschrieben?
 4:   if (ar.IsStoring())
 5:   {
 6:     // Ja, Variable schreiben
 7:     ar << m_MyVar;
 8:   }
 9:   else
10:   {
11:     // Nein, Variable lesen
12:     ar >> m_MyVar;
13:   }
14: }
```

Eine Serialize-Funktion können Sie in alle von Ihnen erstellten Klassen aufnehmen und damit deren Serialize-Funktion von der Funktion Serialize des Dokuments aus aufrufen. Wenn Sie Ihre benutzerdefinierten Objekte in ein Objektarray stellen, wie zum Beispiel das in der Zeichenanwendung der letzten beiden Tage verwendete CObArray, lässt sich die Serialize-Funktion des Arrays von der Serialize-Funktion des Dokuments aufrufen. Das Objektarray ruft seinerseits die Serialize-Funktion des jeweiligen Objekts auf, das im Array gespeichert ist.

Objekte serialisierbar machen

Als Sie am Tag 10, »SDI- und MDI-Anwendungen«, die Klasse CLine erstellt haben, mussten Sie zwei Makros einbinden, bevor Sie die Zeichnungen speichern und wiederherstellen konnten. Die beiden Makros DECLARE_SERIAL und IMPLEMENT_SERIAL binden in Ihre Klassen die erforderliche Funktionalität ein, damit die Funktion Serialize ordnungsgemäß arbeiten kann.

Das Makro DECLARE_SERIAL einbinden

In die Klassendeklaration ist das Makro DECLARE_SERIAL gemäß Listing 12.2 einzubinden. Dieses Makro übernimmt als einziges Argument den Klassennamen und fügt automatisch in Ihre Klasse bestimmte Standardfunktionen und Operatordeklarationen ein, die für den korrekten Ablauf der Serialisierung notwendig sind.

Listing 12.2: Das Makro DECLARE_SERIAL in die Klassendeklaration einbinden

```
1: class CMyClass : public  CObject
2: {
3:   DECLARE_SERIAL (CMyClass)
4: public:
5:   virtual void Serialize(CArchive &ar);
6:   CMyClass();
7:   virtual ~CMyClass();
8: };
```

DECLARE_SERIAL ändert den Zugriffsbereich auf public. Wenn Sie nach dem Makro DECLARE_SERIAL noch Elemente in Ihre Klassendeklaration einfügen, müssen Sie den Zugriff für sie explizit deklarieren, da Sie sich nicht mehr auf den Standardzugriff private verlassen können.

Das Makro IMPLEMENT_SERIAL einbinden

Das Makro IMPLEMENT_SERIAL ist in die Implementierung Ihrer Klasse einzubinden. Es muss außerhalb aller anderen Klassenfunktionen erscheinen, da es den Code für diejenigen Klassenfunktionen hinzufügt, die mit dem Makro DECLARE_SERIAL deklariert werden.

Das Makro IMPLEMENT_SERIAL übernimmt drei Argumente:

- den Klassennamen, wie beim Makro DECLARE_SERIAL.,
- den Namen der Basisklasse, von der Ihre Klasse abgeleitet ist,
- eine Versionsnummer, aus der man bestimmen kann, ob eine in die Anwendung einzulesende Datei in der richtigen Version vorliegt. Die Versionsnummer muss positiv sein und sollte jedesmal inkrementiert werden, wenn Sie Änderungen an der Serialisierungsmethode der Klasse vornehmen und sich dabei die mit einer Datei auszutauschenden Datenformate ändern.

Listing 12.3 zeigt einen typischen Einsatz des Makros IMPLEMENT_SERIAL.

Listing 12.3: Das Makro IMPLEMENT_SERIAL in die Implementierung der Klasse einbinden

```
 1: #include "stdafx.h"
 2: #include "MyClass.h"
 3:
 4: IMPLEMENT_SERIAL (CMyClass, CObject, 1)
 5:
 6: CMyClass::CMyClass()
 7: {
 8: }
 9:
10: CMyClass::~CMyClass()
11: {
12: }
```

Die Funktion Serialize definieren

Neben den beiden Makros müssen Sie eine Serialize-Funktion in Ihre Klasse einbinden. Diese Funktion sollte vom Typ void mit einem einzelnen Argument vom Typ CArchive mit dem Namen &ar, dem Zugriffsstatus public und eingeschaltetem Kontrollkästchen virtual deklariert sein, woraus die Funktionsdeklaration in Listing 12.2 hervorgeht. Wenn Sie die Funktion Serialize für Ihre Klasse implementieren, gehen Sie normalerweise genauso vor, wie bei der in Listing 12.1 gezeigten Dokumentklasse. Auch hier ist zu prüfen, ob die Datei zu lesen oder zu schreiben ist.

MFC-Exkurs: Die Klasse CFile

Die Klasse CFile ist die wichtigste Klasse, mit der Sie Dateien lesen und schreiben. Sie verwendet mehrere abgeleitete Klassen, um bestimmte Dateitypen zu lesen und zu schreiben, von denen manche nicht einmal Dateien sind. Mit manchen, wie CSocketFile, können Sie Netzwerkkommunikation mit der gleichen Funktionalität betreiben, mit der Sie Dateien auf einem Laufwerk lesen oder schreiben. Sie müssen die Klasse CFile nicht zusammen mit der Klasse CArchive verwenden, um Dateien durch Serialisierung zu lesen und zu schreiben (das Hauptthema des heutigen Tages), sondern können direkt mit der Klasse CFile Dateien in jedem gewünschten Format lesen und schreiben.

Dateien öffnen und schließen

Sie haben einige Optionen für das Öffnen von Dateien. Sie können die Datei beim Erzeugen des CFile-Objekts öffnen oder danach. Wenn Sie die Datei beim Erzeugen des CFile-Objekts öffnen wollen, müssen Sie einen Zeiger auf ein CFile-Objekt deklarieren und dann den CFile-Konstruktor mit den Informationen zum Öffnen aufrufen:

```
CFile *pComFile = new CFile(strFile, CFile::modeRead);
```

Wenn Sie die Datei serialisieren, müssen Sie statt einen Zeiger zu verwenden eine Instanz des CFile-Objekts deklarieren:

```
CFile fComFile(strFile, CFile::modeRead);
```

Der erste Parameter, den Sie dieser Version des CFile-Konstruktors übergeben, ist der Name und Pfad der Datei. Der zweite Parameter ist ein Flag, der kontrolliert, wie die Datei geöffnet wird. Sie können mehr als ein Flag, mit | (binäres Oder) verknüpft, verwenden. In Tabelle 12.1 ind die möglichen Flageinstellungen für diesen Parameter aufgelistet.

Flag	Beschreibung
CFile::modeCreate	Erstellt eine neue Datei. Wenn die Datei schon existiert, wird sie gelöscht.
CFile::modeNoTruncate	Wenn die Datei schon existiert, wird sie nicht gelöscht. Es wird gewöhnlich mit CFile::modeCreate zusammen verwendet.
CFile::modeRead	Öffnet die Datei nur zum Lesen (Read only)
CFile::modeReadWrite	Öffnet die Datei zum Lesen und Schreiben

Tabelle 12.1: CFile-Flags für den Modus beim Öffnen

Flag	Beschreibung
CFile::modeWrite	Öffnet die Datei nur zum Schreiben (Write only)
CFile::modeNoInherit	Verhindert, dass die Datei von untergeordneten Prozessen ererbt wird
CFile::shareDenyNone	Erlaubt anderen Prozessen, die Datei zu öffnen, während Sie sie geöffnet haben
CFile::shareDenyWrite	Verhindert, dass andere Prozesse die Datei im Schreibmodus öffnen können (andere Prozesse können die Datei immer noch im Lesemodus öffnen)
CFile::shareDenyRead	Verhindert, dass andere Prozesse die Datei im Lesemodus öffnen können (andere Prozesse können die Datei immer noch im Schreibmodus öffnen)
CFile::shareDenyExclusive	Verhindert, dass andere Prozesse die Datei öffnen, während Sie sie geöffnet haben
CFile::typeText	Die Datei wird im »Text«-Modus geöffnet, das heißt Sie können keine binären Daten aus der Datei lesen oder in sie schreiben. Wird nicht in der Klasse CFile verwendet, aber in abgeleiteten Klassen. Das Zeichen STRG-Z wird als Endmarkierung der Datei interpretiert, die einzelnen Zeilenvorschubzeilen (LF "\n") werden bei der Ausgabe in die Zeichenkombination Zeilenrücklauf/Zeilenvorschub (CR/LF "\r\n") umgewandelt.
CFile::typeBinary	Die Datei wird im »Binär«-Modus geöffnet, das heißt Sie können binäre Daten aus der Datei lesen oder in sie schreiben. Wird nicht in der Klasse CFile verwendet, aber in abgeleiteten Klassen
CFile::osNoBuffer	Die Datei wird ohne Puffer oder Zwischenspeicherung geöffnet.
CFile::osWriteThrough	Das System schreibt direkt auf die Festplatte, Zwischenspeicher wird umgangen.
CFile::osRandomAccess	Auf die Datei wird zufällig zugegriffen. Dem Betriebssystem wird signalisiert, dass der Zwischenspeicher für zufälligen Zugriff optimiert werden soll.
CFile::osSequentialScan	Auf die Datei wird sequenziell von Anfang bis Ende zugegriffen. Dem Betriebssystem wird signalisiert, dass der Zwischenspeicher für sequenziellen Zugriff optimiert werden soll.

Tabelle 12.1: CFile-Flags für den Modus beim Öffnen (Forts.)

Tritt beim Öffnen der Datei mit dieser Methode ein Problem auf, wird eine CFileException- Ausnahme erzeugt. Daher sollten Sie diese Methode zum Öffnen immer in einen try...catch-Rahmen einschließen:

```
CFile *pComFile;
try
{
  pComFile = new CFile(strFile, CFile::modeRead);
...
}
catch (CFileException *e)
{
...
  e->Delete();
}
```

oder

```
try
{
  CFile fComFile(strFile, CFile::modeRead);
```

```
...
}
catch (CFileException *e)
{
...
  e->Delete();
}
```

Die andere Möglichkeit, Dateien zu öffnen, ist die Verwendung der Methode Open, die folgende Syntax besitzt:

```
BOOL Open(LPCTSTR strFile, UINT iOpenFlags, CFileException* pErr);
```

Die ersten beiden Parameter, strFile und iOpenFlags, sind die gleichen wie bei der ersten Methode zum Öffnen der Datei. Der dritte Parameter, pErr, ist ein Zeiger auf eine CFileException-Instanz, die entstehende Fehler beim Versuch, die Datei zu öffnen, empfängt. Dieser Ansatz unterscheidet sich von der ersten Methode, da Sie hier keine try...catch-Behandlung zum Abfangen von auftretenden Fehlern verwenden müssen. Stattdessen müssen Sie den Rückgabewert überprüfen, um zu sehen, ob die Datei erfolgreich geöffnet wurde. Wenn nicht, überprüfen Sie das in die Funktion Open übergebene Ausnahmeobjekt, um das mögliche Problem zu ermitteln.

Wenn Sie die Datei wieder schließen wollen, verwenden Sie die Funktion Close. Close hat keine Parameter und gibt kein Ergebnis zurück, kann jedoch beim Auftreten von Problemen eine CFileException-Ausnahme erzeugen. Für Fälle, in denen Sie die Datei unabhängig von Situation und Bedingung schließen und eventuelle Ausnahmebedingungen ignorieren müssen, verwenden Sie die Methode Abort. Abort übernimmt ebenfalls keine Parameter und gibt kein Ergebnis zurück, erzeugt jedoch niemals eine Ausnahme, unter welchen Umständen auch immer.

Das Zerstören eines Cfile-Objektes bewirkt nicht, dass die zugehörige Datei von Betriebssystem geschlossen wird.

Aus Dateien lesen und in Dateien schreiben

Wenn Sie aus geöffneten Dateien lesen und in sie schreiben möchten, können Sie die Methoden Read und Write verwenden. Die Funktion Read hat folgende Syntax:

```
UINT Read(void* lpBuf, UINT iCount);
```

Der erste Parameter, lpBuf, ist ein Zeiger auf einen Puffer, in den Sie die Datei einlesen können, wenn Sie das wollen. Der zweite Parameter, iCount, ist die Größe des Puffers. Die Funktion Read gibt die Anzahl der in den Puffer eingelesenen Bytes zurück. Häufig wird diese Funktion folgendermaßen verwendet:

```
char buf[100];
int iRead;
iRead = file.Read(buf, sizeof(buf));
```

Tritt beim Lesen der Datei ein Problem auf, beispielsweise beim Erreichen des Endes, wird ein CFileException ausgelöst. Daher müssen Sie die Funktion Read in eine try...catch-Struktur einschließen.

Wenn Sie Daten in eine Datei schreiben wollen, verwenden Sie die Funktion Write, die im Grunde der Funktion Read entspricht. Write besitzt folgende Syntax:

```
void Write(const void* lpBuf, UINT iCount);
```

Der Hauptunterschied zwischen Read und Write ist, dass Write nicht die Anzahl der geschriebenen Bytes

zurückgibt. Der erste Parameter ist ein Zeiger auf einen Puffer, der die zu schreibenden Daten enthält. Der zweite Parameter gibt an, wie viele Daten aus dem Puffer geschrieben werden sollen. Tritt beim Schreiben der Daten in die Datei ein Problem auf, wird eine CFileException ausgelöst.

In Dateien navigieren

Von Zeit zu Zeit, abhängig von der Arbeitsweise Ihrer Anwendung, müssen Sie möglicherweise in der Datei navigieren. Das ist bei Dateien mit einem Header-Abschnitt üblich, in dem Informationen zum Beispiel über die Anzahl der Einträge in der Datei oder Ähnliches enthalten sind. Manchmal müssen Sie auch spezielle Werte aus der Datei lesen oder in sie schreiben. In diesen Situationen lesen oder schreiben Sie die Datei nicht von Anfang bis Ende. Stattdessen lesen Sie Daten an einer Stelle in der Datei, gehen zu einer anderen Stelle und lesen dort wieder und gehen dann wieder zu einer anderen Stelle, um dort Daten zu lesen. Datenbanken arbeiten für gewöhnlich auf diese Weise, das Datenbank-Programm navigiert in einer sehr großen Datei, um bestimmte Einträge und Werte aus dieser Datei zu lesen.

Die ersten Funktionen, die Sie für diese Art Funktionalität verwenden müssen, liefern Ihnen die benötigten Informationen. Die erste benötigte Information ist die Größe der Datei. Diese können Sie mit der Funktion GetLength ermitteln, die die Größe der geöffneten Datei in Bytes zurückgibt. Die nächste Funktion, GetPosition, gibt Ihre aktuelle Position in der Datei zurück. Die Position wird in Bytes vom Anfang der Datei aus angegeben.

Es gibt drei Funktionen, mit denen man in einer Datei navigieren kann: Seek, SeekToBegin und SeekToEnd. Die Funktion Seek übernimmt zwei Parameter:

```
long Seek(long lOff, UINT iFrom);
```

Der erste Parameter, lOff, ist die Anzahl der Bytes, um die die Position verschoben werden soll, entweder in positiver oder in negativer Richtung (negative Werte verschieben die Position näher zum Dateianfang, positive Werte in Richtung Dateiende). Der zweite Parameter, iFrom, ist die Stelle in der Datei, die als Startpunkt für die Bewegung verwendet werden soll. In Tabelle 12.2 sind die möglichen Werte für diesen Parameter aufgelistet.

Wert	Beschreibung
CFile::begin	Verschiebt um die angegebene Bytezahl vom Anfang der Datei aus. Bei dieser Startposition muss der erste Parameter positiv sein.
CFile::current	Verschiebt um die angegebene Bytezahl von der aktuellen Position in der Datei aus
CFile::end	Verschiebt um die angegebene Bytezahl vom Ende der Datei aus. Ist der erste Parameter kleiner als 0, wird der Zeiger in die Datei hinein verschoben, ist der Wert gleich 0, wird das Dateiende erreicht, ist der Wert größer als 0, wird hinter das Ende der Datei verschoben.

Tabelle 12.2: Werte für Startposition in der CFile-Funktion Seek

Der Rückgabewert der Funktion Seek ist die aktuelle Position in der Datei.

Die anderen beiden Funktionen, SeekToBegin und SeekToEnd, verschieben die aktuelle Position in der Datei auf den Anfang bzw. das Ende der Datei. Diese Funktionen übernehmen keine Parameter. SeekToBegin positioniert Sie am Anfang der Datei und gibt keinen Ergebniswert zurück. SeekToEnd positioniert Sie am Ende der Datei und gibt die Gesamtlänge der Datei als Ergebnis zurück.

Alle Funktionen für die Navigation in Dateien können, genau wie die für die Ermittlung von Informationen wie Dateigröße oder Position in der Datei, beim Auftreten eines Fehlers ein CFileException auslösen. Es ist also zu raten, die gesamte Dateinavigation in einer try...catch-Struktur einzuschließen.

Dateien verwalten

Wenn Sie in Ihrer Anwendung Dateien verwalten müssen, können Ihnen ein paar Funktionen bei den grundlegenden Operationen helfen. Diese Funktionen sind statisch (static) und werden nicht mit einer aktiven Instanz der Klasse CFile zusammen, sondern unabhängig verwendet.

Wenn Sie eine Datei umbenennen möchten, können Sie die Funktion Rename verwenden:

```
CString strOldName, strNewName;
CFile::Rename(strOldName, strNewName);
```

Der erste Parameter der Funktion Rename ist der aktuelle Name der Datei. Der zweite Parameter ist der neue Name für die Datei. Diese Funktion gibt kein Ergebnis zurück, aber wenn sie auf ein Problem trifft, löst sie eine CFileException-Ausnahme aus. Die Funktion arbeitet nur mit Dateien, nicht mit Verzeichnissen.

Wenn Sie eine Datei löschen wollen, rufen Sie die Funktion Remove auf:

```
CString strName;
CFile::Remove(strName);
```

Der einzige Parameter, den die Funktion Remove übernimmt, ist der Name der zu löschenden Datei. Wie Rename gibt auch die Funktion Remove kein Ergebnis zurück, löst aber eine CFileException-Ausnahme aus, wenn ein Problem auftritt.

MFC-Exkurs: Die Klasse CFileException

Tritt während der Arbeit mit der Klasse CFile eine Ausnahme auf, wird eine CFileException- Ausnahme ausgelöst. Der größte Teil der Funktionalität, die Sie verwenden werden, ist eigentlich Teil der Vorfahr-Klasse CException, doch ein paar spezielle Funktionen der Klasse CFileException sind von Interesse.

CException-Funktionen

Wahrscheinlich werden Sie in den meisten Ausnahmesituationen drei Member- Funktionen des Vorfahr-Objekts CException verwenden. Die erste dieser Funktionen ist GetErrorMessage mit folgender Syntax:

```
BOOL GetErrorMessage(LPTSTR pstrError, UINT iMaxErr);
```

Der erste Parameter, pstrError, ist ein Zeiger auf ein Zeichenarray, das die Fehlernachricht empfangen wird. Der zweite Parameter, iMaxErr, ist die Größe des Zeichenarrays. Sie verwenden die Funktion folgendermaßen:

```
CFile *pComFile;
try
{
  pComFile = new CFile(strFile, CFile::modeRead);
...
}
catch (CFileException *e)
{
  char pMsg[255];
  if (e->GetErrorMessage(pMsg, sizeof(pMsg)))
    AfxMessageBox(pMsg);
...
  e->Delete();
}
```

Die Funktion ReportError bietet in einer einzigen Funktion die Funktionalität von GetErrorMessage und AfxMessageBox. ReportError hat folgende Syntax:

```
int ReportError(UINT iType, UINT iMsgID);
```

Beide Parameter dieser Funktion sind völlig optional. Der erste Parameter, iType, besteht aus den Meldungsboxtyp-Flags, die angeben, wie die Meldungsbox für den Benutzer angezeigt wird. Der zweite Parameter, iMsgID, ist die Ressourcen-ID eines Strings in der Stringtabelle, der als Standardmeldung verwendet wird, wenn die aufgetretene Ausnahme keine eigene Fehlermeldung besitzt. Der Rückgabewert dieser Funktion ist der gleiche wie der von MessageBox oder AfxMessageBox: die ID der ausgewählten

Schaltfläche.

Die letzte Funktion für das CException-Vorfahr-Objekt ist Delete, das immer aufgerufen werden sollte, um Ausnahmeobjekte nach ihrer Verarbeitung zu löschen. Die Funktion Delete übernimmt keine Parameter und führt die passenden Aufräumarbeiten für das Ausnahmeobjekt aus, abhängig von der Art, auf die das Objekt erzeugt wurde.

Elemente von CFileException

Mit dem CFileException-Objekt können Sie auf ein paar Klassenelemente zugreifen, um die Art des aufgetretenen Problems zu ermitteln. Das erste dieser Elemente ist m_cause. Sie können diesen Wert mit mehreren Konstanten vergleichen, um die Art des Problems und die als Reaktion auf die Ausnahme zu ergreifenden Maßnahmen festzustellen. In Tabelle 12.3 sind die möglichen Werte für die Variable m_cause aufgelistet.

Wert	Beschreibung
CFileException::none	Es ist kein Fehler aufgetreten.
CFileException::generic	Ein unbestimmbarer Fehler ist aufgetreten.
CFileException::fileNotFound	Die Datei wurde nicht gefunden.
CFileException::badPath	Der für die Datei angegebene Pfad ist ungültig.
CFileException::tooManyOpenFiles	Es sind zu viele Dateien geöffnet.
CFileException::accessDenied	Es konnte nicht auf die Datei zugegriffen werden, der Zugriff wurde verweigert.
CFileException::invalidFile	Das Datei-Handle ist ungültig. Das passiert meist, wenn Sie in einer Datei zu lesen oder zu schreiben versuchen, die nicht erfolgreich geöffnet wurde.
CFileException::removeCurrentDir	Das aktuelle Verzeichnis kann nicht gelöscht werden.
CFileException::directoryFull	Das Verzeichnis ist voll.
CFileException::badSeek	Beim Versuch, die Position des Dateizeigers zu setzen, trat ein Fehler auf.
CFileException::hardIO	Ein Hardware-Problem ist aufgetreten.

Tabelle 12.3: Werte für CFileException-Gründe

Ein weiteres möglicherweise interessantes Element der Klasse CFileException ist m_strFileName, das den Namen der Datei enthält, in der die Ausnahme aufgetreten ist.

MFC-Exkurs: Die Klasse CArchive

Wenn Sie einen Großteil des Lesens und Schreibens von Objekten aus und in Dateien automatisieren möchten und kein spezielles Dateiformat benötigen, ist es oft effizienter, das Lesen und Schreiben mithilfe der Funktion CArchive auszuführen. Sie haben CArchive bereits verwendet, um Dateien in der SDI-Anwendung zu lesen und zu schreiben, doch in diesem Fall stellte das MFC-Gerüst die gesamte Erstellungs- und Verwaltungsfunktionalität für Sie bereit. Sie selbst mussten nur noch die Serialisierungsfunktionalität in jeder Klasse bereitstellen, die in der Datei gespeichert werden sollte.

Eine CArchive-Instanz erzeugen und schließen

Der Schlüssel zur Arbeit mit einem CArchive-Objekt ist die Art, auf die es erstellt wird. Sie müssen bereits mit der CFile-Klasse eine Datei geöffnet haben, bevor Sie Ihre Instanz der CArchive-Klasse erzeugen. Ein CArchive-Objekt wird mit folgender Syntax erstellt:

```
CArchive(CFile* pFile, UINT iMode, int iBufSize, void* pBuf);
```

Der erste Parameter, pFile, ist ein Zeiger auf das CFile-Objekt für die zu lesende oder zu schreibende Datei. Der zweite Parameter, iMode, gibt an, ob das CArchive-Objekt zum Lesen oder zum Schreiben verwendet wird. In Tabelle 12.4 sind die möglichen Werte für dieses Flag aufgelistet.

Modus	Beschreibung
CArchive::load	Liest Daten aus der Datei
CArchive::store	Schreibt Daten in die Datei
CArchive::bNoFlushOnDelete	Verwenden Sie dieses Flag mit Vorsicht. Es verhindert, dass das CArchive-Objekt automatisch alle anhängigen Datenschreibvorgänge auf die Festplatte überträgt, bevor es zerstört wird. Wenn Sie dieses Flag verwenden, müssen Sie den Code bereitstellen, um alle Daten vor der Zerstörung des CArchive-Objekts auf die Festplatte zu schreiben.

Tabelle 12.4: Werte für den CArchive-Modus

Modus	Beschreibung
CFileException::sharingViolation	Es trat ein Problem bei der gemeinsamen Nutzung der Datei auf.
CFileException::lockViolation	Es wurde versucht, einen bereits gesperrten Bereich zu sperren.
CFileException::diskFull	Die Festplatte ist voll.
CFileException::endOfFile	Das Dateiende wurde erreicht.

Tabelle 12.4: Werte für den CArchive-Modus (Forts.)

Der dritte Parameter, iBufSize, gibt die Größe des zu verwendenden Datenpuffers an. Der Standardwert für diesen Parameter ist 4096. Sie müssen diesen Parameter nur übergeben, wenn Sie eine andere Puffergröße benötigen. (Wenn Sie sehr große Dateien lesen oder schreiben, könnten Sie aus Effizienzgründen einen größeren Puffer benötigen.) Der letzte Parameter, pBuf, ebenfalls optional, ist ein Zeiger auf einen Puffer, wenn Sie einen eigenen Puffer übergeben möchten oder müssen, der vom CArchive-Objekt für das Lesen und Schreiben verwendet werden soll.

Wenn Sie ein CArchive-Objekt schließen wollen, können Sie die Methode Close verwenden. Die Funktion Close und der Konstuktor können beide eine CFileException- oder CArchiveException-Ausnahme erzeugen. Der Konstruktor könnte darüber hinaus eine CMemoryException-Ausnahme (Unzureichender Speicher) erzeugen. Wenn Sie ein CArchive ohne Rücksicht auf mögliche Ausnahmen schließen wollen, verwenden Sie die Funktion Abort.

Lesen und Schreiben

Die Funktion IsStoring haben Sie bereits verwendet, um festzustellen, ob die Datei gelesen oder geschrieben wird. Sie haben auch schon über die Operatoren >> und << mithilfe von I/O Streams in Dateien geschrieben und aus ihnen gelesen (auch wenn Ihnen wahrscheinlich nicht klar war, dass Sie das tun). Es gibt auch noch die Funktion IsLoading. Dabei handelt es sich um das Gegenstück der Funktion IsStoring, das die gleichen Informationen liefert. Die Klasse CArchive besitzt auch Read- und Write- Funktionen, die grundsätzlich denen in der Klasse CFile entsprechen. Zwei Funktionen der Klasse CArchive, die Sie möglicherweise nutzen werden, sind ReadString und WriteString.

Die Funktion WriteString schreibt eine einzelne Textzeile in die Datei. Sie hat folgende Syntax:

```
void WriteString(LPCTSTR strText);
```

Wenn ein Fehler auftritt, wird eine CFileException-Ausnahme erzeugt. Die Funktion ReadString liest eine einzelne Textzeile:

```
BOOL ReadString(CString& strText);
```

Diese Funktion liest eine Textzeile aus der Datei und schreibt sie in die ihr als Parameter übergebene CString-Variable. Tritt ein Fehler auf, wird eine CFileException-Ausnahme erzeugt.

MFC-Exkurs: Die Klasse CArchiveException

Die Klasse CArchiveException stellt die Member-Variable m_cause bereit, genau wie die Klasse CFileException auch. Darüber hinaus besitzt sie die gesamte Funktionalität ihrer Vorfahr-Klasse CFileException. In Tabelle 12.5 sind die möglichen Werte für die Variable m_cause aufgelistet.

Wert	Beschreibung
CArchiveException::none	Es ist kein Fehler aufgetreten.
CArchiveException::generic	Ein unbestimmbarer Fehler ist aufgetreten.
CArchiveException::readOnly	Es wurde versucht, in ein Archiv zu schreiben, das zum Lesen geöffnet wurde.
CArchiveException::endOfFile	Das Dateiende wurde erreicht.
CArchiveException::writeOnly	Es wurde versucht, aus einem Archiv zu lesen, das zum Schreiben geöffnet wurde.
CArchiveException::badIndex	Die Datei besitzt ein ungültiges Format.
CArchiveException::badClass	Es wurde versucht, ein Objekt in die falsche Klasse einzulesen.
CArchiveException::badSchema	Es wurde versucht, ein Objekt mit einer anderen Versionsnummer zu lesen.

Tabelle 12.5: Cause-Werte von CArchiveException

C++-Exkurs: I/O-Streams

Am vierten Tag, »Maus und Tastatur«, haben Sie gelernt, wie man die Operatoren >> und << verwendet, um Bits nach links oder rechts zu verschieben. In den letzten Tagen haben Sie diese beiden Operatoren verwendet, um Variablen über die CArchive-Klasse in Dateien zu schreiben.

Die zweite Verwendung dieser Operatoren wurde zusammen mit I/O-Streams (Input/ Output-Streams - Ein-/Ausgabe-Datenströme) eingeführt, als die Programmiersprache C++ entwickelt wurde. Damals wurden die meisten C++-Anwendungen für eine Kommandozeilen-Schnittstelle erstellt, ohne dass man sich Gedanken um GUI-Aspekte machen musste. Die Idee war, Eingabe und Ausgabe gegenüber dem Aufwand in C zu vereinfachen. In C++ konnten Sie mit dem <<-Operator Meldungen auf den Bildschirm schreiben, indem Sie die Standardausgabe (stdout, auch Bildschirm genannt) links vom Operator und die auszugebende Variable rechts davon platzierten. Sie konnten sogar mit einer Reihe von <<-Operatoren mehrere Variablen verketten. Solange das Ausgabeziel am linken Ende der Zeile stand, war alles in Ordnung.

Mit der entgegengesetzten Methode konnte man Benutzereingaben von der Tastatur einlesen. Der >>-Operator wurde mit der Eingabequelle, gewöhnlich der Standardeingabe (stdin, auch Tastatur genannt), links vom Operator und der Variable, die die Eingabe aufnehmen sollte, rechts davon verwendet. Solange alle Benutzereingaben für den verwendeten Variablentyp einen Sinn ergaben, funktionierte alles perfekt.

Da es sich bei den drei wichtigsten Ein-/Ausgabe-Quellen und -Zielen - stdin, stdout und stderr - nur um Sonderfälle von Datei-Handles handelt, war die gleiche Verwendung der Operatoren << und >> auch für das Lesen und Schreiben von Dateien verfügbar. Tatsächlich können die meisten Operationen für das Lesen und Schreiben auf jedem anderen Gerät - sei es eine Datei, eine Netzwerkverbindung zu einem Programm oder zu einem anderen Computer oder ein Drucker - mithilfe dieser Operatoren ausgeführt werden. Es gibt Einschränkungen, wann und wo die Operatoren auf diese Art verwendet werden können, da die betroffenen Objekte in der Lage sein müssen, sie verstehen und Eingaben annehmen sowie Ausgaben erzeugen zu können.

12.2 Eine serialisierbare Klasse implementieren

Wenn Sie eine neue Anwendung entwerfen, müssen Sie als eine der ersten Aufgaben entscheiden, wie die Daten in der Dokumentklasse, die Ihre Anwendung erzeugt und mit der sie arbeitet, zu speichern sind. Erstellen Sie zum Beispiel eine datenbezogene Anwendung, die in der Art einer Kontakt-Datenbank Datenmengen vom Benutzer entgegennimmt, wie wollen Sie diese Daten im Arbeitsspeicher der Anwendung halten? Oder wenn Sie eine Textverarbeitung erstellen - wie wollen Sie das geschriebene Dokument im Speicher unterbringen? Oder ein Tabellenblatt? Oder ein Zeichenprogramm? Oder - na ja, Sie ahnen schon, worum es geht.

Nachdem Sie die Datenstrukturen entworfen haben, mit denen Ihre Anwendung arbeitet, können Sie bestimmen, wie sich Ihre Anwendung und die Klassen am besten serialisieren lassen. Wenn Sie alle Daten direkt in der Dokumentklasse aufbewahren, brauchen Sie sich nur darum zu kümmern, wie die Daten über das CArchive-Objekt in der Funktion Serialize des Dokuments geschrieben und gelesen werden. Wenn Sie Ihre eigene Klasse erstellen, um die Anwendungsdaten darin zu halten, müssen Sie die Funktionalität der Serialisierung in Ihre Datenklassen einbinden, damit sich die Objekte selbst speichern und wiederherstellen können.

In der heutigen Beispielanwendung erstellen Sie eine einfache, lineare Datenbankanwendung, die verdeutlicht, wie man eine Mischung von Datentypen in einem einzigen Datenstrom bei der Serialisierung der Anwendung kombinieren kann. Die Anwendung zeigt ein paar Datenfelder an, von denen einige als Strings mit variabler Länge und andere als Integer oder Bool festgelegt sind. Diese Variablen werden über einen einzigen Datenstrom in das CArchive-Objekt geschrieben bzw. daraus wiederhergestellt.

Eine serialisierte Anwendung erstellen

Für eine SDI- oder MDI-Anwendung kann man eigene Klassen erzeugen, die sich ebenfalls serialisieren lassen. Kurz gesagt kann jede Anwendung, die in irgendeiner Form mit Daten arbeitet, ob es sich nun um eine Datenbank oder ein Dokument handelt, serialisiert werden. Sie erstellen nun eine einfache, lineare Datenbankanwendung, die Sie dann serialisieren.

Eine lineare - oder »flache« - Datenbank ist der Urahn aller Datenbanken. Es handelt sich dabei um eine einfache, dateibasierte Datenbank, bei der die Datensätze sequentiell gespeichert sind und jeweils an den vorherigen Datensatz anschließen. Man vermisst hier die märchenhafte Funktionalität von relationalen Datenbanken, die mittlerweile zum Standard in der Datenbanktechnologie avanciert sind. Die Datenbank, die Sie in der heutigen Lektion erstellen, kommt einer alten - ohne Indexe arbeitenden - dBASE- oder Paradox-Datenbank näher als einer Datenbank wie Access oder SQL Server.

Das Anwendungsgerüst erstellen

Um mit Ihrer Anwendung zu beginnen, folgen Sie diesen Schritten:

1. Erstellen Sie ein neues MFC-Anwendung-Visual-C++-Projekt. Geben Sie dem Projekt den Namen Serialize.

2. Im Bereich Anwendungstyp wählen Sie die Option Einfaches Dokument.
3. Im Bereich Zeichenfolgen für Dokumentvorlagen geben Sie eine Dateinamenserweiterung für die Dateien an, die Ihre Anwendung erzeugt und liest (beispielsweise contact für Kontaktdatenbank, serial für serialisieren oder flat für flat- file database - lineare Datenbank).
4. Im Bereich Erstellte Klassen geben Sie die Basisklasse als CFormView an und klicken Sie auf Fertig stellen. Der MFC-Anwendungs-Assistent erzeugt das Anwendungsgerüst.

Wenn Sie CFormView als Basisklasse für die Ansichtsklasse angeben, erhalten Sie das informative Dialogfeld in Abbildung 12.2. Diese Meldung teilt Ihnen mit, dass das erstellte

Gerüst keine standardmäßig integrierte Druckunterstützung haben wird. Wenn Ihre Anwendung drucken können soll, müssen Sie dieses Merkmal selbst einfügen.

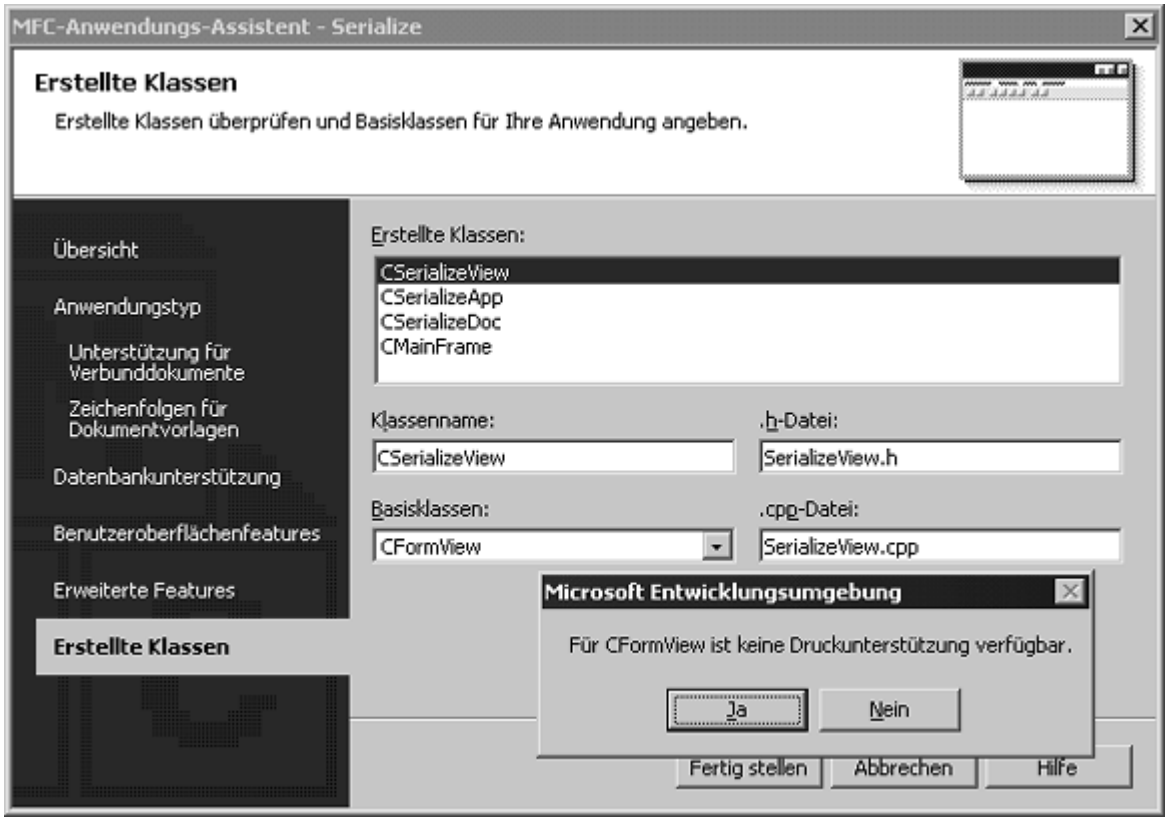


Abbildung 12.2: Die verwendete Ansichtsklasse und was daraus folgt.

5. Nachdem Sie mit dem Anwendungs-Assistenten das Gerüst Ihrer Anwendung erstellt haben, öffnen Sie den Dialog IDD_SERIALIZE_FORM. Sie sehen im Dialog-Designer eine große Fensterfläche wie bei einer dialogfeldbasierten Anwendung, nur dass die Schaltflächen OK und Abbrechen nicht vorhanden sind (siehe Abbildung 12.3).

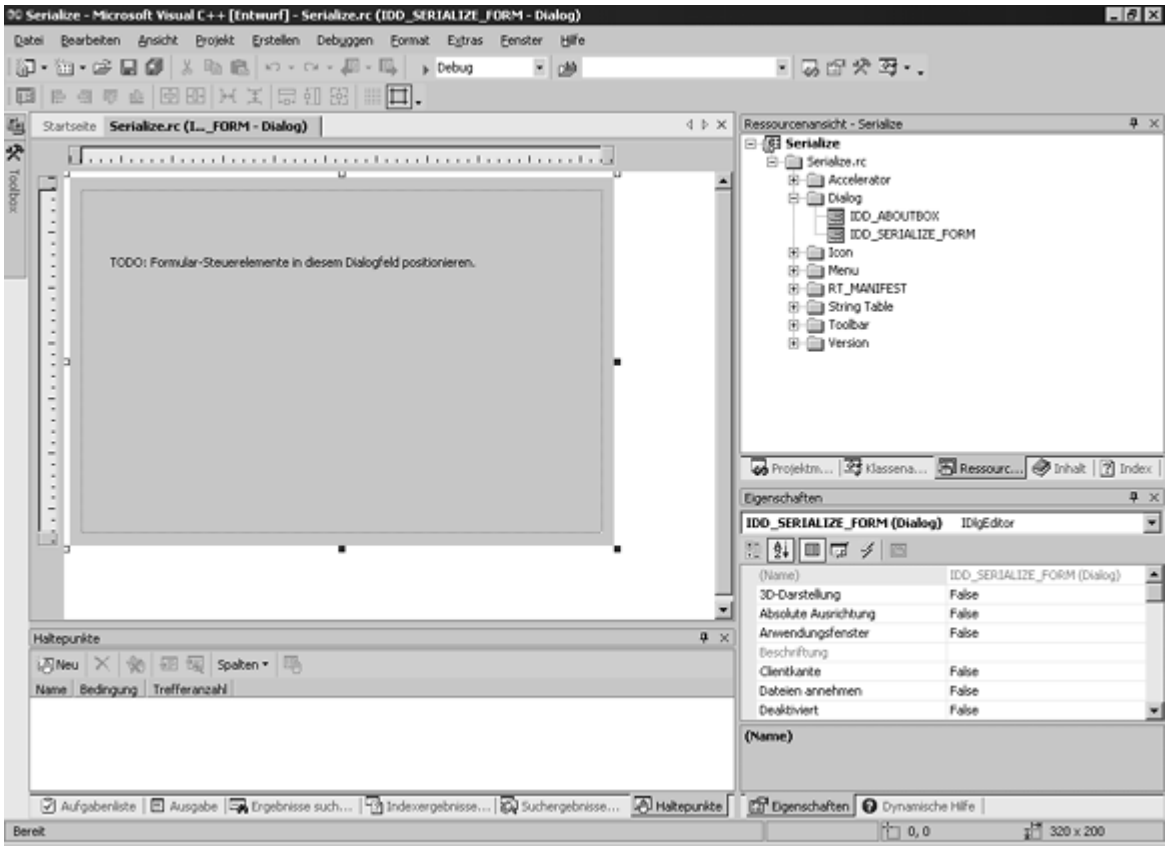


Abbildung 12.3: Der Dialog-Designer für eine SDI-Anwendung

Das Anwendungsfenster entwerfen

Nachdem Sie eine SDI- oder MDI-Anwendung erstellt haben, in der die Ansichtsklasse auf der Klasse CFormView basiert, müssen Sie nun die Ansicht der Anwendung entwerfen. Das geht fast genauso wie beim Entwurf eines Dialogfelds, allerdings brauchen Sie sich nicht um irgendwelche Schaltflächen zum Schließen des Fensters zu kümmern, wobei der Benutzer die Arbeit entweder speichert oder abbricht. Bei einer SDI- oder MDI- Anwendung ist die gesamte Funktionalität zum Speichern und Verlassen des Fensters traditionsgemäß in den Anwendungsmenüs oder der Symbolleiste untergebracht. Daher brauchen Sie nur Steuerelemente für die vom Anwendungsfenster zu realisierenden Funktionen vorzusehen.

Wenn Sie dialogfeldbasierte Anwendungen erstellen, nimmt der Anwendungs-Assistent keinerlei Code für die Serialisierung in Ihr Anwendungsgerüst auf. Wenn Sie eine derartige Anwendung serialisieren wollen, sind Sie für den entsprechenden Code selbst verantwortlich.

Für die heute zu erstellende Beispielanwendung platzieren Sie die Steuerelemente in der Fensterfläche gemäß Abbildung 12.4. Die Eigenschaften der jeweiligen Steuerelemente sind in Tabelle 12.6 aufgeführt.

Objekt	Eigenschaft	Einstellung
Text	Beschriftung	&Name
Eingabefeld	ID	IDC_ENAME
Text	Beschriftung	&Alter
Eingabefeld	ID	IDC_EAGE
Text	Beschriftung	Familienstand:

Optionsfeld	ID	IDC_RSINGLE
	Beschriftung	&Ledig
	Gruppe	True
Optionsfeld	ID	IDC_RMARRIED
	Beschriftung	&Verheiratet
Optionsfeld	ID	IDC_RDIVORCED
	Beschriftung	&Geschieden
Optionsfeld	ID	IDC_RWIDOWED
	Beschriftung	Ver&witwet
Kontrollkästchen	ID	IDC_CBEMPLOYED
	Beschriftung	Erwerbs&tätig
Schaltfläche	ID	IDC_BFIRST
	Beschriftung	&Erster
Schaltfläche	ID	IDC_BPREV
	Beschriftung	Vor&heriger
Schaltfläche	ID	IDC_BNEXT
	Beschriftung	N&ächster
Schaltfläche	ID	IDC_BLAST
	Beschriftung	Let&zter
Text	ID	IDC_SPOSITION
	Beschriftung	Datensatz 0 von 0

Tabelle 12.6: Einstellungen der Steuerelementeigenschaften

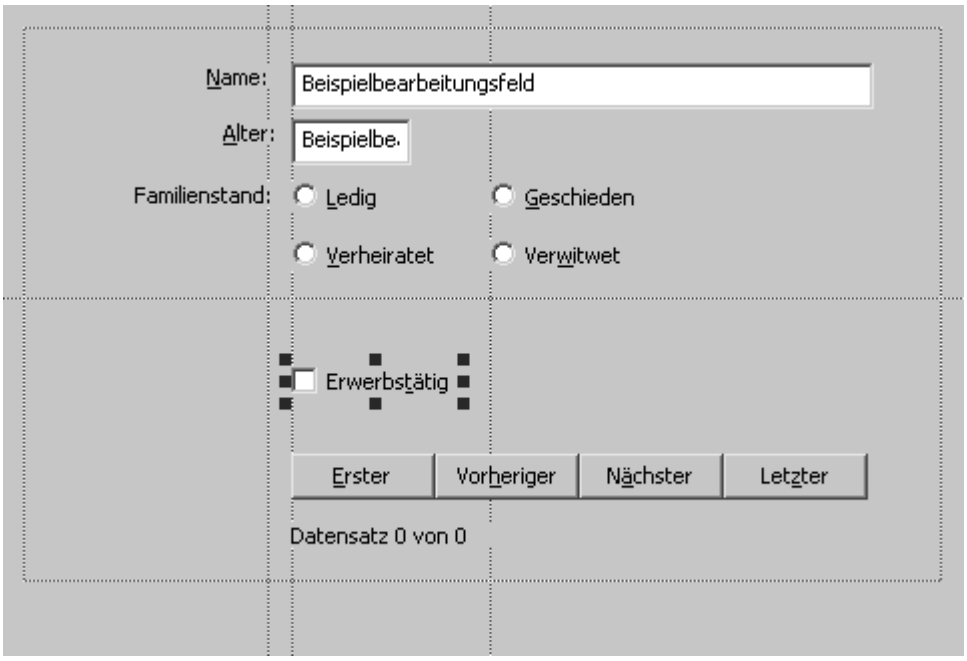


Abbildung 12.4: Das Layout der Beispielanwendung

Wenn Sie dialogfeldbasierte Anwendungen oder Fenster entwickeln, weisen Sie den Steuerelementen im Fenster Variablen in der Dialogfeldklasse zu. Wie sieht es nun bei einer SDI- oder MDI-Anwendung aus? In welcher Klasse erzeugen Sie die Variablen? Da die Funktion UpdateData ein Element der Klasse CWnd ist und die Ansichtsklasse von der Klasse CWnd abgeleitet ist, auch wenn das nicht für das Dokument zutrifft, bietet sich die Ansichtsklasse als logischer Platz an, um die Variablen aufzunehmen, die Sie den Steuerelementen im Fenster zuweisen.

Um Variablen mit den Steuerelementen in Ihrer Beispielanwendung zu verbinden, folgen Sie diesen Schritten:

1. Markieren Sie ein Steuerelement, klicken Sie mit der rechten Maustaste darauf und wählen Sie Variable hinzufügen aus dem Kontextmenü.

2. Nehmen Sie die in Tabelle 12.7 aufgeführten Variablen für die angegebenen Steuerelemente auf.

Objekt	Name	Kategorie	Typ	Zugriff
IDC_ENAME	m_strName	Value	CString	public
IDC_EAGE	m_iAge	Value	int	public
IDC_RSINGLE	m_iMaritalStatus	Value	int	public
IDC_CBEMPLOYED	m_bEmployed	Value	BOOL	public
IDC_SPOSITION	m_strPosition	Value	CString	public

Tabelle 12.7: Variablen für die Steuerelemente

Wenn Sie sich den Quellcode für die Ansichtsklasse ansehen, stellen Sie fest, dass es keine Funktion OnDraw gibt. Wenn Sie für Ihre SDI- oder MDI-Anwendung die Basisklasse CFormView verwenden, brauchen Sie sich um die Funktion OnDraw nicht zu kümmern. Stattdessen behandeln Sie die Ansichtsklasse genauso wie die Dialogfeldklasse in einem Dialogfenster oder einer dialogfeldbasierten Anwendung. Der Hauptunterschied besteht darin, dass die Daten, mit denen Sie die Steuerelemente im Fenster füllen, nicht in der Ansichtsklasse, sondern in der Dokumentklasse gespeichert sind. Im Ergebnis müssen Sie die Interaktion zwischen diesen beiden Klassen realisieren, um die Daten für die Steuerelemente in beiden Richtungen zu übertragen.

Eine serialisierbare Klasse erstellen

Wenn Sie eine formularbasierte Anwendung erstellen, kann man davon ausgehen, dass Ihre Anwendung mehrere Datensätze im Formular aufnimmt und der Benutzer durch die Datensätze navigieren kann, um Änderungen vorzunehmen. Der Benutzer kann zusätzliche Datensätze einfügen oder sogar Datensätze aus dem Recordset entfernen. Das eigentlich Interessante in dieser Phase der Anwendungsentwicklung ist, wie man diese Menge von Datensätzen darstellt und dabei die gesamte erforderliche Funktionalität unterstützt.

Eine Lösung besteht darin, eine Klasse zu erzeugen, die jeden Datensatz verkapselt, und dann diese Datensätze in ein Array aufzunehmen. Dieses Vorgehen entspricht der Zeichenanwendung, die Sie in den vergangenen Tagen erstellt und erweitert haben. Diese Klasse ist von der Klasse CObject abzuleiten und muss Variablen für alle Steuerelementvariablen enthalten, die Sie der Ansichtsklasse hinzugefügt haben. Außerdem gehören noch die Methoden dazu, um diese Variablen lesen und schreiben zu können. Sie müssen aber nicht nur die Methoden hinzufügen, um die Variablen zu setzen und zu lesen, Sie müssen auch die Klasse serialisierbar machen, indem Sie sowohl die Funktion Serialize als auch die beiden Makros, die die Serialisierung der Klasse realisieren, in die Klasse aufnehmen.

Die Basisklasse erstellen

Wie Sie in Lektion 10, »SDI- und MDI-Anwendungen«, gesehen haben, können Sie das Projekt in der Klassenansicht markieren, mit der rechten Maustaste klicken und Hinzufügen / Klasse hinzufügen aus dem Kontextmenü wählen, wenn Sie eine neue Klasse erstellen möchten.

Im Dialogfeld Klasse hinzufügen legen Sie den Typ der Klasse fest, dh. ob es sich um eine MFC-Klasse, eine allgemeine Klasse oder eine andere Klasse handelt. Um eine Klasse zu erzeugen, die die Daten eines Datensatzes enthalten kann, müssen Sie höchstwahrscheinlich eine allgemeine Klasse erstellen. Weiterhin müssen Sie Ihrer Klasse einen Namen verleihen und die Basisklasse angeben, von der die neue Klasse abzuleiten ist.

Folgen Sie für Ihre Beispielanwendung diesen Schritten:

1. Erstellen Sie eine neue Klasse, indem Sie mit der rechten Maustaste auf den Projekt- Knoten der Klassenansicht im Arbeitsbereich klicken und aus dem Kontextmenü Hinzufügen / Klasse hinzufügen wählen.

2. Wählen Sie Allgemeine Klasse als Klassentyp. Klicken Sie auf Öffnen, um den Generischen C++ Klassen-Assistent zu öffnen.
3. Nennen Sie Ihre Klasse CPerson. Geben Sie CObject als Basisklasse an.
4. Klicken Sie auf Fertig stellen, um die Klasse zu erstellen.

Nachdem Sie die neue Klasse erstellt haben, sind noch die Variablen hinzuzufügen, um die auf dem Bildschirm für den Benutzer anzuzeigenden Datenelemente aufzunehmen. Gemäß einem guten objektorientierten Entwurf deklariert man diese Variablen als privat, damit sie sich nicht durch andere Klassen manipulieren lassen. Die Variablentypen sollten den Typen der Variablen der Ansichtsklasse, die mit den Steuerelementen des Fensters verbunden sind, entsprechen.

Für die Beispielanwendung folgen Sie diesen Schritten:

1. Klicken Sie mit der rechten Maustaste auf den Klassenknoten CPerson in der Klassenansicht im Arbeitsbereich.
2. Fügen Sie die Variablen gemäß Tabelle 12.8 hinzu.

Name	Typ
m_bEmployed	BOOL
m_iAge	int
m_strName	CString
m_iMaritalStatus	int

Tabelle 12.8: Klassenvariablen für die Klasse CPerson

Methoden zum Lesen und Schreiben von Variablen hinzufügen

Nachdem die Klasse erstellt ist, müssen Sie Mittel bereitstellen, um die Variablen in der Klasse zu lesen und zu schreiben. Am einfachsten lässt sich das mit Inline-Funktionen in der Klassendefinition realisieren. Mit der einen Gruppe von Inline-Funktionen setzen Sie die Variablen, während Sie mit einer anderen Gruppe die aktuellen Werte der Variablen abrufen.

Eine Inline-Funktion ist eine kurze C++-Funktion, bei der der Funktionskörper beim Kompilieren der Anwendung anstelle des Funktionsaufrufs kopiert wird. Wenn die kompilierte Anwendung dann läuft, wird der Funktionscode ausgeführt, ohne dass ein Kontextsprung zur Funktion und nach Beendigung der Funktion wieder zurück erfolgen muss. Damit verringert sich der Overhead in der laufenden Anwendung, die Ausführungsgeschwindigkeit ist etwas besser, aber die ausführbare Anwendungsdatei nimmt etwas an Umfang zu. Je öfter die Inline-Funktion aufgerufen wird, desto größer ist schließlich die Anwendung.

Es gibt zwei Möglichkeiten, Inline-Funktionen zu erstellen. Bei der einen muss die gesamte Funktionsdefinition selbst in der Klassendeklaration eingebunden werden. Bei der anderen wird das Schlüsselwort inline verwendet. Wenn Sie bei der Erstellung von Release-Versionen Ihrer Anwendung festgelegt haben, dass nach Größe optimiert werden soll, können einige oder alle Inline-Funktionen in gewöhnliche Funktionen umgewandelt werden.

Wenn Sie die Variablenfunktionen Get und Set für Ihre Klasse CPerson der Beispielanwendung implementieren wollen, folgen Sie diesen Schritten:

1. Bearbeiten Sie die Header-Datei Person.h und fügen Sie hier die Zeilen aus Listing 12.4 ein.

Listing 12.4: Erweiterungen zur Klasse CPerson

```
 1: class CPerson :
 2:     public CObject
 3: {
 4: public:
 5:     // Funktionen zum Setzen der Variablen
 6:     void SetEmployed(BOOL bEmployed)
 7:                          {  m_bEmployed = bEmployed;}
 8:     void SetMaritalStatus(int iStat)
 9:                          {  m_iMaritalStatus = iStat;}
10:     void SetAge(int iAge) {  m_iAge = iAge;}
11:     void SetName(CString sName) { m_strName = sName;}
12:     // Funktionen zum Holen der aktuellen Variableneinstellungen
13:     BOOL GetEmployed() {  return m_bEmployed;}
14:     int GetMaritalStatus() {  return m_iMaritalStatus;}
15:     int GetAge() {  return m_iAge;}
16:     CString GetName() {  return m_strName;}
17:     CPerson(void);
18:     ~CPerson(void);
19: private:
20:     // Ist die Person erwerbstätig?
21:     BOOL m_bEmployed;
22:     // Das Alter der Person
23:     int m_iAge;
24:     // Der Namen der Person
25:     CString m_strName;
26:     // Der Familienstand der Person
27:     int m_iMaritalStatus;
28: };
```

Mit diesen Methoden können Sie nun die Werte von Variablen in Ihrer benutzerdefinierten Klasse setzen und abrufen. Normalerweise initialisiert man die Variablen auch, wenn die Klasse erstmalig erzeugt wird. Der MFC Anwendungs-Assistent tut das für Sie, indem er allen Variablen in der Klasse einen Standardwert zuweist (siehe Listing 12.5).

Listing 12.5: Der Konstruktor der Klasse CPerson

```
1: CPerson::CPerson(void)
2: : m_bEmployed(false)
3: , m_iAge(0)
4: , m_strName(_T(""))
5: , m_iMaritalStatus(0)
6: {
7: }
```

Die Klasse serialisieren

Nachdem Sie Ihre benutzerdefinierte Klasse mit allen Variablen definiert und initialisiert haben, müssen Sie die Klasse serialisierbar machen. Das läuft in drei Schritten ab. Im ersten Schritt nehmen Sie die Funktion Serialize in die Klasse auf. Diese Funktion schreibt die Variablenwerte mithilfe von C++-Streams in das CArchive-Objekt und liest sie auf die gleiche Weise von dort wieder aus. Bei den beiden anderen Schritten fügen Sie die Makros DECLARE_SERIAL und IMPLEMENT_SERIAL hinzu. Mit diesen Elementen ist Ihre benutzerdefinierte Klasse serialisierbar und bereit zum Einsatz.

Um die Funktion Serialize in die benutzerdefinierte Klasse aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie über die Klassenansicht eine Member-Funktion hinzu.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen als Serialize fest. Fügen Sie einen Parameter vom Typ Carchive& namens ar ein und legen Sie den Zugriffsstatus als public fest.
3. Klicken Sie auf Fertig stellen. Die Funktion Serialize wird nun hinzugefügt und im Editor angezeigt, bereit zur Anpassung gemäß Listing 12.6.

Listing 12.6: Die Funktion CPerson.Serialize

```
 1: // Die Serialisierungsfunktion
 2: void CPerson::Serialize(CArchive& ar)
 3: {
 4:   // Funktion der Basisklasse aufrufen
 5:   CObject::Serialize(ar);
 6:
 7:   // Wird geschrieben?
 8:   if (ar.IsStoring())
 9:     // Alle Variablen in der richtigen Reihenfolge schreiben
10:     ar << m_strName << m_iAge << m_iMaritalStatus
11:                               << m_bEmployed;
12:   else
13:     // Alle Variablen in der richtigen Reihenfolge lesen
14:     ar >> m_strName >> m_iAge >> m_iMaritalStatus
15:                               >> m_bEmployed;
16: }
```

In Listing 12.6 wird zuerst die Funktion Serialize in der Basisklasse aufgerufen. Wenn Sie die Funktion der Basisklasse zuerst aufrufen, werden alle gespeicherten Basisinformationen wiederhergestellt. Damit steht die erforderliche Unterstützung für Ihre Klasse bereit, bevor die Variablen in Ihrer Klasse wiederhergestellt werden. Nach dem Aufruf der Funktion der

Basisklasse müssen Sie ermitteln, ob die Klassenvariablen zu lesen oder zu schreiben sind. Dazu können Sie die Methode IsStoring der Klasse CArchive aufrufen. Diese Funktion liefert TRUE zurück, wenn das Archiv zu schreiben ist, und FALSE, wenn es zu lesen ist. Wenn die Funktion IsStoring den Wert TRUE zurückgibt, können Sie C++-I/O-Streams einsetzen, um alle Klassenvariablen in das Archiv zu schreiben. Liefert die Funktion FALSE, lesen Sie mit C++-Streams aus dem Archiv. Insbesondere ist darauf zu achten, dass die Variablen sowohl beim Lesen als auch beim Schreiben in derselben Reihenfolge erscheinen.

Nachdem die Funktion Serialize an Ort und Stelle ist, müssen Sie noch die Makros in die benutzerdefinierte Klasse aufnehmen. Das erste Makro - DECLARE_SERIAL - gehört in die Header-Datei der Klasse und übernimmt den Klassennamen als einziges Argument.

Um das Makro DECLARE_SERIAL in die benutzerdefinierte Klasse CPerson der Beispielanwendung einzufügen, folgen Sie diesen Schritten:

1. Öffnen Sie die Header-Datei Person.h.

2. Schreiben Sie das Makro direkt nach dem Anfang der Klassendeklaration, wo es den Standardzugriff für die Klasse erhält.
3. Legen Sie den Klassennamen CPerson als einziges Argument an das Makro fest, wie es Listing 12.7 zeigt.

Listing 12.7: Die Deklaration der serialisierten CPerson-Klasse

```
 1: class CPerson :
 2:     public CObject
 3: {
 4:     DECLARE_SERIAL (CPerson)
 5: public:
 6:     void Serialize(CArchive& ar);
...
```

Die Standardzugriffsberechtigung für Funktionen und Variablen in C++-Klassen ist private. Alle Funktionen und Variablen, die vor der ersten Zugriffsdeklaration deklariert werden, sind standardmäßig private. Sie könnten einfach alle privaten Klassenfunktionen und -variablen in diesem Bereich der Klassendeklaration unterbringen, doch es ist eine bessere Praxis, die Zugriffsberechtigung für alle Funktionen und Variablen explizit zu deklarieren - so entstehen wenige bis gar keine Unsicherheiten über die Sichtbarkeit einer Klassenfunktion oder -variablen.

Die meisten C++-Funktionen benötigen ein Semikolon am Ende der Code-Zeile. Bei den beiden Serialisierungsmakros ist das nicht der Fall, da der C-Präprozessor sie durch den entsprechenden Code ersetzt, bevor er die Anwendung kompiliert. Es schadet nicht, die Semikola an dieser Stelle trotzdem zu verwenden, sie werden einfach ignoriert.

Um die Serialisierung der benutzerdefinierten Klasse fertig zu stellen, müssen Sie noch das Makro IMPLEMENT_SERIAL in die Klassendefinition aufnehmen. Der beste Platz für dieses Makro ist vor der Konstruktordefinition in der .cpp-Datei, die den Quellcode der Klasse enthält. Das Makro übernimmt drei Argumente: den Namen der benutzerdefinierten Klasse, den Namen der Basisklasse und die Versionsnummer. Wenn Sie irgendwelche Änderungen an der Funktion Serialize vornehmen, sollten Sie die

an das Makro IMPLEMENT_SERIAL übergebene Versionsnummer inkrementieren. Die Versionsnummer gibt an, wenn eine Datei mit einer vorherigen Version der Funktion Serialize geschrieben wurde und sich daher unter Umständen nicht durch die aktuelle Version der Anwendung lesen lässt.

Wenn Sie eine Datei mit einer älteren Version der Funktion Serialize in Ihrer Klasse geschrieben haben, erzeugt die Anwendung in der Praxis eine Ausnahme, die man mit den üblichen Verfahren der Ausnahmebehandlung in C++ abfangen kann. Damit können Sie in Ihrer Anwendung Code vorsehen, um ältere Dateiversionen zu erkennen und geeignet zu konvertieren.

Wenn Sie in mehreren Versionen Ihrer serialisierten Klasse lesen können wollen, damit Sie in der Lage sind, ältere Versionen zu lesen, verwenden Sie die Konstante VERSIONABLE_SCHEMA und verknüpfen Sie über OR mit der aktuellen Versionsnummer im IMPLEMENT_SERIAL-Makro:

```
IMPLEMENT_SERIAL(CMyObject, CObject, VERSIONABLE_SCHEMA | 1)
```

Wenn Sie sich in Ihrer Serialize-Funktion befinden, können Sie die CArchive-Methode GetObjectSchem

```
void CMyObject::Serialize(CArchive &ar)
{
  if (ar.IsLoading())
  {
    switch (ar.GetObjectSchema())
    {
      case 1:
        // In Version 1 lesen
        break;
      case 2:
        // In Version 2 lesen
        break;
    }
  }
  else
  // Üblicher Code zum Schreiben
}
```

Um das Makro IMPLEMENT_SERIAL in die Beispielanwendung aufzunehmen, folgen Sie diesen Schritten:

1. Öffnen Sie die Datei Person.cpp und scrollen Sie zu ihrem Anfang.

2. Fügen Sie das Makro IMPLEMENT_SERIAL unmittelbar vor dem Klassenkonstruktor von CPerson ein.
3. Übergeben Sie CPerson (den Klassennamen) als erstes Argument, CObject (die Basisklasse) als zweites Argument und 1 als Versionsnummer, wie es Listing 12.8 zeigt.

Listing 12.8: Das Makro IMPLEMENT_SERIAL im Code von CPerson

```
 1: #include "StdAfx.h"
 2: #include "person.h"
 3:
 4: IMPLEMENT_SERIAL (CPerson, CObject, 1)
 5:
 6: CPerson::CPerson(void)
 7: : m_bEmployed(false)
 8: , m_iAge(0)
 9: , m_strName(_T(""))
10: , m_iMaritalStatus(0)
```

```
11: {
12: }
```

Unterstützung in der Dokumentklasse

Wenn Sie eine formularbasierte Anwendung erstellen, in der das Formular im Fenster der vorrangige Platz für die Benutzerinteraktion mit der Anwendung ist, gibt es eine stillschweigende Übereinkunft, dass Ihre Anwendung dem Benutzer die Arbeit mit einer Anzahl von Datensätzen gestattet. Sie müssen also eine Unterstützung vorsehen, um die Datensätze zwischenzuspeichern und die Navigation durch den Recordset zu ermöglichen. Am einfachsten lässt sich das realisieren, indem man ein Objektarray als Variable in die Dokumentklasse aufnimmt, wie Sie es am Tag 10 gesehen haben. Damit kann man bei Bedarf zusätzliche Datensatzobjekte hinzufügen. Die Navigation implementiert man als Funktionen, die das erste, letzte, nächste oder vorherige Datensatzobjekt abrufen. Schließlich müssen Sie noch die Funktionalität realisieren, um bestimmen zu können, welchen Datensatz im Recordset der Benutzer gerade bearbeitet.

Um diese Funktionalität zu unterstützen, braucht die Dokumentklasse zunächst einmal zwei Variablen für das Objektarray und die aktuelle Datensatznummer im Array. Diese beiden Variablen bieten die erforderliche Unterstützung, um den Recordset zu speichern und die Navigation zu ermöglichen.

In die Beispielanwendung fügen Sie der Klasse CSerializeDoc die folgenden beiden Variablen hinzu und legen Sie für beide den Zugriff als private fest.

Name	Typ
m_iCurPosition	int
m_oaPeople	CObArray

Weiterhin ist in der Dokumentklasse noch zu gewährleisten, dass das Dokument über das zu speichernde Datensatzobjekt informiert ist. Dazu binden Sie in die Quelldatei der Dokumentklasse die Header-Datei der benutzerdefinierten Klasse ein und zwar vor der Header-Datei für die Dokumentklasse. Da die Dokumentklasse Aktionen in der Ansichtsklasse auslösen muss, empfiehlt es sich, auch die Header-Datei für die Ansichtsklasse in die Dokumentklasse aufzunehmen.

Um diese Header-Dateien in die Beispielanwendung einzubinden, folgen Sie diesen Schritten:

1. Öffnen Sie die Quellcode-Datei für die Dokumentklasse, SerializeDoc.cpp, und fügen Sie die #include-Anweisung gemäß Listing 12.9 ein.

Listing 12.9: Die Ansichtsklasse in die Implementierung der Dokumentklasse einbinden

```
 1: // SerializeDoc.cpp : Implementierung der Klasse CSerializeDoc
 2: //
 3:
 4: #include "stdafx.h"
 5: #include "Serialize.h"
 6:
 7: #include "SerializeDoc.h"
 8: #include "SerializeView.h"
 9:
10: #ifdef _DEBUG
11: #define new DEBUG_NEW
12: #endif
13:
14: // CSerializeDoc
```

2. Öffnen Sie die Header-Datei der Dokumentklasse, SerializeDoc.h, und fügen Sie die #include-Anweisung gemäß Listing 12.10 ein.

Listing 12.10: Die benutzerdefinierte Klasse in den Header der Dokumentklasse einbinden

```
1: // SerializeDoc.h : Schnittstelle der Klasse CSerializeDoc
2: //
3:
4: #pragma once
5: #include "Person.h"
6:
7: class CSerializeDoc : public CDocument
```

Neue Datensätze hinzufügen

Bevor Sie durch den Recordset navigieren können, müssen Sie neue Datensätze in das Objektarray aufnehmen können. Wenn Sie eine private Funktion für das Hinzufügen neuer Datensätze vorsehen, können Sie neue Datensätze dynamisch in den Recordset je nach Bedarf einfügen. Da neue Datensätze dem Benutzer leere Datenfelder präsentieren sollten, brauchen Sie keine Datensatzvariablen zu setzen, wenn Sie einen neuen Datensatz in das Objektarray aufnehmen. Damit können Sie auf den Standardkonstruktor zurückgreifen.

Nach der gleichen Logik, mit der Sie neue Liniendatensätze am Tag 10 hinzugefügt haben, nehmen Sie in der heutigen Beispielanwendung einen neuen Personendatensatz in das Objektarray der Dokumentklasse auf. Nachdem Sie einen neuen Datensatz hinzugefügt haben, können Sie einen Zeiger darauf zurückgeben, sodass die Ansichtsklasse direkt die Variablen im Datensatzobjekt aktualisieren kann.

Sobald der neue Datensatz hinzugefügt ist, setzen Sie den aktuellen Datensatzzeiger auf den neuen Datensatz im Array. Auf diese Weise lässt sich die aktuelle Datensatznummer leicht anhand des Positionszählers bestimmen.

Falls irgendwelche Probleme beim Erstellen des neuen Personen-Objekts auftreten, weisen Sie den Benutzer darauf hin, dass der Anwendung nicht mehr ausreichend Speicher zur Verfügung steht, und löschen das reservierte Objekt genau wie am Tag 10.

Um diese Funktionalität in der Beispielanwendung zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie der Dokumentklasse eine neue Member-Funktion hinzu.

2. Legen Sie den Typ als Zeiger auf Ihre benutzerdefinierte Klasse, CPerson*, fest. Geben Sie der Funktion den Namen AddNewRecord. Diese Funktion benötigt keine Argumente. Legen Sie den Zugriff auf die Funktion mit private fest, da sie nur von anderen Funktionen innerhalb der Dokumentklasse aufgerufen wird.
3. In die Funktion übernehmen Sie den Code aus Listing 12.11.

Listing 12.11: Die Funktion CSerializeDoc.AddNewRecord

```
 1: CPerson* CSerializeDoc::AddNewRecord(void)
 2: {
 3:   // Ein neues CPerson-Objekt erzeugen
 4:   CPerson *pPerson = NULL;
 5:   try
 6:   {
 7:     pPerson = new CPerson();
 8:     // Neue Person in Objektarray einfügen
 9:     m_oaPeople.Add(pPerson);
10:     // Dokument als bearbeitet kennzeichnen
11:     SetModifiedFlag();
12:     // Neue Positionsmarke setzen
13:     m_iCurPosition = (m_oaPeople.GetSize() - 1);
14:   }
15:   // Speicherausnahme aufgetreten?
16:   catch (CMemoryException* perr)
17:   {
18:     // Benutzer über schlechte Neuigkeiten informieren
19:     AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK);
20:     // Wurde Person-Objekt erzeugt?
```

```
21:     if (pPerson)
22:     {
23:       // Objekt löschen
24:       delete pPerson;
25:       pPerson = NULL;
26:     }
27:     // Ausnahmeobjekt löschen
28:     perr->Delete();
29:   }
30:   return pPerson;
31: }
```

Die aktuelle Position ermitteln

Um dem Benutzer die Navigation durch den Recordset zu erleichtern, ist es immer hilfreich, einen Anhaltspunkt zu liefern, wo sich der Benutzer im Recordset befindet. Um diese Informationen bereitzustellen, müssen Sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze aus dem Dokument ermitteln und sie dem Benutzer anzeigen.

Die betreffenden Funktionen sind recht einfach. Für die Gesamtzahl der Datensätze im Objektarray brauchen Sie lediglich die Größe des Arrays zu ermitteln und diesen Wert an den Aufrufer zurückgeben.

Folgen Sie für die Beispielanwendung folgenden Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp als int, den Funktionsnamen mit GetTotalRecords und den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 12.12.

Listing 12.12: Die Funktion CSerializeDoc.GetTotalRecords

```
1: int CSerializeDoc::GetTotalRecords(void)
2: {
3:   // Anzahl der Datensätze im Array zurückgeben
4:   return m_oaPeople.GetSize();
5: }
```

Die aktuelle Datensatznummer lässt sich genauso einfach ermitteln. Wenn Sie einen Positionszähler in der Dokumentklasse verwalten, enthält diese Variable die Nummer des Datensatzes, den der Benutzer gerade bearbeitet. Letztendlich brauchen Sie nur den Wert dieser Variablen an den Aufrufer zurückzugeben. Da das Objektarray mit Position 0 beginnt, addieren Sie zunächst eine 1 zur aktuellen Position, bevor Sie den Wert zurückgeben und dem Benutzer anzeigen.

Um Ihrer Beispielanwendung diese Funktion hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine weitere Member-Funktion in die Dokumentklasse.

2. Legen Sie den Typ mit int, den Funktionsnamen mit GetCurRecordNbr und den Zugriff als public fest.
3. Übernehmen Sie den Code aus Listing 12.13 in die Funktion.

Listing 12.13: Die Funktion CSerializeDoc.GetCurRecordNbr

```
1: int CSerializeDoc::GetCurRecordNbr(void)
2: {
3:   // Aktuelle Position zurückgeben
4:   return (m_iCurPosition + 1);
5: }
```

Durch den Recordset navigieren

Damit die Anwendung auch einen echten Nutzen bringt, müssen Sie dem Benutzer eine Möglichkeit bieten, durch den Recordset zu navigieren. Für die Minimalausstattung erstellen Sie einen Satz von Funktionen in der Dokumentklasse, um die Zeiger auf bestimmte Datensätze im Recordset zu ermitteln. Die erste Funktion holt einen Zeiger auf den aktuellen Datensatz. Zwei weitere Funktionen liefern die Zeiger auf den ersten bzw. letzten Datensatz im Recordset. Schließlich sind noch Funktionen erforderlich, um den vorherigen und den nächsten Datensatz im Recordset anzusprechen. Wenn der Benutzer bereits den letzten Datensatz im Recordset bearbeitet und versucht, zum nächsten Datensatz weiterzuschalten, können Sie automatisch einen neuen Datensatz in den Recordset aufnehmen und dem Benutzer als neuen, leeren Datensatz anzeigen.

Als Erstes erstellen wir die Funktion, die den aktuellen Datensatz zurückgibt. Diese Funktion muss den Wert im Datensatzzeiger untersuchen, um sicherzustellen, dass der aktuelle Datensatz eine gültige Array-Position ist. Nachdem Sie geprüft haben, dass die aktuelle Position gültig ist, kann die Funktion einen Zeiger auf den aktuellen Datensatz im Array zurückgeben.

Um diese Funktion zu Ihrer Beispielanwendung hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp mit CPerson* (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration als GetCurRecord und den Zugriff als public fest.
3. Übernehmen Sie den Code aus Listing 12.14 in die Funktion.

Listing 12.14: 6Die Funktion CSerializeDoc.GetCurRecord

```
 1: CPerson* CSerializeDoc::GetCurRecord(void)
 2: {
 3:   // Ist Datensatznummer gültig?
 4:   if (m_iCurPosition >= 0)
 5:     // Ja, aktuellen Datensatz zurückgeben
 6:     return (CPerson*)m_oaPeople.GetAt(m_iCurPosition);
 7:   else
 8:     // Nein, NULL zurückgeben
 9:     return NULL;
10: }
```

Als nächste Aufgabe realisieren Sie die Funktion, die den ersten Datensatz im Array zurückgibt. In dieser Funktion müssen Sie prüfen, ob das Array überhaupt Datensätze enthält. Sind Datensätze im Array vorhanden, setzen Sie den aktuellen Datensatzzeiger auf 0 und liefern einen Zeiger auf den ersten Datensatz im Array zurück.

Um diese Funktion in Ihre Beispielanwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp als CPerson* (ein Zeiger auf die benutzerdefinierte Klasse), den Funktionsnamen als GetFirstRecord und den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 12.15.

Listing 12.15: Die Funktion CSerializeDoc.GetFirstRecord

```
 1: CPerson* CSerializeDoc::GetFirstRecord(void)
 2: {
 3:   // Enthält das Array Datensätze?
 4:   if (m_oaPeople.GetSize() > 0)
 5:   {
 6:     // Ja, zur Position 0 gehen
 7:     m_iCurPosition = 0;
 8:     // Datenzsatz bei Position 0 zurückgeben
 9:     return (CPerson*)m_oaPeople.GetAt(0);
10:   }
11:   else
12:     // Keine Datensätze, NULL zurückgeben
```

```
13:     return NULL;
14: }
```

Um zum nächsten Datensatz im Recordset zu navigieren, müssen Sie den aktuellen Datensatzzeiger inkrementieren und dann prüfen, ob das Ende des Arrays noch nicht überschritten ist. Wenn der Zeiger innerhalb der Array-Grenzen liegt, geben Sie einen Zeiger auf den aktuellen Datensatz im Array zurück. Ist das Ende des Arrays überschritten, fügen Sie einen neuen Datensatz an das Array an.

Um diese Funktion in Ihre Beispielanwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp als CPerson* (ein Zeiger auf die benutzerdefinierte Klasse), den Funktionsnamen als GetNextRecord, und den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 12.16.

Listing 12.16: Die Funktion CSerializeDoc.GetNextRecord

```
 1: CPerson*  CSerializeDoc::GetNextRecord(void)
 2: {
 3:   // Arraygrenze überschritten nach Inkrementieren
 4:   // des Positionszählers?
 5:   if (++m_iCurPosition < m_oaPeople.GetSize())
 6:     // Nein, Datensatz an der neuen aktuellen Position
 7:     // zurückgeben
 8:     return (CPerson*)m_oaPeople.GetAt(m_iCurPosition);
 9:   else
10:     // Ja, neuen Datensatz hinzufügen
11:     return AddNewRecord();
12: }
```

In der Funktion, mit der Sie zum vorherigen Datensatz im Array navigieren, sind verschiedene Prüfungen auszuführen. Erstens ist zu testen, ob das Array über Datensätze verfügt. Ist das der Fall, dekrementieren Sie den aktuellen Datensatzzeiger. Wird dieser Zeiger kleiner als Null, setzen Sie den aktuellen Datensatzzeiger gleich 0 und zeigen damit auf den ersten Datensatz im Array. Nach diesen Schritten können Sie einen Zeiger auf den aktuellen Datensatz im Array zurückgeben.

Um diese Funktion in Ihre Beispielanwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp als CPerson* (ein Zeiger auf die benutzerdefinierte Klasse), den Funktionsnamen mit GetPrevRecord und den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 12.17.

Listing 12.17: Die Funktion CSerializeDoc.GetPrevRecord

```
 1: CPerson*  CSerializeDoc::GetPrevRecord(void)
 2: {
 3:   // Enthält das Array Datensätze?
 4:   if (m_oaPeople.GetSize() > 0)
 5:   {
 6:     // Position nach Dekrementieren der aktuellen Position
 7:     // kleiner als 0?
 8:     if (--m_iCurPosition < 0)
 9:       // Ja, Datensatzzeiger auf 0 setzen
10:       m_iCurPosition = 0;
11:     // Datensatz an der neuen aktuellen Position zurückgeben
12:     return (CPerson*)m_oaPeople.GetAt(m_iCurPosition);
13:   }
14:   else
```

```
15:     // Keine Datensätze, NULL zurückgeben
16:     return NULL;
17: }
```

Für die Funktion, die zum letzten Datensatz im Array navigiert, müssen Sie ebenfalls prüfen, ob Datensätze im Array vorhanden sind. Ist das der Fall, können Sie die aktuelle Größe des Arrays ermitteln und den aktuellen Datensatzzeiger auf die um 1 verminderte Zahl der Datensätze im Array setzen. Dabei handelt es sich tatsächlich um den letzten Datensatz im Array, da die Zählung der Datensätze im Array bei 0 beginnt. Nachdem Sie den aktuellen Datensatzzeiger gesetzt haben, können Sie einen Zeiger auf den letzten Datensatz im Array zurückgeben.

Um diese Funktion in Ihre Beispielanwendung einzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie eine neue Member-Funktion in die Dokumentklasse auf.

2. Legen Sie den Funktionstyp als CPerson* (ein Zeiger auf die benutzerdefinierte Klasse), den Funktionsnamen mit GetLastRecord und den Zugriff als public fest.
3. In die Funktion schreiben Sie den Code aus Listing 12.18.

Listing 12.18: Die Funktion CSerializeDoc.GetLastRecord

```
 1: CPerson*  CSerializeDoc::GetLastRecord(void)
 2: {
 3:   // Enthält das Array Datensätze?
 4:   if (m_oaPeople.GetSize() > 0)
 5:   {
 6:     // Zur letzten Position im Array gehen
 7:     m_iCurPosition = m_oaPeople.GetUpperBound();
 8:     // Datensatz auf dieser Position zurückgeben
 9:     return (CPerson*)m_oaPeople.GetAt(m_iCurPosition);
10:   }
11:   else
12:     // Keine Datensätze, NULL zurückgeben
13:     return NULL;
14: }
```

Den Recordset serialisieren

Wenn Sie die Funktion Serialize in die Dokumentklasse aufnehmen, brauchen Sie nichts weiter zu tun, als das CArchive-Objekt an die Funktion Serialize des Objektarrays zu übergeben, genauso wie Sie es von Tag 10 her kennen.

Beim Lesen der Daten aus dem Archiv fragt das Objektarray das CArchive-Objekt ab, um zu bestimmen, welcher Objekttyp und wie viele Objekte zu erzeugen sind. Das Objektarray erzeugt dann jedes Objekt im Array, ruft dessen Serialize-Funktion auf und übergibt dabei das CArchive-Objekt. Damit können die Objekte im Objektarray ihre eigenen Variablenwerte aus dem CArchive-Objekt in der gleichen Reihenfolge lesen, wie sie in die Datei geschrieben wurden.

Beim Schreiben von Daten in das Dateiarchiv ruft das Objektarray alle Serialize- Funktionen der Reihe nach auf und übergibt das CArchive-Objekt (genau wie beim Lesen aus dem Archiv). Damit kann das Objekt im Array die eigenen Variablen in der erforderlichen Form und Reihenfolge in das Archiv schreiben.

Für die Beispielanwendung bearbeiten Sie die Funktion Serialize der Dokumentklasse, um das Objekt CArchive an die Funktion Serialize des Objektarrays zu übergeben, wie es Listing 12.19 zeigt.

Listing 12.19: Die Funktion CSerializeDoc.Serialize

```
1: void CSerializeDoc::Serialize(CArchive& ar)
2: {
3:   // Serialisierung an das Objektarray übergeben
4:   m_oaPeople.Serialize(ar);
```

```
5: }
```

Aufräumarbeiten

Es ist noch der Code zu realisieren, um das Dokument »aufzuräumen«, wenn es geschlossen oder ein neues Dokument geöffnet wird. Dazu sind alle Objekte im Objektarray zu durchlaufen und zu löschen. Anschließend kann man das Objektarray über die Funktion RemoveAll zurücksetzen.

Um diese Funktionalität in der Beispielanwendung zu realisieren, folgen Sie diesen Schritten:

1. Markieren Sie die Dokumentklasse, CSerializeDoc, in der Klassenansicht des Arbeitsbereichs und wählen Sie im Eigenschaftenfenster den Modus Überschreibungen.

2. Wählen Sie die Basisfunktion DeleteContents zum Überschreiben aus.
3. In diese Funktion übernehmen Sie den Code aus Listing 12.20.

Listing 12.20: Die Funktion CSerializeDoc.DeleteContents

```
 1: void CSerializeDoc::DeleteContents()
 2: {
 3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein
 4:   // und/oder rufen Sie die Basisklasse auf.
 5:   // Anzahl der Datensätze im Objektarray ermitteln
 6:   int iCount = m_oaPeople.GetSize();
 7:   int iPos;
 8:
 9:   // Schleife durch das Array, dabei jedes Objekt löschen
10:   for (iPos = 0; iPos < iCount; iPos++)
11:     delete (CPerson*)m_oaPeople.GetAt(iPos);
12:   // Array zurücksetzen
13:   m_oaPeople.RemoveAll();
14:   CDocument::DeleteContents();
15: }
```

Ein neues Dokument öffnen

Wenn der Benutzer ein neues Dokument anlegt, soll ein leeres Formular erscheinen, in das er neue Informationen eingeben kann. Um diesen leeren Datensatz für die Aufnahme neuer Informationen vorzubereiten, nimmt man einen Datensatz in das Objektarray auf, der im Übrigen leer ist. Damit steht im Objektarray ein einziger Datensatz. Im Anschluss muss man die Ansicht modifizieren, um den neuen Datensatz auch tatsächlich anzuzeigen. Andernfalls zeigt die Ansicht den letzten bearbeiteten Datensatz aus dem vorherigen Recordset an (und der Benutzer wundert sich, warum die Anwendung keinen neuen Datensatz angelegt hat).

Um diese Funktionalität zu implementieren, ist die Funktion OnNewDocument in der Dokumentklasse zu bearbeiten. Diese Funktion ist bereits in der Dokumentklasse vorhanden, sodass man sie nicht extra über den Klassen-Assistenten hinzufügen muss. In der Funktion fügt man zunächst einen neuen Datensatz in das Objektarray ein. Anschließend holt man einen Zeiger auf das Ansichtsobjekt. Die Position desselben lässt sich mit der Funktion GetFirstViewPosition ermitteln. Mit der für das Ansichtsobjekt zurückgegebenen Position ruft man die Funktion GetNextView auf, um einen Zeiger auf dieses Objekt zu erhalten. Mit dem gültigen Zeiger kann man eine Funktion - die Sie in der Ansichtsklasse erstellen - aufrufen, um die Ansicht anzuweisen, die aktuellen Datensatzinformationen auf dem Formular anzuzeigen.

Wenn Sie diesen Code schreiben, sollten Sie beachten, dass der Typ des Zeigers auf die Ansicht in einen Zeiger der Klasse des Ansichtsobjekts umzuwandeln ist. Die Funktion GetNextView liefert einen Zeiger vom Typ CView, sodass keine Aufrufe der zusätzlichen

Funktionen in der Ansichtsklasse möglich sind, bis Sie den Zeiger in Ihre Ansichtsklasse umwandeln. Die Typumwandlung des Zeigers teilt dem Compiler mit, dass der Zeiger eigentlich ein Zeiger auf die Objekte Ihrer Ansichtsklasse ist und demzufolge alle Funktionen enthält, die Sie hinzugefügt haben. Wenn Sie den Zeiger nicht umwandeln, nimmt der Compiler an, dass das Ansichtsobjekt keine der von Ihnen hinzugefügten Funktionen enthält und lässt das Kompilieren der Anwendung nicht zu.

Um die oben besprochene Funktionalität einzufügen, folgen Sie diesen Schritten:

1. Suchen Sie die Funktion OnNewDocument im Quellcode der Dokumentklasse auf und übernehmen Sie in diese Funktion den Code aus Listing 12.21.

Bevor Sie Ihre Anwendung kompilieren können, müssen Sie die Funktion NewDataSet in die Ansichtsklasse aufnehmen. Wenn Sie versuchen, die Anwendung zu kompilieren, nachdem Sie den Code in Listing 12.21 hinzugefügt haben, erhalten Sie eine Fehlermeldung, die besagt, dass die Funktion NewDataSet unbekannt ist. Diese Funktion werden Sie der Ansichtsklasse in Kürze hinzufügen.

Listing 12.21: Die Funktion CSerializeDoc.OnNewDocument

```
 1: BOOL CSerializeDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Lässt sich kein neuer Datensatz einfügen, FALSE zurückgeben
 9:   if (!AddNewRecord())
10:     return FALSE;
11:
12:   // Einen Zeiger auf die Ansicht holen
13:   POSITION pos = GetFirstViewPosition();
14:   CSerializeView* pView = (CSerializeView*)GetNextView(pos);
15:   // Der Ansicht mitteilen, dass neue Datenmenge
16:   // eingetroffen ist
10:   if (pView)
10:     pView->NewDataSet();
10:
20:   return TRUE;
21: }
```

Wenn Sie einen vorhandenen Recordset öffnen, brauchen Sie keinen neuen Datensatz hinzuzufügen. Dennoch müssen Sie das Ansichtsobjekt anweisen, den anzuzeigenden Datensatz zu aktualisieren. Praktisch können Sie den gleichen Code in die Funktion OnOpenDocument schreiben wie in die Funktion OnNewDocument, nur dass Sie den ersten Teil weglassen, der einen neuen Datensatz in das Objektarray einfügt.

1. Markieren Sie die Dokumentklasse, CSerializeDoc, in der Klassenansicht des Arbeitsbereichs und wählen Sie im Eigenschaftenfenster den Modus Überschreibungen.

2. Nehmen Sie eine Behandlungsroutine für das Ereignis OnOpenDocument auf.
3. Schreiben Sie in diese Funktion den Code gemäß Listing 12.22.

Listing 12.22: Die Funktion CSerializeDoc.OnOpenDocument

```
 1: BOOL CSerializeDoc::OnOpenDocument(LPCTSTR lpszPathName)
 2: {
 3:   if (!CDocument::OnOpenDocument(lpszPathName))
 4:     return FALSE;
 5:
 6:   // TODO: Fügen Sie Ihren spezialisierten Erstellcode hier ein.
 7:   // Einen Zeiger auf die Ansicht holen
 8:   POSITION pos = GetFirstViewPosition();
 9:   CSerializeView* pView = (CSerializeView*)GetNextView(pos);
10:   // Der Ansicht mitteilen, dass neue Datenmenge
11:   // eingetroffen ist
12:   if (pView)
13:     pView->NewDataSet();
14:
15:   return TRUE;
16: }
```

Navigation und Bearbeitung in der Ansichtsklasse unterstützen

In die Dokumentklasse haben Sie die Unterstützung für den Recordset aufgenommen. Diese Funktionalität ist nun noch in der Ansichtsklasse zu realisieren, um durch die Datensätze zu navigieren, diese anzuzeigen und zu aktualisieren. Beim ersten Entwurf Ihrer Ansichtsklasse haben Sie eine Reihe von Steuerelementen im Fenster untergebracht, die für die Anzeige und Bearbeitung der verschiedenen Datenelemente der Datensätze vorgesehen sind. Weiterhin haben Sie Steuerelemente für die Navigation durch den Recordset aufgenommen. Diesen Steuerelementen müssen Sie nun »Leben einhauchen«, damit sie die Navigation durch die Datensätze und die Aktualisierung eines Datensatzes mit den vom Benutzer vorgenommenen Änderungen durchführen.

Durch den Umfang der direkten Interaktion, die das Formular mit dem Datensatzobjekt hat (Lesen der Variablenwerte aus dem Datensatz und Schreiben der neuen Werte in den Datensatz), ist es sinnvoll, einen Datensatzzeiger auf die Ansichtsklasse als private Variable hinzuzufügen. Folgen Sie für die Beispielanwendung diesen Schritten:

1. Nehmen Sie eine neue Member-Variable in die Klasse CSerializeView auf.

2. Legen Sie den Typ mit CPerson*, den Namen mit m_pCurPerson und den Zugriff mit private fest.

Den aktuellen Datensatz anzeigen

Fügen Sie der Ansichtsklasse zuerst die Funktionalität hinzu, um den aktuellen Datensatz anzuzeigen. Da verschiedene Stellen in der Ansichtsklasse auf diese Funktionalität zurückgreifen, erstellen Sie am besten eine eigene Funktion dafür. In dieser Funktion holen Sie die aktuellen Werte aller Variablen im Datensatzobjekt und schreiben sie in die Variablen der Ansichtsklasse, die mit den Steuerelementen im Fenster verbunden sind. Weiterhin ermitteln Sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze im Recordset und zeigen diese Werte an, damit der Benutzer die relative Position innerhalb des Recordsets kennt.

Folgen Sie für Ihre Beispielanwendung diesen Schritten:

1. Fügen Sie der Ansichtsklasse eine neue Member-Funktion hinzu.

2. Legen Sie den Funktionstyp mit void, den Funktionsnamen mit PopulateView und den Zugriff als private fest.
3. Nehmen Sie den Code aus Listing 12.23 in die Funktion auf.

Listing 12.23: Die Funktion CSerializeView.PopulateView

```
 1: void CSerializeView::PopulateView(void)
 2: {
 3:   // Einen Zeiger auf das aktuelle Dokument holen
 4:   CSerializeDoc* pDoc = GetDocument();
```

```
 5:   if (pDoc)
 6:   {
 7:     // Aktuelle Datensatzposition in der Menge anzeigen
 8:     m_sPosition.Format("Datensatz %d von %d",
 9:        pDoc->GetCurRecordNbr(), pDoc->GetTotalRecords());
10:   }
11:   // Ist Datensatzobjekt gültig?
12:   if (m_pCurPerson)
13:   {
14:     // Ja, alle Werte des Datensatzes holen
15:     m_bEmployed = m_pCurPerson->GetEmployed();
16:     m_iAge = m_pCurPerson->GetAge();
17:     m_sName = m_pCurPerson->GetName();
18:     m_iMaritalStatus = m_pCurPerson->GetMaritalStatus();
19:   }
20:   // Anzeige aktualisieren
21:   UpdateData(FALSE);
22: }
```

Die Funktion in Listing 12.23 ermittelt einen gültigen Zeiger auf das Dokumentobjekt. Dann gibt sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze im Recordset - die sie mit den bereits in die Dokumentklasse hinzugefügten Funktionen GetCurRecordNbr und GetTotalRecords ermittelt - formatiert aus. Mit einem gültigen Zeiger auf ein Datensatzobjekt setzt sie als Nächstes alle Variablen auf die Werte ihrer korrespondierenden Felder im Datensatzobjekt. Anschließend wird das Fenster mit den Variablenwerten aktualisiert.

Durch den Recordset navigieren

Wenn Sie beim Entwurf des Formulars Schaltflächen zur Navigation hinzugefügt haben, lässt sich die Navigation in einfacher Weise realisieren, indem Sie Behandlungsroutinen für alle Navigations-Schaltflächen hinzufügen und die korrespondierende Navigationsfunktion im Dokument aufrufen. Nachdem das Dokument zum entsprechenden Datensatz im Recordset gegangen ist, müssen Sie die Funktion in Listing 12.23 aufrufen, um den aktuellen Datensatz anzuzeigen. Wenn die Navigationsfunktionen des Dokuments Zeiger auf das neue aktuelle Datensatzobjekt zurückgeben, sollten Sie diese Zeiger zwischenspeichern, bevor Sie die Funktion zur Anzeige des aktuellen Datensatzes aufrufen.

Um diese Funktionalität zu Ihrer Beispielanwendung hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie mit dem Eigenschaftenfenster eine Behandlungsfunktion für das Klickereignis der Schaltfläche Erster hinzu.

2. Nehmen Sie den Code aus Listing 12.24 in die Funktion auf.

Listing 12.24: Die Funktion CSerializeView.OnBnClickedBfirst

```
 1: void CSerializeView::OnBnClickedBfirst()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für
 4:   // die Benachrichtigung ein.
 5:   // Einen Zeiger auf das aktuelle Dokument holen
 6:   CSerializeDoc* pDoc = GetDocument();
 7:   if (pDoc)
 8:   {
 9:     // Den ersten Datensatz aus dem Dokument holen
10:     m_pCurPerson = pDoc->GetFirstRecord();
11:     if (m_pCurPerson)
```

```
12:     {
13:       // Aktuellen Datensatz anzeigen
14:       PopulateView();
15:     }
16:   }
17: }
```

Die Funktion holt einen gültigen Zeiger auf das Dokumentobjekt. Dann ruft sie die Funktion GetFirstRecord desselben auf und speichert den zurückgegebenen Objektzeiger in der Zeigervariablen CPerson der Ansicht ab. Wenn sie einen gültigen Zeiger erhält, ruft sie die Funktion PopulateView auf, um die Daten des Datensatzes anzuzeigen.

Für die Schaltfläche Letzter folgen Sie diesen Schritten:

1. Führen Sie die gleichen Schritte aus wie für die Schaltfläche Erster, rufen aber die Funktion GetLastRecord des Dokumentobjekts auf, wie es Listing 12.25 zeigt.

Listing 12.25: Die Funktion CSerializeView.OnBnClickedBlast

```
 1: void CSerializeView::OnBnClickedBlast()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für
 4:   // die Benachrichtigung ein.
 5:   // Zeiger auf das aktuelle Dokument holen
 6:   CSerializeDoc* pDoc = GetDocument();
 7:   if (pDoc)
 8:   {
 9:     // Den ersten Datensatz aus dem Dokument holen
10:     m_pCurPerson = pDoc->GetLastRecord();
11:     if (m_pCurPerson)
12:     {
13:       // Aktuellen Datensatz anzeigen
14:       PopulateView();
15:     }
16:   }
17: }
```

2. Die gleichen Schritte wiederholen Sie für die Schaltflächen Vorheriger und Nächster, wobei Sie die Funktionen GetPrevRecord bzw. GetNextRecord des Dokumentobjekts aufrufen.

Diese abschließenden Schritte realisieren in Ihrer Anwendung die gesamte Funktionalität, die man für die Navigation durch den Recordset benötigt. Da außerdem der Aufruf der Funktion GetNextRecord des Dokuments für den letzten Datensatz im Recordset automatisch einen neuen Datensatz in den Recordset hinzufügt, haben Sie auch die Möglichkeit, bei Bedarf neue Datensätze in den Recordset aufzunehmen.

Änderungen speichern

Wenn der Benutzer Änderungen an den Daten in den Steuerelementen auf dem Bildschirm vornimmt, müssen diese Änderungen irgendwie in den aktuellen Datensatz im Dokument gelangen. Wenn Sie im Ansichtsobjekt einen Zeiger auf das aktuelle Datensatzobjekt verwalten, können Sie die verschiedenen Funktionen zum Setzen von Werten des Datensatzobjekts aufrufen und dabei den neuen Wert übergeben, um den Wert im Datensatzobjekt zu setzen.

Um dies in der Beispielanwendung zu implementieren, folgen Sie diesen Schritten:

1. Fügen Sie mit dem Klassenassistent eine Behandlungsroutine für das Ereignis BN_CLICKED für das

Kontrollkästchen Erwerbstätig hinzu.

2. Nehmen Sie in die Funktion den Code aus Listing 12.26 auf.
3. Wiederholen Sie die gleichen Schritte für die anderen Steuerelemente, wobei Sie die jeweiligen Funktionen des Datensatzobjekts aufrufen.

Listing 12.26: Die Funktion CSerializeView.OnBnClickedCbemployed

```
 1: void CSerializeView::OnBnClickedCbemployed()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für
 4:   // die Benachrichtigung ein.
 5:   // Daten im Formular mit den Variablen synchronisieren
 6:   UpdateData(TRUE);
 7:   // Wenn gültiges Person-Objekt vorhanden,
 8:   // Daten an das Objekt übergeben
 9:   if (m_pCurPerson)
10:     m_pCurPerson->SetEmployed(m_bEmployed);
11: }
```

Die Funktion ruft zuerst UpdateData auf, um die Werte aus dem Formular in die Variablen der Ansicht zu kopieren. Sie prüft, ob sie einen gültigen Zeiger auf das aktuelle Datensatzobjekt erhält, und ruft dann die entsprechende Set-Funktion auf demselben auf (in diesem Fall die Funktion SetEmployed).

Für die Eingabefelder Name und Alter verwendet die Funktion eine Behandlungsroutine für das Ereignis EN_CHANGE und ruft die Funktionen SetName und SetAge auf. Für die Optionsfelder Familienstand verwendet sie eine Behandlungsroutine für das BN_CLICKED-Ereignis und ruft für alle vier Optionsfelder dieselbe Behandlungsfunktion auf. Diese Funktion ruft auch die Funktion SetMaritalStatus im Datensatzobjekt auf.

In vorherigen Versionen von Visual C++ konnten Sie die Assistenten verwenden, um mehrere Optionsfelder oder andere Steuerelemente auf die gleiche Behandlungsroutine zeigen zu lassen. Es scheint jedoch als hätte Microsoft diese Funktionalität in der neuen Version geändert. Daher müssen Sie jetzt, um diese Funktionalität zu implementieren, die Behandlungsroutine für das erste Optionsfeld hinzufügen und dann Kopien der Einträge in der Nachrichtenzuordnungstabelle erstellen, wobei Sie die IDs der Steuerelemente ändern (wie Sie es an Tag 9, »Bilder, Zeichnungen und Bitmaps«, getan haben).

Einen neuen Recordset anzeigen

Als letzte Funktionalität fügen Sie eine Funktion hinzu, um die Ansicht zurückzusetzen, wenn der Benutzer einen neuen Recordset anlegt oder öffnet, damit der alte nicht mehr angezeigt wird. Dazu rufen Sie die Behandlungsroutine für die Schaltfläche Erster auf und erzwingen, dass die Ansicht den ersten Datensatz im neuen Recordset anzeigt.

Um diese Funktionalität in der Beispielanwendung umzusetzen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Ansichtsklasse ein.

2. Legen Sie den Funktionstyp mit void, den Namen mit NewDataSet und den Zugriff als public fest (damit sich die Funktion aus der Dokumentklasse aufrufen lässt).
3. In der Funktion rufen Sie die Behandlungsroutine für die Schaltfläche Erster auf, wie es Listing 12.27 zeigt.

Listing 12.27: Die Funktion CSerializeView.NewDataSet

```
1: void CSerializeView::NewDataSet(void)
2: {
3:   // Ersten Datensatz in der Menge anzeigen
4:   OnBnClickedBfirst();
5: }
```

Abbildung 12.5: Die laufende Serialisierungsanwendung

Jetzt können Sie mit Ihrer Anwendung Gruppen von Datensätzen hinzufügen, bearbeiten, speichern und wiederherstellen. Wenn Sie die Anwendung kompilieren und ausführen, können Sie Datensätze für sich selbst, für alle Ihre Familienangehörigen, Ihre Freunde oder jeden, den Sie in diese Anwendung aufnehmen möchten, anlegen. Wenn Sie den erstellten Recordset speichern und erneut öffnen, sobald Sie das nächste Mal mit der Anwendung arbeiten, sollten die Datensätze in genau dem gleichen Zustand wiederhergestellt sein, wie Sie sie eingegeben haben (siehe Abbildung 12.5).

12.3 Zusammenfassung

Die heutige Lektion hat im Detail gezeigt, was Serialisierung ist und wie sie funktioniert. Sie haben gelernt, wie man eine Klasse serialisierbar macht und warum und wie man die beiden Makros einsetzt, die zur Serialisierung einer Klasse erforderlich sind. Weiterhin wurde dargestellt, wie man eine formularbasierte SDI-Anwendung entwirft und erstellt, wobei eine Gruppe von Datensätzen in einer linearen Datenbank für den Einsatz in der Anwendung verwaltet wurden. Sie haben gelernt, wie man die Serialisierung einsetzt, um die lineare Datenbank zu erstellen und zu verwalten und wie man die Funktionalität in der Dokument- und der Ansichtsklasse realisiert, um die Navigation und Bearbeitung für diese Recordsets bereitzustellen.

12.4 Workshop

Fragen und Antworten

Frage:
Wenn ich irgendwelche Änderungen an einem der Datensätze in meinem Recordset vornehme, nachdem ich den Recordset gespeichert und dann die Anwendung geschlossen habe oder einen anderen Recordset öffne, fordert die Anwendung keine Bestätigung, ob ich die Änderungen speichern möchte. Wie kann ich erreichen, dass die Anwendung eine entsprechende Bestätigung abruft, wenn Daten geändert wurden?

Antwort:
Den Schlüssel zu diesem Problem liefert ein Funktionsaufruf in der Funktion AddNewRecord im Dokumentobjekt. Nachdem Sie einen neuen Datensatz in das Objektarray aufgenommen haben, rufen Sie die Funktion SetModifiedFlag auf. Diese Funktion markiert das Dokument als geändert. Wenn Sie den Recordset speichern, wird die Markierung automatisch wieder zurückgesetzt (außer wenn die Anwendung den Recordset aus irgendwelchen Gründen nicht speichern kann). Beim Übernehmen der Änderungen müssen Sie das Dokument in den geänderten Zustand setzen, damit die Anwendung weiß, dass das Dokument ungesicherte Änderungen enthält.

Antwort:
Schreiben Sie zu diesem Zweck den entsprechenden Code in die Behandlungsroutinen für Ihre Datensteuerelemente. Nachdem Sie den neuen Wert im aktuellen Datensatz gespeichert haben, holen Sie einen Zeiger auf das Dokumentobjekt und rufen die Funktion SetModifiedFlag des Dokuments auf, wie es Listing 12.28 zeigt. Wenn Sie die gleichen Änderungen an allen Behandlungsroutinen für Datensteuerelemente vornehmen, fragt Ihre Anwendung nach, ob Sie die Änderungen, die Sie seit der letzten Sicherung des Recordsets vorgenommen haben, speichern möchten.

Listing 12.28: Die modifizierte Funktion CSerializeView.OnBnClickedCbemployed

```
 1: void CSerializeView::OnBnClickedCbemployed()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für
 4:   // die Benachrichtigung ein.
 5:   // Daten im Formular mit den Variablen synchronisieren
 6:   UpdateData(TRUE);
 7:   // Wenn gültiges Person-Objekt vorhanden,
 8:   // Daten an das Objekt übergeben
 9:   if (m_pCurPerson)
10:     m_pCurPerson->SetEmployed(m_bEmployed);
11:   // Zeiger auf das Dokument holen
12:   CSerializeDoc * pDoc = GetDocument();
13:   if (pDoc)
14:     // Veränderungsflag im Dokument setzen
15:     pDoc->SetModifiedFlag();
16: }
```

Frage:
Warum muss ich die Versionsnummer im Makro IMPLEMENT_SERIAL ändern, wenn ich die Funktion Serialize in der benutzerdefinierten Datensatzklasse überarbeite?

Antwort:
Ob Sie die Versionsnummer inkrementieren müssen, hängt von der Art der vorgenommenen Änderungen ab. Wenn Sie beispielsweise ein berechnetes Feld in die Datensatzklasse aufnehmen und den Code hinzufügen, um diese neue Variable aus den Werten zu berechnen, die Sie in die Variablen aus dem CArchive-Objekt eingelesen haben, brauchen Sie die Versionsnummer eigentlich nicht zu inkrementieren, weil sich die Variablen und die Reihenfolge der Variablen, die Sie in das Archiv schreiben oder daraus lesen, nicht geändert haben. Wenn Sie allerdings ein neues Feld in die Datensatzklasse aufnehmen und das neue Feld in den I/O-Stream übernehmen, der in das CArchive-Objekt geschrieben oder daraus gelesen wird, dann hat sich das Format der mit dem Archiv ausgetauschten Daten geändert, und Sie müssen die Versionsnummer inkrementieren.

Antwort:
Wenn Sie die Versionsnummer unverändert lassen und Dateien lesen, die mit vorherigen Versionen Ihrer Anwendung erzeugt wurden, dann erhalten Sie eine Meldung wie »Ungültiges Dateiformat« und die Datei wird nicht gelesen. Nachdem Sie die Versionsnummer erhöht haben und eine Datei lesen, die mit einer älteren Versionsnummer geschrieben wurde, erhalten Sie zwar die gleiche Meldung, haben aber die

Möglichkeit, die Ausnahme mit eigenem Code zu behandeln und das Archiv in eine Umwandlungsroutine umzuleiten, die die Datei in das neue Dateiformat konvertiert.

Quiz

1. Welche zwei Makros müssen Sie in eine Klasse einfügen, um sie serialisierbar zu machen?

2. Wie kann man bestimmen, ob das CArchive-Objekt aus der Archivdatei zu lesen oder in die Archivdatei zu schreiben ist?
3. Welche Argumente sind an das Makro IMPLEMENT_SERIAL zu übergeben?
4. Von welcher Klasse müssen Sie die Ansichtsklasse ableiten, damit Sie ein Formular für das Hauptfenster einer SDI- oder MDI-Anwendung mit dem Dialog-Designer erstellen können?
5. In welchen Dateityp schreibt CArchive per Vorgabe?

Übung

Nehmen Sie Optionsfelder in das Formular auf, um das Geschlecht der Person festzulegen, wie es Abbildung 12.6 zeigt. Binden Sie diese Änderungen in die Klasse CPerson ein, um das Feld dauerhaft zu speichern.

Abbildung 12.6: Die laufende Serialisierungsanwendung mit dem Geschlecht einer Person

Tag 13

Datenbanken per ADO bearbeiten

Eine große Zahl von Anwendungen greift auf Datenbanken zu. Angefangen beim persönlichen Organizer bis hin zu großen Personalsystemen in Unternehmen speichern und verwalten Datenbanken alle Datensätze, die eine Anwendung verwendet und manipuliert. Visual C++ bietet vier verschiedene Technologien für die Verwendung und den Zugriff auf Datenbanken in Ihren Anwendungen: Data Access Objects (DAO), Open Database Connectivity (ODBC), OLE DB und ActiveX Data Objects (ADO). Heute lernen Sie etwas über ADO. ADO ist für alle Programmier- und Scriptingwerkzeuge von Microsoft vorgesehen und bietet dem Visual C++-Programmierer neue Herausforderungen bei der Datenbankprogrammierung, wobei gleichzeitig die Funktionalität vertraut bleibt.

Heute lernen Sie, wie ...

- ADO arbeitet und wie es auf die Technologie OLE DB zurückgreift, um einen einfachen Datenbankzugriff zu bieten,
- man auf die ADO-Objekte zugreift und sie verwendet, um eine flexible Datenbankanwendung zu erstellen.
- sich mit speziellen ADO-Makros benutzerdefinierte Datensatzklassen für Datenbankanwendungen erstellen lassen.

13.1 Datenbankzugriff

Die meisten Anwendungen im geschäftlichen Bereich arbeiten mit Daten. Diese Anwendungen verwalten und manipulieren Datensätze, die in Datenbanken gespeichert sind. Wenn Sie eine geschäftliche Anwendung erstellen, kommen Sie um den Zugriff auf eine Datenbank nicht herum. Die Frage ist, welche Datenbank?

Auf dem Markt ist eine ganze Anzahl von Datenbanken verfügbar. Wenn Sie eine Einzelbenutzeranwendung erstellen müssen, die eigenständig auf einem einzelnen Computer läuft, können Sie die verschiedenen PC-basierten Datenbanken wie Access oder FoxPro von Microsoft oder Paradox von Borland ins Auge fassen. Bei Anwendungen, die auf große, gemeinsam genutzte Datenbanken zugreifen, kommt höchstwahrscheinlich nur eine SQL-basierte Datenbank wie SQL Server oder Oracle in Frage. Alle Datenbanken bieten grundsätzlich die gleiche Funktionalität, um Datensätze zu verwalten. Bei jeder Datenbank lassen sich je nach Bedarf einzelne oder mehrere Datensätze abrufen. Man kann auch beliebige Datensätze hinzufügen, aktualisieren oder löschen. Alle Datenbanken können die Anforderungen Ihrer Anwendungen erfüllen, sodass Sie in der Lage sein sollten, eine bestimmte Datenbank für die eine Anwendung einzusetzen und für die nächste Anwendung zu einer anderen Datenbank zu wechseln, je nachdem, welche konkreten Anforderungen in Ihrer Anwendung zu realisieren sind, welche Datenbank sich am besten für die Anwendung eignet oder welche Datenbank Ihr Auftraggeber bevorzugt.

In der Praxis gibt es zahlreiche Unterschiede der heutzutage verfügbaren Datenbanken zu beachten. Jede Datenbank hat spezielle Stärken und Schwächen, was die eine Datenbank für eine bestimmte Situation geeigneter erscheinen lässt als eine andere. Allerdings würde es im Rahmen dieses Buches zu weit führen, auf die konkreten Unterschiede zwischen den einzelnen Produkten näher einzugehen. In Bezug auf die heute zu behandelnden Datenbankfunktionen kann man davon ausgehen, dass alle Datenbanken funktionell gleichwertig und gegeneinander austauschbar sind.

Wenn man von einer Datenbank zu einer anderen wechselt, begegnet man dem Problem, dass jede Datenbank mit einer anderen Schnittstelle für den Zugriff arbeitet. Das bedeutet, dass man sich mit komplett neuen Programmierverfahren und Funktionen für jede einzusetzende Datenbank beschäftigen müsste. Diese Probleme lassen sich mit den eigens für diesen Zweck entwickelten Schnittstellen wie ODBC und ADO in den Griff bekommen.

13.2 Was ist ADO?

Microsoft hat vor einigen Jahren OLE DB als neue Technologie des Datenbankzugriffs auf den Markt gebracht. Diese Technologie zielte nicht nur darauf ab, Daten einfach mit Datenbanken auszutauschen, sondern sollte auch den Zugriff auf Daten ermöglichen, die an einer beliebigen Stelle gespeichert sind. Über OLE DB konnte man auf Mail- Nachrichten, Tabellenblätter, Dateien usw. zugreifen. Diese Technologie war eine der ersten, die zur Erforschung und Entwicklung des objektorientierten Dateisystems im Herzen von Windows 2000 und jetzt auch von Windows XP führte.

Wie man sich gut vorstellen kann, ist es bei der umfangreichen Funktionalität, die OLE DB für den Zugriff auf Daten in den unterschiedlichsten Quellen braucht, ziemlich kompliziert, mit dieser Technologie zu arbeiten. An dieser Stelle kommt ADO ins Spiel. Das Konzept von ADO stellt praktisch eine zusätzliche Schicht über OLE DB dar, die speziell für den Datenbankzugriff vorgesehen ist.

Eines der Ziele bei der Entwicklung von ADO bestand darin, ein Steuerelement zu schaffen, das den Datenzugriff und die Steuerung in Webseiten gestattet, wobei die Datensätze auf dem Client zwischengespeichert werden. Zum Teil begründet sich dieses Ziel damit, dass der Benutzer eines Webbrowsers auf eine ganze Gruppe von Datensätzen zugreifen kann, ohne die einzelnen Datensätze nacheinander herunterladen zu müssen, um durch die Datensätze navigieren und Änderungen daran vornehmen zu können. Aus diesen Gründen gehört das ADO-Steuerelement zum Lieferumfang des Webbrowsers Internet Explorer von Microsoft (ab der Version 4.0 aufwärts).

13.3 ADO und ADO.NET

Mit der Einführung der .NET-Umgebung hat Microsoft mit ADO.NET ein weiteres Konzept zur Behandlung und Programmierung von Datenbanken vorgestellt. Dieses Konzept stellt jedoch keinen Ersatz sondern eine Ergänzung des bestehenden ADO dar. Während ADO für die Programmierung von Frontend-Applikationen gedacht ist, die direkt mit einer Datenquelle arbeiten, fokussiert sich ADO.NET vorwiegend auf den Bereich der Komponenten-Entwicklung. Mit ADO.NET können Daten auch komplett ohne Verbindung zu einer Datenquelle verarbeitet werden, etwa um Arbeitsdaten innerhalb des Speichers zu verwalten.

Aufbauend auf der Erfahrung bei der Entwicklung und Anwendung von ADO wurde die Programmierung von ADO.NET optimiert. Anstelle eines gemeinsamen Datentyps Variant (mehr dazu im Abschnitt »Das Field-Objekt«) für alle Arten von Daten unterstützt ADO.NET nun native Datentypen was zu einem schnelleren und typensicheren Umgang mit einzelnen Daten führt. Sowohl intern als auch in der externen Kommunikation unterstützt ADO.NET die Verwendung von XML (eXtended Markup Language - erweiterte attributierte Sprache, also die Kombination von Daten und Datenschreibung), dem Standard für die Übertragung von Daten über Netzwerke. Mit ADO.NET können Sie z.B. direkt Daten einer XML-Datei verarbeiten. Diese Datei ist dann auf viele andere Applikationen und Betriebssysteme portierbar - im Gegensatz zu den gestern behandelten serialisierten Objekten in einer Datei. Der Themenkreis ADO.NET ist so umfangreich, dass es über dieses Thema spezielle Bücher gibt und geben wird.[1]

Eine Beschreibung der Unterschiede zwischen ADO und ADO.NET finden Sie beispielsweise in »ADO.NET for the ADO Programmer« (nur auf Englisch verfügbar) unter http://msdn.microsoft.com/library/en-us/dndotnet/html/ADONETProg.asp.

ADO-Objekte

Um den Einsatz von ADO in Scriptsprachen wie VBScript genauso einfach zu gestalten wie in Programmierumgebungen wie Visual Basic, hat Microsoft versucht, die Anzahl der Objekte auf einem Minimum zu halten. Im Ergebnis haben Sie es mit einer kleinen Anzahl von Basisobjekten zu tun:

- Connection
- Error
- Command

- Parameter
- Recordset
- Field

Daneben existieren noch Sammelobjekte für die Aufnahme von Kollektionen der Objekte Error, Parameter und Field.

Das Connection-Objekt

Das Connection-Objekt dient dazu, eine Verbindung zu einer Datenbank einzurichten und zu verwalten. Man konfiguriert das Objekt mit der Verbindungsinformation einschließlich dem Standort der Datenbank, der Benutzer-ID und dem Kennwort, bevor man die Verbindung öffnet. Wenn alle Angaben ordnungsgemäß konfiguriert sind, sollte das Verbindungsobjekt seine Open-Methode aufrufen, um die Verbindung zu öffnen. Wenn das Connection-Objekt seinen Gültigkeitsbereich verliert, wird die Verbindung nicht automatisch geschlossen. Sie müssen das Schließen der Datenbankverbindung selbst kontrollieren, indem Sie die Close-Methode des Connection-Objekts aufrufen.

Über das Connection-Objekt wird auch die gesamte Funktionalität der höheren Ebenen gesteuert. Dazu gehört die Kontrolle der Transaktionsverarbeitung mit den Methoden BeginTrans, CommitTrans und RollbackTrans des Connection-Objekts. Viele Eigenschaften des Connection-Objekts können bei der Arbeit mit Datenbanken hilfreich sein. In Tabelle 13.1 sind die nützlicheren aufgelistet.

In dieser und den folgenden Listen von Eigenschaften und Methoden wird ADO Version 2.7 verwendet. Wenn Sie mit einer älteren Version von ADO arbeiten, sind diese Methoden und Eigenschaften möglicherweise nicht verfügbar.

Sie greifen auf Eigenschaften von COM-Objekten (wie bei allen ADO-Objekten) nicht zu, wie Sie es in Visual Basic tun würden. Auf jede Eigenschaft kann mit zwei möglichen Syntaxen zugegriffen werden. Um den Wert der Eigenschaft zu lesen, fügen Sie vor dem Namen der Eigenschaft Get und nach dem Namen ein leeres Klammernpaar ein. Wenn Sie beispielsweise den Wert einer Eigenschaft namens Timeout lesen möchten, sieht das so aus:

```
myVar = obj.GetTimeout();
```

Um den Wert einer Eigenschaft zu setzen, fügen Sie ein Put vor dem Namen der Eigenschaft ein und übergeben ihr den neuen Wert als Argument:

```
obj.PutTimeout(myVal);
```

Der Grund dafür ist, dass alle Objekteigenschaften in C++ eigentlich Funktionen sind.

Eigenschaft	Beschreibung
CommandTimeout	Die Anzahl von Sekunden, die das ADO-Objekt auf eine Befehlsausführung wartet,. Wenn Ihre Anwendung SQL-Abfragen oder gespeicherte Prozeduren (in der Datenbank gespeicherte SQL-Funktionen) ausführen soll, könnte das etwas Zeit benötigen, weshalb Sie diesen Wert anpassen sollten. Standard sind 30 Sekunden. Wenn Sie diese Eigenschaft auf 0 setzen, wartet die ADO-Verbindung für immer darauf, dass SQL-Befehle und -Prozeduren fertig werden. Manchmal ist das eine gute Sache, manchmal aber auch nicht.

ConnectionString	Der String, der die Datenbank angibt, mit der verbunden werden soll. Er entspricht einem ODBC-Verbindungsstring. Das Format dieses Strings ist eine Reihe von durch Semikola getrennter argument=wert-Anweisungen, z.B. Provider=MSDASQL;Data Source=VCPP21DB.
ConnectionTimeout	Wie CommandTimeout gibt diese Eigenschaft die Anzahl der Sekunden an, die auf den Aufbau der Verbindung zur Datenbank gewartet wird. Wenn Sie mit einer Datenbank mit schlechten Antwortzeiten oder mit einer weit entfernten Datenbank arbeiten, müssen Sie diese Eigenschaft möglicherweise anpassen, damit die Verbindung mehr Zeit hat, sich aufzubauen. Der Standardwert dieser Eigenschaft ist 15 Sekunden. Wenn Sie die Eigenschaft auf 0 setzen, wartet Ihre Anwendung für immer auf den Aufbau der Verbindung.
CursorLocation	Gibt die Position des Cursors an (wird zur Navigation durch eine Datenmenge verwendet). Diese Eigenschaft hat zwei mögliche Einstellungen: adUseClient und adUseServer.
Mode	Gibt den Modus an, in dem Sie sich mit der Datenbank verbinden. In Tabelle 13.2 sind die möglichen Einstellungen dieser Eigenschaft aufgelistet. Diese Einstellungen wirken sich gewöhnlich nur auf den Modus aus, in dem Sie bestimmte Datensätze in Ihren Recordsets öffnen. Sie können diese Moduseinstellungen mit OR verknüpfen, um Ihren Modus und den Modus der gemeinsamen Nutzung festzulegen.
State	Zeigt den Status der Verbindung zur Datenbank an. In Tabelle 13.3 sind die möglichen Werte dieser Eigenschaft aufgelistet.
Version	Gibt die ADO-Versionsnummer an

Tabelle 13.1: Eigenschaften des Connection-Objekts

Modus	**Beschreibung**
adModeRead	Nur Lesen
adModeReadWrite	Lese- und Schreibberechtigung
adModeRecursive	Wird mit den »Share«-Modi verwendet, um Berechtigungen an alle untergeordneten Datensätze in der Datenbank weiterzugeben
adModeShareDenyNone	Erlaubt anderen, die Datensätze in beliebigen Modi zu öffnen
adModeShareDenyRead	Hindert andere daran, die gleichen Datensätze im Lesemodus zu öffnen
adModeShareDenyWrite	Hindert andere daran, die gleichen Datensätze im Schreibmodus zu öffnen
adModeShareExclusive	Hindert andere daran, die gleichen Datensätze zu öffnen, während Sie sie geöffnet haben
adModeUnknown (Standard)	Zeigt an, dass die Berechtigungen unbekannt sind und wahrscheinlich vom Datenbankadministrator gesetzt werden
adModeWrite	Nur Schreibberechtigung

Tabelle 13.2: Connection-Moduseinstellungen

Status	**Beschreibung**
adStateClosed	Die Datenbankverbindung ist geschlossen.
adStateOpen	Die Datenbankverbindung ist offen.
adStateConnecting	Die Datenbankverbindung wird hergestellt.
adStateExecuting	Die Datenbank führt derzeit einen Befehl aus.
adStateFetching	Es werden derzeit Datenzeilen aus der Datenbank abgerufen.

Tabelle 13.3: Connection-Statuswerte

Wichtige Connection-Methoden

Das Connection-Objekt besitzt mehrere nützliche Methoden, von denen die meisten keine Parameter benötigen.

Die Methode Open stellt die Verbindung zur Datenbank her. Sie übernimmt folgende Parameter:

```
Open(_bstr_t ConnectionString, _bstr_t UserID, _bstr_t Password,
     long Options);
```

Der erste Parameter der Methode Open, ConnectionString, gibt die zu kontaktierende Datenbank an. Dieser Parameter überschreibt die Eigenschaft ConnectionString. Der zweite Parameter, UserID, ist der für die Verbindung zur Datenbank zu verwendende Benutzername. Der dritte Parameter, Password, ist das Datenbankpasswort für den im zweiten Parameter angegebenen Benutzer. (Desktop-Datenbanken wie Access benötigen zwar oft keine Benutzer-IDs und Passwörter, um eine Verbindung zu öffnen, doch bei der Arbeit mit großen SQL-Datenbanken wie SQL Server oder Oracle werden Sie fast immer eine Benutzer-ID und ein Passwort angeben müssen.) Der letzte Parameter, Options¸ gibt an, ob die Datenverbindung synchron oder asynchron sein soll. (Mit anderen Worten, soll diese Funktion vor dem Aufbau der Verbindung zurückkehren oder warten, bis die Verbindung entweder hergestellt ist oder ein Timeout erreicht wurde?) Die möglichen Werte für diesen letzten Parameter sind adAsyncConnect (sofort zurückkehren) und adConnectUnspecified (die Standardeinstellung, wartet vor dem Zurückkehren auf den Aufbau der Verbindung).

Was hat es mit diesem neuen Datentyp, _bstr_t, auf sich? Er verkapselt den Stringtyp BSTR, den String-Datentyp. Meistens können Sie einen CString verwenden und in einen LPCTSTR-Datentyp umwandeln, wenn Sie einer COM-Methode oder -Eigenschaft einen String übergeben müssen.

Die zweite Methode, die Parameter übernimmt, ist Execute. Sie können die Methode Execute verwenden, um einen beliebigen SQL-Befehl auf der Datenbank auszuführen. Es kann sich dabei um eine SQL-Abfrage, einen DDL-Befehl (Data Definition Language, wird zur Erstellung von Tabellen und Indizes in der Datenbank verwendet) oder um eine gespeicherte Prozedur handeln. Die Methode hat folgende Syntax:

```
_RecordsetPtr Execute(_bstr_t CommandText, VARIANT* RecordsAffected,
                      long Options);
```

Der erste Parameter dieser Methode, CommandText, ist der auszuführende SQL-Befehl. Der zweite Parameter, RecordsAffected, ist ein Zeiger auf eine Variant-Variable, in der die Anzahl der von dem Befehl betroffenen Datensätze gespeichert ist. Der dritte Parameter, Options, ist ein Flag-Wert, der angibt, wie der Befehl durch die Datenbank interpretiert und ausgeführt werden soll. In Tabelle 13.4 sind die möglichen Werte aufgelistet. Wenn der Befehl einen Recordset erzeugt, gibt sie einen Zeiger auf diesen Recordset als Ergebnis zurück.

Option	Beschreibung
adCmdUnspecified	Der Befehlstyp ist unspezifisch.
adCmdText	Der Befehl wird als textuelle Definition eines Befehls oder einer gespeicherten Prozedur ausgewertet.
adCmdTable	Der Befehl ist ein Tabellenname, wobei die gesamte Tabelle zurückgegeben werden soll.
adCmdStoredProc	Der Befehl ist eine auszuführende gespeicherte Prozedur.

adCmdUnknown	Der Standardwert, der Befehlstyp, ist nicht bekannt.
adCmdFile	Der Befehl ist der Dateiname eines gespeicherten Recordsets.
adAsyncExecute	Der Befehl soll asynchron ausgeführt werden (die Methode wartet nicht, bis der Befehl ausgeführt ist, sondern kehrt sofort zurück).
adAsyncFetch	Die nach der ersten Menge (von der Eigenschaft CacheSize begrenzt) übrig bleibenden Zeilen werden asynchron abgerufen.
adExecuteNoRecords	Der Befehl liefert keine Datenzeilen zurück. Wenn er Daten zurückgibt, werden diese verworfen.
adExecuteRecord	Der Befehl gibt nur eine Datenzeile zurück.
adOptionUnspecified	Der Befehl ist unspezifisch.

Tabelle 13.4: Optionswerte von Execute

Die restlichen Methoden des Connection-Objekts übernehmen keine Parameter. Sie sind in Tabelle 13.5 aufgelistet.

Methode	Beschreibung
Close	Die Verbindung zur Datenbank schließen
Cancel	Bricht alle derzeit ausgeführten asynchronen Aktionen ab (Verbindung, Datenabruf oder Befehlsausführung)
BeginTrans	Startet eine Transaktion in der Datenbank
CommitTrans	Übergibt eine Transaktion und macht damit alle Änderungen dauerhaft
RollbackTrans	Macht eine Transaktion rückgängig und setzt alle geänderten Daten auf die Werte vor dem Beginn der Transaktion zurück

Tabelle 13.5: Methoden des Connection-Objekts

Das Error-Objekt

Bei jedem Datenbankfehler werden die Fehlerinformationen aus der Datenbank in ein ADO Error-Objekt geschrieben. Bei den Fehlerinformationen im Error-Objekt handelt es sich um die Fehlerinformationen aus der Datenbank und nicht um Fehlerinformationen von ADO. Wenn Sie einen Fehler erhalten und anhand der Fehlerinformationen die Ursachen ermitteln wollen, müssen Sie die Fehlercodes der Datenbank und nicht die von ADO prüfen. In Tabelle 13.6 sind die Eigenschaften des Error-Objekts aufgelistet.

Eigenschaft	Beschreibung
Description	Ein String, der die Beschreibung des Fehlers enthält
NativeError	Ein long-Wert mit einem für den Datenbank-Provider spezifischen Fehlercode. Dieser Wert muss mit den Fehlercodes des Datenbankvertreibers verglichen werden, um Möglichkeiten der Fehlerbehebung zu ermitteln.
Number	Ein long-Wert, der einen ADO-, OLE-DB- oder OLE-Fehler angibt
Source	Ein String, der den Namen des Objekts oder der Anwendung enthält, das oder die den Fehler ausgelöst hat
SQLState	Ein String aus fünf Zeichen, der den ANSI-SQL-Standard für Fehlercodes angibt. Dieser Code muss mit der Dokumentation der Datenbank verglichen werden, um den aufgetretenen Fehler zu ermitteln.

Tabelle 13.6: Eigenschaften des Error-Objekts

Das Command-Objekt

Über das Command-Objekt werden die Befehle in der Datenbank ausgeführt. Mit diesem Objekt lassen sich SQL-Anweisungen ausführen oder gespeicherte Prozeduren aufrufen. Bei jedem Befehl, der Datenzeilen zurückgibt, müssen Sie das Command-Objekt mit einem Recordset-Objekt verbinden, in dem sich die zurückgegebenen Daten ablegen lassen.

Beim Aufruf einer gespeicherten Prozedur sind oftmals - wie bei Funktionen in anderen Programmiersprachen - Parameter zu übergeben. Dazu weist man dem Command-Objekt eine Reihe von Parameter-Objekten zu. Jedes Parameter-Objekt verfügt über den Namen des Parameters, für den es den Wert aufnimmt, sowie den Wert, der an die Datenbank für den betreffenden Parameter zu übergeben ist.

Viele Eigenschaften des Command-Objekts können sich bei der Arbeit mit Datenbanken als nützlich erweisen. In Tabelle 13.7 sind einige der wichtigeren aufgelistet.

Eigenschaft	Beschreibung
ActiveConnection	Ein Zeiger auf das aktive Connection-Objekt, mit dem dieses Command-Objekt verbunden ist
CommandText	Ein String, der den durch die Datenbank auszuführenden Befehl enthält
CommandTimeout	Die Anzahl der Sekunden, die die ADO-Verbindung auf die Ausführung eines Befehls wartet. Wenn Ihre Anwendung SQL-Abfragen oder gespeicherte Prozeduren ausführen soll, die eine Weile dauern können, sollten Sie diese Eigenschaft anpassen. Standard sind 30 Sekunden. Wenn Sie diese Eigenschaft auf 0 setzen, wartet die ADO-Verbindung für immer darauf, dass SQL-Befehle und Prozeduren fertig werden. Manchmal ist das eine gute Sache, manchmal aber auch nicht.
CommandType	Der Typ des auszuführenden Befehls. In Tabelle 13.8 sind die möglichen Werte dieser Eigenschaft aufgelistet.
State	Der Status der Datenbankverbindung. In der schon früher gezeigten Tabelle 13.3 sind die möglichen Werte dieser Eigenschaft aufgelistet.

Tabelle 13.7: Eigenschaften des Command-Objekts

Option	Beschreibung
adCmdUnspecified	Der Befehlstyp ist unspezifisch.
adCmdText	Der Befehl wird als textuelle Definition eines Befehls oder einer gespeicherten Prozedur ausgewertet.
adCmdTable	Der Befehl ist ein Tabellenname, wobei die gesamte Tabelle zurückgegeben werden soll.
adCmdStoredProc	Der Befehl ist eine auszuführende gespeicherte Prozedur.
adCmdUnknown (Standard)	Der Befehlstyp ist unbekannt.
adCmdFile	Der Befehl ist der Dateiname eines gespeicherten Recordsets.

Tabelle 13.8: Command-Typwerte

Execute ist die wichtigste Methode des Command-Objekts. Diese Methode hat folgende Syntax:

```
_RecordsetPtr Execute(VARIANT* RecordsAffected, VARIANT* Parameters,
                      long Options);
```

Der erste Parameter dieser Funktion, RecordsAffected, ist eine VARIANT-Variable, in der die Anzahl der von dem Befehl betroffenen Datensätze gespeichert ist. Der zweite Parameter, Parameters, ist ein Array von Parameter-Objekten, die (wenn benötigt) als Parameter an den Befehl übergeben werden sollen. Der dritte Parameter, Options, gibt an, wie der Befehl ausgeführt werden soll. Die möglichen Werte dieses Parameters wurden bereits in Tabelle 13.4 aufgelistet.

Beachten Sie, dass es keinen Parameter für den Befehlstext gibt. Der tatsächlich auszuführende Befehl muss bereits vor dem Aufruf der Execute-Methode an die Eigenschaft CommandText übergeben worden sein.

Sie können die Methode CreateParameter verwenden, um einen Parameter für die Übergabe an die Execute-Methode zu erzeugen. Diese Methode hat folgende Syntax:

```
_ParameterPtr CreateParameter(_bstr_t Name, DataTypeEnum Type,
    ParameterDirectionEnum Direction, long Size, _variant_t &Value);
```

Der erste Parameter, Name, ist der Name des Parameters. Der zweite Parameter, Type, gibt den Datentyp des Parameters an. In Tabelle 13.9 sind die möglichen Werte für diesen Parameter aufgelistet. Der dritte Parameter, Direction, gibt den Parametertyp an. In Tabelle 13.10 sind die möglichen Werte für diesen Parameter aufgelistet. Der vierte Parameter, Size, gibt die maximale Länge für den Parameterwert an, entweder in Zeichen oder in Bytes (abhängig vom Datentyp). Der letzte Parameter, Value, ist der eigentliche Wert des Parameterobjekts.

Datentyp	**Beschreibung**
adBigInt	Ein 8 Byte großer Integer mit Vorzeichen
adBinary	Ein binärer Wert
adBoolean	Ein Boolescher Wert
adBSTR	Ein nullterminierter Zeichenstring
adChar	Ein Stringwert
adCurrency	Ein Währungswert (einige Datenbanken besitzen einen Währungs-Datentyp)
adDate	Ein Datumswert, gespeichert als double, der die Anzahl der Tage seit dem 30. Dezember 1899 angibt
adDBDate	Ein Datumswert im Format yyyymmdd
adDBTime	Ein Zeitwert im Format hhmmss
adDBTimeStamp	Ein Datum-/Zeit-Stempel im Format yyyymmddhhmmss zuzüglich eines Bruchs, der Milliardstelsekunden angibt
adDecimal	Ein exakter numerischer Wert mit fester Genauigkeit und Skalierung
adDouble	Ein Fließkommawert doppelter Genauigkeit
adEmpty	Kein Wert
adError	Ein 32 Bit großer Fehlercode
adFileTime	Ein 64 Bit großer Wert, der die Anzahl von 100-Nanosekunden-Intervallen seit dem 1. Januar 1601 angibt
adInteger	Ein 4 Byte großer Integer-Wert mit Vorzeichen
adLongVarBinary	Ein langer binärer Wert (nur Parameter-Objekt)
adLongVarChar	Ein langer Stringwert (nur Parameter-Objekt)
adLongVarWChar	Ein langer Unicode-Stringwert (nur Parameter-Objekt)
adNumeric	Ein exakter numerischer Wert mit fester Genauigkeit und Skalierung
adSingle	Ein Fließkommawert mit einfacher Genauigkeit
adSmallInt	Ein 2 Byte großer Integer-Wert mit Vorzeichen
adTinyInt	Ein 1 Byte großer Integer-Wert mit Vorzeichen
adUnsignedBigInt	Ein 8 Byte großer Integer-Wert ohne Vorzeichen

adUnsignedSmallInt	Ein 2 Byte großer Integer-Wert ohne Vorzeichen
adUnsignedTinyInt	Ein 1 Byte großer Integer-Wert ohne Vorzeichen
adUserDefined	Eine benutzerdefinierte Variable
adVarBinary	Ein Binärwert
adVarChar	Ein Stringwert
adVarNumeric	Ein numerischer Wert
adVarWChar	Ein Unicode-Stringwert
adWChar	Ein Unicode-Stringwert

Tabelle 13.9: Parameter-Datentypen

Typ	Beschreibung
adParamInput (Standard)	Der Parameter ist ein Eingabe-Parameter.
adParamInputOutput	Der Parameter ist sowohl ein Eingabe- als auch ein Ausgabe-Parameter.
adParamOutput	Der Parameter ist ein Ausgabe-Parameter.
adParamReturnValue	Der Parameter repräsentiert den Rückgabewert.
adParamUnknown	Der Parametertyp ist unbekannt.

Tabelle 13.10: Parameter-Typen

Die letzte Methode des Command-Objekts, Cancel, bricht alle derzeit ausgeführten asynchronen Aktionen ab (Verbindung, Datenabruf oder Befehlsausführung).

Das Parameter-Objekt

Mit dem Parameter-Objekt lassen sich Variablen für den Aufruf gespeicherter Prozeduren oder parametrisierter Abfragen übergeben. Diese werden an ein Command-Objekt gebunden und beim Aufruf des Befehls verwendet, der im Command-Objekt programmiert wurde. In Tabelle 13.11 sind die wichtigsten Eigenschaften des Parameter-Objekts aufgelistet.

Eigenschaft	Beschreibung
Direction	Gibt an, ob es sich bei dem Parameter um einen Eingabe-, Ausgabe- oder Rückgabewert-Parameter handelt. Die möglichen Werte dieser Eigenschaft wurden bereits in Tabelle 13.10 aufgelistet.
Name	Der Name des Parameter-Objekts
NumericScale	Ein Byte-Wert, der die Anzahl von Dezimalstellen angibt, nach denen numerische Werte aufgelöst werden
Precision	Ein Byte-Wert, der die Anzahl der für die Darstellung numerischer Werte verwendeten Stellen angibt
Size	Ein long-Wert, der die maximale Anzahl von Bytes oder Zeichen angibt, die für den Wert des Parameter-Objekts verwendet werden können
Type	Der Datentyp des Parameter-Objekts. Die möglichen Werte dieser Eigenschaft wurden bereits in Tabelle 13.9 aufgelistet.
Value	Der tatsächliche Wert des Parameter-Objekts

Tabelle 13.11: Die Eigenschaften des Parameter-Objekts

Das Recordset-Objekt

Das Recordset-Objekt enthält eine Gruppe von Datensätzen aus der Datenbank. Die Menge der Datensätze ist das Ergebnis eines Befehls, der an die Datenbank gesendet wurde und die Rückgabe einer Gruppe von Datensätzen bewirkt. Durch den Recordset können Sie genauso navigieren, wie es bei Recordset-Objekten in anderen Technologien des Datenbankzugriffs üblich ist. Auf die einzelnen Felder eines Datensatzes im Recordset kann man über die Field-Objekte zugreifen, die mit dem Recordset verbunden sind. Man kann die Datensätze im Recordset aktualisieren und dann mit ihm die Datenbank auf den neuesten Stand bringen. Es lassen sich auch neue Datensätze in den Recordset einfügen oder daraus löschen. Diese Änderungen kann man dann an die Datenbank übertragen.

Bei der Arbeit mit Datenbanken können sich viele der Eigenschaften des Recordset- Objekts als nützlich erweisen. In Tabelle 13.12 sind einige der wichtigeren aufgelistet.

Eigenschaft	Beschreibung
ActiveCommand	Ein Zeiger auf das aktive Command-Objekt, mit dem das Recordset-Objekt verbunden ist
ActiveConnection	Ein Zeiger auf das aktive Connection-Objekt, mit dem das Recordset-Objekt verbunden ist
BOF, EOF	Boolesche Werte, die angeben, ob sich die aktuelle Position im Recordset vor dem ersten Datensatz (Beginning-of-File - BOF) oder hinter dem letzten Datensatz (End-of-File - EOF) befindet
CursorType	Der derzeit im Recordset verwendete Cursortyp. In Tabelle 13.13 sind die möglichen Werte aufgelistet, die kontrollieren, wie Sie durch den Recordset navigieren und ihn aktualisieren können.
EditMode	Der Bearbeitungsstatus des aktuellen Datensatzes. Die möglichen Werte sind adEditNone (keine Bearbeitung durchgeführt), adEditInProgress (Daten wurden geändert, jedoch nicht in die Datenbank gespeichert), adEditAdd (der aktuelle Datensatz ist neu und wurde der Datenbank noch nicht hinzugefügt) und adEditDelete (der aktuelle Datensatz wurde gelöscht).
Filter	Ein Stringwert, der für das Filtern der Datensätze im Recordset verwendete Kriterien enthält
LockType	Der bei der Bearbeitung von Datensätzen im Recordset verwendete Sperrentyp. In Tabelle 13.14 sind die möglichen Werte dieser Eigenschaft aufgelistet.
MaxRecords	Die maximale Anzahl der Datensätze, die eine Abfrage an einen Recordset zurückgibt. Diese Eigenschaft begrenzt die Anzahl der Datensätze, die zurückgegeben werden können, unabhängig von der Anzahl der Datensätze, die den Kriterien des SQL-Befehls entsprechen.
PageCount	Die Anzahl der im Recordset enthaltenen Datenseiten
PageSize	Die Anzahl der Datensätze, aus denen eine Datenseite besteht
RecordCount	Die Anzahl der Datensätze in einem Recordset
Sort	Ein Stringwert, der einen oder mehrere Feldnamen angibt, nach denen der Recordset sortiert wird. Die Feldnamen werden von ASC für aufsteigende Reihenfolge oder DESC für absteigende Reihenfolge gefolgt und durch Kommata voneinander getrennt.
State	Gibt den Status des Recordsets an. Die möglichen Werte dieser Eigenschaft wurden bereits in Tabelle 13.3 aufgelistet.

Tabelle 13.12: Eigenschaften des Recordset-Objekts

Typ	Beschreibung
adOpenDynamic	Verwendet einen dynamischen Cursor, wobei von anderen Benutzern durchgeführte Hinzufügungen, Änderungen und Löschvorgänge sichtbar und alle Arten der Bewegung im Recordset erlaubt sind

adOpenForwardOnly	Ein Nur-Lesen-Recordset, das nur vom ersten zum letzten Datensatz durchlaufen werden kann
adOpenKeySet	Verwendet einen Keyset-Cursor. Dieser ähnelt einem dynamischen Cursor, macht jedoch keine Änderungen anderer Benutzer sichtbar.
adOpenStatic	Ein Nur-Lesen-Recordset
adOpenUnspecified	Der verwendete Cursortyp wird nicht angegeben.

Tabelle 13.13: Cursor-Typen

Typ	Beschreibung
adLockBatchOptimistic	Zeigt optimistische Stapel-Aktualisierungen an. Wird für Stapel-Aktualisierungen benötigt
adLockOptimistic	Optimistische Sperre, die beim Aufruf der Methode Update einzelne Datensätze sperrt
adLockPessimistic	Pessimistische Sperre, die einzelne Datensätze sperrt, sobald sie bearbeitet werden
adLockReadOnly	Macht die Daten Read-only, sodass sie nicht bearbeitet werden können
adLockUnspecified	Kein Sperrentyp angegeben

Tabelle 13.14: Sperrentypen

Die meiste Funktionalität ist im Recordset-Objekt enthalten, da Sie hauptsächlich mit einer Datenbank arbeiten, um die in dieser Datenbank gespeicherten Daten zu verwenden. Das Recordset-Objekt enthält diese Daten. Weil in diesem Objekt so viel Funktionalität enthalten ist, macht es Sinn, sie in verschiedene Bereiche der Funktionalität aufzuteilen.

Recordsets öffnen und schließen

Zum Öffnen eines Recordsets können Sie die Methode Open verwenden. Diese Methode hat folgende Syntax:

```
Open(_variant_t &Source, _variant_t &ActiveConnection,
    CursorTypeEnum CursorType, LockTypeEnum LockType, long Options);
```

Der erste Parameter, Source, kann ein Command-Objekt, eine SQL-Anweisung, ein Tabellenname oder der Aufruf einer gespeicherten Prozedur sein. Der zweite Parameter, ActiveConnection, gibt die zu verwendende Datenbankverbindung an. Es kann sich dabei entweder um ein Connection-Objekt oder um einen Verbindungsstring handeln, der die zu kontaktierende Datenbank angibt, aus der der Recordset abgerufen werden soll. Der dritte Parameter, CursorType, gibt den Typ des zu verwendenden Cursors an. Die möglichen Werte dieses Parameters wurden bereits in Tabelle 13.13 aufgelistet. Der vierte Parameter, LockType, gibt das auf dem Recordset zu verwendende Sperrenschema an. Die möglichen Werte für diesen Parameter wurden bereits in Tabelle 13.14 aufgelistet. Der letzte Parameter, Options, gibt an, wie die Datenbank den Parameter Source auswerten soll. Die möglichen Werte für diesen Parameter wurden bereits in Tabelle 13.4 aufgelistet.

Nachdem Sie die Arbeit mit einem Recordset beendet haben, können Sie ihn mit der Methode Close schließen. Wenn noch asynchrone Operationen anstehen, können Sie diese mit der Methode Cancel abbrechen. Keine dieser beiden Methoden übernimmt Parameter oder gibt Ergebnisse zurück.

Durch Recordsets navigieren

Wenn Sie einen Recordset aus der Datenbank abgerufen haben, müssen Sie darin navigieren (falls der Recordset nicht nur einen Datensatz enthält). Das Recordset-Objekt stellt mehrere Funktionen für die

Navigation im Recordset bereit, mit denen Sie jeden Datensatz erreichen können. In Tabelle 13.15 sind diese Funktionen aufgelistet.

Funktion	Beschreibung
MoveFirst	Geht zum ersten Datensatz im Recordset
MoveLast	Geht zum letzten Datensatz im Recordset
MoveNext	Geht zum nächsten Datensatz im Recordset
MovePrevious	Geht zum vorhergehenden Datensatz im Recordset
Move	Geht eine bestimmte Anzahl von Datensätzen vom aktuellen oder ersten Datensatz aus weiter

Tabelle 13.15: Recordset-Navigationsfunktionen

Von all diesen Funktionen übernimmt nur Move zwei Argumente, wobei eine gewisse Ähnlichkeit mit den im vorigen Kapitel vorgestellten Seek-Funktionen des CFile-Objekts unverkennbar ist:

- Die Anzahl der Zeilen, um die der Datensatzzeiger bewegt werden soll - diese Zahl kann positiv oder negativ sein; eine negative Zahl gibt eine Rückwärtsbewegung im Recordset an.
- Die Startposition, von der aus Sie sich durch die Zeilen bewegen möchten - dieser Parameter kann entweder ein Bookmark oder einer der in Tabelle 13.16 aufgelisteten vordefinierten Bookmarks sein.

Position	Beschreibung
adBookmarkCurrent	Geht die angegebene Anzahl von Zeilen von der aktuellen Zeile aus vor oder zurück
adBookmarkFirst	Geht die angegebene Anzahl von Zeilen von der ersten Zeile aus weiter
adBookmarkLast	Geht die angegebene Anzahl von Zeilen von der letzten Zeile aus weiter

Tabelle 13.16: Startpositionen für Move

Recordsets aktualisieren

Die Navigation durch die Datensätze aus einer Datenbank ist nur ein Teil dessen, was Sie können müssen. Sie müssen auch neue Datensätze in den Recordset einfügen, existierende Datensätze bearbeiten und aktualisieren und Datensätze löschen können. Diese Aktionen werden durch die verschiedenen Funktionen ermöglicht, die das Recordset-Objekt bereitstellt. In Tabelle 13.17 sind die Funktionen aufgelistet, mit denen Sie Ihren Benutzern diese Funktionalität ermöglichen können.

Funktion	Beschreibung
AddNew	Fügt einen neuen Datensatz in den Recordset ein. Übernimmt zwei Parameter: ein Array mit Feldnamen und ein Array mit Werten für den neuen Datensatz
Delete	Löscht den aktuellen Datensatz aus dem Recordset. Übernimmt einen Parameter, um die von der Löschoperation betroffenen Datensätze anzugeben. In Tabelle 13.18, »Spezifizierer für Delete«, sind die möglichen Werte aufgelistet.
Update	Speichert die aktuellen Änderungen in die Datenbank. Kann zwei optionale Parameter übernehmen: ein Array mit Feldnamen und ein Array mit Werten für den neuen Datensatz
Requery	Startet die aktuelle SQL-Abfrage neu, um den Recordset auf den neuesten Stand zu bringen. Übernimmt einen Parameter, der angibt, wie die Abfrage ausgeführt werden soll. In Tabelle 13.4: Optionswerte von Execute wurden die möglichen Werte dieses Parameters bereits aufgelistet.

Tabelle 13.17: Recordset-Bearbeitungsfunktionen

Wert	Beschreibung
adAffectAll	Wenn kein Filter existiert, sind alle Datensätze im Recordset betroffen. Existiert ein Filter, sind nur sichtbare Datensätze betroffen.
adAffectCurrent	Betrifft nur den aktuellen Datensatz
adAffectGroup	Betrifft nur die Datensätze, die der aktuellen Einstellung der Eigenschaft Filter entsprechen

Tabelle 13.18: Spezifizierer für Delete

Um der Datenbank einen neuen Datensatz hinzuzufügen, rufen Sie die Funktion AddNew auf. Dann setzen Sie Standardwerte in alle Felder, die Werte benötigen, wie beispielsweise die Schlüsselfelder. Nachdem Sie den neuen Datensatz zum Recordset hinzugefügt haben, müssen Sie zum letzten Datensatz im Recordset gehen, da dies der gerade hinzugefügte ist.

Wenn Sie den aktuellen Datensatz löschen wollen, rufen Sie die Funktion Delete auf. Nachdem Sie den aktuellen Datensatz gelöscht haben, müssen Sie zu einem anderen Datensatz gehen, damit der Benutzer nicht weiterhin den gerade gelöschten Datensatz sieht. Wenn Sie den aktuellen Datensatz löschen, existiert kein aktueller Datensatz, bis Sie zu einem anderen gehen.

Das Field-Objekt

Das Field-Objekt repräsentiert eine einzelne Spalte im Recordset. Jedes Field-Objekt enthält den Spaltennamen, den Datenwert und die Darstellungsart desselben. Da ADO für den Einsatz in den Scriptsprachen von Microsoft vorgesehen ist und in diesen lediglich der Datentyp Variant zur Verfügung steht, enthalten Field-Objekte immer einen Datenwert vom Typ Variant. Der Datenwert wird automatisch in den jeweiligen Datentyp konvertiert, wenn die Datenbank aktualisiert wird. Als Programmierer, der mit den ADO-Objekten arbeitet, müssen Sie den Wert des Datentyps Variant in den benötigten Datentyp umwandeln und beim Aktualisieren des Wertes diese Umwandlung in der anderen Richtung vornehmen. In Tabelle 13.19 sind die wichtigsten Eigenschaften des Field- Objekts aufgelistet.

Eigenschaft	Beschreibung
ActualSize	Die tatsächliche Länge des Feldwerts
DefinedSize	Die maximal mögliche Länge des Feldwerts
Name	Der Name des Field-Objekts
NumericScale	Ein Byte-Wert, der die Anzahl der Dezimalstellen angibt, nach der numerische Werte aufgelöst werden
OriginalValue	Der ursprüngliche Wert des Felds, bevor Änderungen vorgenommen werden
Precision	Ein Byte-Wert, der die Anzahl der zur Repräsentation numerischer Werte verwendeten Stellen angibt
Status	Der Status des Feldwerts. In Tabelle 13.20 sind die möglichen Werte dieser Eigenschaft aufgelistet.
Type	Der Datentyp des Felds. Die möglichen Werte für diese Eigenschaft wurden bereits in Tabelle 13.9 aufgelistet.
UnderlyingValue	Der Wert für dieses Feld in der Datenbank
Value	Der aktuelle Wert des Felds

Tabelle 13.19: Eigenschaften des Field-Objekts

Status	Beschreibung
adFieldAlreadyExists	Das Feld existiert bereits.

adFieldBadStatus	Es wurde ein ungültiger Feldstatus von ADO an den OLE-DB-Provider übergeben.
adFieldCannotComplete	Der Server konnte die Operation nicht fertig stellen.
adFieldCannotConvertValue	Das Feld kann nicht ohne Datenverlust abgerufen oder gespeichert werden.
adFieldCantCreate	Das Feld kann nicht hinzugefügt werden, weil der Provider eine Begrenzung überschritten hat.
adFieldDataOverflow	Die vom Provider zurückgegebenen Daten haben einen Overflow im Datentyp des Felds verursacht.
adFieldDefault	Der Standardwert des Felds wurde verwendet.
adFieldDoesNotExist	Das angegebene Feld existiert nicht.
adFieldIgnore	Das Feld wurde beim Setzen von Datenwerten in der Quelle übersprungen.
adFieldIntegrityViolation	Das Feld kann nicht gesetzt werden, weil es sich um eine berechnete oder abgeleitete Einheit handelt.
adFieldInvalidURL	Die URL der Datenquelle enthält unzulässige Zeichen.
adFieldIsNull	Das Feld enthält einen NULL-Wert, was bedeutet, dass ein Feld oder eine Spalte keinen Wert enthält.
adFieldOK (Standard)	Das Feld wurde erfolgreich hinzugefügt oder gelöscht.
adFieldPendingDelete	Das Feld wird gelöscht, wenn die Methode Update aufgerufen wird.
adFieldPendingInsert	Das Feld wird eingefügt, wenn die Methode Update aufgerufen wird.
adFieldPendingUnknown	Der Provider kann nicht feststellen, durch welche Operation das Feld gesetzt wurde.
adFieldPendingUnknownDelete	Der Provider kann nicht feststellen, durch welche Operation das Feld gesetzt wurde und das Feld wird gelöscht, wenn die Methode Update aufgerufen wird.
adFieldPermissionDenied	Das Feld kann nicht geändert werden.
adFieldReadOnly	Das Feld ist als Read-only definiert.
adFieldResourceExists	Der Provider konnte die Operation nicht durchführen, weil ein Objekt bereits existiert und nicht überschrieben werden kann.
adFieldResourceLocked	Der Provider konnte die Operation nicht durchführen, weil die Datenquelle gesperrt ist.
adFieldResourceOutOfScope	Die Datenquelle befindet sich außerhalb des Geltungsbereichs des aktuellen Datensatzes.
adFieldSchemaViolation	Der Wert verletzte die auf das Feld angewendeten Einschränkungen.
adFieldSignMismatch	Der vom Provider zurückgegebene Datenwert enthielt ein Vorzeichen, der Feldwert ist jedoch vorzeichenlos.
adFieldTruncated	Ein Wert variabler Länge wurde beim Lesen aus der Datenquelle abgeschnitten.
adFieldUnavailable	Der Wert des Felds konnte nicht ermittelt werden.
adFieldVolumeNotFound	Der Provider konnte das in der URL angegebene Speichermedium nicht finden.

Tabelle 13.20: Field-Statuswerte

Die ADO-DLL importieren

Wenn Sie sich in der MFC-Klassenhierarchie umsehen, werden Sie Klassen für den Einsatz von ADO

vermissen. Sie müssen diese Klassen nicht selbst erstellen, weil Microsoft andere Werkzeuge bereitgestellt hat, um Klassen für die Objekte in ADO zu erstellen und zu verwenden. Zu diesem Zweck gibt es eine neue C++-Präcompilerdirektive namens #import.

Mit dieser Direktive kann man eine ActiveX Dynamic Link Library (DLL) importieren, die mit der Schnittstellenbeschreibung IDispatch erstellt wurde. Diese Direktive weist den Visual C++-Compiler an, die angegebene DLL zu importieren und die Objektinformationen aus der DLL herauszuziehen, wobei eine Reihe von Header-Dateien erstellt und automatisch in Ihr Projekt aufgenommen werden. Diese Header-Dateien weisen die Dateinamenserweiterungen .tlh und .tli auf und befinden sich im Ausgabeverzeichnis Ihres Projekts (dem Verzeichnis Debug oder Release, demselben Verzeichnis, in dem Sie die ausführbare Anwendung finden, nachdem Sie das Projekt kompiliert haben). Diese beiden Dateien enthalten Klassendefinitionen für alle Objekte in der DLL, die Sie in Ihrem Code verwenden können. Die Direktive #import weist den Compiler auch an, die DLL als Teil des Projekts einzubinden, wodurch die Notwendigkeit entfällt, die .lib-Datei für die DLL in das Projekt aufzunehmen.

Um die ADO-DLL zu importieren, schreiben Sie den folgenden Code an den Beginn der Header-Datei, in der Sie alle Datenbankobjekte definieren:

```
#define INITGUID
#import "C:\Programme\Gemeinsame Dateien\System\ADO\msado15.dll"
        rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"
```

Der hier angegebene Pfadname kann sich bei Ihrem Computer unterscheiden, je nach verwendeter Sprachversion und Systemalaufwerk. Suchen Sie gegebenenfalls nach der Datei mit dem Namen msado15.dll.

In diesen vier Zeilen definiert die erste Direktive eine Konstante, die für ADO erforderlich ist. Die Konstante INITGUID kontrolliert, wie das Makro DEFINE_GUID definiert wird, mit dem GUIDs (Globally Unique IDs - global eindeutige IDs) für verschiedene COM-Dienste und -Objekte erstellt werden. Die zweite Zeile importiert die ADO-DLL und erstellt die erwähnten zwei Header-Dateien. Nach dem Namen der zu importierenden Datei enthält diese Direktive zwei Attribute zur #import-Direktive:

- rename_namespace benennt den Namensbereich um, in den die DLL importiert wurde. Daran schließt sich die Zeile nach der #import-Direktive an, wo der umbenannte Namensbereich als der verwendete spezifiziert wird;
- rename benennt ein Element in den Header-Dateien um, die mit der #import-Direktive erzeugt wurden. Die Elemente in diesen Header-Dateien sind umzubenennen, um zu verhindern, dass Konflikte mit einem anderen Element, das an anderer Stelle benannt wurde, auftreten. Wenn Sie sich die Header-Datei ansehen, ist das angegebene Element nicht in der Datei umbenannt, wenn aber der Compiler die Datei liest, benennt er das Element um.

Die letzte Zeile bindet die ADO-Header-Datei mit den Definitionen verschiedener Makros ein, die Sie beim Schreiben Ihrer ADO-Anwendungen benötigen.

Mit einer Datenbank verbinden

Bevor man ADO-Objekte verwenden kann, muss man die COM-Umgebung für die Anwendung initialisieren. Dazu ruft man die API-Funktion CoInitialize auf und übergibt NULL als einzigen Parameter:

```
::CoInitialize(NULL);
```

Damit können Sie nun ActiveX-Objekte aufrufen. Wenn Sie diese Code-Zeile in Ihrer Anwendung auslassen oder erst nach der Interaktion mit den Objekten schreiben, erhalten Sie einen COM-Fehler, sobald Sie die Anwendung ausführen.

Wenn Sie alle Aktivitäten in Bezug auf ADO beendet haben, müssen Sie auch die COM- Umgebung herunterfahren. Rufen Sie dazu die Funktion CoUninitialize auf:

```
CoUnitialize();
```

Diese Funktion räumt die COM-Umgebung auf und bereitet Ihre Anwendung auf das Herunterfahren vor.

Sobald die COM-Umgebung initialisiert ist, können Sie eine Verbindung zur Datenbank herstellen. Statt eine Connection-Objektvariable zu erzeugen, deklariert man besser einen Connection-Objektzeiger, _ConnectionPtr, und verwendet ihn für alle Zugriffe auf das Connection-Objekt. Diesen Connection-Objektzeiger können Sie dann mit einer Instanz des Connection-Objekts initialisieren. Dazu rufen Sie die Funktion CreateInstance auf und übergeben die UUID (Universally Unique Identifier - universell eindeutige IDs) des Connection-Objekts als einzigen Parameter:

```
_ConnectionPtr ptrConn;
ptrConn.CreateInstance(__uuidof(Connection));
```

Eine weitere Möglichkeit ist folgende:

```
_ConnectionPtr ptrConn("ADODB.Connection");
```

Wenn Sie mit diesen Objekten und Funktionen arbeiten, ist auf die richtige Anzahl von Unterstrichen vor den verschiedenen Objekt- und Funktionsnamen zu achten. Das Objekt _ConnectionPtr hat nur einen einzigen Unterstrich, während in __uuidof zwei Unterstriche zu schreiben sind.

Das _ConnectionPtr-Objekt wird als intelligenter Zeiger (smart pointer) bezeichnet. Ein intelligenter Zeiger ist ein Wrapper (to wrap - umwickeln) um einen COM-Schnittstellenzeiger. Ein Schnittstellenzeiger ist ein Zeiger auf eine COM-Schnittstelle. Ein intelligenter Zeiger erzeugt über den Schnittstellenzeiger einen Wrapper um die Schnittstelle, sodass Sie den Schnittstellenzeiger als Klasseninstanz behandeln können. Das vereinfacht die Arbeit mit einer COM-Schnittstelle, jedoch kann die Bezeichnung »Schnittstellenzeiger« verwirrend sein, da Sie mit ihm nicht wie mit einem gewöhnlichen C++-Zeiger interagieren.

Nachdem Sie das Objekt erzeugt haben, können Sie die Funktion Open aufrufen, um die Verbindung zur Datenbank einzurichten. Die Funktion Open übernimmt vier Parameter:

- Die Verbindungszeichenfolge, die die OLE-DB-Datenquelle für die Datenbank definiert. Das kann ein ODBC-OLE-DB-Treiber sein, wobei OLE DB über einer ODBC-Datenquelle sitzt, wie Sie sie in Ihrer Beispielanwendung einsetzen. Wenn Sie mit SQL Server oder Oracle-Datenbanken arbeiten, kann es eine direkte Verbindung zur OLE-DB-Schnittstelle sein, die von der Datenbank selbst bereitgestellt wird.
- Die Benutzer-ID für die Verbindung zur Datenbank
- Das Kennwort für die Verbindung zur Datenbank
- Der Cursortyp, der mit der Datenbank zum Einsatz kommt. Diese Typen sind in der Header-Datei msado15.tlh definiert, die durch die #import-Direktive angelegt wird.

Das folgende Beispiel zeigt einen typischen Einsatzfall der Funktion Open, um eine Verbindung zu einer ODBC-Datenquelle herzustellen, die weder Benutzer-ID noch Kennwort benötigt:

```
ptrConn->Open(L"Provider=MSDASQL.1;Data Source=VCPP21DB", L"", L"", adConnectUnsp
```

Beachten Sie, daß auf diese Weise das Kennwort im Klartext im fertigen Programm stehen würde! In einer sicherheitsrelevanten Umgebung sollten Sie entweder die Eingabe des Kennwortes vom Benutzer verlangen oder das Kennwort im Programmtext verschlüsselt abspeichern.

Befehle ausführen und Daten abrufen

Nachdem Sie die Verbindung geöffnet haben, können Sie mit einem Command-Objekt SQL-Befehle an die Datenbank übergeben. Das ist die normale Methode, um SQL- Befehle mit ADO auszuführen. Um ein Command-Objekt zu erzeugen, gehen Sie genauso vor wie bei einem Connection-Objekt. Sie deklarieren einen Command-Objektzeiger, _CommandPtr, und erzeugen dann eine Instanz dieses Objekts mit der UUID des Command- Objekts:

```
_CommandPtr ptrCmd;
ptrCmd.CreateInstance(__uuidof(Command));
```

oder

```
_CommandPtr ptrCmd("ADODB.Command");
```

Wenn Sie Ihr Command-Objekt erstellt und bereits eine Verbindung zur Datenbank eingerichtet haben, setzen Sie die Eigenschaft ActiveConnection (für die aktive Verbindung) des Command-Objekts auf den geöffneten Connection-Objektzeiger:

```
ptrCmd->ActiveConnection = ptrConn;
```

Als Nächstes können Sie den auszuführenden SQL-Befehl spezifizieren, indem Sie die CommandText-Eigenschaft des Command-Objekts festlegen:

```
ptrCmd->CommandText = "SELECT * FROM Adressen";
```

Jetzt haben Sie zwei Möglichkeiten, um diesen Befehl auszuführen und die Datensätze abzurufen. Erstens können Sie die Methode Execute des Command-Objekts aufrufen. Die Methode gibt ein neues Recordset-Objekt zurück, das Sie auf einen Recordset- Objektzeiger setzen:

```
_RecordsetPtr ptrRs;
ptrRs = ptrCmd->Execute();
```

Die andere Lösung, den Befehl auszuführen und die Datensätze abzurufen, besteht darin, das Command-Objekt als Quelle für die Datensätze im Recordset festzulegen. Dazu ist es erforderlich, das Recordset-Objekt wie folgt zu erstellen:

```
_RecordsetPtr ptrRs;
ptrRs.CreateInstance(__uuidof(Recordset));
ptrRs->PutRefSource(ptrCmd);
```

oder

```
_RecordsetPtr ptrRs("ADODB.Recordset");
ptrRs->PutRefSource(ptrCmd);
```

Jetzt sind zwei NULL-Werte vom Typ Variant zu erzeugen, um sie als die beiden ersten Parameter an die Methode Open des Recordset-Objekts zu übergeben. Der dritte Parameter ist der zu verwendende Cursortyp, gefolgt von der zu verwendenden Sperrmethode. Schließlich enthält der fünfte Parameter an die Methode Open des Recordset-Objekts ein Options-Flag, das angibt, wie die Datenbank den übergebenen Befehl auswerten soll. Dazu schreiben Sie folgenden Code:

```
// NULL vom Typ Variant erzeugen
_variant_t vNull;
```

```
vNull.vt = VT_ERROR;
vNull.scode = DISP_E_PARAMNOTFOUND;
// Recordset öffnen
ptrRs->Open(vNull, vNull, adOpenDynamic, adLockOptimistic,
            adCmdUnknown);
```

Es gibt eine weitere Lösung, um alle obigen Aufgaben in nur ein paar Code-Zeilen zu formulieren. Man lässt die Objekte Command und Connection einfach links liegen und schreibt die gesamten Verbindungsinformationen in die Funktion Open des Recordset- Objekts. Den SQL-Befehl kann man als ersten Parameter spezifizieren und die Verbindungsinformationen als zweiten, statt der beiden NULL-Werte, die Sie vorher übergeben haben. Bei dieser Methode reduziert sich der gesamte obige Code auf die folgenden Zeilen:

```
_RecordsetPtr ptrRs;
ptrRs.CreateInstance(__uuidof(Recordset));
ptrRs->Open(_T("Provider=MSDASQL.1;Data Source=VCPP21DB"),
          _T("SELECT * FROM Adressen"), adOpenDynamic,
           adLockOptimistic, adCmdUnknown);
```

Auch wenn es bei einer einfachen Anwendung - wie der heute zu erstellenden - ohne weiteres möglich ist, alle Befehls- und Verbindungsinformationen in der Funktion Open des Recordset-Objekts unterzubringen, empfiehlt es sich bei Anwendungen, die mehr als ein paar Datenbankabfragen ausführen, das Connection-Objekt zu verwenden. Damit kann man eine einzelne Datenbankverbindung einrichten und diese Verbindung zur gesamten Interaktion mit der Datenbank nutzen.

Durch den Recordset navigieren

Nachdem Sie eine Gruppe von Datensätzen aus der Datenbank abgerufen und in einem Recordset-Objekt abgelegt haben, müssen Sie durch die Datensätze navigieren. Diese Funktionalität ist erwartungsgemäß über die Funktionen MoveFirst (zum ersten Datensatz), MoveLast (zum letzten Datensatz), MovePrevious (zum vorherigen Datensatz) und MoveNext (zum nächsten Datensatz) verfügbar. Die Funktionen übernehmen keine Parameter, da sie genau die beschriebenen Aufgaben ausführen.

Neben diesen Funktionen verfügt das Recordset-Objekt noch über die beiden Eigenschaften BOF und EOF (die Sie normalerweise umbenennen sollten, um Konflikte mit der Standarddefinition von EOF zu vermeiden). Aus diesen Eigenschaften geht hervor, ob der aktuelle Datensatz in der Gruppe außerhalb der Datensatzgruppe liegt.

Die Konstante EOF steht für End Of File (Dateiende). Es handelt sich dabei um eine in der Standard-E/A-Funktionalität für die Sprachen C und C++ definierte Konstante. Um die Konstante in der Datenbankfunktionalität zu verwenden, muss sie neu definiert werden, sodass sie im Recordset-Objekt das Ende des Recordsets definieren kann.

Auf Feldwerte zugreifen

Sobald Sie auf die Datenwerte in den einzelnen Feldern zugreifen müssen, beginnt das Arbeiten mit ADO in Visual C++ interessant zu werden. Das Konzept von ADO zielt auf einen einfachen Einsatz in den Scriptsprachen VBScript und JScript von Microsoft ab, die nur den Datentyp Variant kennen, sodass alle Datenelemente, die Sie aus den Feldern im ADO-Recordset abrufen, Variant-Werte sind. Man muss sie in die Datentypen konvertieren, die man eigentlich benötigt.

Dazu gibt es zwei Wege. Beim einfacheren ruft man die Werte in einen Variant ab und konvertiert sie dann, wie es der folgende Code zeigt:

```
_variant_t vFirstName;
CString strFirstName;
vFirstName = ptrRs->GetCollect(_variant_t("FirstName"));
vFirstName.ChangeType(VT_BSTR);
strFirstName = vFirstName.bstrVal;
```

Der etwas mühsamere Weg ist tatsächlich der bessere und erleichtert auf lange Sicht die Arbeit. Microsoft hat eine Reihe von Makros geschaffen, die die Konvertierung automatisch vornehmen und einen Satz von Variablen der Datensätze in der Ergebnismenge verwalten. Um damit zu arbeiten, definiert man eine neue Klasse als Schnittstelle zum Recordset. Diese Klasse leitet man von der Klasse CADORecordBinding ab, die in der Header-Datei icrsint.h definiert ist und die man unmittelbar nach der #import- Direktive einbindet. Diese Klasse weist weder Konstruktor noch Destruktor auf, enthält aber verschiedene Makros und eine Anzahl von Variablen. Jedes Feld in der Datensatzmenge hat zwei Variablen, eine unsigned long, die den Status der Variablen verwaltet, und die Feldvariable an sich. Diese Variablen müssen reguläre C-Variablen sein und dürfen nicht als C++-Klassen wie etwa CString angelegt sein. Der folgende Code zeigt ein einfaches Beispiel für diese Klassendeklaration:

```
class CCustomRs :
    public CADORecordBinding
{
BEGIN_ADO_BINDING(CCustomRs)
    ADO_FIXED_LENGTH_ENTRY(1, adInteger, m_lAddressID,
        lAddressIDStatus, FALSE)
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szFirstName,
        sizeof(m_szFirstName), lFirstNameStatus, TRUE)
    ADO_FIXED_LENGTH_ENTRY(3, adDate, m_dtBirthdate,
        lBirthdateStatus, TRUE)
    ADO_FIXED_LENGTH_ENTRY(4, adBoolean, m_bSendCard,
        lSendCardStatus, TRUE)
END_ADO_BINDING()
public:
    LONG m_lAddressID;
    ULONG lAddressIDStatus;
    CHAR m_szFirstName[51];
    ULONG lFirstNameStatus;
    DATE m_dtBirthdate;
    ULONG lBirthdateStatus;
    VARIANT_BOOL m_bSendCard;
    ULONG lSendCardStatus;
};
```

Den Aufbau dieser Klasse definieren Sie entsprechend der Anordnung der Datensätze, die die Datenbankabfrage zurückgibt. Dann können Sie eine Variable dieser Klasse für Ihre Anwendung wie folgt deklarieren:

```
CCustomRs m_rsRecSet;
```

Als Nächstes ist ein Zeiger auf eine IADORecordBinding-Schnittstelle zu erzeugen:

```
IADORecordBinding *picRs = NULL;
```

Das ist ein Zeiger auf eine COM-Schnittstelle, die Teil des ADO-Recordset-Objekts ist. Wenn Sie die Gruppe der Datensätze abrufen, müssen Sie den Zeiger auf die IADORecordBinding-Schnittstelle abrufen und die benutzerdefinierte Recordset-Klasse an das Recordset-Objekt wie im folgenden Code-Beispiel binden:

```
if (FAILED(pRs->QueryInterface(__uuidof(IADORecordBinding),
    (LPVOID *)&picRs)))
  _com_issue_error(E_NOINTERFACE);
picRs->BindToRecordset(&m_rsRecSet);
```

Wenn Sie nun durch die Datensätze in der Ergebnismenge navigieren, brauchen Sie nur auf die Member-Variablen Ihrer benutzerdefinierten Datensatzklasse zuzugreifen, um den aktuellen Wert für jedes Feld abzurufen.

Die Makros BEGIN_ADO_BINDING und END_ADO_BINDING

Den Schlüssel zur zweiten Methode des Zugriffs auf Datenwerte im Recordset bilden die Makros, mit denen Sie die Datensatzklasse definieren. Die Gruppe der Makros beginnt mit BEGIN_ADO_BINDING, das den Klassennamen als einzigen Parameter übernimmt. Das Makro richtetet die Strukturdefinition ein, die mit den nachstehend beschriebenen restlichen Makros erzeugt wird.

Das Makro END_ADO_BINDING schließt die Makrogruppe ab. Dieses Makro übernimmt keine Parameter und beendet die Definition der Datensatzbindungsstruktur, die in der Klasse erzeugt wurde. In den übrigen Makros, die zwischen diesen beiden stehen, läuft die eigentliche Arbeit ab.

Die ADO_FIXED_LENGTH_ENTRY-Makros

Das Makro ADO_FIXED_LENGTH_ENTRY kommt bei Datenbankfeldern mit fester Größe zum Einsatz. Es arbeitet mit Feldern vom Typ Datum oder Boole sowie mit Textfeldern, denen eine feste Größe ohne die Möglichkeit einer Änderung in der Datenbank zugewiesen ist. Das Makro existiert in zwei Versionen, wobei man für die zweite Version eine 2 an den Makronamen anhängt (ADO_FIXED_LENGTH_ENTRY2).

```
ADO_FIXED_LENGTH_ENTRY(Ordinal, DataType, Buffer, Status, Modify)

ADO_FIXED_LENGTH_ENTRY2(Ordinal, DataType, Buffer, Modify)
```

Beide Versionen erfordern die gleichen ersten drei und den letzten Parameter. In der ersten Version ist ein zusätzlicher Parameter erforderlich, der in der zweiten Version optional ist.

Der erste Parameter, Ordinal, gibt die Ordinalzahl des Feldes im Recordset an. Es handelt sich dabei um die Feldreihenfolge, wie sie die SQL-Abfrage zurückgibt, die zum Füllen des Recordsets ausgeführt wird. Der zweite Parameter, Data Type, bezeichnet den Datentyp des Feldes. Die verfügbaren Datentypen sind in der Header-Datei definiert, die durch die #import-Direktive erzeugt wird. Im dritten Parameter, Buffer, steht die Variable, in die der Datenwert kopiert wird.

Bei der ersten Version des Makros ist der vierte Parameter, Status, die Variable für den Feldstatus (der unsigned long, den Sie mit der Variablen für den eigentlichen Wert definiert haben).

Die letzte Variable, Modify, ist ein Boolescher Wert, der angibt, ob das Feld modifiziert werden kann.

Die ADO_NUMERIC_ENTRY-Makros

Die ADO_NUMERIC_ENTRY-Makros verwendet man nur in Verbindung mit numerischen Feldern. Wie die ADO_FIXED_LENGTH_ENTRY-Makros liegen sie ebenfalls in zwei Versionen vor, die in der gleichen Weise benannt sind. Die ersten fünf und der letzte Parameter sind in beiden Versionen gleich. Wie bei den ADO_FIXED_LENGTH_ENTRY-Makros hat die erste Version einen zusätzlichen Parameter, der in der zweiten Version nicht verwendet wird.

```
ADO_NUMERIC_ENTRY(Ordinal, DataType, Buffer, Precision, Scale, Status, Modify)

ADO_NUMERIC_ENTRY2(Ordinal, DataType, Buffer, Precision, Scale, Modify)
```

Die drei ersten Parameter für die ADO_NUMERIC_ENTRY-Makros entsprechen den Parametern für die ADO_

Die ADO_VARIABLE_LENGTH_ENTRY-Makros

Die ADO_VARIABLE_LENGTH_ENTRY-Makros bilden den Abschluss in der Reihe der Makros. Diese Makros verwendet man für Datenbankfelder, die sich in der Länge ändern können. Bei einer SQL-basierten Datenbank kommen diese Makros bei allen Spalten vom Typ varchar (Strings variabler Länge) zum Einsatz.

Das Makro existiert in vier Versionen. Alle vier Versionen stimmen in den ersten vier und dem letzten

Parameter überein. In den dazwischen vorkommenden Parametern unterscheiden sich die Makros.

```
ADO_VARIABLE_LENGTH_ENTRY(Ordinal, DataType, Buffer, Size, Status, Length, Modify
ADO_VARIABLE_LENGTH_ENTRY2(Ordinal, DataType, Buffer, Size, Status, Modify)
ADO_VARIABLE_LENGTH_ENTRY3(Ordinal, DataType, Buffer, Size, Length, Modify)
ADO_VARIABLE_LENGTH_ENTRY4(Ordinal, DataType, Buffer, Size, Modify)
```

Der erste Parameter, Ordinal, gibt die Ordinalzahl der Spalte im Recordset an, wie er von der SQL-Abfrage zurückgegeben wird. Der zweite Parameter, DataType, gibt den Datentyp an. Im dritten Parameter, Buffer, steht die Variable, in die der Datenwert zu übernehmen ist. Der vierte Parameter, Size, spezifiziert für alle Versionen des Makros die Größe der Variablen, in die der Wert einzutragen ist. Damit wird verhindert, dass die Daten über das Ende der Variablen, die für die Aufnahme des Wertes definiert wurde, hinausgeschrieben werden. Wie bei den vorherigen Makros gibt der letzte Parameter, Modify, an, ob das Feld aktualisierbar ist.

In der ersten Version des Makros stehen zwei Parameter zwischen dem vierten und dem letzten Parameter. Die zweite Version besitzt nur den ersten dieser Parameter, während in der dritten Version des Makros nur der zweite der beiden zusätzlichen Parameter vorkommt. Der erste der beiden Parameter, Status, ist die Statusvariable für das betreffende Feld. Im zweiten dieser beiden Parameter, Length, steht die Länge des Feldes in der Datenbank. Das obige Beispiel verwendet die zweite Version des Makros.

Datensätze aktualisieren

Wenn man Werte in einem Datensatz des Recordsets aktualisieren muss, hängt dessen Behandlung von den beiden Methoden ab, nach denen man die Datenelemente aus dem Recordset abruft. Wenn Sie jedes Feld abgerufen und in eigener Regie aus einem Variant konvertiert haben, müssen Sie jedes einzelne Feld, das sich geändert hat, aktualisieren. Zu diesem Zweck verwendet man die Methode Update des Recordset-Objekts. Diese Methode übernimmt zwei Variablen: das zu aktualisierende Feld und den neuen Wert für das Feld. Der folgende Code zeigt ein Beispiel für eine derartige Aktualisierung:

```
_variant_t vName, vValue;
vName.SetString("FirstName");
vValue.SetString("John");
ptrRs->Update(vName, vValue);
```

Haben Sie eine Datensatzklasse erzeugt und an den Recordset gebunden, vereinfacht sich die Aktualisierung des Datensatzes etwas. Nachdem Sie die neuen Werte in die Variablen der Datensatzklasse kopiert haben, können Sie die datensatzgebundene Version der Funktion Update aufrufen, wie es das folgende Beispiel zeigt:

```
picRs->Update(&m_rsRecSet);
```

Hier erhält das zu aktualisierende Recordset-Objekt die Werte aus der Datensatzklasse, die Sie an den Recordset gebunden haben.

Hinzufügen und Löschen

Datensätze lassen sich einem ADO-Recordset in ähnlicher Weise hinzufügen und löschen, wie man es von anderen Technologien des Datenbankzugriffs kennt. Allerdings gibt es kleinere Besonderheiten zu beachten, wenn man neue Datensätze hinzufügt.

Um den aktuellen Datensatz zu löschen, ruft man die Delete-Methode des Recordset- Objekts auf. Diese Methode erfordert einen einzelnen Parameter, der festlegt, wie das Löschen erfolgen soll. Höchstwahrscheinlich übergeben Sie den Wert adAffectCurrent, sodass nur der aktuelle Datensatz im Recordset gelöscht wird, wie es der folgende Code zeigt:

```
ptrRs->Delete(adAffectCurrent);
ptrRs->MovePrevious();
```

Analog zu anderen Technologien des Datenbankzugriffs gibt es keinen aktuellen Datensatz mehr, nachdem Sie ihn gelöscht haben. Man muss zu einem anderen Datensatz navigieren, bevor man dem Benutzer weitere Aktionen gestatten kann.

Um einen neuen Datensatz hinzuzufügen, ruft man die Methode AddNew des Recordset- Objekts auf. Der neu hinzugefügte wird zum aktuellen Datensatz im Recordset. Wenn Sie die Variablen in der von Ihnen erzeugten Datensatzklasse untersuchen, stellen Sie fest, dass sie alle leer sind. Allerdings können Sie nicht einfach beginnen, Datenwerte in diese Felder einzugeben. Um dem Benutzer die sofortige Eingabe der verschiedenen Datenelemente in den neuen Datensatz zu ermöglichen, leeren Sie die Werte in der Datensatzklasse und übergeben diese Variable als einzigen Parameter an die Methode AddNew. Diese müssen Sie über den datensatzgebundenen Schnittstellenzeiger wie im folgenden Beispiel aufrufen:

```
m_rsRecSet.m_lAddressID = 0;
strcpy(m_rsRecSet.m_szFirstName, " ");
m_rsRecSet.m_dtBirthdate = NULL;
m_rsRecSet.m_bSendCard = VARIANT_FALSE;
picRs->AddNew(&m_rsRecSet);
```

Auf diese Weise kann man dem Benutzer einen leeren Datensatz zur Verfügung stellen, der zur Bearbeitung bereit ist. Nachdem der Benutzer die verschiedenen Werte in den Datensatz eingetragen hat, kopieren Sie alle diese Werte zurück in die Datensatzvariable. Rufen Sie dann die Methode Update auf, um den Datensatz zu speichern.

Die Objekte Recordset und Connection schließen

Wenn Sie die Arbeit an einem Recordset beendet haben, schließen Sie den Recordset durch Aufruf der Methode Close, wie es der folgende Code zeigt:

```
ptrRs->Close();
```

Haben Sie die Datenbankinteraktion für die gesamte Anwendung beendet, schließen Sie auch die Verbindung zur Datenbank. Rufen Sie dazu die Methode Close des Connection- Objekts auf:

```
ptrConn->Close();
```

13.4 Eine Datenbankanwendung mit ADO erstellen

Die Beispielanwendung, die Sie heute erstellen, ist eine SDI-Anwendung ohne Datenbankunterstützung (Sie werden die Datenbankunterstützung selbst mit ADO hinzufügen). Sie verwenden ADO, um einen Recordset aus einer Access-Datenbank abzurufen und stellen die Funktionalität bereit, um durch den Recordset zu navigieren. Der Benutzer kann Änderungen an den Daten im Recordset vornehmen und diese Änderungen in die Datenbank übernehmen. Außerdem lassen sich Datensätze nach Belieben neu hinzufügen und löschen. Alles das erreichen Sie mit ADO als Mittel für den Zugriff auf die Datenbank, mit der über den ODBC-Treiber kommuniziert wird.

Vorbereitung der Datenbank

Bevor Sie beginnen können, eine Anwendung zu erstellen, die eine Datenbank verwendet, brauchen Sie eine Datenbank. Fast jede Datenbank, die Sie für Ihre Anwendung erwerben können, bringt Werkzeuge für die Erstellung einer neuen Datenbank mit. Sie müssen diese Werkzeuge verwenden, um Ihre Datenbank zu erstellen und dann mithilfe des ODBC- Administrators eine ODBC-Datenquelle für Ihre neue Datenbank zu konfigurieren.

Für das folgende Beispiel finden Sie eine bereits mit Daten gefüllte Access-Datenbank- Datei auf der beigefügten CD zum Buch. Alternativ können Sie diese Datei auch über die Webseite http://www.mut.de auf den zu diesem Buch gehörenden Seiten laden.

Sie können jede beliebige Access-Version verwenden, die Sie besitzen. Wenn Sie eine

Datenbank mit einer anderen Access-Version erstellen, müssen Sie den Beispielcode für bestimmte Datenfelder in Ihrer Datenbank modifizieren. Die englische Version der für dieses Beispiel verwendeten Datenbank ist zusammen mit dem gesamten Beispielcode online auf den Seiten der amerikanischen Originalausgabe unter http://www.samspublishing.com erhältlich (geben Sie im Suchfeld die Original-ISBN dieses Buchs - 0672321971 - ein, um alle für dieses Buch verfügbaren Downloads zu finden). Unter dieser Adresse können Sie auch nachschauen, ob es vom Autor dieses Buches Updates zu seinen Beispielapplikationen gibt.

Nachdem Sie die Datenbank erstellt haben, müssen Sie eine ODBC-Datenquelle konfigurieren, um auf die eben erzeugte Datenbank zu verweisen. Folgen Sie diesen Schritten:

1. Starten Sie den ODBC-Datenquellen-Administrator aus der Systemsteuerung von Windows. Wenn Sie Windows 2000 oder Windows XP verwenden, finden Sie den ODBC-Datenquellen-Administrator unter dem Namen Datenquellen (ODBC) im Verzeichnis Verwaltung.

2. Fügen Sie im ODBC-Datenquellen-Administrator eine neue Datenquelle hinzu, indem Sie auf die Schaltfläche Hinzufügen klicken (siehe Abbildung 13.1).

Abbildung 13.1: Der ODBC-Datenquellen-Administrator

3. Wählen Sie den Datenbanktreiber für die neue Datenquelle aus (siehe Abbildung 13.2). Für die heutige Beispielanwendung wählen Sie den Microsoft Access Driver, da die Datenbank mit Access erstellt wurde.

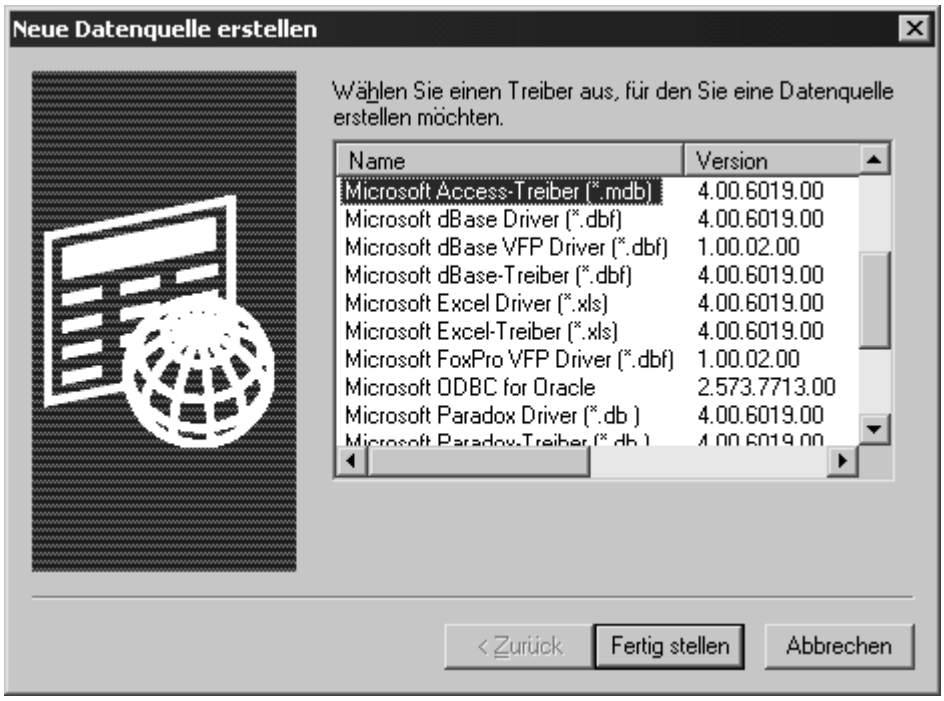


Abbildung 13.2: Das Dialogfeld Neue Datenquelle erstellen

4. Klicken Sie auf die Schaltfläche Fertig stellen.

Geben Sie im Dialogfeld ODBC Microsoft Access Setup (siehe Abbildung 13.3) einen kurzen, einfachen Namen für die Datenquelle ein. Ihre Anwendung wird diesen Namen verwenden, um die für die Datenbankverbindung zu verwendende ODBC- Datenquellenkonfiguration anzugeben, also sollte er die Funktion wiedergeben, der die Datenbank dient, oder dem Namen der Anwendung ähneln, die die Datenbank verwendet.

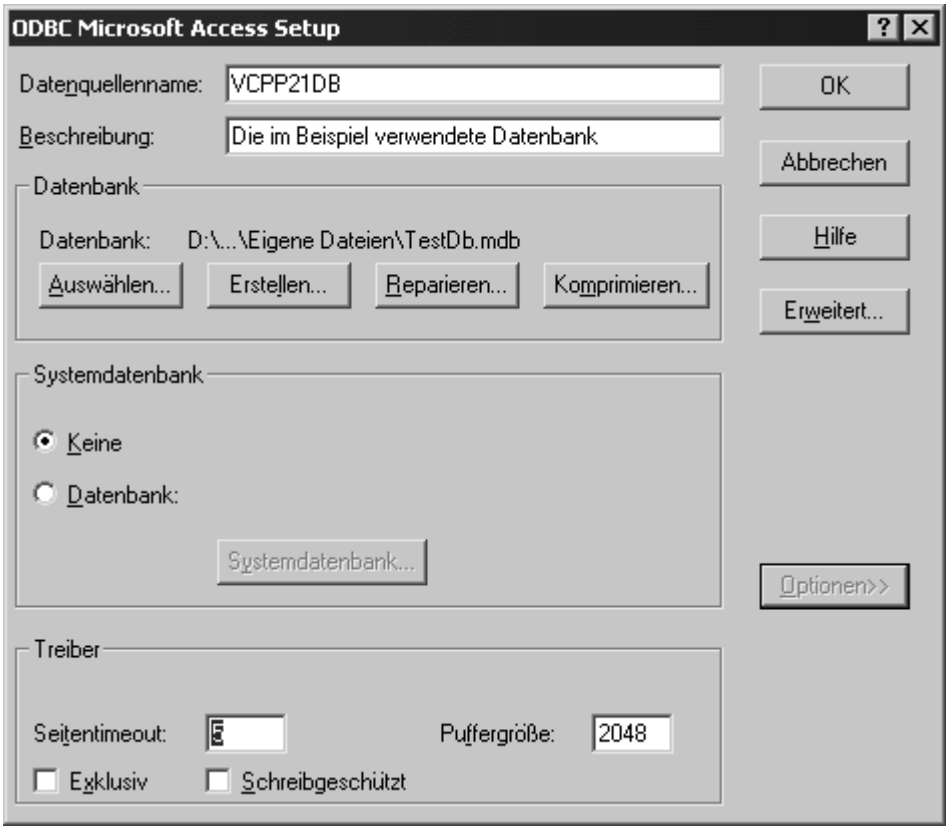


Abbildung 13.3: Das Dialogfeld ODBC Microsoft Access Setup

Fahren Sie nun mit folgenden Schritten fort:

1. Nennen Sie Ihre Datenquelle VCPP21DB (für »Visual C++ in 21 Tagen - Datenbank«) und geben Sie im nächsten Feld eine Beschreibung der Datenbank ein.

2. Um anzugeben, wo sich die Datenbank befindet, klicken Sie auf die Schaltfläche Auswählen und wählen Sie die Access-Datenbank an, die Sie erstellt haben oder wählen Sie eine bestehende Access-Datenbank aus.
3. Nachdem Sie die ODBC-Datenquelle für Ihre Datenbank konfiguriert haben, klicken Sie auf Ok, um die neue Datenquelle zum ODBC-Administrator hinzuzufügen.
4. Klicken Sie noch einmal auf Ok, um die Eingabe zu beenden und den ODBC- Datenquellen-Administrator zu schließen.

Nun können Sie Ihre Aufmerksamkeit der Erstellung Ihrer Anwendung zuwenden.

Das Anwendungsgerüst erstellen

Die heutige Beispielanwendung ist als SDI-Anwendung konzipiert. Wie bei verschiedenen anderen Anwendungen, die Sie im Verlauf dieses Buches erstellen, lassen sich alle Elemente der heutigen Anwendung ebenso auf MDI-Anwendungen oder dialogbasierte Anwendungen übertragen. Als Erstes erstellen Sie für die Anwendung ein Gerüst mit dem MFC-Anwendungs-Assistenten, wobei Sie die meisten der vorgegebenen Einstellungen für eine SDI-Anwendung übernehmen:

1. Erstellen Sie ein neues MFC-Anwendung-Visual-C++-Projekt. Nennen Sie das Projekt AdoDatabase.

2. Im Bereich Anwendungstyp wählen Sie die Option Einfaches Dokument.
3. Achten Sie darauf, im Bereich Datenbankunterstützung die Standardeinstellung Keine zu übernehmen.
4. Legen Sie im Bereich Erstellte Klassen die Basisklasse mit CFormView fest und klicken Sie auf Fertig stellen. Der MFC-Anwendungs-Assistent erzeugt das Anwendungsgerüst.
5. Nachdem Sie das Anwendungsgerüst erstellt haben, entwerfen Sie das Hauptdialogfeld. Fügen Sie gemäß Abbildung 13.4 die Standard-Steuerelemente für alle Felder der Tabelle Adressen aus der Datenbank hinzu (falls Sie eine andere Datenbank erstellt haben, nehmen Sie die Steuerelemente für die Felder auf, die Sie aus der erstellten Datenbank auswählen). Legen Sie die Eigenschaften der Steuerelemente entsprechend Tabelle 13.21 fest.

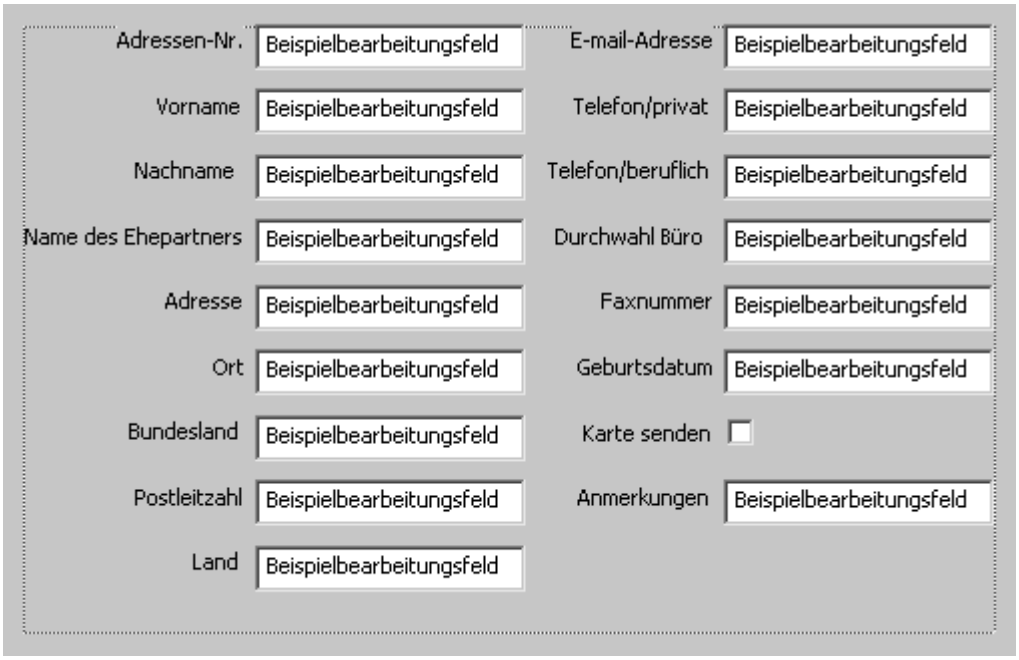


Abbildung 13.4: Das Layout des Hauptformulars

Wenn Ihre Zeit knapp bemessen ist, können Sie das Beispiel auch vereinfachen und verschiedene Steuerelemente und Datenbankfelder auslassen. Als Schlüsselfelder brauchen Sie Adressen-Nr., Vorname, Nachname, Geburtsdatum und Karte senden. Die anderen Felder können Sie in der Anwendung ohne weiteres weglassen. Diese Felder müssen Sie in die Klasse CCustomRs einbinden, die Sie weiter hinten in diesem Kapitel erstellen werden.

Objekt	Eigenschaft	Einstellung
Text	Beschriftung	Addressen-Nr.
Eingabefeld	ID	IDC_EADDRESSID
Text	Beschriftung	Vorname
Eingabefeld	ID	IDC_EFIRSTNAME
Text	Beschriftung	Nachname
Eingabefeld	ID	IDC_ELASTNAME
Text	Beschriftung	Name des Ehepartners
Eingabefeld	ID	IDC_ESPOUSENAME
Text	Beschriftung	Adresse
Eingabefeld	ID	IDC_EADDRESS
Text	Beschriftung	Ort
Eingabefeld	ID	IDC_ECITY
Text	Beschriftung	Bundesland
Eingabefeld	ID	IDC_ESTATEORPROVINCE
Text	Beschriftung	Postleitzahl
Eingabefeld	ID	IDC_EPOSTALCODE
Text	Beschriftung	Land
Eingabefeld	ID	IDC_ECOUNTRY
Text	Beschriftung	E-mail-Adresse
Eingabefeld	ID	IDC_EEMAILADDRESS
Text	Beschriftung	Telefon/privat
Eingabefeld	ID	IDC_EHOMEPHONE
Text	Beschriftung	Telefon/beruflich
Eingabefeld	ID	IDC_EWORKPHONE
Text	Beschriftung	Durchwahl Büro
Eingabefeld	ID	IDC_EWORKEXTENSION
Text	Beschriftung	Faxnummer
Eingabefeld	ID	IDC_EFAXNUMBER
Text	Beschriftung	Geburtsdatum
Eingabefeld	ID	IDC_EBIRTHDATE
Text	Beschriftung	Karte senden
Kontrollkästchen	ID	IDC_CBSENDCARD
Text	Beschriftung	Anmerkungen

Eingabefeld	ID	IDC_ENOTES

Tabelle 13.21: Eigenschaften der Steuerelemente für das Hauptformular der Datenbankanwendung

6. Nachdem Sie diese Steuerelemente auf dem Formular untergebracht haben, weisen Sie den Steuerelementen Variablen zu, wie es aus Tabelle 13.22 hervorgeht. Die Variablen sollten mit den Datentypen der Datenbankspalten, die das jeweilige Steuerelement anzeigt, übereinstimmen.

Objekt	Name	Kategorie	Typ
IDC_EADRESSID	m_lAddressID	Value	long
IDC_EFIRSTNAME	m_strFirstName	Value	CString
IDC_ELASTNAME	m_strLastName	Value	CString
IDC_ESPOUSENAME	m_strSpouseName	Value	CString
IDC_EADDRESS	m_strAddress	Value	CString
IDC_ECITY	m_strCity	Value	CString
IDC_ESTATEORPROVINCE	m_strStateOrProvince	Value	CString
IDC_EPOSTALCODE	m_strPostalCode	Value	CString
IDC_ECOUNTRY	m_strCountry	Value	CString
IDC_EEMAILADDRESS	m_strEmailAddress	Value	CString
IDC_EHOMEPHONE	m_strHomePhone	Value	CString
IDC_EWORKPHONE	m_strWorkPhone	Value	CString
IDC_EWORKEXTENSION	m_strWorkExtension	Value	CString
IDC_EFAXNUMBER	m_strFaxNumber	Value	CString
IDC_EBIRTHDATE	m_oledtBirthdate	Value	COleDateTime
IDC_CBSENDCARD	m_bSendCard	Value	BOOL
IDC_ENOTES	m_strNotes	Value	CString

Tabelle 13.22: Variablen der Steuerelemente

Eine benutzerdefinierte Datensatzklasse erstellen

Bevor Sie weiter an Ihrer Anwendung arbeiten, müssen Sie erst Ihre benutzerdefinierte Datensatzklasse erstellen, die Sie an den Recordset binden. In dieser Klasse sind öffentliche Variablen für alle Spalten in der ausgewählten Datenbanktabelle sowie Statusvariablen für diese Spalten erforderlich. Weiterhin erstellen Sie die Gruppe von Makros, um die Spaltenwerte zwischen dem Recordset und den Klassenvariablen auszutauschen. Um die Klasse zu erstellen, folgen Sie diesen Schritten:

1. Legen Sie eine neue Klasse nach der gleichen Methode wie in den vergangenen Tagen an. Als Typ wählen Sie Allgemeine Klasse.

2. Nennen Sie die Klasse CCustomRs und legen Sie CADORecordBinding als Basisklasse mit Zugriffsstatus public fest.
3. Löschen Sie den Konstruktor und den Destruktor sowohl aus den Header- als auch den Quellcode-Dateien für die neue Klasse.
4. Bearbeiten Sie die Header-Datei für die neue Klasse, um die ADO-DLL zu importieren und um die Makros und Variablen aufzunehmen, wie es Listing 13.1 zeigt. Möglicherweise müssen Sie den Pfad zur Datei msado15.dll anpassen.

Listing 13.1: Die benutzerdefinierte Datensatzklasse

```
#define INITGUID
#import "C:\Programme\Gemeinsame Dateien\System\ADO\msado15.dll"
        rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include <icrsint.h>
class CCustomRs :
    public CADORecordBinding
{
    BEGIN_ADO_BINDING(CCustomRs)
    ADO_FIXED_LENGTH_ENTRY(1, adInteger, m_lAddressID,
        lAddressIDStatus, FALSE)
    ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szFirstName,
        sizeof(m_szFirstName), lFirstNameStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szLastName,
        sizeof(m_szLastName), lLastNameStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_szSpouseName,
        sizeof(m_szSpouseName), lSpouseNameStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(5, adVarChar, m_szAddress,
        sizeof(m_szAddress), lAddressStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(6, adVarChar, m_szCity,
        sizeof(m_szCity), lCityStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(7, adVarChar,
                               m_szStateOrProvince,
        sizeof(m_szStateOrProvince), lStateOrProvinceStatus,
                               TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(8, adVarChar, m_szPostalCode,
        sizeof(m_szPostalCode), lPostalCodeStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(9, adVarChar, m_szCountry,
        sizeof(m_szCountry), lCountryStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(10, adVarChar, m_szEmailAddress,
        sizeof(m_szEmailAddress), lEmailAddressStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(11, adVarChar, m_szHomePhone,
        sizeof(m_szHomePhone), lHomePhoneStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(12, adVarChar, m_szWorkPhone,
        sizeof(m_szWorkPhone), lWorkPhoneStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(13, adVarChar, m_szWorkExtension,
        sizeof(m_szWorkExtension), lWorkExtensionStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(14, adVarChar, m_szFaxNumber,
        sizeof(m_szFaxNumber), lFaxNumberStatus, TRUE)
    ADO_FIXED_LENGTH_ENTRY(15, adDate, m_dtBirthdate,
        lBirthdateStatus, TRUE)
    ADO_FIXED_LENGTH_ENTRY(16, adBoolean, m_bSendCard,
        lSendCardStatus, TRUE)
    ADO_VARIABLE_LENGTH_ENTRY2(17, adLongVarChar, m_szNotes,
        sizeof(m_szNotes), lNotesStatus, TRUE)
END_ADO_BINDING()
public:
    LONG m_lAddressID;
    ULONG lAddressIDStatus;
    CHAR m_szFirstName[51];
    ULONG lFirstNameStatus;
    CHAR m_szLastName[51];
    ULONG lLastNameStatus;
    CHAR m_szSpouseName[51];
    ULONG lSpouseNameStatus;
    CHAR m_szAddress[256];
    ULONG lAddressStatus;
    CHAR m_szCity[51];
    ULONG lCityStatus;
    CHAR m_szStateOrProvince[51];
    ULONG lStateOrProvinceStatus;
    CHAR m_szPostalCode[21];
    ULONG lPostalCodeStatus;
    CHAR m_szCountry[51];
```

```
65:     ULONG lCountryStatus;
66:     CHAR m_szEmailAddress[51];
67:     ULONG lEmailAddressStatus;
68:     CHAR m_szHomePhone[31];
69:     ULONG lHomePhoneStatus;
70:     CHAR m_szWorkPhone[31];
71:     ULONG lWorkPhoneStatus;
72:     CHAR m_szWorkExtension[21];
73:     ULONG lWorkExtensionStatus;
74:     CHAR m_szFaxNumber[31];
75:     ULONG lFaxNumberStatus;
76:     DATE m_dtBirthdate;
77:     ULONG lBirthdateStatus;
78:     VARIANT_BOOL m_bSendCard;
79:     ULONG lSendCardStatus;
80:     CHAR m_szNotes[65536];
81:     ULONG lNotesStatus;
82: };
```

5. Nachdem Sie diese Klasse erstellt haben, fügen Sie eine Variable in die Dokumentklasse ein, wobei Sie den Variablentyp als CCustomRs, den Namen als m_rsRecSet und den Zugriff als private festlegen.
6. Damit Sie in der Dokumentklasse eine Variable vom Typ CCustomRs verwenden können fügen Sie an den Anfang der Headerdatei AdoDatabaseDoc.h eine #include- Anweisung für die Datei CustomRs.h ein.

Sie müssen außerdem der Ansicht eine Möglichkeit bieten, einen Zeiger auf die Datensatzklasse von der Dokumentklasse zu erhalten. Diese Funktion sollte einen Zeiger auf die Variable der Datensatzklasse zurückgeben. Um die Funktion in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie der Dokumentklasse eine neue Member-Funktion hinzu, wobei Sie den Funktionstyp mit CCustomRs*, den Funktionsnamen mit GetRecSet und den Zugriff als public festlegen.

2. In die Funktion schreiben Sie den Code aus Listing13.2.

Listing 13.2: Die Funktion GetRecSet

```
1: CCustomRs* CAdoDatabaseDoc::GetRecSet(void)
2: {
3:    // Zeiger auf das Datensatzobjekt zurückgeben
4:    return &m_rsRecSet;
5: }
```

Der letzte Teil der Funktionalität, die Sie realisieren müssen, bevor es wirklich in medias res mit der ADO-Programmierung geht, ist die Funktion, die Fehler von ADO und der Datenbank meldet. Diese Funktion zeigt dem Benutzer eine Meldung mit dem Fehlercode und einer Beschreibung an. Um Ihrer Anwendung diese Funktion hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie dazu eine neue Member-Funktion in die Dokumentklasse ein. Legen Sie den Funktionstyp als void und den Funktionsnamen als GenerateError fest und fügen Sie zwei Parameter hinzu. Legen Sie den ersten Parametertyp als HRESULT, den Namen als hr fest und den zweiten Parametertyp als PWSTR und den Namen als pwszDescription. Geben Sie den Funktionszugriff als public an.

Der PWSTR-Datentyp ist ein Zeiger auf einen UNICODE-String.

2. Geben Sie in die Funktion den Code gemäß Listing 13.3 ein.

Listing 13.3: Die Funktion CAdoDatabaseDoc.GenerateError

```
 1: void* CAdoDatabaseDoc::GenerateError(HRESULT hr,
 2:       PWSTR pwszDescription)
 3: {
 4:   CString strError;
 5:   // Fehlermeldung formatieren und anzeigen
 6:   strError.Format("Laufzeitfehler '%d (%x)'", hr, hr);
 7:   strError += "\n\n";
 8:   strError += pwszDescription;
 9:
10:   AfxMessageBox(strError);
11: }
```

Verbinden und Daten abrufen

In der Funktion OnNewDocument in der Dokumentklasse lassen sich alle Aufgaben in Bezug auf die Verbindung zur Datenbank und das Abrufen des Recordsets realisieren. Bevor Sie diese Funktionalität hinzufügen können, müssen Sie weitere Variablen in die Dokumentklasse aufnehmen. Es sind ein Recordset-Objektzeiger, ein IADORecordBinding- Schnittstellenzeiger, verschiedene Stringvariablen zur Aufnahme der Verbindungszeichenfolge für die Datenbank und des auszuführenden SQL-Befehls, der den Recordset füllt, erforderlich. Diese Variablen fügen Sie der Dokumentklasse gemäß Tabelle 13.23 hinzu.

Name	Typ	Zugriff
m_ptrRs	_RecordsetPtr	private
m_piAdoRecordBinding	IADORecordBinding*	private
m_strConnection	CString	private
m_strCmdText	CString	private

Tabelle 13.23: Member-Variablen der Dokumentklasse

Die Funktion OnNewDocument führt verschiedene Schritte aus, um die Verbindung herzustellen und den Recordset abzurufen. Zuerst legt sie die Strings für die Datenbankverbindung und den auszuführenden SQL-Befehl fest. Als Nächstes initialisiert sie die COM-Umgebung und setzt die beiden Zeiger auf den Anfangswert NULL. Mit der Funktion CreateInstance erzeugt OnNewDocument das Recordset-Objekt. Sie öffnet das Recordset-Objekt, wobei sie gleichzeitig die Verbindung zur Datenbank herstellt und den SQL-Befehl ausführt. Sie bindet die Datensatzklasse mithilfe des Schnittstellenzeigers IADORecordBinding an den Recordset. Schließlich weist die Funktion die Ansichtsklasse an, die gebundenen Daten zu aktualisieren und dabei mit einer in Kürze zu erstellenden Funktion den anfänglichen Datensatz für den Benutzer anzuzeigen.

Um in der Klasse CAdoDatabaseDoc Zugriff auf die Ansichtsklasse zu bekommen fügen Sie am Anfang der Datei AdoDatabaseDoc.cpp die in Listing 13.4 fett gedruckte #include- Anweisung ein:

Listing 13.4: Die Ansichtsklasse in der Dokumenrklasse bekannt machen

```
1: // AdoDatabaseDoc.cpp : Implementierung der Klasse
2: // doDatabaseDoc
3:
4:
5: #include "stdafx.h"
6: #include "AdoDatabase.h"
7:
8: #include "AdoDatabaseDoc.h"
9: #include "AdoDatabaseView.h"
```

Um die genannte Funktionalität zu realisieren, bearbeiten Sie die Funktion OnNewDocument in der Dokumentklasse und nehmen den fett gedruckten Code von Listing 13.5 auf.

Listing 13.5: Die Funktion CAdoDatabaseDoc.OnNewDocument

```
 1: BOOL* CAdoDatabaseDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:   // TODO: Hier Code zur Reinitialisierung einfügen
 6:   // (SDI-Dokumente verwenden dieses Dokument)
 7:   // Verbindungszeichenfolge und SQL-Befehl festlegen
 8:   m_strConnection =
 9:            _T("Provider=MSDASQL.1;Data Source=VCPP21DB ");
10:   m_strCmdText = _T("select * from Adressen");
11:
12:   // Zeiger auf Recordset und Bindung initialisieren
13:   m_ptrRs = NULL;
14:   m_piAdoRecordBinding = NULL;
15:   // COM-Umgebung initialisieren
16:   ::CoInitialize(NULL);
17:   try
18:   {
19:     // Datensatzobjekt erzeugen
20:     m_ptrRs.CreateInstance(__uuidof(Recordset));
21:
22:     // Datensatzobjekt öffnen
23:     m_ptrRs->Open((LPCTSTR)m_strCmdText,
24:                   (LPCTSTR)m_strConnection,
25:                  adOpenDynamic,
26:                  adLockOptimistic, adCmdUnknown);
27:     // Zeiger auf Bindungsschnittstelle des Datensatzes holen
28:     if (FAILED(m_ptrRs->QueryInterface(
29:                         __uuidof(IADORecordBinding),
30:                           (LPVOID *)&m_piAdoRecordBinding)))
31:             _com_issue_error(E_NOINTERFACE);
32:     // Datensatzklasse an Recordset binden
33:     m_piAdoRecordBinding->BindToRecordset(&m_rsRecSet);
34:
35:     // Zeiger auf die Ansicht holen
36:     POSITION pos = GetFirstViewPosition();
37:     CAdoDatabaseView* pView =
38:                        (CAdoDatabaseView*)GetNextView(pos);
39:     if (pView)
40:       // Datensatzgruppe mit Formular synchronisieren
41:       pView->RefreshBoundData();
42:   }
43:   // Fehler vorhanden?
44:   catch (_com_error &e)
45:   {
46:     // Fehler anzeigen
47:     GenerateError(e.Error(), e.Description());
48:   }
49:
50:   return TRUE;
51: }
```

In Listing 13.5 haben Sie den MSDASQL-Provider angewiesen, die Access-Datenbank zu verwenden. Dabei handelt es sich um den ODBC-Provider für die OLE-DB-Technologie, auf der ADO aufbaut. Wenn Sie direkt eine SQL Server 2000-Datenbank verwendeten, könnten Sie den SQLOLEDB-Provider nutzen. Andere Provider können aus der mit der Datenbank mitgelieferten Dokumentation ermittelt werden.

Bevor Sie weitergehen, sollten Sie zunächst mit entsprechendem Code sicherstellen, dass beim Schließen der Anwendung alles ordnungsgemäß aufgeräumt wird. Sie müssen den Recordset schließen und den Zeiger auf die Bindungsschnittstelle des Datensatzes freigeben. Weiterhin ist die COM-Umgebung herunterzufahren. Um diese Funktionalität hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie wie an den vergangenen Tagen eine Überschreibungs-Funktion für die Funktion DeleteContents in die Dokumentklasse ein.

2. Nehmen Sie in die Funktion den Code aus Listing 13.6 auf.

Listing 13.6: Die Funktion CAdoDatabaseDoc.DeleteContents

```
 1: void CAdoDatabaseDoc::DeleteContents()
 2: {
 3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein
 4:   // und/oder rufen Sie die Basisklasse auf.
 5:   // Recordset schließen
 6:   if (m_ptrRs)
 7:     m_ptrRs->Close();
 8:   // Haben wir einen gültigen Zeiger auf eine Datensatzbindung?
 9:   if (m_piAdoRecordBinding)
10:     // Zeiger freigeben
11:     m_piAdoRecordBinding->Release();
12:   // Datensatzzeiger auf NULL setzen
13:   m_ptrRs = NULL;
14:
15:   // COM-Umgebung herunterfahren
16:   CoUninitialize();
17:   CDocument::DeleteContents();
18: }
```

Das Formular füllen

Um die Spaltenwerte des Datensatzes für den Benutzer anzuzeigen, fügen Sie eine Funktion hinzu, die die Werte aus der Datensatzklasse in die Ansichtsvariablen kopiert. Diese Funktion muss zunächst einen Zeiger auf die Datensatzklasse aus der Dokumentklasse ermitteln. Als Nächstes prüft die Funktion den Status der einzelnen Felder in der Datensatzklasse, um sich zu vergewissern, dass der Kopiervorgang starten kann. Anschließend wird der Wert kopiert. Nachdem Sie alle Werte kopiert haben, rufen Sie die Funktion UpdateData auf, um die Werte in den Steuerelementen auf dem Formular anzuzeigen. Um diese Funktionalität hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Ansichtsklasse ein. Legen Sie den Funktionstyp als void, den Funktionsnamen mit RefreshBoundData und den Zugriff als public fest.

2. In die Funktion übernehmen Sie den Code aus Listing 13.7.

Listing 13.7: Die Funktion CAdoDatabaseView.RefreshBoundData

```
 1: void CAdoDatabaseView::RefreshBoundData(void)
 2: {
 3:   CCustomRs* pRs;
 4:   // Zeiger auf das Dokumentobjekt holen
 5:   pRs = GetDocument()->GetRecSet();
 6:
 7:   // Ist das Feld OK?
 8:   if (adFldOK == pRs->lAddressIDStatus)
 9:     // Wert kopieren
10:     m_lAddressID = pRs->m_lAddressID;
11:   else
12:     // Andernfalls Wert auf 0 setzen
13:     m_lAddressID = 0;
14:   // Ist das Feld OK?
15:   if (adFldOK == pRs->lFirstNameStatus)
16:     // Wert kopieren
17:     m_strFirstName = pRs->m_szFirstName;
18:   else
19:     // Andernfalls Wert auf 0 setzen
```

```
20:     m_strFirstName = _T("");
21:   // Ist das Feld OK?
22:   if (adFldOK == pRs->lLastNameStatus)
23:     // Wert kopieren
24:     m_strLastName = pRs->m_szLastName;
25:   else
26:     // Andernfalls Wert auf 0 setzen
27:     m_strLastName = _T("");
28:   // Ist das Feld OK?
29:   if (adFldOK == pRs->lSpouseNameStatus)
30:     // Wert kopieren
31:     m_strSpouseName = pRs->m_szSpouseName;
32:   else
33:     // Andernfalls Wert auf 0 setzen
34:     m_strSpouseName = _T("");
35:   // Ist das Feld OK?
36:   if (adFldOK == pRs->lAddressStatus)
37:     // Wert kopieren
38:     m_strAddress = pRs->m_szAddress;
39:   else
40:     // Andernfalls Wert auf 0 setzen
41:     m_strAddress = _T("");
42:   // Ist das Feld OK?
43:   if (adFldOK == pRs->lCityStatus)
44:     // Wert kopieren
45:     m_strCity = pRs->m_szCity;
46:   else
47:     // Andernfalls Wert auf 0 setzen
48:     m_strCity = _T("");
49:   // Ist das Feld OK?
50:   if (adFldOK == pRs->lStateOrProvinceStatus)
51:     // Wert kopieren
52:     m_strStateOrProvince = pRs->m_szStateOrProvince;
53:   else
54:     // Andernfalls Wert auf 0 setzen
55:     m_strStateOrProvince = _T("");
56:   // Ist das Feld OK?
57:   if (adFldOK == pRs->lPostalCodeStatus)
58:     // Wert kopieren
59:     m_strPostalCode = pRs->m_szPostalCode;
60:   else
61:     // Andernfalls Wert auf 0 setzen
62:     m_strPostalCode = _T("");
63:   // Ist das Feld OK?
64:   if (adFldOK == pRs->lCountryStatus)
65:     // Wert kopieren
66:     m_strCountry = pRs->m_szCountry;
67:   else
68:     // Andernfalls Feld auf 0 setzen
69:     m_strCountry = _T("");
70:   // Ist das Feld ok?
71:   if (adFldOK == pRs->lEmailAddressStatus)
72:     // Wert kopieren
73:     m_strEmailAddress = pRs->m_szEmailAddress;
74:   else
75:     // Andernfalls Wert auf 0 setzen
76:     m_strEmailAddress = _T("");
77:   // Ist das Feld OK?
78:   if (adFldOK == pRs->lHomePhoneStatus)
79:     // Wert kopieren
80:     m_strHomePhone = pRs->m_szHomePhone;
81:   else
82:     // Andernfalls Wert auf 0 setzen
83:     m_strHomePhone = _T("");
```

```
 84:   // Ist das Feld OK?
 85:   if (adFldOK == pRs->lWorkPhoneStatus)
 86:     // Wert kopieren
 87:     m_strWorkPhone = pRs->m_szWorkPhone;
 88:   else
 89:     // Andernfalls Wert auf 0 setzen
 90:     m_strWorkPhone = _T("");
 90:   // Ist das Feld OK?
 90:   if (adFldOK == pRs->lWorkExtensionStatus)
 90:     // Wert kopieren
 90:     m_strWorkExtension = pRs->m_szWorkExtension;
 90:   else
 90:     // Andernfalls Wert auf 0 setzen
 90:     m_strWorkExtension = _T("");
 90:   // Ist das Feld OK?
 90:   if (adFldOK == pRs->lFaxNumberStatus)
100:     // Wert kopieren
101:     m_strFaxNumber = pRs->m_szFaxNumber;
102:   else
103:     // Andernfalls Wert auf 0 setzen
104:     m_strFaxNumber = _T("");
105:   // Ist das Feld OK?
106:   if (adFldOK == pRs->lBirthdateStatus)
107:     // Wert kopieren
108:     m_oledtBirthdate = pRs->m_dtBirthdate;
109:   else
110:     // Andernfalls Wert auf 0 setzen
111:     m_oledtBirthdate = 0L;
112:   // Ist das Feld OK?
113:   if (adFldOK == pRs->lSendCardStatus)
114:     // Wert kopieren
115:     m_bSendCard = VARIANT_FALSE ==
116:                   pRs->m_bSendCard ? FALSE : TRUE;
117:   else
118:     // Andernfalls Wert auf 0 setzen
119:     m_bSendCard = FALSE;
120:   // Ist das Feld OK?
121:   if (adFldOK == pRs->lNotesStatus)
122:     // Wert kopieren
123:     m_strNotes = pRs->m_szNotes;
124:   else
125:     // Andernfalls Wert auf 0 setzen
126:     m_strNotes = _T("");
127:
128:   // Daten mit Steuerelementen synchronisieren
129:   UpdateData(FALSE);
120: }
```

Aktualisierungen speichern

Wenn Sie Änderungen zurück in den Recordset kopieren müssen, verläuft das Kopieren der Daten aus den Steuerelementen auf dem Formular in die Variablen der Datensatzklasse in umgekehrter Richtung. Dabei können Sie alle Werte kopieren, ob diese sich geändert haben oder nicht, oder Sie vergleichen die beiden Werte auf Änderungen, um zu entscheiden, welche zurückkopiert werden müssen. Die betreffende Funktion rufen Sie auf, bevor der Benutzer zu anderen Datensätzen im Recordset navigieren kann, damit alle vom Benutzer vorgenommenen Änderungen in der Datenbank gespeichert werden. Um diese Funktionalität in Ihre Anwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Ansichtsklasse ein. Legen Sie den Funktionstyp als void, den Funktionsnamen mit UpdateBoundData und den Zugriff als private fest.

2. Nehmen Sie den Code aus Listing 13.8 in die Funktion auf.

Listing 13.8: Die Funktion CAdoDatabaseView.UpdateBoundData

```
 1: void CAdoDatabaseView::UpdateBoundData(void)
 2: {
 3:   CCustomRs* pRs;
 4:   // Zeiger auf das Dokument holen
 5:   pRs = GetDocument()->GetRecSet();
 6:
 7:   // Steuerelemente mit den Variablen synchronisieren
 8:   UpdateData(TRUE);
 9:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
10:   if (m_lAddressID != pRs->m_lAddressID)
11:     pRs->m_lAddressID = m_lAddressID;
12:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
13:   if (m_strFirstName != pRs->m_szFirstName)
14:     strcpy(pRs->m_szFirstName, (LPCTSTR)m_strFirstName);
15:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
16:   if (m_strLastName != pRs->m_szLastName)
17:     strcpy(pRs->m_szLastName, (LPCTSTR)m_strLastName);
18:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
19:   if (m_strSpouseName != pRs->m_szSpouseName)
20:     strcpy(pRs->m_szSpouseName, (LPCTSTR)m_strSpouseName);
21:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
22:   if (m_strAddress != pRs->m_szAddress)
23:     strcpy(pRs->m_szAddress, (LPCTSTR)m_strAddress);
24:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
25:   if (m_strCity != pRs->m_szCity)
26:     strcpy(pRs->m_szCity, (LPCTSTR)m_strCity);
27:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
28:   if (m_strStateOrProvince != pRs->m_szStateOrProvince)
29:     strcpy(pRs->m_szStateOrProvince,
30:            (LPCTSTR)m_strStateOrProvince);
31:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
32:   if (m_strPostalCode != pRs->m_szPostalCode)
33:     strcpy(pRs->m_szPostalCode, (LPCTSTR)m_strPostalCode);
34:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
35:   if (m_strCountry != pRs->m_szCountry)
36:     strcpy(pRs->m_szCountry, (LPCTSTR)m_strCountry);
37:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
38:   if (m_strEmailAddress != pRs->m_szEmailAddress)
39:     strcpy(pRs->m_szEmailAddress, (LPCTSTR)m_strEmailAddress);
40:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
41:   if (m_strHomePhone != pRs->m_szHomePhone)
42:     strcpy(pRs->m_szHomePhone, (LPCTSTR)m_strHomePhone);
43:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
44:   if (m_strWorkPhone != pRs->m_szWorkPhone)
45:     strcpy(pRs->m_szWorkPhone, (LPCTSTR)m_strWorkPhone);
46:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
47:   if (m_strWorkExtension != pRs->m_szWorkExtension)
48:     strcpy(pRs->m_szWorkExtension, (LPCTSTR)m_strWorkExtension);
49:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
50:   if (m_strFaxNumber != pRs->m_szFaxNumber)
51:     strcpy(pRs->m_szFaxNumber, (LPCTSTR)m_strFaxNumber);
52:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
53:   if (((DATE)m_oledtBirthdate) != pRs->m_dtBirthdate)
54:     pRs->m_dtBirthdate = (DATE)m_oledtBirthdate;
55:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
56:   if (m_bSendCard == TRUE)
57:     pRs->m_bSendCard = VARIANT_TRUE;
58:   else
59:     pRs->m_bSendCard = VARIANT_FALSE;
60:   // Hat sich das Feld geändert? Wenn ja, Wert zurückkopieren
61:   if (m_strNotes != pRs->m_szNotes)
62:     strcpy(pRs->m_szNotes, (LPCTSTR)m_strNotes);
```

```
63: }
```

Dieser Code berücksichtigt keine Strings, die die in der Datenbank definierten Feldlängen überschreiten. Dazu müssten Sie Code einfügen, der die Länge überprüft, um Strings abzuschneiden, die die Länge des Datenbankfelds überschreiten, in dem sie gespeichert werden sollen. Sie können auch die Funktion strncpy verwenden, der ein dritter Parameter mit der Anzahl der vom zweiten in den ersten String zu kopierenden Zeichen übergeben wird.

Durch den Recordset navigieren

Die Navigation durch den Recordset unterstützen Sie in Ihrer Anwendung mit einer Reihe von Menüeinträgen für die vier grundlegenden Navigationsbefehle: Erster, Vorheriger, Nächster und Letzter. Da das Recordset-Objekt und die Schnittstellenzeiger zur Datensatzbindung im Dokumentobjekt angelegt sind, muss man die Nachrichten für diese Menüs an die Dokumentklasse weiterreichen, um den aktuellen Datensatz zu aktualisieren und dann zum ausgewählten Datensatz weiterzuschalten. Allerdings muss die Ansichtsklasse die Nachrichten zuerst empfangen, da sie die geänderten Werte aus den Steuerelementen auf dem Formular zurückkopieren muss, bevor der Recordset aktualisiert wird. Ist die Navigation beendet, muss die Ansicht wiederum das Formular mit den neuen Spaltenwerten des Datensatzes in Übereinstimmung bringen. Wenn man die Sequenz berücksichtigt, entsprechend der die Nachrichten zu übergeben sind, ist es am sinnvollsten, die Behandlungsroutine in der Ansichtsklasse unterzubringen und von hier aus die Behandlungsroutine für die Dokumentklasse aufzurufen.

Um die beschriebene Funktionalität in der Anwendung zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie vier Menübefehle für die Navigation durch den Recordset (Erster, Vorheriger, Nächster und Letzter) und die entsprechenden Symbolleisten- Schaltflächen hinzu.

2. Erstellen Sie in der Ansichtsklasse Behandlungsroutinen für die Nachrichten der vier Menübefehle.
3. In die Behandlungsroutine für den Menübefehl Erster nehmen Sie den Code aus Listing 13.9 auf.

Listing 13.9: Die Funktion CAdoDatabaseView.OnDatenErster

```
 1: void CAdoDatabaseView::OnDatenErster()
 2: {
 3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
 4:    // Aktuellen Datensatz aktualisieren
 5:    UpdateBoundData();
 6:    // Zum ersten Datensatz gehen
 7:    GetDocument()->MoveFirst();
 8:    // Formular mit den Daten des neuen Datensatzes aktualisieren
 9:    RefreshBoundData();
10: }
```

Als Nächstes fügen Sie die Funktion MoveFirst (zum ersten Datensatz gehen) in die Dokumentklasse ein und realisieren die eigentliche Funktionalität für den Recordset für diese Funktion:

1. Nehmen Sie dazu eine Member-Funktion in die Dokumentklasse Ihrer Anwendung auf. Legen Sie den Funktionstyp als void, den Namen mit MoveFirst und den Zugriff als public fest.

2. In die Funktion schreiben Sie den Code aus Listing 13.10.

Listing 13.10: Die Funktion CAdoDatabaseView.MoveFirst

```
 1: void CAdoDatabaseDoc::MoveFirst(void)
 2: {
```

```
 3:    try
 4:    {
 5:      // Aktuellen Datensatz aktualisieren
 6:      m_piAdoRecordBinding->Update(&m_rsRecSet);
 7:      // Zum ersten Datensatz gehen
 8:      m_ptrRs->MoveFirst();
 9:    }
10:    // Fehler vorhanden?
11:    catch (_com_error &e)
12:    {
13:      // Fehlermeldung erzeugen
14:      GenerateError(e.Error(), e.Description());
15:    }
16: }
```

3. Für die ADO-Funktionen MovePrevious (zum vorherigen Datensatz), MoveNext (zum nächsten) und MoveLast (zum letzten) fügen Sie entsprechende Funktionspaare der Ansichts- und Dokumentklasse hinzu.

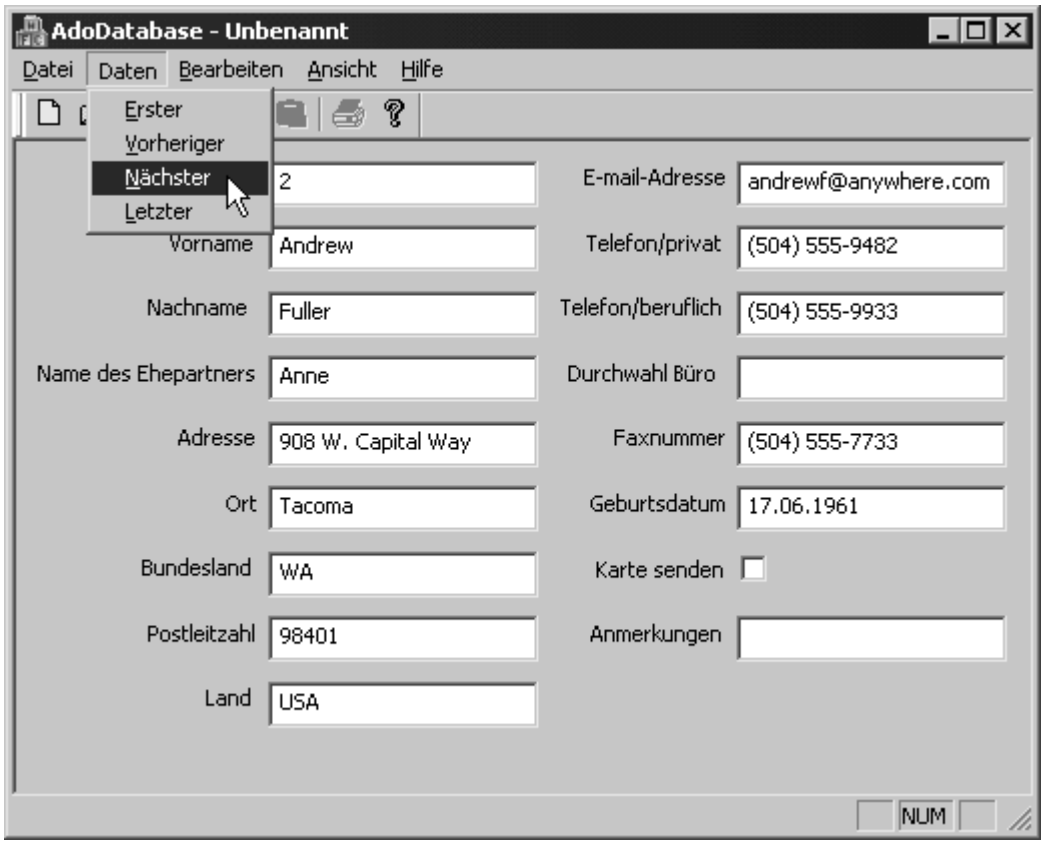


Abbildung 13.5: Die laufende Anwendung

Wenn Sie diese Schritte abgeschlossen haben, können Sie Ihre Anwendung kompilieren und ausführen. Mit der Anwendung ist es jetzt möglich, die Datenbanktabelle Adressen zu öffnen und die einzelnen Datensätze anzuzeigen, zu bearbeiten und zu aktualisieren (siehe Abbildung 13.5).

Neue Datensätze hinzufügen

In der momentanen Entwicklungsphase der Anwendung können Sie Datensätze aus der Datenbanktabelle abrufen und in der Ergebnismenge - dem Recordset - navigieren. Es wäre wünschenswert, wenn man auch neue Datensätze in die Tabelle aufnehmen könnte. Diese Funktionalität lässt sich in genau der gleichen Weise umsetzen, wie Sie es für die Navigation vorgenommen haben. Zu diesem Zweck fügen Sie einen Menübefehl hinzu, lösen über diesen Menübefehl eine Behandlungsroutine in der Ansichtsklasse aus, schreiben die aktuellen Werte des Datensatzes zurück in den Recordset, rufen eine Funktion in der Dokumentklasse auf und bringen den aktuellen Datensatz aus dem Recordset auf den neuesten Stand.

In Bezug auf das Menü und die Ansichtsklasse ändert sich gegenüber den Menübefehlen und Funktionen zur Navigation lediglich die ID des Menüs und der Name der aufgerufenen Funktion, praktisch genau wie bei den verschiedenen Navigationsfunktionen. In der Funktion der Dokumentklasse liegen die eigentlichen funktionellen Unterschiede.

Damit sich ein neuer Datensatz hinzufügen lässt, stellen Sie sicher, dass nach dem Aktualisieren des momentanen Datensatzes in der Dokumentklasse das Hinzufügen eines neuen Datensatzes möglich ist. Wenn diese Möglichkeit besteht, bauen Sie einen leeren Datensatz zusammen und fügen ihn in den Recordset ein. Nachdem Sie den leeren Datensatz aufgenommen haben, navigieren Sie zum letzten Datensatz in der Ergebnismenge, da dieser den neuen Datensatz repräsentiert. Jetzt können Sie die Funktion verlassen und der Ansichtsklasse die Aufgabe übertragen, das Formular mit den Datenwerten aus dem neuen - leeren - Datensatz zu aktualisieren.

Um diese Funktionalität in der Anwendung zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie einen neuen Menübefehl für das Hinzufügen eines neuen Datensatzes ein.

2. In die Ansichtsklasse nehmen Sie eine Behandlungsroutine für den neuen Menübefehl auf. In diese Funktion schreiben Sie den gleichen Code wie für die Navigationsfunktionen, rufen aber die Funktion AddNew in der Dokumentklasse auf.
3. Erstellen Sie die Funktion AddNew in der Dokumentklasse. Nehmen Sie dazu eine neue Member-Funktion in die Dokumentklasse auf. Legen Sie den Typ als void, den Namen mit AddNew und den Zugriff als public fest.
4. In die Funktion schreiben Sie den Code aus Listing 13.11.

Listing 13.11: Die Funktion CAdoDatabaseDoc.AddNew

```
 1: void CAdoDatabaseDoc::AddNew(void)
 2: {
 3:   try
 4:   {
 5:     // Aktuellen Datensatz aktualisieren
 6:     m_piAdoRecordBinding->Update(&m_rsRecSet);
 7:     // Können wir einen neuen Datensatz hinzufügen?
 8:     if (m_ptrRs->Supports(adAddNew))
 9:     {
10:       // Leeren Datensatz erzeugen
11:       CreateBlankRecord();
12:       // Leeren Datensatz hinzufügen
13:       m_piAdoRecordBinding->AddNew(&m_rsRecSet);
14:       // Zum letzten Datensatz gehen
15:       m_ptrRs->MoveLast();
16:     }
17:   }
18:   // Fehler vorhanden?
19:   catch (_com_error &e)
20:   {
21:     // Fehlermeldung erzeugen
22:     GenerateError(e.Error(), e.Description());
23:   }
24: }
```

Als Nächstes fügen Sie die Funktion hinzu, die den leeren Datensatz erzeugt. In dieser Funktion setzen Sie alle Feldvariablen in der Datensatzklasse auf einen fast leeren String. Um diese Funktion in die Klasse aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein. Legen Sie den Typ als void, den Namen mit CreateBlankRecord und den Zugriff als private fest.

2. Übernehmen Sie in diese Funktion den Code aus Listing 13.12.

Listing 13.12: Die Funktion CAdoDatabaseDoc.CreateBlankRecord

```
 1: void CAdoDatabaseDoc::CreateBlankRecord(void)
 2: {
 3:   // Alle Werte im Datensatzobjekt setzen
 4:   m_rsRecSet.m_lAddressID = 0;
 5:   strcpy(m_rsRecSet.m_szFirstName, " ");
 6:   strcpy(m_rsRecSet.m_szLastName, " ");
 7:   strcpy(m_rsRecSet.m_szSpouseName, " ");
 8:   strcpy(m_rsRecSet.m_szAddress, " ");
 9:   strcpy(m_rsRecSet.m_szCity, " ");
10:   strcpy(m_rsRecSet.m_szStateOrProvince, " ");
11:   strcpy(m_rsRecSet.m_szPostalCode, " ");
12:   strcpy(m_rsRecSet.m_szCountry, " ");
13:   strcpy(m_rsRecSet.m_szEmailAddress, " ");
14:   strcpy(m_rsRecSet.m_szHomePhone, " ");
15:   strcpy(m_rsRecSet.m_szWorkPhone, " ");
16:   strcpy(m_rsRecSet.m_szWorkExtension, " ");
17:   strcpy(m_rsRecSet.m_szFaxNumber, " ");
18:   m_rsRecSet.m_dtBirthdate = NULL;
19:   m_rsRecSet.m_bSendCard = VARIANT_FALSE;
20:   strcpy(m_rsRecSet.m_szNotes, " ");
21: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie neue Datensätze in die Datenbanktabelle einfügen und bearbeiten.

Datensätze löschen

Als krönenden Abschluss der heutigen Beispielanwendung realisieren wir das Löschen des aktuellen Datensatzes aus dem Recordset. Diese Funktion entspricht weitgehend den Funktionen zur Navigation und zum Hinzufügen. Die Anwendung erhält einen neuen Menübefehl, der eine Behandlungsroutine in der Ansichtsklasse auslöst. Die Funktion in der Ansichtsklasse realisiert die gleichen Aufgaben wie die vorherigen Funktionen: sie aktualisiert den aktuellen Datensatz, ruft die korrespondierende Funktion in der Dokumentklasse auf und bringt dann den aktuellen Datensatz im Formular auf den neuesten Stand.

Die Funktion der Dokumentklasse verfolgt beim Löschen des Datensatzes fast den gleichen Weg wie beim Hinzufügen. Sie aktualisiert den momentanen Datensatz, prüft, ob der zum Löschen freigegeben ist, holt vom Benutzer eine Bestätigung ein, ob der Datensatz wirklich gelöscht werden soll, ruft dann die Funktion Delete auf und navigiert zu einem anderen Datensatz im Recordset. Um diese Funktionalität in Ihrer Anwendung zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie einen neuen Menübefehl für die Löschen-Funktion hinzu und verbinden Sie ihn mit einer Behandlungsfunktion in der Ansichtsklasse.

2. In die Funktion schreiben Sie den gleichen Code wie bei den Funktionen zum Navigieren und Hinzufügen, rufen aber die Funktion Delete in der Dokumentklasse auf.
3. In die Dokumentklasse nehmen Sie deshalb eine neue Member-Funktion auf, für die Sie den Typ als void, den Namen mit Delete und den Zugriff als public festlegen.
4. In die Funktion schreiben Sie den Code aus Listing 13.13.

Listing 13.13: Die Funktion CAdoDatabaseDoc.Delete

```
 1: void CAdoDatabaseDoc::Delete(void)
 2: {
 3:   try
 4:   {
 5:     // Aktuellen Datensatz aktualisieren
 6:     m_piAdoRecordBinding->Update(&m_rsRecSet);
 7:     // Können wir Datensätze löschen?
 8:     if (m_ptrRs->Supports(adDelete))
 9:     {
10:       // Sicherstellen, dass der Benutzer diesen Datensatz
11:       // löschen will
```

```
12:       if (AfxMessageBox(
13:           "Wollen Sei diesen Datensatz wirklich löschen?",
14:           MB_YESNO | MB_ICONQUESTION) == IDYES)
15:       {
16:          // Datensatz löschen
17:          m_ptrRs->Delete(adAffectCurrent);
18:          // Zum vorherigen Datensatz gehen
19:          m_ptrRs->MovePrevious();
20:       }
21:     }
22:   }
23:   // Fehler vorhanden?
24:   catch (_com_error &e)
25:   {
26:     // Fehlermeldung erzeugen
27:     GenerateError(e.Error(), e.Description());
28:   }
29: }
```

Jetzt können Sie Ihre Anwendung kompilieren und ausführen. Das Löschen beliebiger Datensätze aus dem Recordset sollte nun funktionieren. Wenn Sie alle Datensätze aus dem Recordset löschen, wird bei der Bewegung zum vorherigen Datensatz ein Fehler erzeugt. Sie müssen vor dem Aufruf dieser Methode überprüfen, ob im Recordset noch Datensätze vorhanden sind.

13.5 Zusammenfassung

Heute haben Sie die neueste Technologie für den Datenbankzugriff von Microsoft kennen gelernt: ActiveX Data Objects. Sie haben die DLL importiert, die eine umfangreiche Palette von Funktionen für den Datenzugriff bereitstellt, und haben erfahren, wie man diese Funktionalität in eigenen Anwendungen verwendet und steuert. Mit der heutigen Beispielanwendung ist es möglich, Datensätze abzurufen, die Datensätze im Recordset (der Ergebnismenge) zu manipulieren und die Änderungen in die Datenbank zu übernehmen. Für den Zugriff und die Aktualisierung der Datenwerte in einem Datensatz des Recordsets haben Sie zwei verschiedene Wege kennen gelernt. Vor allem wurde gezeigt, wie man mit etwas mehr Aufwand zu Beginn einen beträchtlichen Teil der Arbeit bei der Anwendungsentwicklung auf lange Sicht einspart.

13.6 Workshop

Fragen und Antworten

Frage:
Visual C++ bietet für ADO keine Unterstützung in Form von Assistenten. Warum sollte ich dennoch auf ADO zurückgreifen?

Antwort:
Mit ADO gibt Microsoft den Trend für die Technologien des Datenbankzugriffs vor. Zukünftig wird diese Technologie in allen Programmiersprachen und Anwendungen zu finden sein. Vielleicht werden letztendlich auch Assistenten integriert, die einen Teil der Datenbank-Zugriffsfunktionalität für Sie realisieren.

Frage:
Wenn ADO den Zugriff auf meine Datenbank über ODBC realisiert, warum geht man dann nicht gleich den direkten Weg über die ODBC-Schnittstelle, um auf die Datenbank zuzugreifen?

Antwort:
ADO kann ODBC nutzen, um auf diejenigen Datenbanken zuzugreifen, die keine native OLE DB-Schnittstelle haben. Wenn Sie mit SQL Server oder Oracle arbeiten, sind OLE DB-Schnittstellen vorhanden. In diesen Fällen greift ADO nicht über ODBC auf die Datenbank zu und bringt für die Anwendung ein besseres Leistungsverhalten.

Quiz

1. Was bezeichnet die Abkürzung ADO?

2. Was verwendet ADO für den Datenbankzugriff?
3. Welche Objekte sind in ADO vorhanden?
4. Wie initialisiert man die COM-Umgebung?
5. Wie verbindet man ein Connection-Objekt mit einem Command-Objekt?
6. Wie verbindet man ein Recordset-Objekt mit einem Command-Objekt und füllt den Recordset mit Daten?

Übung

Aktivieren und deaktivieren Sie die Menübefehle und Symbolleisten-Schaltflächen zur Navigation, je nachdem, ob der Recordset am Beginn der Datei (BOF) oder am Ende der Datei (EOF, umbenannt zu EndOfFile) steht.

1

Zum Beispiel »Jetzt lerne ich ADO.NET« von Ralf Westphal, ISBN: 3-8272-6229-1, erscheint Oktober 2002.

Tag 14

DLLs

Manchmal ist eine Gruppe von Funktionen zu erstellen, die in einer Anwendung, an der ein anderer Programmierer arbeitet, zum Einsatz kommen. Es kann sein, dass die Funktionen in einer ganzen Reihe von Anwendungen benötigt werden. Weiterhin ist es möglich, dass Sie eine bestimmte Funktionsgruppe aus organisatorischen Gründen von der übrigen Anwendung abtrennen möchten. Diese funktionellen Einheiten können Sie dann separat entwickeln und den Code an die betreffenden Mitarbeiter weitergeben. Das hat allerdings den Nachteil, dass Änderungen, die Sie an der Funktionsgruppe vornehmen, auch noch in den bereits weitergegebenen Code eingebunden werden müssen. Es wäre praktischer, wenn man eine kompilierte Version der Funktionsgruppe an den anderen Programmierer weitergeben könnte, sodass Sie bei einer Aktualisierung auf Ihrer Seite lediglich die neu kompilierte Datei aushändigen müssten. Die neue Datei könnte einfach die vorherige Version ersetzen, ohne dass der andere Programmierer seinen bisher erstellten Code ändern müsste (so jedenfalls die schöne Theorie ...).

Oftmals weist eine Anwendungsfamilie gleiche Funktionen auf. Wenn Sie diese gemeinsam genutzte Funktionalität in DLLs statt in Bibliotheksmodulen unterbringen, können alle Anwendungen auf die gleiche Funktionalität zugreifen, wozu nur ein Exemplar des Funktionspakets in Form von DLLs zu vertreiben ist, statt die gleiche Funktionalität mehrfach für die einzelnen Anwendungen bereitzustellen. Dieses Verfahren spart Platz auf allen Systemen, auf denen die Anwendungen installiert sind.

Die heutige Lektion zeigt, ...

- wie Sie verschiedene Typen von DLLs mit Visual C++ erstellen können und wie Sie den am besten geeigneten Typ auswählen,
- wie man zwei dieser DLL-Typen erstellt und wie sich die Lösungen bei diesen Typen unterscheiden,
- wie man die Funktionalität für beide DLL-Typen in einer Visual C++-Anwendung einsetzt,
- wie man bestimmt, wann eine Anwendung bei Änderungen einer DLL, auf die eine Anwendung zurückgreift, neu zu linken ist.

14.1 Klassen entwerfen

In den vergangenen Tagen haben Sie bereits eigene Klassen entworfen und erstellt, sodass dieses Thema nicht grundsätzlich neu für Sie ist. Warum haben Sie diese Klassen erzeugt? Die neu erzeugten Klassen haben alle eine Gruppe von Funktionalität verkapselt, die als selbstständige Einheit agiert. Diese Einheiten bestehen sowohl aus Daten als auch als Funktionen, die in ihrer Gesamtheit das Objekt definieren.

Kapselung

Der objektorientierte Software-Entwurf verfolgt das Konzept, nach dem alles in der uns umgebenden Welt aufgebaut ist. Betrachten Sie zum Beispiel ein Auto, das aus einer Sammlung von Objekten besteht: Motor, Karosserie, Radaufhängung usw. Jedes dieser Objekte besteht aus einem Bündel anderer Objekte. Beispielsweise enthält der Motor den Vergaser und die Einspritzventile, die Verbrennungskammer und die Kolben usw. Auch hier bestehen die einzelnen Objekte wiederum aus noch mehr Objekten.

Alle diese Objekte haben eine Funktion, die sie ausführen. Jedes Objekt weiß, wie die eigenen Funktionen auszuführen sind, ohne oder nur mit geringen Kenntnissen, wie die anderen Objekte ihre Funktionen wahrnehmen. Jedes Objekt weiß, wie es mit den anderen Objekten in Wechselwirkung treten kann und wie es mit den anderen Objekten verbunden ist, mehr ist aber nicht über die anderen Objekte bekannt. Wie jedes dieser Objekte intern arbeitet, bleibt vor den anderen Objekten verborgen. Die Bremsen im Auto wissen zum Beispiel nichts darüber, wie das Getriebe funktioniert, bei einem Automatikgetriebe wissen aber die Bremsen, wie sie dem Getriebe mitteilen, dass sie aktiv sind, und das Getriebe entscheidet, wie auf diese Informationen zu reagieren ist. (Eigentlich verlangsamen die Bremsen nur das Auto; das Getriebe entscheidet dann, wie es auf den neuen Zustand der Fahrzeugbewegung reagiert.)

Auf die gleiche Weise gehen Sie den Entwurf neuer Klassen für Ihre Anwendungen an. Die übrigen Anwendungsobjekte brauchen nicht zu wissen, wie Ihre Objekte arbeiten, sie müssen nur wissen, wie sie mit Ihren Objekten in Wechselwirkung treten. Diese so genannte Kapselung gehört zu den Grundprinzipien der objektorientierten Software.

Vererbung

Ein weiteres Schlüsselprinzip des objektorientierten Software-Entwurfs ist die Vererbung. Ein Objekt kann von einem anderen Objekt vererbt sein. Das abgeleitete Objekt erbt die gesamte Funktionalität des Basisobjekts. Damit lässt sich das abgeleitete Objekt in Form von Änderungen gegenüber dem Basisobjekt definieren.

Sehen wir uns dieses Konzept am Beispiel eines Thermostaten an. Nehmen wir einen einfachen Thermostaten an, den Sie in nahezu jeder Einstellung verwenden können. Es lässt sich die Temperatur vorgeben, die der Thermostat regeln soll. Zu diesem Zweck schaltet er die Heizung oder Klimaanlage ein. Jetzt wollen wir einen Thermostat für einen Gefrierschrank erstellen. Man könnte nun einen eigens entwickelten Thermostaten von Grund auf neu aufbauen - oder man nimmt den vorhandenen Thermostaten und legt fest, wie sich die Gefrierschrankversion vom Original unterscheidet. Zu diesen Unterschieden kann gehören, dass der Thermostat nur noch die Klimaanlage einschalten muss und die Hei- zung überhaupt nicht mehr. Vielleicht möchten Sie auch den Temperaturbereich streng eingrenzen, auf den sich der Thermostat einstellen lässt, beispielsweise auf Temperaturen unterhalb 0° Celsius (aus Gründen der Vereinfachung werden Geräte mit Aubtauautomatik in diesem Buch nicht weiter beachtet). Wenn Sie analog dazu einen Thermostat für eine Büroheizung brauchen, grenzen Sie wahrscheinlich den Temperaturbereich auf die üblichen Raumtemperaturen ein und erlauben keine Werte für extreme Kälte oder Hitze.

Das gleiche Prinzip verfolgen Sie, wenn Sie eigene Klassen per Vererbung erstellen. Nach Möglichkeit sollten Sie mit einer vorhandenen C++-Klasse, die bereits die erforderliche Basisfunktionalität mitbringt, beginnen. Dann programmieren Sie, wie sich Ihre Klasse von der Basisklasse, von der Sie erben, unterscheidet. Dabei können Sie neue Datenelemente hinzufügen, die vorhandene Funktionalität erweitern oder vorhandene Funktionen überschreiben, wie es gerade in Ihr Konzept passt.

Klassentypen in Visual C++

Wenn Sie eine neue Klasse erzeugen, stehen Ihnen in den meisten Anwendungsprojekten verschiedene Optionen zum Typ der zu erzeugenden Klasse zur Verfügung. Zu diesen Optionen gehören:

- Allgemeine Klasse
- MFC-Klasse
- ATL-Klasse

Von welchem Typ Sie Ihre Klasse erzeugen, hängt von Ihren Anforderungen und den Aufgaben der Klasse ab. Weiterhin ist zu beachten, ob die Klasse von einer MFC-Klasse abzuleiten ist.

Allgemeine Klassen

Aus einer allgemeinen (oder generischen) Klasse erstellen Sie eine Klasse, die von einer bereits erstellten Klasse abgeleitet ist. Mit diesem Klassentyp erzeugen Sie in erster Linie Klassen, die von keiner MFC-Klasse abgeleitet sind (obwohl Sie bereits gesehen haben, wo man mit diesem Typ Klassen erstellt, die auf MFC-Klassen basieren). Wenn Sie eine speziellere Version der Klasse CLine erstellen wollen, zum Beispiel eine Klasse CRedLine, die nur in Rot zeichnet, erzeugen Sie eine allgemeine Klasse, weil sie von einer anderen

Klasse, die Sie erzeugt haben, abgeleitet ist.

MFC-Klassen

Wenn Sie eine wiederverwendbare Klasse erstellen möchten, die auf einer vorhandenen MFC-Klasse aufbaut, wie etwa ein Eingabefeld, das automatisch Zahlen als Währung formatiert, erstellen Sie eine MFC-Klasse. Der Typ der MFC-Klasse dient der Erstellung neuer Klassen, die von existierenden MFC-Klassen abgeleitet sind.

> *Vor der aktuellen Version von Visual C++ ließen sich über den MFC-Klassen-Assistenten nur bestimmte MFC-Klassen als Basisklassen verwenden. Wenn Sie von einer nicht im Assistenten enthaltenen MFC-Klasse erben wollten, mussten Sie eine allgemeine Klasse erstellen. In der neuen Ausgabe von Visual C++ wurde das bedeutend erweitert, sodass Sie viel mehr MFC-Klassen als Basisklassen verwenden können. Mit der neuen Version von Visual C++ können Sie Ihre CLine-Klasse als MFC-Klasse mit CObject als Basisklasse erstellen. Bisher waren Sie in diesem Fall gezwungen, eine allgemeine Klasse zu erstellen.*

ATL-Klassen

Der Typ der ATL-Klasse (Active Template Library) wird für die Erstellung von Komponenten und Diensten verwendet. Es handelt sich dabei um eine Alternative zur MFC-Klassenbibliothek, die zur Erstellung von Komponenten in Anwendungen, nicht für vollständige benutzerorientierte Anwendungen verwendet wird (man kann damit Dienste erstellen, die für den Benutzer nicht sichtbar sind). Was ATL ist und wie man damit Komponenten und Dienste erstellt, lernen Sie an Tag 20, »ATL-Komponenten«.

14.2 Warum DLLs erstellen?

Wenn Sie neue Klassen für Ihre Anwendung erstellen, können diese Klassen auch in anderen Anwendungen nützlich sein. Oftmals lassen sich die zu erzeugenden Klassen mit etwas Nachdenken und geringem Mehraufwand so flexibel gestalten, dass man sie in anderen Anwendungen einsetzen kann. In diesem Fall brauchen Sie eine Möglichkeit, um die Klassen für andere Anwendungen zusammenzupacken, ohne dabei den Quellcode aus der Hand geben zu müssen. Für einen solchen Fall sind DLLs prädestiniert. Hiermit können Sie Ihre Klassen und Module in eine kompilierte Objektcode-Bibliothek kompilieren, die sich mit jeder anderen Visual C++-Anwendung linken lässt.

Microsoft hat bereits in den frühen Tagen von Windows so genannte dynamische Linkbibliotheken (DLLs - Dynamic Link Libraries) eingeführt. DLLs stimmen mit Bibliotheksmodulen darin überein, dass sie beide eine Funktionsgruppe verpacken. In der Bindung der Bibliothek zur Anwendung unterscheiden sich beide Typen. Wenn Sie eine Anwendung mit einem Bibliotheksmodul (LIB) erstellen, wird die Funktionalität der Bibliothek während des Prozesses des Kompilierens und der Erstellung zur Anwendung gelinkt. Die in der Bibliotheksdatei enthaltene Funktionalität wird Teil der ausführbaren Anwendungsdatei. Bei einer DLL dagegen bindet die Anwendung die in der Bibliotheksdatei enthaltene Funktionalität zur Laufzeit der Anwendung ein. Die Bibliotheksdatei bleibt eine separate Datei, die von der Anwendung referenziert und aufgerufen wird.

Es gibt mehrere Gründe, warum man DLLs erstellen sollte:

- Die Größe der ausführbaren Anwendungsdateien lässt sich verringern, indem man die von mehreren Anwendungen gemeinsam genutzte Funktionalität in DLLs unterbringt.
- Die Funktionalität in den DLLs kann man aktualisieren und modifizieren, ohne die ausführbare Anwendung auf den neuesten Stand bringen zu müssen (vorausgesetzt, dass die von der DLL exportierte Schnittstelle unverändert bleibt).
- Wenn man die richtige Art von DLL erstellt, kann man DLLs mit nahezu jeder anderen Programmiersprache unter Windows verwenden, wodurch die von Ihnen erstellte Funktionalität einer breiteren Masse von Programmierern zur Verfügung steht und nicht nur den Anhängern von Visual C++.

In all unseren bislang erstellten Applikationen wurden die MFC-Klassen aus verschiedenen DLLs geladen (z.B. mfc70.dll und msvcr70.dll).

DLLs erstellen und einsetzen

DLLs sind Bibliotheksdateien mit kompiliertem Code, auf den andere Anwendungen zurückgreifen können. Die DLLs legen bestimmte Funktionen und Klassen für diese Anwendungen frei, indem sie die Funktion exportieren. Eine exportierte Funktion wird in eine Tabelle eingetragen, die zur DLL gehört. Diese Tabelle enthält den Standort aller exportierten Funktionen, die in der DLL vorhanden sind. Alle nicht exportierten Funktionen erscheinen auch nicht in der Tabelle und eine Anwendung außerhalb der DLL kann diese weder sehen noch aufrufen.

Eine Anwendung kann die Funktionen in der DLL nach zwei Verfahren aufrufen. Bei der komplizierteren Methode sucht die Anwendung nach dem Standort der gewünschten Funktion in der DLL und holt einen Zeiger auf diese Funktion. Über den Zeiger findet dann der Aufruf der Funktion statt. Das hat zur Folge, daß neuere Versionen der DLL weitere Funktionen zu dieser Tabelle hinzufügen können, ohne daß alte Applikationen neu erstellt werden müßten.

Der andere und wesentlich einfachere Weg (und der einzige Weg, den Sie in allen Beispielen in diesem Buch gehen) ist, die Anwendung mit der LIB-Datei, die mit der DLL erzeugt wird, zu verknüpfen. Diese LIB-Datei behandelt der Linker als normale Bibliotheksdatei. Allerdings enthält diese LIB-Datei Funktionsrümpfe (Stub) für alle exportierten Funktionen der DLL. Diese Pseudofunktionen haben den gleichen Namen und die gleiche Argumentliste wie die echten Funktionen, enthalten aber im Inneren nur einen winzigen Code-Abschnitt, der die echte Funktion in der DLL aufruft und alle an die Pseudofunktion übergebenen Parameter an die echte Funktion weiterreicht. Damit können Sie die Funktionen in der DLL genauso behandeln, als wären sie Teil des Anwendungscodes und nicht einer separaten Datei.

Die LIB-Datei für eine DLL wird automatisch beim Kompilieren der DLL erzeugt. Von Ihrer Seite sind keine weiteren Schritte dafür erforderlich.

Es ist nicht nur leichter, Anwendungen mithilfe von LIB-Dateien für alle verwendeten DLLs zu erstellen, sondern auch sicherer, wenn Sie die Anwendung ausführen. Wenn Sie mit LIB-Dateien arbeiten, werden alle DLLs, auf die Sie in Ihrer Anwendung zurückgreifen, zu dem Zeitpunkt in den Arbeitsspeicher geladen, zu dem die Anwendung startet. Sollte irgendeine DLL fehlen, informiert Windows den Benutzer automatisch über dieses Problem und Ihre Anwendung startet nicht. Wenn Sie dagegen nicht mit den LIB-Dateien arbeiten, müssen Sie die DLL selbst in den Speicher laden und sind auch dafür verantwortlich, alle Fehler zu behandeln, die bei einer fehlenden DLL auftreten können.

Zwei DLL-Typen lassen sich mit Visual C++ sehr einfach erstellen. Dabei handelt es sich um erweiterte MFC-DLLs und Standard-DLLs.

Mit Visual C++ lassen sich auch andere DLL-Typen erstellen. Diese umfassen aber einen beträchtlichen Teil der ActiveX-Funktionalität, sodass wir im Rahmen dieses Buches nicht darauf eingehen können. Wenn Sie mit ActiveX-Technologie so genannte prozessinterne

Server-DLLs oder andere Arten von ActiveX-DLLs erstellen wollen, sollten Sie sich nach einem weiterführenden Buch zu Visual C++ umsehen, das sich speziell diesen Themen widmet. Bei Markt+Technik ist zu diesem Thema »Visual C++ 6 Kompendium« (ISBN: 3-8272-5467-1) erschienen.

Erweiterte MFC-DLLs

MFC-DLLs lassen sich am einfachsten programmieren und erstellen, da man sie genauso behandeln kann wie jede andere Sammlung von Klassen. Bei allen Klassen, die Sie aus einer DLL exportieren wollen, brauchen Sie lediglich das Makro AFX_EXT_CLASS wie folgt in die Klassendeklaration einzufügen:

```
class AFX_EXT_CLASS CMyClass
{
...
};
```

Dieses Makro exportiert die Klasse und macht sie damit für Visual C++-Anwendungen verfügbar. Das Makro ist in die Header-Datei einzubinden, die von den Anwendungen, die auf die DLL zurückgreifen, verwendet wird. Damit importieren Sie die Klasse aus der DLL, sodass sie sich verwenden lässt. Klassen, die dieses Makro nicht in der Klassendeklaration enthalten, werden nicht exportiert und sind damit nicht für andere Anwendungen sichtbar, die die DLL verwenden.

Ein Nachteil bei der Erstellung von erweiterten MFC-DLLs besteht darin, dass sie sich nicht in anderen Programmiersprachen nutzen lassen, sondern nur mit anderen C++- Compilern, solange diese MFC unterstützen (wie das bei den Compilern von Borland und Symantec der Fall ist).

Standard-DLLs

Standard-DLLs exportieren Standardfunktionen aus der DLL und keine C++-Klassen. Im Ergebnis kann dieser DLL-Typ etwas mehr Nachdenken und Planung erfordern als eine erweiterte MFC-DLL. Innerhalb der DLL selbst können Sie allerdings alle Klassen wie gewünscht verwenden, für die externen Anwendungen müssen Sie jedoch reine Funktionsaufrufe vorsehen.

Um eine Funktion zu exportieren, ist sie als Exportfunktion zu deklarieren. Dazu versehen Sie den Funktionsnamen mit folgendem Vorsatz:

```
extern "C" function_type PASCAL EXPORT function_declaration
```

Diese zusätzlichen Angaben sind sowohl in der Header-Datei für den Prototyp der Funktion als auch im eigentlichen Quellcode erforderlich.

- Der Abschnitt extern "C" deklariert, dass es sich um einen normalen Funktionsaufruf in C handelt, damit bei der C++-Namensanalyse nicht der Funktionsname zerrissen wird.
- PASCAL weist den Compiler an, alle Funktionsargumente in der PASCAL-Reihenfolge zu übergeben, bei der die Argumente auf dem Stapel in umgekehrter Reihenfolge gegenüber der Schreibweise platziert werden.
- Schließlich teilt EXPORT dem Compiler mit, dass diese Funktion aus der DLL zu exportieren ist und von außerhalb der DLL aufgerufen werden kann.

Beim Exportieren von Funktionen aus der DLL sind darüber hinaus alle exportierten Funktionsnamen in die DEF-Datei für das DLL-Projekt einzutragen. Diese Datei ist für das Erstellen der LIB-Datei mit den Pseudofunktionen sowie der Exporttabelle in der DLL erforderlich. Sie enthält den Namen der DLL oder Bibliothek, eine kurze Beschreibung der DLL und die Namen aller Funktionen, die exportiert werden. Für die Datei ist ein spezielles Format vorgeschrieben, sodass Sie die vom DLL-Assistenten automatisch erzeugte Standard-DEF-Datei nur insofern verändern sollten, als Sie exportierte Funktionsnamen hinzufügen. Eine typische DEF-Datei sieht folgendermaßen aus:

```
LIBRARY     "mydll"
DESCRIPTION 'meine Windows Dynamic Link Library'
EXPORTS
    ; Explizite Exporte können hier eingefügt werden
```

```
    MyFunc1
    MyFunc2
```

Wenn Sie vor Ihrer Funktionsdeklaration __declspec(dllexport) einfügen, benötigen Sie die DEF-Datei nicht. Diese Direktive wurde vor ein paar Jahren zu Visual C++ hinzugefügt, um den komplizierten Export von Funktionen in DLLs zu erleichtern. Die Direktive __declspec(dllexport) weist den Compiler an, die darauf folgende Funktion in die DLL zu exportieren, genauso als würden Sie den Funktionsnamen in die DEF-Datei einfügen.

Wenn Sie MFC-Klassen in Ihren Standard-DLLs verwenden, müssen Sie das Makro AFX_MANAGE_STATE als erste Code-Zeile in allen exportierten Funktionen aufrufen. Das ist erforderlich, um die exportierten Funktionen thread-sicher zu machen. Damit können Ihre Klassenfunktionen gleichzeitig durch mehrere Threads aufgerufen werden (was Threads sind erfahren Sie am Tag 17 »Multitasking«). Das Makro AFX_MANAGE_STATE übernimmt als einziges Argument einen Zeiger auf eine AFX_MODULE_STATE-Struktur, die sich durch Aufruf der Funktion AfxGetStaticModuleState abrufen lässt. Eine typische exportierte Funktion, die mit MFC arbeitet, sieht wie folgt aus:

```
extern "C" void PASCAL EXPORT MyFunc(...)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    // Hier normaler Funktionsrumpf
...
}
```

DLLs entwerfen

Wenn Sie Ihre DLLs entwerfen, sollten Sie sich dessen bewusst sein, dass alle Funktionen in den DLLs gleichzeitig durch mehrere Threads, die zur gleichen Zeit laufen, aufgerufen werden können. Daraus folgt, dass die gesamte Funktionalität in allen von Ihnen erstellten DLLs Thread-sicher sein muss.

Alle Variablen, die Werte über eine separate Funktion hinaus aufnehmen, sind durch die Anwendung und nicht von der DLL zu speichern und zu verwalten. Alle Anwendungsvariablen, die von der DLL manipuliert werden müssen, sind an die DLL als Funktionsargumente zu übergeben. Alle globalen Variablen, die innerhalb der DLL verwaltet werden, können bei laufender Funktion mit Variablen aus anderen Anwendungsprozessen ausgetauscht werden, was zu unvorhersagbaren Ergebnissen führt. Daher muss die Manipulation globaler Variablen durch Mechanismen der Thread- Synchronisierung geschützt werden, wie es in Kapitel 17, »Multitasking«, besprochen wird.

14.3 Erweiterte MFC-DLLs erstellen und einsetzen

Damit Sie sich davon überzeugen können, wie einfach es ist, eine erweiterte MFC-DLL zu erstellen und einzusetzen, erstellen Sie heute eine DLL mit zwei Klassen. Die erste Klasse ist die an Tag 10, »SDI- und MDI-Anwendungen«, erzeugte Klasse CLine. Die zweite Klasse erzeugt die zufälligen Zeichnungen auf der Zeichenoberfläche. Diese Klasse enthält ein Objekt-Array der CLine-Objekte, das sie bei jedem Zeichenansatz erzeugt und bestückt. Diese zweite Klasse benötigt außerdem Funktionalität, um die Zeichnung zu speichern und wiederherzustellen und um die bestehende Zeichnung zu löschen und eine neue zu beginnen. Sie muss die Abmessungen des Zeichenbereichs kennen, um eine Zeichnung erzeugen zu können, die in den Zeichenbereich hineinpasst. Nachdem Sie dieses Modul erstellt haben, werden Sie sehen, wie man es in einem Anwendungsprojekt verwendet. Nachdem Sie gesehen haben, wie einfach sich erweiterte MFC-DLLs erstellen lassen, implementieren Sie die gleiche Funktionalität erneut als Standard-DLL, sodass Sie am Ende die verschiedenen Lösungswege kennen, die bei den beiden DLL-Stilen erforderlich sind.

Die erweiterte MFC-DLL erstellen

Um das Projekt für das heutige Beispiel zu beginnen, folgen Sie diesen Schritten:

1. Erzeugen Sie ein neues Projekt und spezifizieren Sie es als MFC-DLL-Projekt (siehe Abbildung 14.1). Nennen Sie das Projekt ModArtDll und klicken Sie auf Ok, um das Projekt zu erzeugen.

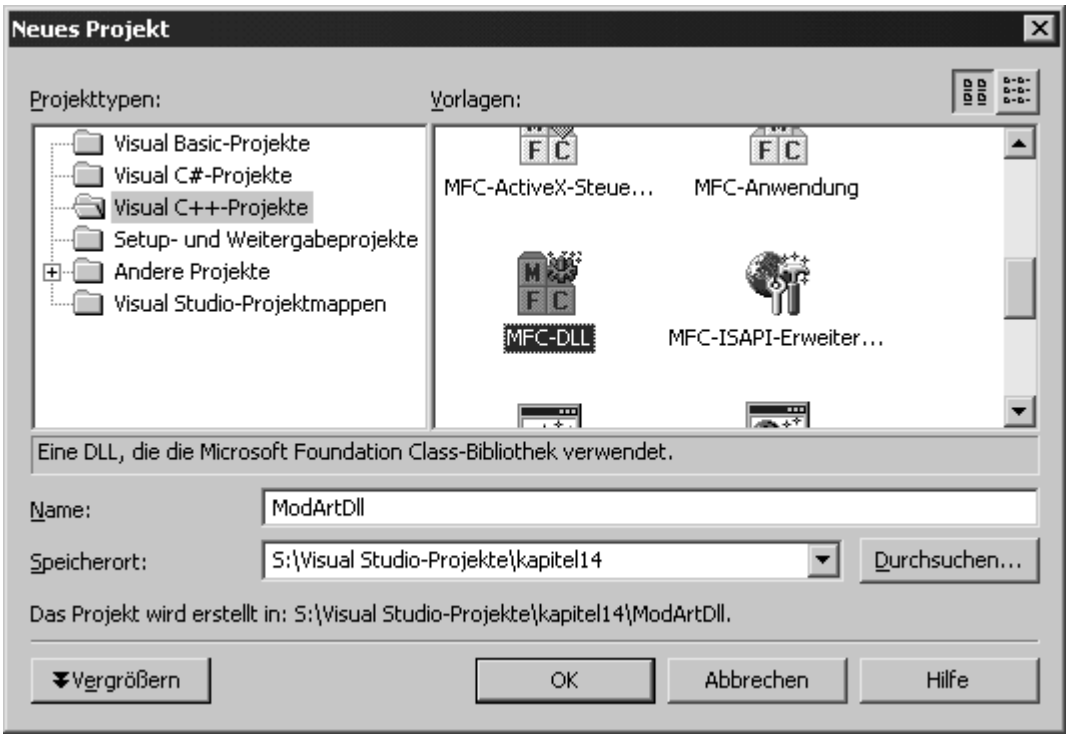


Abbildung 14.1: Ein MFC-DLL-Projekt anlegen

2. Wählen Sie im Bereich Anwendungseinstellungen den Eintrag MFC- Erweiterungs-DLL aus (siehe Abbildung 14.2). Klicken Sie auf Fertig stellen, um das Projekt zu erstellen.

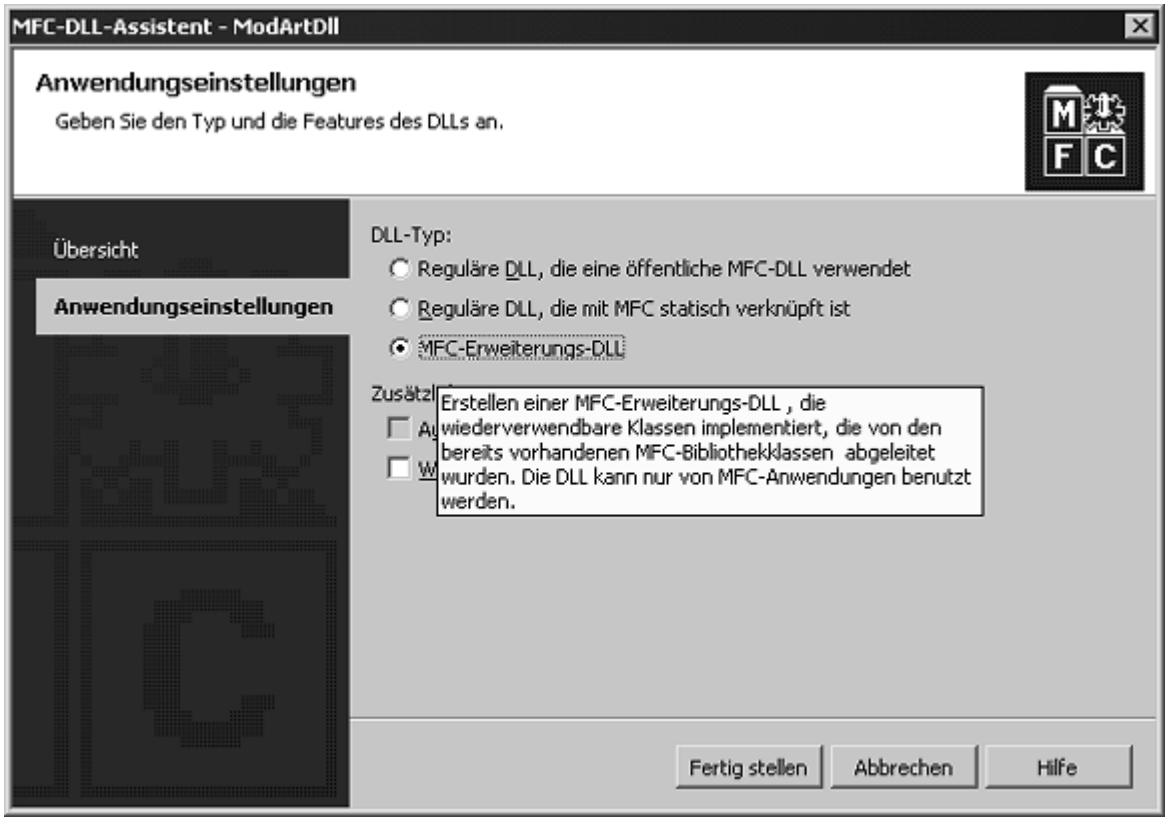


Abbildung 14.2: Den Typ der zu erstellenden DLL festlegen

Nachdem Sie Ihr Projekt erstellt haben, stellen Sie fest, dass Sie mit einem Projekt ohne Klassen arbeiten. Sie haben eine leere Arbeitsfläche, von der aus Sie ein Modul genau nach Ihren Bedürfnissen zurechtschneidern können.

3. Da Sie die Klasse CLine bereits erstellt haben, kopieren Sie sie (sowohl Line.h als auch Line.cpp) aus dem Projektbereich von Tag 10 in das Verzeichnis des heutigen Projekts.
4. Fügen Sie die Header- und Quellcode-Datei in Ihr heutiges Projekt ein, indem Sie Vorhandenes Element hinzufügen aus dem Menü Projekt wählen (Sie können beide Dateien zusammen markieren). Danach sollte die Klasse CLine in der Klassenansicht Ihres Projekts erscheinen.
5. Im Projektmappen-Explorer können Sie sehen, dass die beiden Dateien im Ordner Projektmappen-Elemente abgelegt wurden. Ziehen Sie sie mit der Maus in die beiden Ordner Quelldateien bzw. Headerdateien.

Klassen definieren

Nachdem Sie nun über ein einsatzbereites Projekt verfügen, ist das Modul noch mit Leben zu erfüllen. Mit der Klasse CLine bietet sich eine gute Möglichkeit, eine bereits mit anderen Vorgaben realisierte Funktionalität wiederzuverwenden. Allerdings soll die eigentliche Funktionalität des Moduls darin bestehen, zufällige Zeichnungen - oder Squiggles (auf deutsch etwa: Gekritzel) - zu erzeugen. Für diese Funktionalität benötigen Sie eine neue Klasse. Um mit der neuen Klasse zu beginnen, folgen Sie diesen Schritten:

1. Fügen Sie auf der Registerkarte Klassen-Ansicht eine neue allgemeine Klasse in das Projekt ein (wählen Sie Hinzufügen / Klasse hinzufügen aus dem Kontextmenü).

2. Geben Sie der Klasse den Namen CModArt und legen Sie fest, dass sie von der Klasse CObject als public abgeleitet wird.

Nachdem Sie Ihre Klasse angelegt haben, müssen Sie noch ein paar Variablen in die Klasse aufnehmen. Zuerst ist dafür Sorge zu tragen, dass alle Linien, aus denen die Zeichnung besteht, gespeichert werden können. Zu diesem Zweck legen Sie ein Objektarray an. Zweitens müssen Sie die Abmessungen des Zeichenbereichs kennen. Die entsprechenden Angaben speichern Sie in einer CRect-Struktur. Fügen Sie beide Variablen in Ihre neue Klasse ein und verwenden Sie dabei folgende Einstellungen:

Typ	Name	Zugriff
CRect	m_rDrawArea	private
CObArray	m_oaLines	private

Den Zeichenbereich einrichten

Bevor man überhaupt etwas zeichnen kann, muss man den verfügbaren Zeichenbereich kennen. Dazu fügen Sie in Ihre Klasse eine öffentliche Funktion ein, die die übergebene CRect-Struktur in die Elementvariable CRect kopiert. Um diese Funktion in Ihr Projekt einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in Ihre neue Klasse ein.

2. Legen Sie den Funktionstyp als void und den Namen mit SetRect fest und fügen Sie einen Parameter vom Typ CRect namens rDrawArea ein. Legen Sie den Zugriff als public fest.
3. In die Funktion schreiben Sie den Code aus Listing 14.1:

Listing 14.1: Festlegung des Zeichenbereichs

```
1: void CModArt::SetRect(CRect rDrawArea)
2: {
3:    // Rechteck für Zeichenbereich festlegen
4:    m_rDrawArea=rDrawArea;
5: }
```

Eine neue Zeichnung erstellen

Eines der Schlüsselmerkmale dieses Moduls ist die Fähigkeit, zufällige Squiggles zu generieren, die im Zeichenbereich erscheinen. Durch das Generieren einer ganzen Folge dieser Squiggles kann das Modul eine vollständige Zeichnung erzeugen. Beginnend mit einem einzelnen Squiggle können Sie eine Funktion entwerfen, die ein Squiggle generiert, und dann diese Funktion mehrmals aufrufen, um die gesamte

Zeichnung zu erstellen.

Die erste Funktion, der Squiggle-Generator, muss ermitteln, wie viele Linien zu einem Squiggle gehören. Außerdem sind die Farbe und die Breite des Zeichenstifts für das Squiggle zu bestimmen. Weiterhin braucht die Funktion den Anfangspunkt des Squiggles. Von diesem Punkt aus kann die Funktion in einer Schleife die entsprechende Anzahl von Linien durchlaufen und jeweils einen neuen Zielpunkt generieren, um das Squiggle vom vorherigen Zielpunkt fortzusetzen.

Um die Funktionalität in Ihrem Projekt zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Zeichenklasse ein.

2. Legen Sie den Funktionstyp als void, den Namen mit NewLine und den Zugriffsstatus als private (da die Funktion nur durch die Hauptschleife aufgerufen wird, die bestimmt, wie viele Squiggles in der endgültigen Zeichnung enthalten sein sollen).
3. In die neue Funktion übernehmen Sie den Code aus Listing 14.2.

Listing 14.2: Die Funktion CModArt.NewLine

```
 1: void CModArt::NewLine(void)
 2: {
 3:   int iNumLines;
 4:   int iCurLine;
 5:   int iCurColor;
 6:   UINT nCurWidth;
 7:   CPoint ptTo;
 8:   CPoint ptFrom;
 9:
10:   static COLORREF crColors[8] = {
11:     RGB(   0,   0,   0),    // Schwarz
12:     RGB(   0,   0, 255),    // Blau
13:     RGB(   0, 255,   0),    // Grün
14:     RGB(   0, 255, 255),    // Cyan
15:     RGB( 255,   0,   0),    // Rot
16:     RGB( 255,   0, 255),    // Magenta
17:     RGB( 255, 255,   0),    // Gelb
18:     RGB( 255, 255, 255)     // Weiß
19:   };
20:
21:   // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
22:   m_rDrawArea.NormalizeRect();
23:   // Breite und Höhe des Zeichenbereichs ermitteln
24:   int iWidth = m_rDrawArea.Width();
25:   int iHeight = m_rDrawArea.Height();
26:
27:   // Anzahl der Teile dieses Squiggles bestimmen
28:   iNumLines = rand() % 100;
29:   // Umfasst das Squiggle mindestens ein Teil?
30:   if (iNumLines > 0)
31:   {
32:     // Farbe bestimmen
33:     iCurColor = rand() % 8;
34:     // Stiftbreite bestimmen
35:     nCurWidth = (rand() % 8) + 1;
36:     // Anfangspunkt für Squiggle bestimmen
37:     ptFrom.x = (rand() % iWidth) + m_rDrawArea.left;
38:     ptFrom.y = (rand() % iHeight) + m_rDrawArea.top;
39:     // Schleife durch Anzahl der Segmente
40:     for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
41:     {
42:       // Endpunkt des Segments bestimmen
43:       ptTo.x = ((rand() % 20) - 10) + ptFrom.x;
44:       ptTo.y = ((rand() % 20) - 10) + ptFrom.y;
```

```
45:       // Neues CLine-Objekt erzeugen
46:       CLine *pLine = NULL;
47:       try
48:       {
49:         pLine = new CLine(ptFrom, ptTo, crColors[iCurColor],
50:                           nCurWidth);
51:         // Neue Linie in das Objektarray einfügen
52:         m_oaLines.Add(pLine);
53:       }
54:       // Speicherausnahme?
55:       catch (CMemoryException* perr)
56:       {
57:         // Meldung an Benutzer mit schlechten Neuigkeiten
58:         AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK);
59:         // Wurde ein Linienobjekt erzeugt?
60:         if (pLine)
61:         {
62:           // Löschen
63:           delete pLine;
64:           pLine = NULL;
65:         }
66:         // Ausnahmeobjekt löschen
67:         perr->Delete();
68:       }
69:       // Anfangspunkt auf Endpunkt setzen
70:       ptFrom = ptTo;
71:     }
72:   }
73: }
```

Die Funktion in Listing 14.2 ermittelt zuerst mit den folgenden Zeilen den verfügbaren Zeichenbereich:

```
m_rDrawArea.NormalizeRect();
int iWidth = m_rDrawArea.Width();
int iHeight = m_rDrawArea.Height();
```

Die erste Zeile normalisiert das Rechteck, damit garantiert ist, dass die in den nächsten beiden Zeilen zurückgegebenen Werte für Breite und Höhe positiv sind. Auf Grund des in Windows verwendeten Koordinatensystems kann das Ermitteln der Breite durch Subtraktion der linksseitigen Position von der rechtsseitigen Position zu negativen Zahlen führen. Das Gleiche trifft auf die Höhe zu. Durch die Normalisierung des Rechtecks ist sichergestellt, dass man für die beiden Werte positive Ergebnisse erhält.

Nachdem der Zeichenbereich ermittelt ist, bestimmt die Funktion die Anzahl der Linienabschnitte, die im Squiggle zu verwenden sind:

```
iNumLines = rand() % 100;
```

Die Funktion rand kann Zahlen in einem weiten Bereich zurückgeben (festgelegt durch die Konstante RAND_MAX, bei Visual Studio 32767). Durch die Modulo-Division mit 100 stellt man sicher, dass die resultierende Zahl zwischen 0 (einschließlich) und 100 (ausschließlich) liegt. Dieses Verfahren wendet man im Allgemeinen an, um Zufallszahlen für einen bestimmten Bereich zu erzeugen, wobei man in der Modulo-Funktion die Obergrenze des Wertebereichs angibt. (Ist die Untergrenze ungleich 0, gibt man in der Modulo-Funktion das Ergebnis aus Untergrenze minus Obergrenze an und addiert den Wert der Untergrenze auf das Ergebnis der Modulo-Rechnung). Nach dem gleichen Verfahren bestimmen Sie die Farbe, Breite und Startposition für das Squiggle:

```
iCurColor = rand() % 8;
nCurWidth = (rand() % 8) + 1;
ptFrom.x = (rand() % lWidth) + m_rDrawArea.left;
ptFrom.y = (rand() % lHeight) + m_rDrawArea.top;
```

Beim Berechnen der Startposition wird die linke und obere Position des Zeichenbereichs zur generierten

Position addiert. Damit ist sichergestellt, dass der Startpunkt innerhalb des Zeichenbereichs liegt. Sobald die Funktion in die Schleife eingetreten ist und alle Linienabschnitte im Squiggle generiert, wird der verfügbare Bereich für den nächsten Zielpunkt auf 10 von der aktuellen Position aus begrenzt:

```
ptTo.x = ((rand() % 20) - 10) + ptFrom.x;
ptTo.y = ((rand() % 20) - 10) + ptFrom.y;
CLine *pLine = NULL;
pLine = new CLine(pFrom, pTo, m_crColors[nCurColor], nCurWidth);
m_oaLines.Add(pLine);
```

Diesen Abstand können Sie in einfacher Weise erhöhen, um die Zeichnung verwinkelter zu gestalten. Nachdem die Funktion das nächste Liniensegment generiert hat, erzeugt sie das Linienobjekt und fügt es in das Objektarray ein. Schließlich setzt die Funktion den Anfangspunkt auf den Endpunkt des eben generierten Liniensegments:

```
ptFrom = ptTo;
```

Nun kann die Funktion die Schleife erneut durchlaufen und das nächste Liniensegment generieren, bis alle Liniensegmente in diesem Squiggle generiert sind.

Nachdem Sie nun ein einzelnes Squiggle erzeugen können, ist der verbleibende Rest einfach zu erledigen. Zuerst ermittelt man, aus wie vielen Squiggles die Zeichnung bestehen soll. Als Nächstes durchläuft man eine Schleife für die Anzahl der zu generierenden Squiggles und ruft in jedem Durchlauf die Funktion NewLine einmal für jedes Squiggle auf. Um diese Funktionalität im Beispielprojekt zu realisieren, folgen Sie diesen Schritten:

1. Fügen Sie in die Zeichenklasse eine neue Member-Funktion ein.

2. Legen Sie den Typ als void, den Namen mit NewDrawing und den Zugriff als public fest.
3. In die Funktion schreiben Sie den Code aus Listing 14.3:

Listing 14.3: Die Funktion CModArt.NewDrawing

```
 1: void CModArt::NewDrawing(void)
 2: {
 3:   int iNumLines;
 4:   int iCurLine;
 5:
 6:   // Anzahl der zu erzeugenden Linien ermitteln
 7:   iNumLines = rand() % 10;
 8:   // Sind Linien zu erzeugen?
 9:   if (iNumLines > 0)
10:   {
11:     // Schleife durch die Anzahl der Linien
12:     for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
13:     {
14:       // Neue Linie erzeugen
15:       NewLine();
16:     }
17:   }
18: }
```

Die Zeichnung anzeigen

Um die Gruppe der Squiggles im Zeichenbereich darzustellen, können Sie eine Funktion hinzufügen, die das Objektarray in einer Schleife durchläuft und dabei die Funktion Draw für jedes Liniensegment im Array aufruft. Diese Funktion muss den Gerätekontext als einziges Argument erhalten und ihn an die einzelnen Liniensegmente weiterreichen. Um die Funktion in Ihr Projekt einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie der Zeichenklasse eine neue Member-Funktion hinzu.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen mit Draw fest und fügen Sie einen Parameter vom Typ CDC* namens pDC ein. Legen Sie den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 14.4.

Listing 14.4: Die Funktion CModArt.Draw

```
 1: void CModArt::Draw(CDC* pDC)
 2: {
 3:   // Anzahl der Zeilen im Objektarray ermitteln
 4:   int iCount = m_oaLines.GetSize();
 5:   int iPos;
 6:
 7:   // Enthält das Objektarray Objekte?
 8:   if (iCount)
 9:   {
10:     // Schleife durch das Array, dabei jedes Objekt zeichnen
11:     for (iPos = 0; iPos < iCount; iPos++)
12:       ((CLine*)m_oaLines.GetAt(iPos))->Draw(pDC);
13:   }
14: }
```

Die Zeichnung serialisieren

Da Sie auf die Liniensegmentklasse zurückgreifen, die Sie bereits an früherer Stelle erzeugt und serialisierbar gemacht haben, brauchen Sie keine Makros für die Serialisierung in die Zeichenklasse einzubinden. Allerdings müssen Sie eine Serialize- Funktion hinzufügen, die das Archivobjekt an das Objektarray weiterreicht und diesem sowie den Liniensegmentobjekten die Serialisierung überträgt. Um Ihrem Projekt diese Funktion hinzuzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Zeichenklasse ein.

2. Legen Sie den Funktionstyp mit void und den Namen als Serialize fest und fügen Sie einen Parameter vom Typ CArchive& namens ar ein. Legen Sie den Zugriff als public fest.
3. In die Funktion schreiben Sie folgenden Code:

```
void CModArt::Serialize(CArchive &ar)
{
  // Archivobjekt an Array übergeben
  m_oaLines.Serialize(ar);
}
```

Die Zeichnung löschen

Um die volle Funktionalität bereitzustellen, müssen Sie in der Lage sein, die Zeichnung aus der Zeichenklasse zu löschen, sodass sich eine neue Zeichnung generieren oder eine vorhandene Zeichnung laden lässt. Dazu durchläuft man einfach das Objektarray, zerstört alle Liniensegmentobjekte und setzt dann das Objektarray zurück. Um diese Funktionalität zu Ihrem Projekt hinzuzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie eine weitere Member-Funktion in das Projekt auf.

2. Spezifizieren Sie den Typ als void, den Namen als ClearDrawing und den Zugriff als public.
3. Übernehmen Sie in diese Funktion den Code aus Listing 14.5.

Listing 14.5: Die Funktion CModArt.ClearDrawing

```
 1: void CModArt::ClearDrawing(void)
 2: {
 3:   // Anzahl der Linien im Objektarray ermitteln
 4:   int iCount = m_oaLines.GetSize();
 5:   int iPos;
 6:
```

```
 7:   // Enthält das Array Objekte?
 8:   // Schleife durch das Array, dabei jedes Objekt löschen
 9:   for (iPos = 0; iPos < iCount; iPos++)
10:     delete m_oaLines.GetAt(iPos);
11:   // Array zurücksetzen
12:   m_oaLines.RemoveAll();
13: }
```

Die Klasse fertig stellen

Um die Zeichenklasse zu komplettieren, ist noch der Zufallszahlengenerator zu initialisieren. Die als Zufallszahlengenerator arbeitende Funktion rand generiert eine statistisch zufällige Zahlenfolge, die auf einer Reihe von mathematischen Berechnungen beruht. Wenn man den Zufallszahlengenerator jedes Mal mit derselben Zahl startet, erhält man immer wieder die gleiche Zahlenfolge. Damit der Zufallszahlengenerator bei jedem Start der Anwendung eine andere Zufallszahlenfolge liefert, muss man ihm einen Anfangswert übergeben, der jedes Mal unterschiedlich ist. Normalerweise übergibt man zu diesem Zweck die Systemzeit an die Funktion srand, die den Zufallszahlengenerator bei jedem Start der Anwendung mit einer anderen Zeit initiiert. Den Zufallszahlengenerator muss man nur einmal während der Ausführung einer Anwendung initialisieren. Diese Funktionalität können Sie folgendermaßen hinzufügen:

1. Nehmen Sie folgenden Code in den Konstruktor der Zeichenklasse auf:

```
CModArt::CModArt(void)
{
  // Zufallszahlengenerator initialisieren
  srand((unsigned)time(NULL));
}
```

2. Sie müssen außerdem das Makro AFX_EXT_CLASS in die Klassendeklaration aufnehmen. Um dieses Makro einzufügen, öffnen Sie die Header-Datei (ModArt.h), in der die Definition der Zeichenklasse enthalten ist.
3. Nehmen Sie wie folgt das Makro in die Klassendeklaration auf:

```
class AFX_EXT_CLASS CModArt :
  public CObject
  {
    ...
```

Die DLL ist damit fertig gestellt. Bevor Sie weitergehen können, müssen Sie Ihr Projekt kompilieren. Nach diesem Schritt können Sie noch nichts starten, da Sie erst eine Anwendung erstellen müssen, die Ihre DLL verwendet. Nur so lässt sich der Code ausführen und testen. Um diese Testanwendung zu erstellen, schließen Sie zunächst die Projektmappe, damit Sie für die Testanwendung neu beginnen können.

Sie erhalten beim Kompilieren eine lange Liste von Fehlermeldungen? Dann haben Sie am Anfang der Datei ModArt.cpp vergessen die Headerdatei Line.h einzubinden:

```
#include "StdAfx.h"
#include "modart.h"
#include "line.h"
CModArt::CModArt(void)
```

Eine Testanwendung erstellen

Damit Sie Ihr Modul testen können, brauchen Sie eine Anwendung, die das Modul verwendet. Diese einfache Anwendung muss nur so viel Funktionalität bieten, dass sich das Modul ausreichend testen lässt. Wir wollen lediglich die gesamte Funktionalität des Moduls ausprobieren und können auf eine ausgewachsene Anwendung verzichten.

Wenn Sie die Testanwendung erstellen, müssen Sie die Header-Datei für die Zeichenklasse in die relevanten Klassen in Ihrer Anwendung einbinden. Bei einer typischen SDI- oder MDI-Anwendung bedeutet das, dass man die Header-Datei zumindest in die Quelldateien der Dokumentklasse und gegebenenfalls noch in die der

Ansichtsklasse und der Anwendungsklasse einbindet. Weiterhin ist die Bibliotheksdatei, die Ihr Modul erzeugt hat, in das Anwendungsprojekt einzubinden, damit sie zur Anwendung gelinkt wird.

Das Gerüst der Testanwendung

Der Rahmen für einen Test lässt sich auf einfache Weise mit einem normalen SDI- oder MDI-Anwendungsgerüst realisieren. Um die Beispielanwendung so einfach wie möglich zu halten, empfiehlt es sich, eine SDI-Anwendung zu wählen. Wenn Sie allerdings bestimmte Funktionen in Ihrem Modul vorgesehen haben, die sich speziell auf eine MDI- Anwendung beziehen, dann liegt es auf der Hand, dass Sie mit dem Gerüst einer MDI- Anwendung arbeiten.

Für die Testanwendung des erstellte Beispielmoduls folgen Sie diesen Schritten:

1. Erstellen Sie ein neues MFC-Anwendung-Visual C++-Projekt. Nennen Sie das Projekt TestApp.

2. Wählen Sie im Bereich Anwendungstyp als Anwendungstyp den Eintrag Einfaches Dokument.
3. Geben Sie im Bereich Zeichenfolgen für Dokumentvorlagen eine Dateinamenserweiterung für die Dateien ein, die Ihre Anwendung erzeugen wird.

Nachdem Sie das Anwendungsgerüst erstellt haben, müssen Sie die DLL in das Projekt einbinden. Dazu folgen Sie diesen Schritten:

1. Wählen Sie Vorhandenes Element hinzufügen aus dem Menü Projekt.

2. Legen Sie die Dateitypen mit Alle Dateien fest.
3. Gehen Sie in das Debug-Verzeichnis der DLL, um das Bibliotheksmodul zu lokalisieren, das Sie im vorherigen Projekt erstellt haben. Dazu müssen Sie normalerweise eine Verzeichnisebene höher gehen, das Projektverzeichnis für das Modul aufsuchen und in diesem Verzeichnis nach unten zum Debug-Verzeichnis wechseln. (Wenn Sie die Release-Version des Moduls und der Anwendung erstellen, müssen Sie sinngemäß in die entsprechenden Release-Verzeichnisse wechseln.) Die Bibliotheksdatei für das von Ihnen erstellte Modul sollten Sie entsprechend Abbildung 14.3 lokalisieren können. Markieren Sie das Modul und klicken Sie auf Öffnen, um es in das Projekt einzufügen.

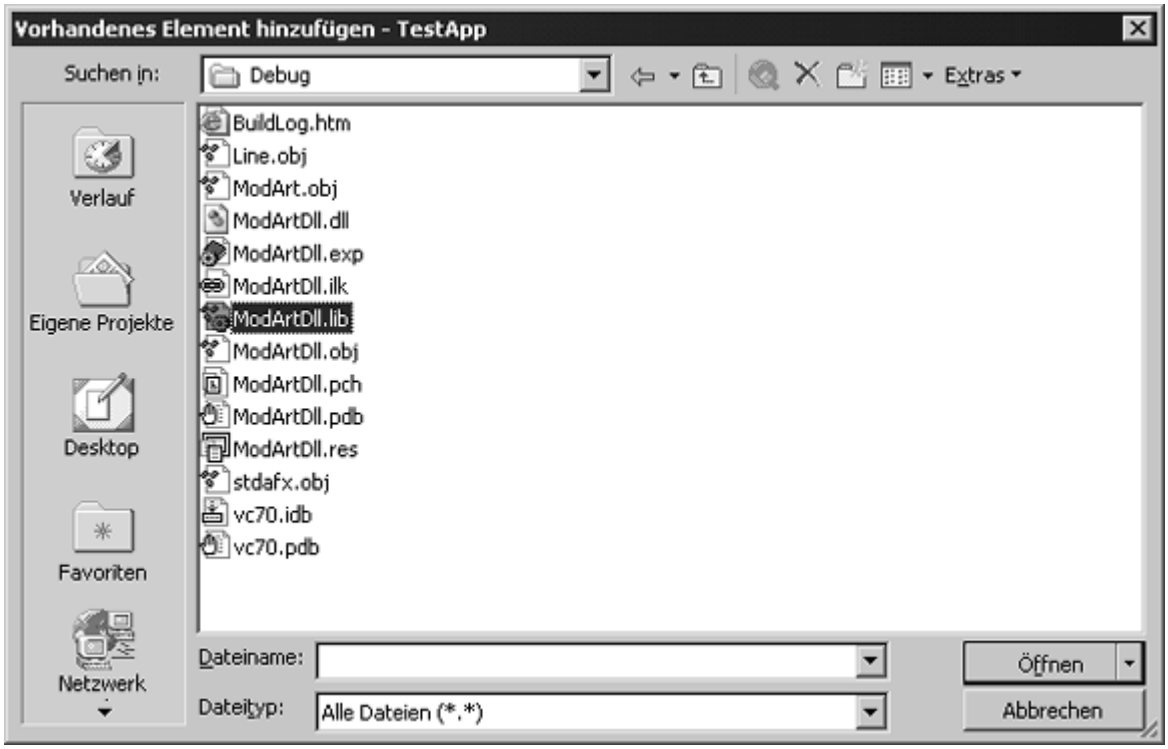


Abbildung 14.3: Eine Bibliotheksdatei in das Projekt aufnehmen

Nachdem Sie eine Bibliotheksdatei in das Projekt aufgenommen haben, müssen Sie noch die Header-

Dateien für alle Klassen im Modul, die in den entsprechenden Quellcode- Dateien der Anwendung zum Einsatz kommen, einbinden. Für die zu erstellende Testanwendung folgen Sie diesen Schritten:

1. Wählen Sie Vorhandenes Element hinzufügen aus dem Menü Projekt.

2. Gehen Sie zum Projektverzeichnis des DLL-Projekts.
3. Markieren Sie die Datei ModArt.h und klicken Sie auf Öffnen, um sie dem Projekt hinzuzufügen.

Sie müssen die Header-Dateien nur für die Klassen in der DLL hinzufügen, die exportiert wurden und auf die Sie in Ihrem Projekt zugreifen. Wenn Sie Header-Dateien für nicht exportierte Klassen hinzufügen, können Sie beim Versuch, Ihre Anwendung zu kompilieren, auf Fehler stoßen.

Um die Vorbereitungen am Anwendungsgerüst abzuschließen, nehmen Sie als Letztes noch eine Variable für alle Klassen aus der DLL auf, die in irgendwelche Klassen der Anwendung eingebunden werden. Für die Testanwendung ist das eine Variable in der Dokumentklasse der Zeichenklasse, die Sie im Projekt des Bibliotheksmoduls erzeugt haben. Um diese Variable in Ihre Anwendung einzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie eine neue Member-Variable in die Dokumentklasse auf.

2. Legen Sie den Variablentyp als CModArt, den Namen mit m_maDrawing und den Zugriff als private fest.

Eine neue Zeichnung erstellen

Auf die Funktionalität des Moduls greifen Sie zuerst beim Erzeugen eines neuen Dokuments zurück. Das ist der Moment, wo eine neue Zeichnung zu generieren ist. Letztendlich sind zwei Dinge zu realisieren:

- den Zeichenbereich der Ansichtsklasse ermitteln und an das Zeichenobjekt übergeben,
- das Zeichenobjekt anweisen, eine neue Zeichnung zu generieren.

Das ist alles ziemlich unkompliziert. Um diese Funktionalität in Ihre Anwendung aufzunehmen, bearbeiten Sie die Funktion OnNewDocument in der Dokumentklasse und fügen die fett gedruckten Zeilen aus Listing 14.6 hinzu.

Listing 14.6: Die Funktion CTestAppDoc.OnNewDocument

```
 1: CTestAppDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Position der Ansicht ermitteln
 9:   POSITION pos = GetFirstViewPosition();
10:   // Ist Position gültig?
11:   if (pos != NULL)
12:   {
13:     // Zeiger auf die Ansicht holen
14:     CView* pView = GetNextView(pos);
15:     RECT rWndRect;
16:     // Rechteck des Anzeigebereichs ermitteln
17:     pView->GetClientRect(&rWndRect);
```

```
18:     // Zeichenbereich setzen
19:     m_maDrawing.SetRect(rWndRect);
20:     // Neue Zeichnung erzeugen
21:     m_maDrawing.NewDrawing();
22:   }
23:   return TRUE;
24: }
```

Eine Zeichnung speichern und löschen

In die Dokumentklasse sind noch die Funktionen einzubauen, um die Zeichnung zu speichern und wiederherzustellen sowie zu löschen. Diese Aufgaben sind die letzten der dokumentbezogenen Funktionalität Ihres Bibliotheksmoduls.

Um die Funktionalität zum Speichern und Wiederherstellen in der Anwendung zu realisieren, bearbeiten Sie die Funktion Serialize in der Dokumentklasse. Löschen Sie den gesamten Inhalt der Funktion und ersetzen Sie ihn durch folgenden Aufruf der Serialize-Funktion des Zeichenobjekts:

```
void CTestAppDoc::Serialize(CArchive& ar)
{
  // Zeichnung serialisieren
  m_maDrawing.Serialize(ar);
}
```

Zum Löschen der Zeichnung - damit sich eine neue Zeichnung generieren oder eine gespeicherte Zeichnung laden lässt - brauchen Sie eine Behandlungsroutine für die Funktion DeleteContents der Dokumentklasse. In dieser Funktion rufen Sie die Funktion ClearDrawing des Zeichenobjekts auf. Um die Funktionalität in die Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie mit den Schritten der vergangenen Tage eine Überschreibungs-Funktion für die Basisfunktion DeleteContents in die Dokumentklasse ein.

2. Ergänzen Sie die Funktion mit folgendem Code:

```
1: void CTestAppDoc::DeleteContents()
2: {
3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein und/oder
4:   // rufen Sie die Basisklasse auf.
5:   // Zeichnung löschen
6:   m_maDrawing.ClearDrawing();
7:
8:   CDocument::DeleteContents();
9: }
```

Eine Zeichnung anzeigen

In die Testanwendung ist noch eine letzte Funktionsgruppe aufzunehmen, bevor Sie Ihre DLL testen können: die Zeichenfunktionalität für die Anwendung. Diese Funktionalität gehört in die Ansichtsklasse, da ja das Objekt weiß, wann es sich selbst neu zeichnen muss. Bevor Sie die Funktionalität in der Ansichtsklasse realisieren können, müssen Sie eine Möglichkeit schaffen, damit die Ansichtsklasse auf das Zeichenobjekt zugreifen kann. Am einfachsten lässt sich das mit einer weiteren Funktion in der Dokumentklasse erreichen, die als Rückgabewert einen Zeiger auf das Zeichenobjekt liefert. Nachdem die Ansicht über diesen Zeiger verfügt, kann sie die zum Zeichenobjekt gehörende Draw-Funktion aufrufen.

Um Ihrer Dokumentklasse zu ermöglichen, einen Zeiger auf das Zeichenobjekt abzurufen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Spezifizieren Sie den Funktionstyp als CModArt*, den Namen als GetDrawing und den Zugriff als public.

3. In die Funktion übernehmen Sie folgenden Code:

```
CModArt* CTestAppDoc::GetDrawing()
{
  // Zeichenobjekt zurückgeben
  return &m_maDrawing;
}
```

Um die Zeichenfunktionalität der Ansichtsklasse zu realisieren, bearbeiten Sie einfach die Funktion OnDraw in der Ansichtsklasse. Die Funktion holt einen Zeiger auf das Zeichenobjekt und ruft dann dessen Draw-Funktion auf. Bearbeiten Sie also die Funktion OnDraw wie folgt:

```
void CTestAppView::OnDraw(CDC* pDC)
{
  CTestAppDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  // TODO: Code zum Zeichnen der ursprünglichen Daten hinzufügen
  // Zeichenobjekt holen
  CModArt* pDrawing = pDoc->GetDrawing();
  // Zeichnung zeichnen
  pDrawing->Draw(pDC);
}
```

Nachdem Sie die gesamte Funktionalität hinzugefügt haben, können Sie die Anwendung kompilieren, um die Funktionalität der DLL zu testen. Nachdem Sie die Testanwendung kompiliert haben, wechseln Sie zum Explorer, suchen die DLL im Debug- Unterverzeichnis des DLL-Projektverzeichnisses und kopieren sie in das Debug- Unterverzeichnis im Projektverzeichnis der Testanwendung. Fehlt die DLL im Verzeichnis so wird direkt beim Starten der Anwendung eine Fehlermeldung wie in Abbildung 14.4 angezeigt.

Abbildung 14.4: Fehlermeldung durch fehlende DLL

Nun sollten Sie Ihre Anwendung und die DLL ausführen können. Immer, wenn Sie Neu aus dem Datei-Menü der Anwendung wählen, wird eine neue Zeichnung wie in Abbildung 14.5 erzeugt.

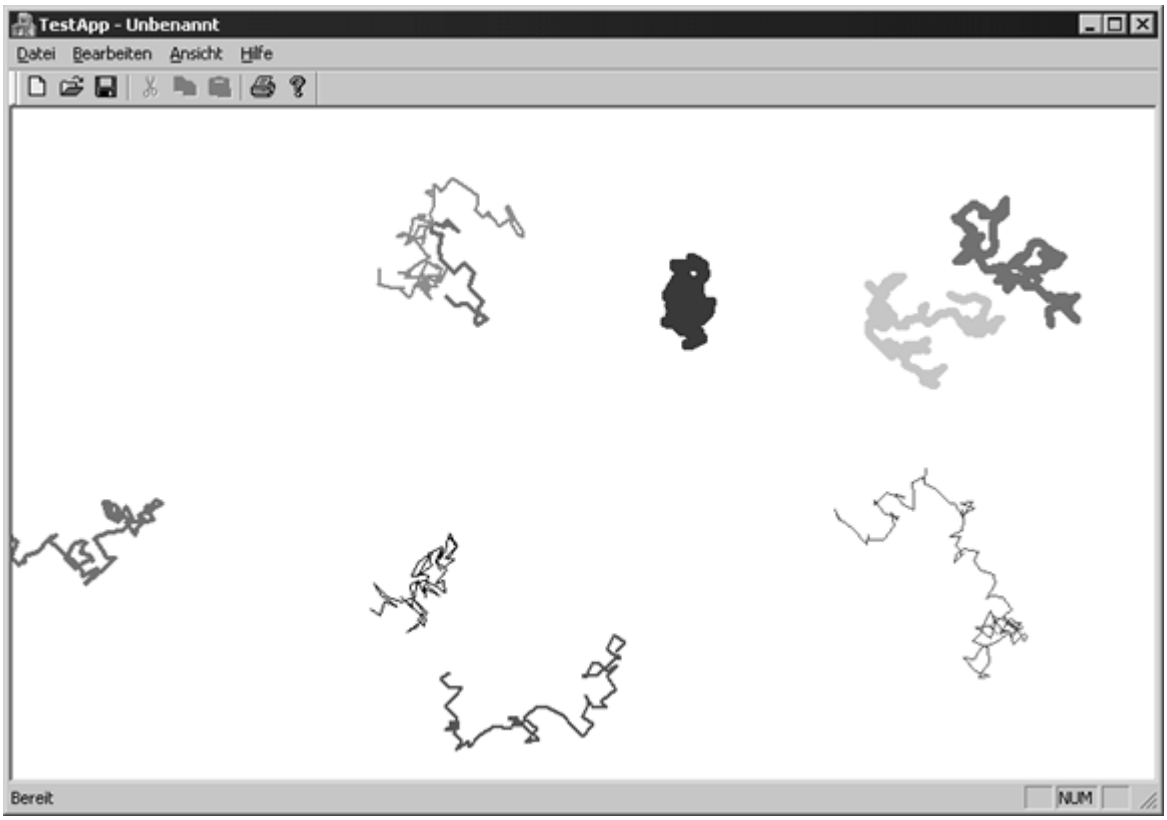


Abbildung 14.5: Zufällige Schnörkelzeichnungen erzeugen

14.4 Eine Standard-DLL erstellen und einsetzen

Vielleicht kommt es Ihnen so vor, als ob Sie die Verwendungsregeln der nicht von der Anwendung besessenen Variablen in der DLL verletzten, als Sie die erweiterte MFC-DLL erzeugt und erstellt haben. Das ist allerdings nicht der Fall. Die Instanz der Zeichenklasse ist ein Element der Dokumentklasse in der Testanwendung. Sie wird von der Anwendung und nicht von der DLL erzeugt und verwaltet. Wenn Sie jetzt darangehen, die gleiche Funktionalität in einer normalen DLL zu implementieren, wird das deutlicher.

Um die erweiterte MFC-DLL in eine Standard-DLL umzuwandeln, müssen Sie die Zeichenklasse in eine Folge von normalen Funktionsaufrufen konvertieren. Das Objektarray muss im Zuge dieser Umwandlung zu einer Elementvariablen der Dokumentklasse der Anwendung werden und ist als Argument an jede exportierte Funktion in der DLL zu übergeben.

Die Standard-DLL erstellen

Beginnen Sie mit einem neuen Projekt, um die erweiterte MFC-DLL in eine Standard- DLL zu konvertieren. Visual C++ muss ein Projekt erstellen, das dem Compiler mitteilt, welcher Dateityp erstellt wird. Dieses neue Projekt können Sie in den gleichen Schritten anlegen, wie Sie das Projekt der erweiterten MFC-DLL erstellt haben. Allerdings geben Sie im DLL-Assistenten an, dass Sie eine Standard-DLL erzeugen möchten. (Damit können Sie die Standardeinstellungen des Assistenten übernehmen.) Nachdem Sie das Projekt angelegt haben, können Sie die Quellcode- und Header-Dateien der Linien- und Zeichenklasse in das Projektverzeichnis kopieren und diese Dateien in das Projekt einfügen. Anschließend beginnen Sie, die Zeichenklasse in eine Folge einfacher Funktionsaufrufe umzuwandeln.

Um mit dem Projekt für das Beispiel zu beginnen, folgen Sie diesen Schritten:

1. Erstellen Sie ein neues Projekt und geben Sie dabei an, dass es sich um ein MFC- DLL-Projekt handelt. Nennen Sie das Projekt ModArtDll2 und klicken Sie auf OK, um es zu erzeugen.

2. Wählen Sie im Bereich Anwendungseinstellungen den Eintrag Reguläre DLL, die eine öffentliche MFC-DLL verwendet aus. Klicken Sie auf Fertig stellen, um das Projekt zu erzeugen.
3. Da Sie die Klassen CLine und CModArt bereits erstellt haben, kopieren Sie sie (Line.h, Line.cpp,

ModArt.h und ModArt.cpp) aus dem ModArt-Projektbereich in das Verzeichnis dieses Projekts.
4. Fügen Sie die Header- und Quellcode-Dateien in dieses Projekt ein, indem Sie Vorhandenes Element hinzufügen aus dem Menü Projekt wählen.

Die Header-Datei abändern

Zunächst einmal müssen Sie die Header-Datei für die Zeichenklasse radikal ändern, damit sie sich für eine Standard-DLL eignet. Dabei müssen Sie alle Spuren der eigentlichen Klasse aus der Header-Datei eliminieren und nur die Funktionsaufrufe beibehalten. Allen Funktionen sind die jeweils erforderlichen Objekte zu übergeben. (Ein Argument jeder Funktion ist das Objektarray.) Als Nächstes modifizieren Sie alle Funktionsnamen etwas, sodass der Compiler nicht durcheinander kommt und irrtümlich eine Elementfunktion irgendeiner Klasse aufruft (wie es zum Beispiel bei der Funktion Serialize der Fall ist). Schließlich sind alle öffentlichen Funktionen als exportierbare Funktionen zu deklarieren. Um diese Änderungen an der Datei ModArt.h vorzunehmen, öffnen Sie die Datei und ersetzen die gesamte Klassendeklaration durch folgende Funktionsprototypen:

```
extern "C" void PASCAL EXPORT ModArtNewDrawing(CRect* pRect,
           CObArray* poaLines);
extern "C" void PASCAL EXPORT ModArtSerialize(CArchive& ar,
           CObArray* poaLines);
extern "C" void PASCAL EXPORT ModArtDraw(CDC* pDC,
           CObArray* poaLines);
extern "C" void PASCAL EXPORT ModArtClearDrawing(
           CObArray* poaLines);
void NewLine(CRect* pRect, CObArray* poaLines);
```

Das Objektarray ist an alle diese Funktionen immer als Zeiger zu übergeben. Da die Funktionen Objekte in das Array einfügen und daraus löschen, müssen sie mit dem eigentlichen Array und nicht mit einer Kopie arbeiten.

Die Funktionen zum Generieren der Zeichnung anpassen

Bei der Umwandlung der Quellcode-Datei (ModArt.cpp) sind zahlreiche kleine und doch signifikante Änderungen an diesen Funktionen vorzunehmen. Beginnen wir bei der Funktion NewDrawing. Hier müssen Sie einen Zeiger auf das CRect-Objekt übergeben, um den Zeichenbereich zu erhalten. Die Funktion für das Festlegen des Zeichenbereichs löschen Sie, da es keine lokalen Variablen gibt, in denen Sie dieses Objekt aufnehmen könnten. Letztendlich sind Sie besser dran, wenn Sie es an die Funktionen zur Generierung der Zeichnung übergeben. Die andere Änderung betrifft die Stelle, wo Sie das Objektarray als weiteres Argument an die Funktion übergeben. Mit diesen Argumenten unternehmen Sie in dieser Funktion nichts weiter, sondern reichen sie lediglich an die Funktion, die das Squiggle generiert, durch. Darüber hinaus ist in dieser Funktion das Makro AFX_MANAGE_STATE als erste Zeile im Rumpf der Funktion hinzuzufügen. Nachdem Sie diese Änderungen vorgenommen haben, sollte die Funktion NewDrawing wie in Listing 14.7 aussehen.

Listing 14.7: Die Funktion ModArtNewDrawing

```
 1: extern "C" void PASCAL EXPORT ModArtNewDrawing(CRect* pRect,
 2:            CObArray* poaLines)
 3: {
 4:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 5:   // Hier normaler Funktionsrumpf
 6:   int iNumLines;
 7:   int iCurLine;
 8:
 9:   // Überprüfen, ob wir einen gültigen Zeiger haben
10:   if (!poaLines) return;
11:
```

```
12:   // Zufallszahlengenerator initialisieren
13:   srand((unsigned)time(NULL));
14:   // Anzahl der zu erzeugenden Linien ermitteln
15:   iNumLines = rand() % 10;
16:   // Sind Linien zu erzeugen?
17:   if (iNumLines > 0)
18:   {
19:     // Schleife durch Anzahl der Linien
20:     for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
21:     {
22:       // Neue Linie erzeugen
23:       NewLine(pRect, poaLines);
24:     }
25:   }
26: }
```

In der Funktion NewDrawing ist eine weitere Änderung hinzugekommen: die Initialisierung des Zufallszahlengenerators in der folgenden Zeile:

```
srand((unsigned)time(NULL));
```

Da es keinen Klassenkonstruktor mehr gibt, können Sie ihm auch keinen Startwert für den Zufallszahlengenerator übergeben. Demzufolge kommt als nächster logischer Platz die Funktion NewDrawing in Frage, bevor irgendwelche Zufallszahlen erzeugt werden.

In der Funktion NewLine werden umfangreichere Änderungen erforderlich.

- Das CRect-Objekt und das Objektarray werden als Zeiger übergeben.
- Da es sich nicht um eine exportierte Funktion handelt, brauchen Sie auch nicht das Makro AFX_MANAGE_STATE hinzuzufügen.
- An allen Stellen, an denen die Elementvariable CRect vorkommt, muss sie durch den CRect-Zeiger ersetzt werden, der als Argument an die Funktion übergeben wird.
- Die Aufnahme von Objekten in das Objektarray ist so zu ändern, dass die als Argument übergebenen Objektarray-Zeiger verwendet werden.

Nach diesen Änderungen sollte die Funktion NewLine Listing 14.8 entsprechen.

Listing 14.8: Die Funktion NewLine

```
 1: void NewLine(CRect* pRect, CObArray* poaLines)
 2: {
 3:   int iNumLines;
 4:   int iCurLine;
 5:   int iCurColor;
 6:   UINT nCurWidth;
 7:   CPoint ptTo;
 8:   CPoint ptFrom;
 9:
10:   COLORREF crColors[8] = {
11:       RGB(   0,   0,   0),    // Schwarz
12:       RGB(   0,   0, 255),    // Blau
13:       RGB(   0, 255,   0),    // Grün
14:       RGB(   0, 255, 255),    // Cyan
15:       RGB( 255,   0,   0),    // Rot
16:       RGB( 255,   0, 255),    // Magenta
17:       RGB( 255, 255,   0),    // Gelb
18:       RGB( 255, 255, 255)     // Weiß
19:   };
20:
21:   // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
22:   pRect->NormalizeRect();
23:   // Breite und Höhe des Zeichenbereichs ermitteln
```

```
24:   int iWidth = pRect->Width();
25:   int iHeight = pRect->Height();
26:
27:   // Anzahl der Teile dieses Squiggles bestimmen
28:   iNumLines = rand() % 100;
29:   // Umfasst das Squiggle mindestens ein Teil?
30:   if (iNumLines > 0)
31:   {
32:     // Farbe bestimmen
33:     iCurColor = rand() % 8;
34:     // Stiftbreite bestimmen
35:     nCurWidth = (rand() % 8) + 1;
36:     // Anfangspunkt für Squiggle bestimmen
37:     ptFrom.x = (rand() % iWidth) + pRect->left;
38:     ptFrom.y = (rand() % iHeight) + pRect->top;
39:     // Schleife durch Anzahl der Segmente
40:     for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
41:     {
42:       // Endpunkt des Segments bestimmen
43:       ptTo.x = ((rand() % 20) - 10) + ptFrom.x;
44:       ptTo.y = ((rand() % 20) - 10) + ptFrom.y;
45:       // Neues CLine-Objekt erzeugen
46:       CLine *pLine = NULL;
47:       try
48:       {
49:         pLine = new CLine(ptFrom, ptTo, crColors[iCurColor],
50:                           nCurWidth);
51:         // Neue Linie in das Objektarray einfügen
52:         poaLines->Add(pLine);
53:       }
54:       // Speicherausnahme?
55:       catch (CMemoryException* perr)
56:       {
57:         // Meldung an den Benutzer mit schlechten Neuigkeiten
58:         AfxMessageBox("Out of memory", MB_ICONSTOP | MB_OK);
59:         // Wurde Linienobjekt erzeugt?
60:         if (pLine)
61:         {
62:           // Löschen
63:           delete pLine;
64:           pLine = NULL;
65:         }
66:         // Ausnahmeobjekt löschen
67:         perr->Delete();
68:       }
69:       // Anfangspunkt auf Endpunkt setzen
70:       ptFrom = ptTo;
71:     }
72:   }
73: }
```

Die übrigen Funktionen anpassen

Die erforderlichen Änderungen an den anderen Funktionen sind weniger kompliziert als die Änderungen an den Funktionen zur Generierung der Zeichnung. Bei den übrigen Funktionen ist ein Zeiger auf das Objektarray als Funktionsargument hinzuzufügen und dann die Verwendung des Arrays auf diese Zeiger anstelle der nicht mehr existierenden Elementvariablen umzustellen. Außerdem müssen Sie das Makro AFX_MANAGE_STATE als erste Zeilen in allen verbleibenden Funktionen hinzufügen.

1. Bearbeiten Sie die restlichen Funktionen wie in den Listing 14.9, 14.10 und 14.11.

Listing 14.9: Die Funktion ModArtDraw

```
 1: extern "C" void PASCAL EXPORT ModArtDraw(CDC* pDC,
 2:            CObArray* poaLines)
 3: {
 4:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 5:   // Überprüfen, ob wir einen gültigen Zeiger haben
 6:   if (!poaLines) return;
 7:
 8:   // Anzahl der Linien im Objektarray ermitteln
 9:   int iCount = poaLines->GetSize();
10:   int iPos;
11:
12:   //Schleife durch Array, dabei jedes Objekt zeichnen
13:   for (iPos = 0; iPos < iCount; iPos++)
14:     ((CLine*)poaLines->GetAt(iPos))->Draw(pDC);
15: }
```

Listing 14.10: Die Funktion ModArtSerialize

```
1: extern "C" void PASCAL EXPORT ModArtSerialize(CArchive& ar,
2:           CObArray* poaLines)
3: {
4:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
5:   // Überprüfen, ob wir einen gültigen Zeiger haben
6:   if (!poaLines) return;
7:   // Archivobjekt an Array übergeben
8:   poaLines->Serialize(ar);
9: }
```

Listing 14.11: Die Funktion ModArtClearDrawing

```
 1: extern "C" void PASCAL EXPORT ModArtClearDrawing(
 2:                                 CObArray* poaLines)
 3: {
 4:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 5:   // Überprüfen, ob wir einen gültigen Zeiger haben
 6:   if (!poaLines) return;
 7:
 8:   // Anzahl der Linien im Objektarray ermitteln
 9:   int iCount = poaLines->GetSize();
10:   int iPos;
11:
12:   // Schleife durch Array, dabei jedes Objekt löschen
13:   for (iPos = 0; iPos < iCount; iPos++)
14:     delete poaLines->GetAt(iPos);
15:   // Array zurücksetzen
16:   poaLines->RemoveAll();
17: }
```

2. Nachdem Sie die Änderungen an diesen Funktionen ausgeführt haben, ist nur noch der gesamte Code für den Konstruktor und den Destruktor der Klasse sowie der Code für die Funktion SetRect zu löschen.

Die Moduldefinitionsdatei erstellen

Bevor Sie die DLL kompilieren, müssen Sie alle Funktionsnamen in die Moduldefinitionsdatei aufnehmen. Diese Datei finden Sie in der Liste der Quelldateien des Projektmappen-Explorers. Wenn Sie diese Datei öffnen, sehen Sie, dass sie kurz das zu erstellende Modul in allgemeiner Form beschreibt. Am Ende der Datei finden Sie eine Stelle, wo Sie die Exportanweisungen für die DLL unterbringen können. Um die exportierbaren Funktionen hinzuzufügen, bearbeiten Sie die Datei ModArtDll2.def wie folgt:

```
; ModArt2.def : Deklariert die Modulparameter für die DLL.
```

```
LIBRARY      "ModArt2"
EXPORTS
    ; Explizite Exporte können hier eingefügt werden
    ModArtNewDrawing
    ModArtSerialize
    ModArtDraw
    ModArtClearDrawing
```

Jetzt können Sie Ihre Standard-DLL kompilieren. Anschließend kopieren Sie sie in das Debug-Verzeichnis der Testanwendung.

Die Testanwendung anpassen

Um die Testanwendung an den Einsatz der neuen DLL, die Sie gerade erstellt haben, anzupassen, sind eine Reihe von Änderungen erforderlich:

1. Ändern Sie die Elementvariable der Dokumentklasse von einer Instanz der Zeichenklasse in das Objektarray.

2. Löschen Sie die Header-Datei ModArt.h aus dem Projekt und fügen Sie die neue Header-Datei ModArt.h als Include-Direktive im Quellcode des Dokuments und der Ansicht ein. Löschen Sie die Include-Direktive für die Header-Datei aus dem Header der Dokumentklasse.
3. Löschen Sie die DLL-LIB-Datei und fügen Sie die LIB-Datei für die neue DLL in das Projekt ein. Ändern Sie alle Funktionsaufrufe der Zeichenklasse in Aufrufe von Funktionen in der neuen DLL.
4. Ändern Sie die Funktion GetDrawing in der Dokumentklasse, sodass sie einen Zeiger auf das Objektarray statt auf das Zeichenobjekt zurückgibt.

Löschen Sie als Erstes die LIB-Datei aus dem Projekt der Testanwendung. Dann folgen Sie diesen Schritten:

1. Fügen Sie die LIB-Datei für die neue DLL mit Projekt / Vorhandenes Element hinzufügen in das Projekt ein.

2. Bearbeiten Sie den Quellcode für die Dokument- und Ansichtsklassen, um die Include-Anweisung für die neue Header-Datei ModArt.h einzufügen:

```
// TestAppDoc.cpp : Implementierung der Klasse CTestAppDoc
//
#include "stdafx.h"
#include "TestApp.h"
#include "TestAppDoc.h"
#include "../ModArtDll2/ModArt.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

3. Nach diesen Änderungen öffnen Sie die Header-Datei für die Dokumentklasse.
4. Bearbeiten Sie die Deklaration der Dokumentklasse wie folgt: Ändern Sie den Typ der Funktion GetDrawing, sodass die Funktion einen Zeiger auf ein Objektarray zurückgibt. Entfernen Sie die Variable der Zeichenklasse und fügen Sie eine Objektarray-Variable hinzu, wie es aus Listing 14.12 hervorgeht. Nehmen Sie nur diese drei Änderungen vor, alles andere in der Klassendeklaration bleibt erhalten.

Listing 14.12: Die Deklaration der Klasse CTestAppDoc

```
 1: class CTestAppDoc : public CDocument
 2: {
 3: protected: // Nur aus Serialisierung erstellen
 4:     CTestAppDoc();
 5:     DECLARE_DYNCREATE(CTestAppDoc)
 6:
 7: // Attribute
 8: public:
```

```
 9:
10: // Operationen
11: public:
12:
13: // Überschreibungen
14: public:
15:     virtual BOOL OnNewDocument();
16:     virtual void Serialize(CArchive& ar);
17:
18: // Implementierung
19: public:
20:     virtual ~CTestAppDoc();
21: #ifdef _DEBUG
22:     virtual void AssertValid() const;
23:     virtual void Dump(CDumpContext& dc) const;
24: #endif
25:
26: protected:
27:
28: // Generierte Funktionen für die Meldungstabellen
29: protected:
30:     DECLARE_MESSAGE_MAP()
31: private:
32:     // Die DLL zum Zeichnen
33:     // CModArt m_maDrawing;
34:     // Das Array mit den Squiggles
35:     CObArray m_oaLines;
36: public:
37:     // Zeiger auf das Objektarray holen
38:     CObArray* GetDrawing(void);
39:     virtual void DeleteContents();
40: };
```

Die Funktionen des Dokuments modifizieren

Nachdem Sie die allgemeinen Änderungen an der Testanwendung vorgenommen haben, sind nun die Änderungen in der Funktionalität an der Reihe. Alle Aufrufe der Klassenmethoden des Zeichenobjekts sind in die entsprechenden Aufrufe von Funktionen in der neuen DLL umzuändern.

In der Funktion OnNewDocument löschen Sie den Funktionsaufruf, der das CRect-Objekt an das Zeichenobjekt übergibt, und ersetzen den Aufruf der Funktion NewDocument durch die neue DLL-Funktion - in diesem Fall ModArtNewDrawing, wie es Listing 14.13 zeigt.

Listing 14.13: Die Funktion CTestAppDoc.OnNewDocument

```
 1: BOOL CTestAppDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Position der Ansicht ermitteln
 9:   POSITION pos = GetFirstViewPosition();
10:   // Ist die Position gültig?
11:   if (pos != NULL)
12:   {
13:     // Zeiger auf Ansicht holen
14:     CView* pView = GetNextView(pos);
15:     RECT rWndRect;
16:     // Anzeigerechteck ermitteln
17:     pView->GetClientRect(&rWndRect);
18:     // Neue Zeichnung erzeugen
```

```
19:     ModArtNewDrawing((CRect*)&rWndRect, &m_oaLines);
20:   }
21:
22:   return TRUE;
23: }
```

In der Funktion Serialize ändern Sie den Aufruf der Serialize-Funktion des Zeichenobjekts in die neue Serialisierungsfunktion der DLL - in diesem Fall ModArtSerialize:

```
void CTestAppDoc::Serialize(CArchive& ar)
{
  // Zeichnung serialisieren
  ModArtSerialize(ar, &m_oaLines);
}
```

In der Funktion DeleteContents ist der Aufruf der Funktion ClearDrawing durch den Aufruf der neuen DLL-Funktion ModArtClearDrawing zu ersetzen:

```
void CTestAppDoc::DeleteContents()
{
  // TODO: Fügen Sie hier Ihren spezialisierten Code ein
  // und/oder rufen Sie die Basisklasse auf.
  // Zeichnung löschen
  ModArtClearDrawing(&m_oaLines);
  CDocument::DeleteContents();
}
```

Schließlich müssen Sie für die Funktion GetDrawing die Funktionsdeklaration ändern, um zu kennzeichnen, dass sie einen Zeiger auf ein Objektarray zurückgibt, genau wie Sie es in der Header-Datei vorgenommen haben. Als Nächstes ändern Sie die Variable, die zurückgegeben wird, in die Objektarray-Variable, die Sie in die Header-Datei aufgenommen haben:

```
CObArray* CTestAppDoc::GetDrawing()
{
  // Zeichenobjekt zurückgeben
  return &m_oaLines;
}
```

Die Funktionen der Ansicht modifizieren

In der Ansichtsklasse ist nur eine einfache Änderung an der Funktion OnDraw vorzunehmen. In dieser Funktion ist der Typ des Zeigers, der aus der Funktion GetDrawing abgerufen wird, von einem Zeichenobjekt in ein Objektarrayobjekt zu ändern, wie es Listing 14.14 zeigt. Unmittelbar danach rufen Sie die DLL-Funktion ModArtDraw auf, um das Zeichnen im Fenster auszuführen.

Listing 14.14: Die Funktion CTestAppView.OnDraw

```
 1: void CTestAppView::OnDraw(CDC* pDC)
 2: {
 3:   CModTestAppDoc* pDoc = GetDocument();
 4:   ASSERT_VALID(pDoc);
 5:
 6:   // TODO: Code zum Zeichnen der ursprünglichen Daten hinzufügen
 7:   // Zeichenobjekt holen
 8:   CObArray* poaLines = pDoc->GetDrawing();
 9:   // Zeichnung zeichnen
10:   ModArtDraw(pDC, poaLines);
11: }
```

Nachdem Sie alle Änderungen an der Testanwendung vorgenommen haben, können Sie die Anwendung kompilieren und testen. Die Anwendung sollte genauso arbeiten wie mit der vorherigen DLL. Sie können auch damit experimentieren, indem Sie zurückgehen und die DLL ändern, die neue DLL in das Debug-Verzeichnis

für die Testanwendung kopieren und verfolgen, wie sich die Änderungen im Verhalten der Testanwendung widerspiegeln.

Die im heutigen Beispiel entwickelte Version einer Standard-DLL ist noch nicht mit anderen Programmiersprachen einsetzbar. Der Grund dafür liegt darin, dass Sie MFC-Klassen als Argumente für alle Funktionen der DLL übergeben. Damit ist die Einsatzbreite auf Anwendungen beschränkt, die mittels MFC erstellt wurden. Um diese DLL tatsächlich portierbar zu machen, müssen Sie die nackten Strukturen anstelle der Klassen übergeben (beispielsweise die RECT-Struktur anstelle der CRect-Klasse) und dann die Strukturen innerhalb der DLL in Klassen konvertieren.

14.5 Zusammenfassung

Heute haben Sie zwei Möglichkeiten kennen gelernt, wie Sie Ihre Funktionalität für andere Programmierer verpacken können. Es wurde dargestellt, wie leicht sich Klassen in einer erweiterten MFC-DLL unterbringen lassen und wie einfach man sie in einer Visual C++-Anwendung einsetzen kann. Sie haben gesehen, wie man Änderungen an der DLL vornehmen kann, ohne die darauf zugreifenden Anwendungen erneut kompilieren zu müssen. Weiterhin haben Sie gelernt, was zum Erstellen einer Standard-DLL gehört, die sich mit anderen - nicht mit Visual C++ erstellten - Anwendungen einsetzen lässt. Sie haben gesehen, wie Sie die exportierten Klassen aus der DLL in normale Funktionen nach C-Konventionen konvertieren mussten und was erforderlich ist, um eine Anwendung, die diese Art von DLL verwendet, anzupassen.

14.6 Workshop

Fragen und Antworten

Frage:
Wie kann ich kontrollieren, welche DLLs von meiner Applikation geladen wurden?

Antwort:
Über die Modulansicht: Debuggen / Fenster / Module.

Frage:
Wie kann ich die Standard-DLL konvertieren, sodass sie sich in Anwendungen einsetzen lässt, die nicht mit Visual C++ erstellt wurden?

Antwort:
Zuerst sind alle Argumente an die Funktionen mit reinen Strukturen anstelle von MFC-Klassen zu übergeben. Um zum Beispiel die Funktion ModArtNewDrawing umzuwandeln, ändern Sie sie dahin gehend, dass sie die RECT-Struktur anstelle der CRect-Klasse sowie einen generischen Zeiger anstelle eines Zeigers auf ein Objektarray erhält. Die Umwandlungen sind in den entsprechenden Klassen in der DLL vorzunehmen, wie es Listing 14.15 zeigt.

Listing 14.15: Die Funktion ModArtNewDrawing

```
 1: extern "C" void PASCAL EXPORT ModArtNewDrawing(RECT* pRect,
 2:             LPVOID lpoaLines)
 3: {
 4:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 5:   CRect rRect;
 6:   rRect.top = pRect->top;
 7:   rRect.left = pRect->left;
 8:   rRect.right = pRect->right;
 9:   rRect.bottom = pRect->bottom;
10:   CObArray* poaLines = (CObArray*)lpoaLines;
```

```
11:    // Normal function body here
12:    int iNumLines;
13:    int iCurLine;
14:
15:    // Zufallszahlengenerator initialisieren
16:    srand((unsigned)time(NULL));
17:    // Anzahl der zu erzeugenden Linien bestimmen
18:    iNumLines = rand() % 10;
19:    // Sind Linien zu erzeugen?
20:    if (iNumLines > 0)
21:    {
22:      // Schleife durch Anzahl der Linien
23:      for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
24:      {
25:        // Neue Linie erzeugen
26:        NewLine(&rRect, poaLines);
27:      }
28:    }
29: }
```

Antwort:
Darüber hinaus müssen Sie bei der Anwendung, die das Objektarray als generischen Zeiger speichert, Funktionen hinzufügen, um dieses zu erzeugen und zu zerstören:

```
extern "C" LPVOID PASCAL EXPORT ModArtInit()
{
  AFX_MANAGE_STATE(AfxGetStaticModuleState());
  // Create the object array
  return (LPVOID)new CObArray;
}
```

Antwort:
Wenn Sie diese Änderungen vorgenommen haben, kann die Client-Anwendung eine long-Variable enthalten, die der DLL an Stelle des Objektarrays übergeben wird. Diese Variable enthält einen Zeiger auf das von der DLL instanziierte Objektarray.

Frage:
Wann muss ich die Anwendungen, die meine DLL verwenden, neu kompilieren?

Antwort:
Immer dann, wenn Sie die Aufrufe von exportierten Funktionen ändern. Wenn Sie Argumente in diesen Funktionen ändern, hinzufügen oder entfernen, müssen Sie die Anwendungen, die auf die DLL zurückgreifen, neu kompilieren. Arbeiten Sie mit einer erweiterten MFC-DLL, ist die Anwendung, die die DLL verwendet, neu zu kompilieren, wenn sich die öffentliche Schnittstelle für die exportierten Klassen ändert oder eine neue Funktion oder Variable hinzugefügt oder entfernt wird. Es spielt keine Rolle, wenn die Anwendung gar nicht auf die geänderten Funktionen zurückgreift. Allerdings ist es übliche Praxis, die Anwendungen neu zu kompilieren, einfach um sicher zu sein.

Quiz

1. Welche Art von DLL müssen Sie erstellen, um Klassen in der DLL für die Anwendungen verfügbar zu machen?

2. Was müssen Sie in die Klasse einfügen, um sie aus einer DLL zu exportieren?
3. Welche Art von DLL lässt sich mit anderen Programmiersprachen nutzen?
4. Müssen Sie Anwendungen, die auf eine DLL zurückgreifen, neu kompilieren, wenn Sie die betreffende DLL verändern?
5. Welche Funktion stellt die LIB-Datei für eine DLL bereit?

Übungen

1. Spalten Sie die Linienklasse in ihre eigene erweiterte MFC-DLL auf und verwenden Sie sie mit der zweiten (Standard-) DLL.

2. Ändern Sie die DLL für die Linienklasse, sodass sie eine einheitliche Linienbreite für alle Linien verwendet.

Woche 2 im Rückblick

Mit Abschluss der zweiten Woche sollten Sie sich an die Arbeit mit Visual C++ gewöhnt haben. Sie sollten verstehen, wie man die MFC-Klassenhierarchie verwendet, um ein gerüttelt Maß an Funktionalität in Ihren Anwendungen bereitzustellen. Sie sollten auch wissen, mit wie viel unterstützender Infrastruktur Ihre Anwendungen starten, wenn Sie sie mit den Visual C++-Wizards erstellen.

An dieser Stelle empfiehlt es sich, das bisher Gelernte selbst auszuprobieren. Erstellen Sie eine MDI-Anwendung und setzen Sie dazu einen Dokumenttyp ein, den Sie selbst entwickeln. Sehen Sie sich an, wie man das Dokument speichern und wiederherstellen sowie verwalten kann. Eigene Übungen sind der Schlüssel dazu, Ihre bisherigen Kenntnisse zu vertiefen. Dabei erkennen Sie auch Schwachstellen und können Themen wiederholen, die Ihnen noch nicht geläufig sind, aber Sie wissen auch, wo Sie bereits sattelfest sind.

Das Erscheinungsbild Ihrer Anwendungen lässt sich mit Text und Schriften flexibler gestalten - und Sie können dem Benutzer die Möglichkeit bieten, das Aussehen der Anwendung an die eigenen Vorstellungen anzupassen. Sie wissen, wie man die verfügbaren Schriften des Computers, auf dem die Anwendung läuft, untersucht, und falls eine von der Anwendung genutzte Schrift nicht verfügbar ist, eine andere Schrift auswählt, die der geforderten möglichst nahe kommt. Sollten noch Fragen in Bezug auf die Infrastruktur von Schriften unter Windows bestehen und wie Sie die Schriften in Ihren Anwendungen einsetzen, wiederholen Sie das Material von Tag 8, »Text und Schriften«.

Die Dokument-/View-Architektur ist Ihnen kein Buch mit sieben Siegeln mehr. Mit dieser Architektur verwalten Sie die Trennung der eigentlichen Daten von der Darstellung der Daten, die für den Benutzer angezeigt werden. Auf der Basis dieses Modells haben Sie sowohl SDI- (Single Document Interface) als auch MDI-(Multiple Document Interface) Anwendungen entwickelt und eingesetzt, um Dateien auf den Datenträger zu schreiben und wieder in die Anwendung einzulesen. Diese Architektur gehört zu den Grundpfeilern beim Erstellen von MFC-Anwendungen mit Visual C++. Sie wissen, wo man die Initialisierungen für einen neuen Satz von Daten unterbringt und wo man Aufräumarbeiten erledigen muss, wenn man einen Satz von Daten schließt.

Weiterhin kennen Sie die Grundzüge von SDI- und MDI-Anwendungen und wissen, wie sie sich untereinander und von dialogbasierten Anwendungen unterscheiden. Sie haben bereits ein Gefühl dafür entwickelt, wann eine zu erstellende Anwendung in einem dieser Stile zu erstellen ist und wann sich ein anderer Stil anbietet. Sie können problemlos Ihre eigenen SDI- und MDI-Anwendungen je nach Ihren Anforderungen erstellen. Falls noch irgendwelche Fragen zu diesen Themen auftauchen, sehen Sie sich noch einmal in den Lektionen 10, »SDI- und MDI-Anwendungen«, an, wie die Dokument-/View-Architektur sowohl in SDI- als auch in MDI-Anwendungen funktioniert.

Zu Ihren Kenntnissen gehört mittlerweile auch, dass Sie in SDI- und MDI-Anwendungen komplexe Datenstrukturen in Dateien auf der Festplatte des Systems speichern und in die Anwendung einlesen können. Sie erstellen Objekte mit gemischten Typen, die Sie im Dokumentobjekt Ihrer Anwendung erzeugen und verwalten. Dazu gehört auch der Umgang mit der Funktion Serialize in Verbindung mit dem CArchive-Objekt, um die Objekte in eine Datei zu schreiben und zu einem späteren Zeitpunkt wiederherstellen zu können. Bei Problemen in diesem Themenbereich wiederholen Sie Tag 12, »Serialisierung«.

Beim Themenkomplex »Lesen und Schreiben von Dateien« haben Sie gelernt, wie man Symbolleisten für SDI- und MDI-Anwendungen entwirft und erstellt. Nunmehr können Sie eigene Symbolleisten entwerfen und erstellen sowie in Ihren Anwendungen einsetzen. Ihnen ist bekannt, dass die ID der Symbolleisten-Schaltfläche mit der ID des Menüs, für das die Schaltfläche als Ersatz gedacht ist, übereinstimmen muss. Außerdem wissen Sie, wie man eigene benutzerdefinierte Elemente in der Statusleiste von SDI- und MDI-Anwendungen erzeugt und verwendet. Die Nachricht UPDATE_COMMAND_UI setzen Sie ein, um den Status von Elementen im Menü, auf Symbolleisten und in der Statusleiste auszuwerten und zu ändern sowie deren Aussehen und Zustand in eigener Regie zu verwalten, wobei Sie diese Elemente nicht selbst einrichten müssen. Wenn Sie bezüglich dieser Dinge noch Fragen haben, gehen Sie noch einmal zu Tag 11, »Symbolleisten und Statusleisten« zurück.

Sie haben gelernt, wie man eine einfache Datenbankanwendung erstellt, mit der Sie Daten aus einer Datenbank über die ADO-Schnittstelle abrufen. Ihnen sind jetzt die Grundlagen bekannt, auf denen man Datenbankanwendungen mit diesem Verfahren erstellt, wie man die Daten verwaltet, wie man neue Datensätze hinzufügt und wie man sie löscht. Sie wissen, wie die Recordset-Klasse die gesamte Interaktion mit der Datenbank abwickelt und wie Sie die Daten über dieses Objekt direkt beeinflussen. Falls Ihnen in diesem Themenbereich noch etwas unklar ist, wiederholen Sie den Stoff von Tag 13, »Datenbanken per ADO bearbeiten«.

Sie haben gelernt, durch die Erstellung von DLLs Funktionalität mit anderen Programmierern zu teilen. Dabei haben Sie zwei verschiedene Arten von DLLs erstellt: Der eine Typ lässt sich nur von anderen Visual C++-Anwendungen nutzen, der andere von beliebigen Anwendungen, unabhängig davon, mit welcher Programmiersprache sie erstellt wurden. Sie haben eine DLL erstellt, die andere Visual C++-Programmierer nutzen können, ohne dass Sie gänzlich anders vorgehen mussten, um die Module zu entwerfen oder zu programmieren. Sie haben aber auch gelernt, dass Ihr Modul grundsätzlich anders aufzubauen ist, damit der Einsatz und die Interaktion in DLLs, die man mit allen Programmiersprachen einsetzen kann, gewährleistet sind. Es wurde gezeigt, wie man gewöhnliche Funktionsaufrufe als Schnittstelle für andere Anwendungen bereitstellt, wobei alle erforderlichen Informationen als Parameter an die Funktionen zu übergeben sind. Ein weiteres Thema waren die Definitionsdateien, die die Namen aller zu exportierenden Funktionen enthalten. Sehen Sie sich Tag 14, »DLLs«, noch einmal an, falls Ihnen noch nicht alle Verfahren in diesem Themenbereich geläufig sind.

Schließlich haben Sie mit einer Zeichenanwendung gearbeitet und dabei gelernt, wie man Linien, Kreise und Quadrate mit einer Vielzahl von Stiften und Pinseln zeichnet. Sie haben sogar einen benutzerdefinierten Pinsel aus einer Bitmap erstellt. Es wurde erläutert, wie man eine Bitmap aus einer Datei lädt und für den Benutzer anzeigt. Vor allem aber ging es darum, was ein Gerätekontext ist und wie man ihn einsetzt, um alle diese Features im Fenster der Anwendung darzustellen. Mit den Methoden eines Gerätekontextes können Sie nun diese und andere Figuren zeichnen, um beliebige Grafiken im Fenster zu erstellen, die der Benutzer ansehen und damit in Wechselwirkung treten kann. Wenn hierzu noch Fragen offen sind, wiederholen Sie Tag 9, »Bilder, Zeichnungen und Bitmaps«.

Mittlerweile haben Sie sich ein umfangreiches Repertoire an Programmierkenntnissen mit Visual C++ erarbeitet. Kleinere Programmieraufgaben können Sie nun ohne weiteres lösen und sich möglicherweise auch bereits mit etwas größeren Programmen auseinander setzen. Sie sind auf dem besten Weg, ein perfekter Visual C++-Programmierer zu werden. Also wäre es ganz falsch, jetzt aufzuhören. Es gibt noch viel zu lernen. Nur noch eine Woche liegt vor Ihnen, also los!

Woche 3 im Überblick

In der dritten und letzten Woche lernen Sie einige der komplizierteren Aspekte beim Erstellen von Anwendungen mit Visual C++ kennen. In Ihrer weiteren Programmierpraxis haben Sie mit manchen Themen öfter zu tun als mit anderen. Wenn Visual C++ aber zu Ihrem täglichen Brot gehört, kommen Sie doch auf lange Sicht mit fast allen dieser Bereiche in Berührung.

Sie beginnen die Woche mit Tag 15, »Eigene ActiveX- Steuerelemente«, und lernen, wie Sie eigene Steuerelemente erstellen, die sich in anderen Anwendungen oder sogar in Webseiten einsetzen lassen. Sie definieren die Eigenschaften und Methoden für das Steuerelement und lösen über Ihr Steuerelement Ereignisse in der Containeranwendung aus.

Am Tag 16, »Funktionen und Webbrowser«, lernen Sie, wie einfach es ist, den Webbrowser Internet Explorer von Microsoft in eigene Visual C++- Anwendungen zu integrieren. Sie werden sehen, wie Visual C++ benutzerdefinierte Klassen um die Steuerelemente herum erstellt, die Sie Ihrem Projekt hinzufügen, sodass Sie mit dem Steuerelement wie mit jedem anderen C++-Objekt interagieren können.

Tag 17, »Multitasking«, zeigt, wie sich in einer Anwendung zwei oder mehrere separate Aufgaben gleichzeitig ausführen lassen. Sie lösen Hintergrundverarbeitungen aus, sobald eine Anwendung im Leerlauf arbeitet, und erstellen unabhängige Threads (Programmfäden), die selbst dann arbeiten, wenn die Anwendung beschäftigt ist.

Am Tag 18, »Internet-Anwendungen und Netzwerk-Kommunikation«, lernen Sie, wie Internet-Anwendungen über die Winsock-Schnittstelle untereinander kommunizieren. Sie setzen die gleiche Schnittstelle ein, um in Ihren Anwendungen den Datenaustausch über ein Netzwerk oder auf ein und derselben Maschine zu realisieren.

Am Tag 19, »Verwalteter Code«, lernen Sie, Anwendungen mit verwaltetem Code zu erstellen. Sie lernen den Unterschied zwischen Standard-C++-Anwendungen und mit der neuen Common Language Runtime (CLR) von Microsoft kompilierten Anwendungen mit verwaltetem Code kennen. So können Sie Anwendungen erstellen, die auf mehrere Plattformen portierbar sind, ohne sie neu kompilieren oder Teile der Funktionalität portieren zu müssen.

Am Tag 20, »ATL«, tauchen Sie in die Welt der Programmierung mit Advanced Template Library (ATL) ein. ATL wird für die Erstellung von Komponenten und komponentenbasierenden Anwendungen, die als Dienste laufen, verwendet. Diese Anwendungen besitzen keine Benutzerschnittstelle, können jedoch von Webseiten oder anderen Anwendungen aus aufgerufen werden.

Schließlich lernen Sie am Tag 21, »Interaktion mit Visual Basic .NET- und C # .NET- Komponenten«, wie Sie die Fähigkeiten der CLR ausnutzen, um mit in anderen Sprachen wie Visual Basic oder der neuen Microsoft-

Sprache C# geschriebenen Komponenten zu interagieren. So können Sie die Arbeit in Ihrer Anwendung nutzen, die von anderen in verschiedenen Sprachen programmiert wurde.

Wenn Sie diese letzte Woche abgeschlossen haben, besitzen Sie Kenntnisse über die meisten Bereiche der Visual C++-Programmierung. Obwohl es noch weitere Gebiete und Technologien gibt, die ein tieferes Eindringen erfordern, haben Sie zumindest eine Vorstellung, worum es im Wesentlichen geht. Sie sind darauf vorbereitet, in allen Bereichen der Visual C++-Programmierung zu arbeiten. Und mittlerweile haben Sie vielleicht einige Gebiete herausgegriffen, über die Sie mehr erfahren möchten, als dieses Buch Ihnen vermitteln kann.

Eine Woche ist noch zu meistern, gehen Sie es an.

Tag 15

Eigene ActiveX- Steuerelemente

Die Softwareindustrie hat in den vergangenen Jahren eine Revolution erlebt. Die Art und Weise, wie man Software erstellt und im Paket zusammenschnürt, hat sich von einem Modell, bei dem alle Anwendungen große, monolithische Stücke von ausführbarem Code waren, zu einem Modell entwickelt, bei dem die meisten Anwendungen aus kleinen Bausteinen bestehen. Diese kleinen Bausteine - oftmals als Komponenten bezeichnet - lassen sich mit den unterschiedlichsten Sprachen erstellen und können die verschiedensten Formen annehmen. Eine der bekanntesten Komponenten ist das ActiveX- Steuerelement. Wenn Sie wissen, wie man eigene ActiveX-Steuerelemente erstellt, können Sie eigene Komponenten erzeugen und anderen Programmierern zur Verfügung stellen.

Heute lernen Sie, wie man ...

- mit den Assistenten von Visual C++ ActiveX-Steuerelemente erstellt,
- Eigenschaften und Methoden mit den Visual Studio-Assistenten in die Steuerelemente aufnimmt,
- das Steuerelement mit den Werkzeugen von Visual C++ testet.

15.1 Was ist ein ActiveX-Steuerelement?

Ein ActiveX-Steuerelement ist eine Softwarekomponente, die sich in die verschiedensten Programme einbauen lässt und die man wie einen nativen Teil des Programms verwenden kann. Das Konzept ist mit den Komponenten einer HiFi-Anlage vergleichbar. Wenn man ein neues Kassettendeck kauft, kann man es einfach mit der bereits vorhandenen Anlage verbinden und deren Funktionalität erweitern, ohne die bisherigen Funktionen zu beeinträchtigen. ActiveX-Steuerelemente bringen die gleichen Möglichkeiten der Zusammenarbeit von Komponenten in Softwareanwendungen.

ActiveX-Steuerelemente firmierten ursprünglich unter der Bezeichnung OLE 2.0. Die von Microsoft entwickelte Technologie diente dazu, zwei oder mehr Anwendungen zu kombinieren und wie eine einzige Anwendung erscheinen zu lassen (oder zumindest eine Umschaltung zwischen den verschiedenartigen Anwendungen innerhalb derselben Benutzeroberfläche zu erreichen). OLE 2.0 war eine Erweiterung der ursprünglichen OLE-Technologie (OLE - Object Linking and Embedding, Objekte verknüpfen und einbetten), die es lediglich erlaubte, die mit verschiedenen Anwendungen erzeugten Dokumente zu einem Dokument zusammenzufassen. Beim Aufpolieren der OLE- Technologien für eine Arbeit in einer verteilten Umgebung (wie etwa dem Internet) hat sich Microsoft dafür entschieden, auch einen neuen Namen zu verwenden. Somit war ActiveX geboren.

ActiveX-Steuerelemente stellen eine Reihe von Schnittstellen bereit, auf die die Containeranwendung zurückgreift, um mit den verschiedenen funktionellen Einheiten, die das Steuerelement bietet, zu arbeiten. Viele dieser Schnittstellen dienen dazu, Ereignisse im Steuerelement oder in der Containeranwendung auszulösen. Andere Schnittstellen sind dafür vorgesehen, die Eigenschaftsseite des Steuerelements zu spezifizieren oder um mitzuteilen, ob das Steuerelement aktiviert wurde. Alles in allem sind in den meisten ActiveX-Steuerelementen derartig viele Schnittstellen eingebaut, dass man eine ganze Zeit brauchen würde, um die Funktionalität für jede dieser Schnittstellen selbst zu programmieren. Glücklicherweise fügen der Anwendungs-Assistent und der Klassen-Assistent von Visual C++ den größten Teil dieser Funktionalität automatisch hinzu und gestatten Ihnen damit, sich auf die spezifische Funktionalität, die man vom Steuerelement erwartet, zu konzentrieren.

Für das zu erstellende Steuerelement müssen Sie trotzdem selbst festlegen, welche Eigenschaften, Methoden und Ereignisse Sie dafür freilegen möchten. Diese Elemente können Sie dem Steuerelement über die verschiedenen Assistenten hinzufügen. Wenn aber irgendwelche Eigenschaften oder Ereignisse speziellen Code von Ihrer Seite erfordern, müssen Sie ihn auch selbst hinzufügen. Wie bei allen Methoden, die Sie Ihren Steuerelementen hinzufügen, müssen Sie den gesamten Code bereitstellen. Der Methoden-Assistent baut automatisch die umgebende Struktur und den Code auf, um der Containeranwendung zu

erlauben, die Methode zu sehen und aufzurufen - genauso wie er Ihrer Anwendung den erforderlichen Code hinzufügt, um beliebige Behandlungsroutinen aufzurufen.

ActiveX und die IDispatch-Schnittstelle

Die ActiveX-Technologie setzt auf der COM-Technologie von Microsoft auf (COM - Component Object Model) und stützt sich auf deren Oberfläche und das Interaktionsmodell, um die ActiveX-Steuerelemente nahtlos zu integrieren. Die COM- Technologie definiert, wie ActiveX-Objekte konstruiert sind und wie ihre Schnittstellen auszusehen haben. Die auf COM aufsetzende Schicht legt fest, welche Schnittstellen die verschiedenartigen Objekte unterstützen sollten und wie unterschiedliche Objekttypen miteinander in Wechselwirkung treten.

Die COM-Technologie von Microsoft definiert, wie Komponenten über die Schnittstellen miteinander interagieren können. Eine Schnittstelle verhält sich wie ein Funktionsaufruf an eine ActiveX-Komponente. COM legt dabei fest, wie dieser Funktionsaufruf aufgebaut sein muss, wie der Aufruf zu erfolgen hat und welche unterstützende Funktionalität ihn begleiten muss.

Über Schnittstellen wie IUnknown, die in jedem COM-Objekt erforderlich sind, fragt man die Komponente ab, um herauszufinden, welche anderen Schnittstellen sie bietet. Jede Schnittstelle unterstützt einen speziellen Satz von Funktionen - eine Schnittstelle behandelt etwa die visuelle Erscheinung des Steuerelements, eine andere steuert, wie die Erscheinung des Steuerelements mit der umgebenden Anwendung in Wechselwirkung tritt, eine weitere löst Ereignisse in der umgebenden Anwendung aus usw.

Zu den Schlüsseltechnologien der ActiveX-Steuerelemente gehört die Automatisierung. Damit kann eine Anwendung, die in eine andere Anwendung eingebettet ist, sich selbst aktivieren, ihren Teil der Benutzeroberfläche oder des Dokuments steuern, Änderungen am Dokument vornehmen und sich dann wieder schließen, wenn der Benutzer zu einem anderen Teil der Anwendung geht, der nicht von der eingebetteten Anwendung gesteuert wird.

Diese Vorgänge laufen beispielsweise ab, wenn man ein Excel-Tabellenblatt in ein Word-Dokument einbettet. Ein Klick auf das Tabellenblatt aktiviert Excel. Das Tabellenblatt lässt sich dann mit Excel bearbeiten, selbst wenn man weiterhin in Word arbeitet. Sobald die Änderungen am Tabellenblatt abgeschlossen sind, schließt sich Excel automatisch und man kann in Word weiterarbeiten.

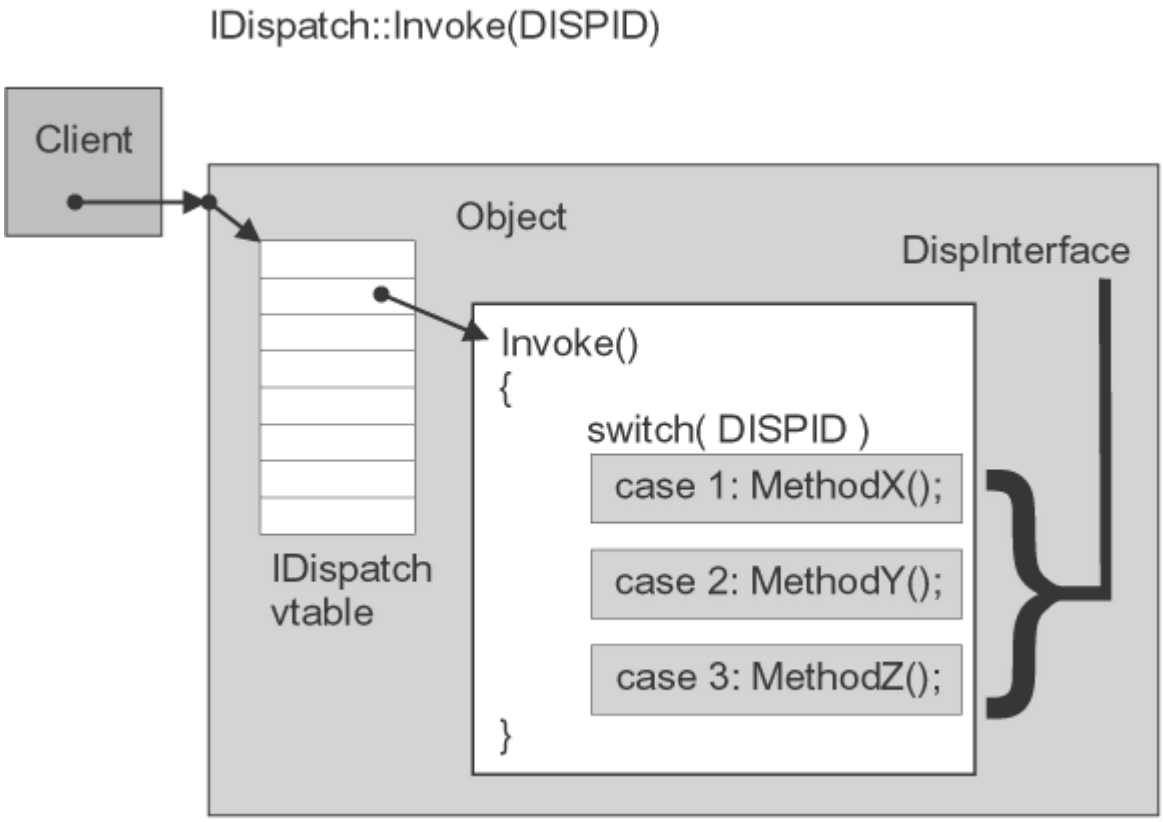


Abbildung 15.1: Die ActiveX-Schnittstelle IDispatch.

Ein Schlüssel zur Arbeitsweise der Automatisierung ist eine spezielle Schnittstelle namens IDispatch (auch als Dispinterface bezeichnet). Die IDispatch-Schnittstelle besteht aus einem Zeiger auf eine Tabelle verfügbarer Methoden, die sich im ActiveX-Steuerelement oder der eingebetteten Anwendung ausführen lassen. Diesen Methoden sind ID-Nummern - so genannte DISPIDs - zugeordnet, die ebenfalls in eine Tabelle geladen werden. In dieser Tabelle kann man dann nach der ID für eine bestimmte Methode suchen. Nachdem man die DISPID für eine bestimmte Methode kennt, kann man diese Methode über die Methode Invoke der IDispatch-Schnittstelle aufrufen. Dabei übergibt man die DISPID, um die auszuführende Methode zu kennzeichnen. Abbildung 15.1 zeigt, wie die IDispatch-Schnittstelle die Methode Invoke verwendet, um Methoden im ActiveX-Objekt auszuführen.

ActiveX-Container und -Server

Um ein ActiveX-Objekt in ein anderes einzubetten, muss man das eingebettete Objekt als ActiveX-Server und das Objekt, das das erste Objekt enthält, als ActiveX-Container implementieren. Ein ActiveX-Objekt, das man in ein anderes einbetten kann, ist ein ActiveX-Server, ob es sich dabei nun um eine vollständige Anwendung oder nur ein kleines ActiveX-Steuerelement handelt. Jedes ActiveX-Objekt, das andere ActiveX-Objekte in sich aufnehmen kann, ist ein ActiveX-Container.

Verwechseln Sie nicht die Begriffe Container und Server mit dem Begriff Client in Abbildung 15.1. Der Client ist das Objekt, das die IDispatch-Schnittstelle des anderen Objekts aufruft. Wie Sie in Kürze sehen werden, rufen sowohl der Container als auch der Server die IDispatch-Schnittstellen des »Partners« auf und machen ihn damit zu ihrem Client.

Diese beiden Typen von ActiveX-Objekten schließen sich nicht gegenseitig aus. Ein ActiveX-Server kann auch ActiveX-Container sein. Ein Beispiel für dieses Konzept liefert der Internet Explorer. Der Internet Explorer ist als ActiveX-Server implementiert, der im Gerüst eines ActiveX-Containers (der ebenfalls Word, Excel, PowerPoint oder eine beliebige andere ActiveX-Server-Anwendung aufnehmen kann) läuft. Zur selben Zeit, da Internet Explorer als ActiveX-Server innerhalb der Browser-Umgebung läuft, kann er andere ActiveX-Steuerelemente enthalten.

ActiveX-Steuerelemente sind eine spezielle Instanz eines ActiveX-Servers. Manche ActiveX-Server sind ebenfalls Anwendungen, die eigenständig laufen können. ActiveX- Steuerelemente können nicht eigenständig laufen und müssen in einen ActiveX-Container eingebettet werden.

Der größte Teil der Interaktion zwischen dem ActiveX-Container und einem ActiveX- Steuerelement findet über die IDispatch-Schnittstellen statt. Eine dieser IDispatch- Schnittstellen gehört zum Steuerelement und wird durch den Container verwendet, um Aufrufe der verschiedenen Methoden auszuführen, die das Steuerelement dem Container verfügbar macht.

Der Container stellt dem Steuerelement zwei IDispatch-Schnittstellen bereit. Das Steuerelement verwendet die erste dieser IDispatch-Schnittstellen, um Ereignisse in der Containeranwendung auszulösen. Die zweite Schnittstelle dient dazu, die Eigenschaften des Steuerelements festzulegen, wie es Abbildung 15.2 darstellt. Die meisten Eigenschaften eines ActiveX-Steuerelements werden eigentlich vom Container bereitgestellt, aber durch das Steuerelement verwaltet. Wenn man eine Eigenschaft für das Steuerelement setzt, ruft der Container eine Methode im Steuerelement auf, um es anzuweisen, die Eigenschaften aus dem Container zu lesen.

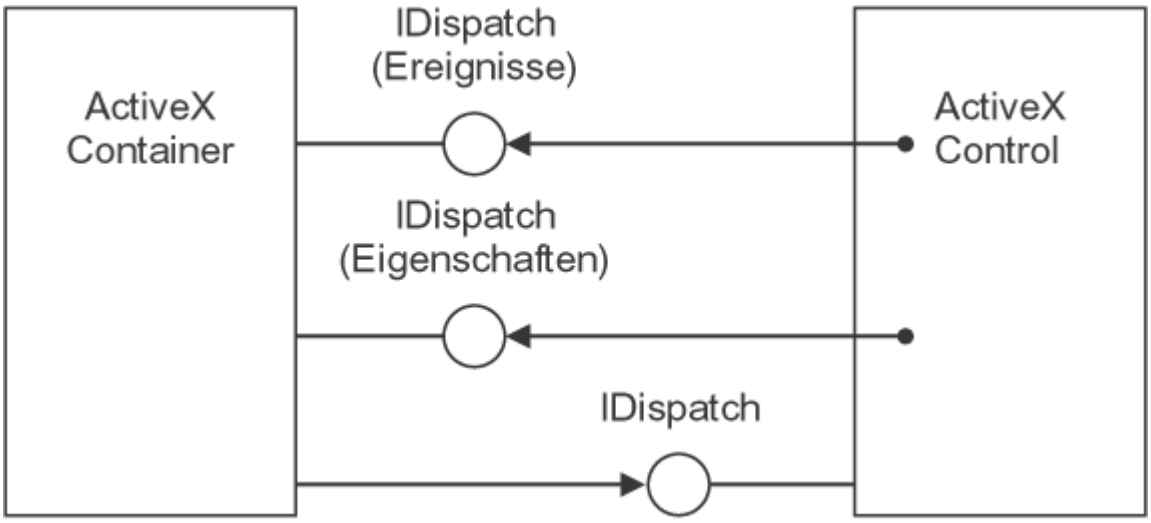


Abbildung 15.2: ActiveX-Container und Steuerelemente treten hauptsächlich über wenige IDispatch-Schnittstellen in Wechselwirkung.

Eigenschaften

Eigenschaften sind die Attribute von Steuerelementen, die für die Containeranwendung sichtbar sind und sich oftmals auch durch diese modifizieren lassen. Die vier grundlegenden Typen der Eigenschaften sind ambient, erweitert, vordefiniert und benutzerdefiniert:

- Die Containeranwendung stellt ambient-Eigenschaften bereit - wie etwa Hintergrundfarbe oder Standardschrift -, sodass das Steuerelement wie ein Teil der Containeranwendung aussieht.
- Erweiterte Eigenschaften, wie die Tabulator-Reihenfolge, sind eigentlich keine Eigenschaften des Steuerelements, sondern werden von der Containeranwendung bereitgestellt und implementiert. Das Steuerelement kann diese Eigenschaften bis zu einem gewissen Grad erweitern. Wenn zum Beispiel das Steuerelement zwei oder mehr Standardsteuerelemente enthält, kann es die Tabulator-Reihenfolge im Steuerelement als Ganzes festlegen und die Kontrolle der Tabulator-Reihenfolge an die Anwendung zurückgeben, nachdem das Steuerelement die interne Tabulator- Reihenfolge durchlaufen hat.
- Vordefinierte Eigenschaften werden durch das ActiveX-Entwicklungskit implementiert, beispielsweise die Schrift oder die Hintergrundfarbe des Steuerelements.
- Mit dem letzten Typ, den benutzerdefinierten Eigenschaften, haben Sie am meisten zu tun, da diese Eigenschaften für Ihr Steuerelement spezifisch sind und sich direkt auf die Funktionalität desselben beziehen.

Über den Assistenten zum Hinzufügen von Eigenschaften können Sie alle Eigenschaften festlegen, die Sie in Ihrem Steuerelement brauchen. Wenn Sie eine neue Eigenschaft über diesen Assistenten in das Steuerelement aufnehmen, spezifizieren Sie mehrere Aspekte der Eigenschaft.

Der erste Aspekt ist der externe Eigenschaftsname. Unter diesem Namen spricht auch die Containeranwendung die Eigenschaft an. Ein weiterer Aspekt ist der, dass man den internen Variablennamen festlegen kann, den man selbst im Code verwendet, aber nur, wenn die Eigenschaft als Member-Variable implementiert ist. Weiterhin legen Sie den Variablentyp für die Eigenschaft fest.

Wenn Sie angeben, die Eigenschaft als Member-Variable zu implementieren (die Eigenschaft ist eine Member-Variable der Steuerelementklasse), dann können Sie den Namen der Benachrichtigungsfunktion spezifizieren, die aufzurufen ist, wenn die Eigenschaft durch die Containeranwendung geändert wird. Ist die Eigenschaft keine Member-Variable der Steuerelementklasse, müssen Sie festlegen, dass sie über die Methoden Get und Set abgerufen und geändert wird, wobei die Containeranwendung eine Get-Methode aufruft, um den aktuellen Wert der Eigenschaft zu holen, und mit einer Set- Methode den Wert der Eigenschaft ändert. Wenn die Eigenschaft über Get- und Set- Methoden verwaltet wird, können Sie die Namen dieser beiden Methoden festlegen (es werden ansonsten Standardnamen vergeben).

Für alle diese Aspekte einer Eigenschaft schlägt der Eigenschaften-Assistent einen passenden Namen vor, nachdem Sie den externen Namen für die Eigenschaft eingegeben haben. Wenn Sie die Standardnamen übernehmen wollen, müssen Sie nur folgende Dinge festlegen: den externen Namen, den Typ und ob die Variable eine Member- Variable ist oder die Methoden Get und Set zu verwenden sind. Wenn Sie eine vordefinierte Eigenschaft aus der Liste der Standardeigenschaften auswählen, werden die übrigen Elemente automatisch spezifiziert. Nachdem Sie alle Informationen angegeben haben, fügt der Eigenschaften-Assistent den gesamten erforderlichen Code und die Variablen in Ihr Steuerelementprojekt ein.

Methoden

Methoden sind Funktionen im Steuerelement, die sich durch die Containeranwendung aufrufen lassen. Die Funktionen werden über die vorhin behandelte IDispatch-Schnittstelle anderen Anwendungen verfügbar gemacht. Auf Grund der Arbeitsweise von IDispatch beim Aufruf der Methoden in einem Steuerelement sind die an die Methode zu übergebenden Variablen in einer Struktur zu verpacken, die an das Steuerelement übergeben wird. Diese Struktur ist maschinenunabhängig, weshalb es keine Rolle spielt, ob das Steuerelement unter Windows 95/98 auf einem Intel Pentium III oder unter Windows NT/2000 mit einem MIPS oder Alpha-Prozessor läuft (wenn NT noch auf einer dieser Plattformen laufen würde; die Portierungen wurden eingestellt). Die Struktur ist immer die gleiche. Jede Seite des Funktionsaufrufs ist selbst dafür verantwortlich, die Parameter nach Bedarf zu konvertieren, um sie korrekt in die Struktur einzutragen oder sie daraus zu extrahieren. Diese Verpackung der Methodenparameter bezeichnet man als Marshalling.

Wenn Sie Ihrem Steuerelement eine neue Methode über den Methoden-Assistenten hinzufügen, fügt der

Assistent sowohl den gesamten erforderlichen Code ein, um das Marshalling der Parameter zu realisieren, als auch andere unterstützende Funktionalitäten, wozu auch der Aufbau der IDispatch-Schnittstelle und der Tabelle gehört.

Der Assistent fordert Sie auf, den externen Namen für die Methode anzugeben, die durch die Containeranwendung aufgerufen wird. Die Methode erhält einen vorgegebenen internen Namen, den Sie mit einem selbst gewählten internen Namen überschreiben können. Andere Aspekte Ihrer Steuerelement-Methoden, die Sie festlegen müssen, sind der Rückgabetyp und die Parameter der Methode. Nachdem Sie diese Informationen eingegeben haben, fügt der Methoden-Assistent den gesamten erforderlichen Code automatisch ein.

Ereignisse

Ereignisse sind Benachrichtigungen, die vom Steuerelement an die Containeranwendung gesendet werden. Sie sind dafür vorgesehen, die Anwendung zu informieren, dass ein bestimmtes Ereignis eingetreten ist; die Anwendung kann bei Bedarf eine Aktion unternehmen. Es lassen sich zwei Ereignisarten vom Steuerelement auslösen: vordefinierte und benutzerdefinierte Ereignisse. Vordefinierte Ereignisse werden vom ActiveX-Steuerelement-Entwicklungskit implementiert und sind als Funktionsaufrufe innerhalb des Steuerelements verfügbar. Über diese vordefinierten Ereignisse können Sie in der Containeranwendung Ereignisse für Maus und Tastatur, Fehler oder Zustandsänderungen auslösen.

Sie können auch eigene benutzerdefinierte Ereignisse hinzufügen, die in der Containeranwendung ausgelöst werden. Diese Ereignisse sollten sich auf die spezifische Funktionalität Ihres Steuerelements beziehen. Man kann zusammen mit dem Ereignis Argumente an die Containeranwendung übergeben, sodass die Anwendung die Daten erhält, die sie für die Reaktion auf die Nachricht benötigt.

Wenn Sie diese Ereignisse auslösen müssen, rufen Sie einfach die interne Ereignisfunktion auf, die das Ereignis initiiert, und übergeben alle erforderlichen Parameter an die Funktion. Der Assistent zum Hinzufügen von Ereignissen fügt auch den gesamten Code hinzu, damit sich die Nachricht vom internen Funktionsaufruf auslösen lässt.

Ereignisse werden über den Assistenten zum Hinzufügen von Eigenschaften hinzugefügt. Im Eigenschaften-Assistenten geben Sie den Ereignisnamen, den internen Funktionsnamen und die mit dem Ereignis an die Containeranwendung zu übergebenden Parameter an.

15.2 Ein ActiveX-Steuerelement erstellen

Im heutigen Beispiel erstellen Sie ein ActiveX-Steuerelement für Kritzelzeichnungen, das Ihnen bereits aus Lektion 14, »DLLs«, als DLL bekannt ist. Dieses Modul wandeln Sie nun in ein ActiveX-Steuerelement um. Dabei legen Sie als Eigenschaften des Steuerelements die Anzahl der von ihm zu zeichnenden Squiggles und auch die maximale Länge der Squiggles frei. Diese Eigenschaften wird die Containeranwendung setzen. Das Steuerelement ist so programmiert, dass es beim Anklicken eine neue Squiggle-Zeichnung erzeugt. Außerdem fügen Sie eine Methode hinzu, um eine Squiggle-Zeichnung in das Steuerelement zu laden, die mit den vorherigen Versionen des Squiggle-Moduls erstellt wurde. Schließlich lassen Sie das Steuerelement ein Ereignis auslösen, um der Containeranwendung mitzuteilen, dass das Steuerelement die Zeichnung geladen hat.

Das Gerüst des Steuerelements erstellen

Sicherlich haben Sie mittlerweile bemerkt, dass eine der Visual C++-Projektvorlagen im Dialogfeld Neues Projekt ein MFC-ActiveX-Steuerelement ist. Es handelt sich um einen weiteren Projekt-Assistenten analog zu den MFC-Assistenten, die Anwendungs- und DLL- (Dynamic Link Library) Projekte erzeugen. Sie können ihn verwenden, um ein Gerüst für beliebige ActiveX-Steuerelemente zu erstellen. Der Assistent legt die erforderlichen Dateien an und konfiguriert das Projekt, sodass der Compiler ein ActiveX-Steuerelement

erstellen kann.

Wenn Sie den MFC-ActiveX-Steuerelement-Assistenten starten, stellt er ein paar Fragen über das Steuerelementprojekt, beispielsweise ob die Steuerelemente mit Laufzeitlizenzen auszustatten sind.

Mit Laufzeitlizenzen kann man sicherstellen, dass der Benutzer des Steuerelements eine Lizenz erworben hat, um es einzusetzen. Steuerelemente, die für den Verkauf an Entwickler vorgesehen sind, haben oftmals Laufzeitlizenzen. Die Lizenz verhindert, dass ein Benutzer das Steuerelement kostenlos nutzt. Wenn Sie das Steuerelement in einer Anwendung einsetzen, wird die Laufzeitlizenz entweder durch die Installationsroutine in der Registrierung des Benutzers eingerichtet oder zusammen mit der Anwendung kompiliert. Auf diese Weise lassen sich mit dem Steuerelement keine neuen Anwendungen erstellen.

Ein anderer Bereich des MFC-ActiveX-Steuerelement-Assistenten stellt etwas tiefer gehende Fragen, die aber dennoch leicht zu beantworten sind. Hier können Sie auf die Schaltfläche Namen bearbeiten klicken, um dem Steuerelement einen aussagekräftigen Namen zu geben, mit dem der Benutzer etwas anfangen kann. In einem dritten Bereich des Steuerelement-Assistenten listet ein Kombinationsfeld eine Reihe von Fensterklassen auf, die von diesem Steuerelement als untergeordnete Klasse definiert werden können. Wenn Sie ein spezielles Eingabefeld erstellen möchten, das bestimmte Editorfunktionen für die Benutzereingaben erlaubt, wählen Sie Bearbeiten aus der Liste der Fensterklassen im Dropdown-Teil des Kombinationsfelds. Im gleichen Bereich finden sich Fragen, deren Beantwortung ein tiefer gehendes Verständnis der ActiveX-Steuerelemente erfordert.

Um mit dem Beispiel-Steuerelementprojekt zu beginnen, folgen Sie diesen Schritten:

1. Legen Sie ein neues Projekt an, wählen Sie die Projektvorlage MFC ActiveX- Steuerelement und geben Sie dem Projekt den Namen Squiggle (siehe Abbildung 15.3).

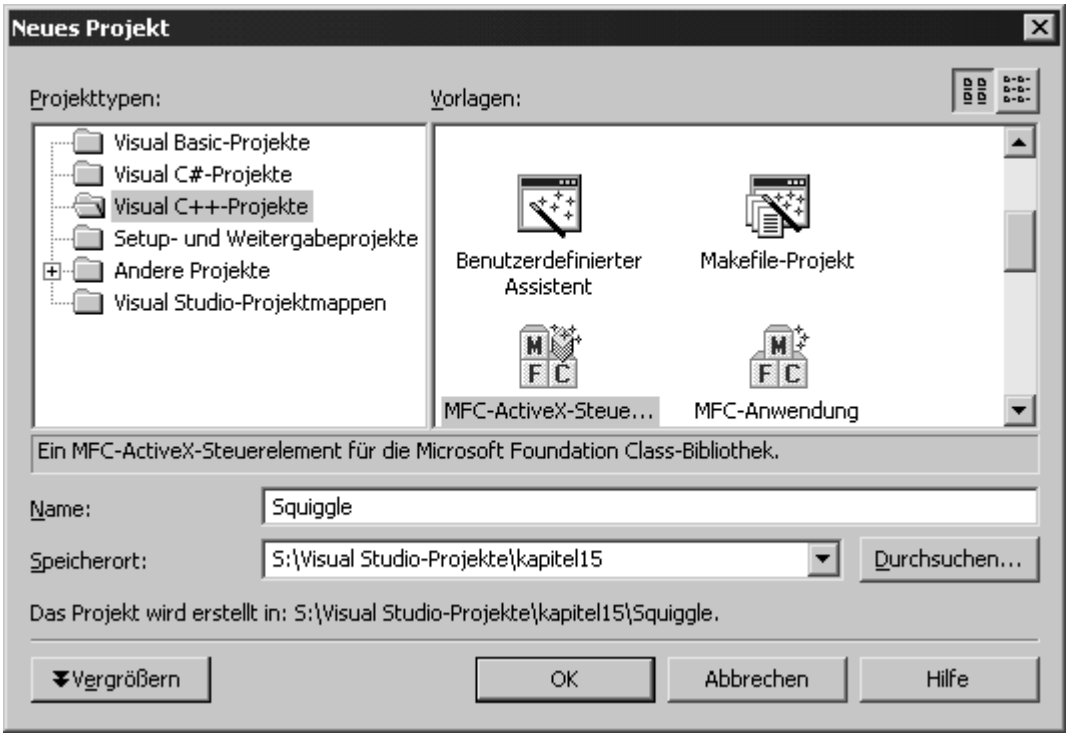


Abbildung 15.3: Das Projekt eines ActiveX-Steuerelements beginnen

2. Klicken Sie auf die Schaltfläche Steuerelementnamen auf der linken Seite des MFC ActiveX-Steuerelement-Assistenten. Stellen Sie sicher, dass der Typ-Name das Steuerelement ausreichend charakterisiert, wie in Abbildung 15.4. Übernehmen Sie für die restlichen Optionen die Standardeinstellungen - Sie erstellen heute nur ein einzelnes Steuerelement und benötigen keine Laufzeitlizenz. Klicken Sie auf Fertig stellen, um das Projekt zu erzeugen.

Abbildung 15.4: Überprüfen Sie den Namen des Steuerelementtyps.

Die Klasse CModArt modifizieren

Nachdem Sie das Gerüst des Steuerelements erstellt haben, kopieren Sie die .cpp und .h Dateien Line und ModArt aus dem MFC-DLL-Projektverzeichnis ModArtDll des am Tag 14 angelegten Projekts. Fügen Sie alle vier Dateien zu dem Steuerelementprojekt hinzu, Sie erhalten somit die Klassen CLine und CModArt in der Klassenansicht.

Bevor Sie diese Klassen verwenden können, müssen Sie die DLL-Klassendeklaration aus der Header-Datei ModArt.h löschen:

1. Öffnen Sie die Header-Datei ModArt.h und löschen Sie die AFX_EXT_CLASS-Deklaration aus der Klassendeklaration, die dann folgendermaßen aussehen sollte:

```
class CModArt :
    public CObject
{
...
```

2. Die Änderungen, die Sie an der Klasse CModArt für Ihr Steuerelement vornehmen müssen, betreffen die Variablen für die Maximalzahl und die Länge der Squiggles, die als Eigenschaften des Steuerelements freigelegt werden können. Zu diesem Zweck fügen Sie der Klasse CModArt zwei Member-Variablen hinzu. Mit der einen Variablen steuern Sie die Länge der Squiggles, mit der anderen die Anzahl. Fügen Sie diese beiden Variablen der Klasse CModArt wie folgt hinzu:

Name	Typ	Zugriff

m_iLength	Int	private
m_iSegments	Int	private

Es ist nun eine Möglichkeit zu schaffen, um die Variablen über die freigelegten Eigenschaften abrufen und aktualisieren zu können. Das bedeutet, dass Sie für beide Werte Funktionen brauchen, um den aktuellen Wert zu holen und den neuen Wert einzustellen.

3. Für die Variable m_iLength nehmen Sie zu diesem Zweck eine Member-Funktion in die Klasse CModArt auf. Legen Sie den Funktionstyp als int, den Namen als GetLength und den Zugriff als public fest.
4. In die Funktion schreiben Sie den Code aus Listing 15.1.

Listing 15.1: Die Funktion CModArt.GetLength

```
1: int CModArt::GetLength(void)
2: {
3:   // Aktuellen Wert der Variablen m_iLength zurückgeben
4:   return m_iLength;
5: }
```

5. Fügen Sie eine weitere Member-Funktion in die Klasse CModArt ein. Legen Sie den Funktionstyp als void und den Namen als SetLength fest und fügen Sie einen Parameter vom Typ int namens iLength ein. Legen Sie den Zugriff als public fest.
6. In die Funktion übernehmen Sie den Code aus Listing 15.2.

Listing 15.2: Die Funktion CModArt.SetLength

```
1: void CModArt::SetLength(int iLength)
2: {
3:   // Aktuellen Wert der Variablen m_iLength setzen
4:   m_iLength = iLength;
5: }
```

7. Analoge Funktionen fügen Sie für die Variable m_iSegments hinzu, sodass sich diese Variable ebenfalls als Eigenschaft des Steuerelements freilegen lässt. Nennen Sie diese Funktionen GetSegments und SetSegments.
8. Um sicherzustellen, dass die beiden Eigenschaften auf vernünftige Werte initialisiert werden, bevor das Steuerelement zum Einsatz kommt, modifizieren Sie den Konstruktor von CModArt gemäß Listing 15.3.

Listing 15.3: Der modifizierte Konstruktur von CModArt

```
1: CModArt::CModArt(void)
2: : m_iLength(200)
3: , m_iSegments(50)
4: {
5:   // Zufallszahlengenerator initialisieren
6:   srand((unsigned)time(NULL));
7: }
```

Schließlich modifizieren Sie die beiden Funktionen, die die Squiggle-Zeichnungen erstellen, sodass sie die Variablen anstelle der momentan fest codierten Werte verwenden. Folgen Sie diesen Schritten:

1. Um die Funktion NewDrawing zu modifizieren, ersetzen Sie die Maximalzahl der Squiggles entsprechend Listing 15.4 durch die Variable m_iSegments.

Listing 15.4: Die Funktion CModArt.NewDrawing

```
 1: void CModArt::NewDrawing(void)
```

```
 2: {
 3:   int iNumLines;
 4:   int iCurLine;
 5:
 6:   // Anzahl der zu erzeugenden Linien ermitteln
 7:   iNumLines = rand() % m_iSegments;
 8:   // Sind Linien zu erzeugen?
 9:   if (iNumLines > 0)
10:   {
11:     // Schleife durch Anzahl der Linien
12:     for (iCurLine = 0; iCurLine < iNumLines; iCurLine++)
13:     {
14:       // Neue Linie erzeugen
15:       NewLine();
16:     }
17:   }
18: }
```

2. Ersetzen Sie die Maximallänge der Squiggles in der Funktion NewLine gemäß Listing 15.5 durch die Variable m_iLength.

Listing 15.5: Die Funktion CModArt.NewLine

```
 1: void CModArt::NewLine()
 2: {
 3:   int lNumLines;
 4: ...
 5:
 6:   // Anzahl der Teile dieses Squiggles bestimmen
 7:   lNumLines = rand() % m_iLength;
 8:   // Besteht das Squiggle aus mindestens einem Teil?
 9: ...
10: }
```

An den Klassen CModArt und CLine haben Sie nun alle erforderlichen Änderungen für Ihr ActiveX-Steuerelement vorgenommen. Jetzt müssen Sie eine Instanz der Klasse CModArt in die Steuerelementklasse als Member-Variable einfügen. Nehmen Sie eine neue Member- Variable in die Steuerelementklasse CSquiggleCtrl auf. Legen Sie den Variablentyp als CModArt, den Namen als m_maDrawing und den Zugriff als private fest.

Eigenschaften hinzufügen

Da die beiden Variablen, die Sie in die Klasse CModArt aufgenommen haben, keine Variablen der Steuerelementklasse (CSquiggleCtrl) sind, sieht man normalerweise Get- und Set-Methoden vor, um den Eigenschaftswert abzurufen bzw. zu setzen. Wären diese Variablen Elemente der Steuerelementklasse, könnten Sie sie über den Assistenten zum Hinzufügen von Member-Variablen als Member-Variablen hinzufügen. Sie wüssten dann immer darüber Bescheid, wann und ob sich die Variablen geändert haben, da in der Steuerelementklasse auch eine Benachrichtigungsmethode enthalten wäre, wie sie bei Änderung des Eigenschaftswerts aufgerufen wird. Weil aber nun die Variablen Elemente einer internen Klasse sind, braucht man etwas mehr Kontrolle über deren Werte.

Selbst wenn die Variablen, die Sie freilegen wollen, Member-Variablen der Steuerelementklasse wären, würde man trotzdem Get- und Set-Methoden für den Zugriff auf die Variablen als Steuerelementeigenschaften vorsehen. Bei Einsatz von Get- und Set-Methoden kann man eine Gültigkeitsprüfung für die neuen Eigenschaftswerte vorsehen und damit testen, ob die Containeranwendung den passenden Wert in die Eigenschaft schreibt.

Um diese Eigenschaften Ihrem Steuerelement hinzuzufügen, folgen Sie diesen Schritten:

1. Erweitern Sie den SquiggleLib-Knoten in der Klassenansicht, um den Knoten der _DSquiggle-Schnittstelle zu sehen.

2. Markieren Sie den _DSquiggle-Knoten und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Eigenschaft hinzufügen aus dem Kontextmenü.

3. Im Assistenten zum Hinzufügen von Eigenschaften (siehe Abbildung 15.5) geben Sie den externen Namen ein, den die Eigenschaft haben soll, SquiggleLength, und legen den Typ als SHORT fest (der Typ int ist nicht verfügbar, nur SHORT und LONG). Klicken Sie auf das Optionsfeld Get/Set-Methoden.
4. Klicken Sie auf Fertig stellen, um die Eigenschaft hinzuzufügen.

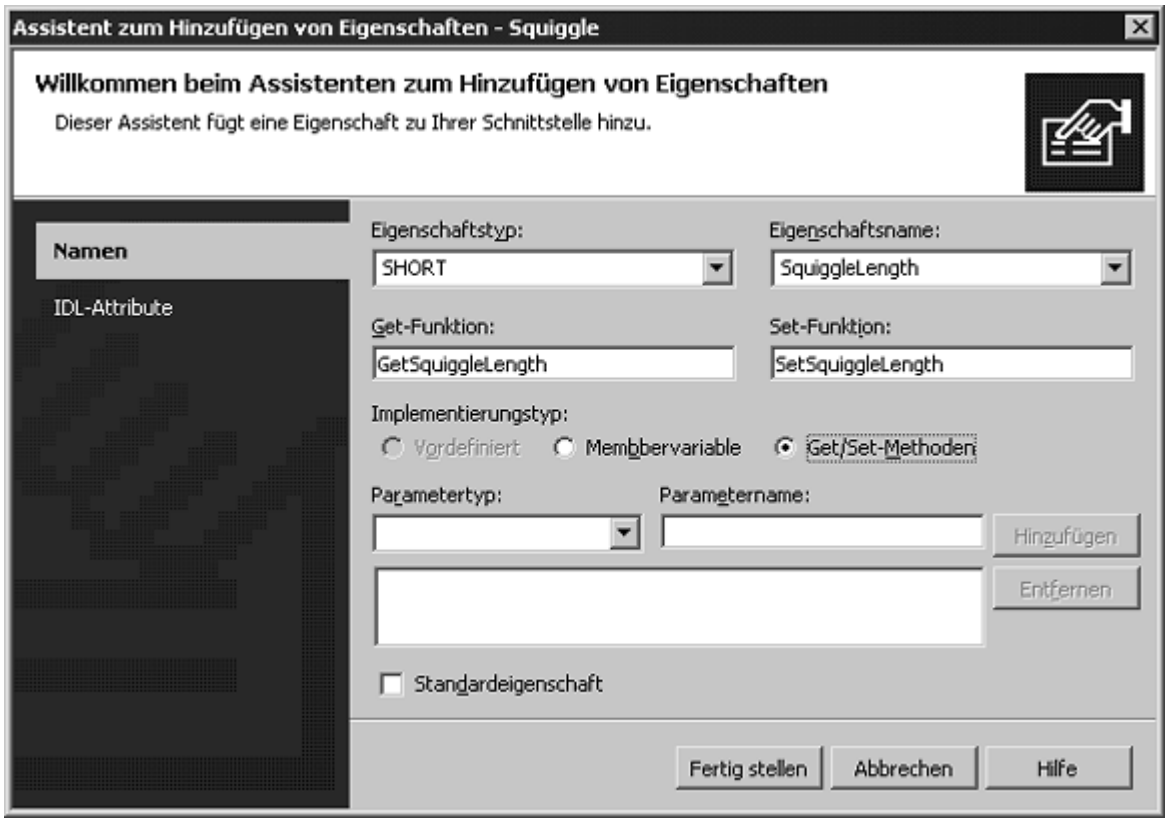


Abbildung 15.5: Der Assistent zum Hinzufügen von Eigenschaften

5. Erweitern Sie die CSquiggleCtrl-Klasse in der Klassenansicht.
6. Doppelklicken Sie auf die Methode GetSquiggleLength, um zur Funktionsimplementierung zu gelangen. Die Implementierung der Funktion SetSquiggleLength sollte in der Datei SquiggleCtrl.cpp unmittelbar auf die von GetSquiggleLength folgen. Rufen Sie in beiden Methoden die Funktionen Get und Set auf, die Sie der Klasse CModArt hinzugefügt haben, um den Zugriff auf die Längenvariable zu kontrollieren.
7. Bearbeiten Sie die beiden Methoden wie in Listing 15.6.

Listing 15.6: Die Funktionen Get/SetSquiggleLength von CSquiggleCtrl

```
 1: SHORT CSquiggleCtrl::GetSquiggleLength(void)
 2: {
 3:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 4:
 5:   // TODO: Fügen Sie hier Ihren Dispatchhandlercode ein.
 6:   // Ergebnis der Funktion GetLength() zurückliefern
 7:   return m_maDrawing.GetLength();
 8: }
 9:
10: void CSquiggleCtrl::SetSquiggleLength(SHORT newVal)
```

```
11: {
12:    AFX_MANAGE_STATE(AfxGetStaticModuleState());
13:
14:    // TODO: Fügen Sie hier Ihren Eigenschaftenhandlercode ein.
15:    // Neuen Längenwert setzen
16:    m_maDrawing.SetLength(newVal);
17:
18:    SetModifiedFlag();
19: }
```

8. Sehen Sie eine weitere Eigenschaft für die Anzahl der Squiggles in der Zeichnung vor, indem Sie die Schritte 2-7 noch einmal ausführen, dabei aber den Eigenschaftsnamen NumberSquiggles verwenden und die Get/SetSegments-Funktionen in der Klasse CModArt aufrufen.
9. Als letzte Eigenschaft soll das Steuerelement eine boolesche Eigenschaft erhalten, über die die Containeranwendung Einfluss auf das Erzeugen neuer Zeichnungen und die Sichtbarkeit der aktuellen Zeichnung hat. Nehmen Sie eine neue Eigenschaft über den Eigenschaften-Assistenten auf, nennen Sie sie KeepCurrentDrawing und legen Sie den Typ als VARIANT_BOOL fest. Übernehmen Sie die vorgeschlagene Implementierung als Member-Variable und klicken Sie auf Fertig stellen. Der Eigenschaften-Assistent nimmt die Variable automatisch in die Steuerelementklasse auf und schreibt auch den erforderlichen Code, um die Variable zu verwalten.

Die Eigenschaftsseite entwerfen und erstellen

Für das Steuerelement ist eine Eigenschaftsseite bereitzustellen, auf die Entwickler zugreifen können, wenn sie mit Ihrem Steuerelement arbeiten. Diese Eigenschaftsseite liefert den Benutzern ein Mittel, um die Eigenschaften des Steuerelements zu setzen, selbst wenn ihre eigenen Entwicklungswerkzeuge keine Einrichtung bieten, um die Eigenschaften in einer anderen Form als per Code zugänglich zu machen.

Eine Eigenschaftsseite lässt sich dem Steuerelement in einfacher Weise hinzufügen. Wenn Sie die Ressourcenansicht wählen und den Ordner Dialog erweitern, finden Sie hier bereits ein Dialogfeld für die Eigenschaften des Steuerelements. Es handelt sich um ein normales Dialogfeld, das Sie mit den Standardsteuerelementen des Dialog-Editors entwerfen können. Um die Eigenschaftsseite für das Steuerelement des Beispiels zu gestalten, nehmen Sie den Entwurf des Dialogfelds wie in Abbildung 15.6 und mit den Eigenschaftseinstellungen gemäß Tabelle 15.1 vor.

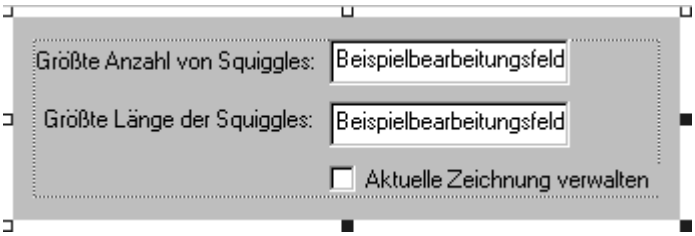


Abbildung 15.6: Das Layout der Eigenschaftsseite für das Steuerelement

Objekt	Eigenschaft	Einstellung
Text	Beschriftung	Größte Anzahl von Squiggles:
Eingabefeld	ID	IDC_ENBRSQUIG
Text	Beschriftung	Größte Länge der Squiggles:
Eingabefeld	ID	IDC_ELENSQUIG
Kontrollkästchen	ID Beschriftung	IDC_CMAINTDRAW Aktuelle Zeichnung verwalten

Tabelle 15.1: Eigenschaftseinstellungen für das Steuerelement

Wenn Sie die Größe des Dialogfeldbereichs für die Eigenschaftsseite verändern, müssen Sie die neue Größe ganz genau im Auge behalten. Die aktuelle Größe des Dialogfelds wird in der rechten unteren Ecke von Visual Studio angezeigt. Das Dialogfeld für die Eigenschaftsseite muss entweder 250 x 62 oder 250 x 110 groß sein. Alles andere ist keine Standardgröße für eine Eigenschaftsseite und kann beim Gebrauch Probleme oder Warnungen auslösen.

Nachdem Sie alle Steuerelemente platziert und die Eigenschaften festgelegt haben, verwenden Sie den Assistenten zum Hinzufügen von Member-Variablen, um den Steuerelementen Variablen zuzuordnen. Wenn Sie einem der Steuerelemente im Dialogfeld Eigenschaftsseite eine Variable hinzugefügt haben, können Sie den Datenaustausch modifizieren, um die Variablen der Eigenschaftsseite mit den Steuerelementeigenschaften zu verbinden. Folgen Sie diesen Schritten:

1. Fügen Sie den Steuerelementen auf der Eigenschaftsseite Ihres Steuerelements Variablen hinzu und binden Sie sie wie in Tabelle 15.2 aufgeführt an die Steuerelementeigenschaften.

Objekt	Name	Kategorie	Typ	Zugriff
IDC_ENBRSQUIG	m_iNbrSquiggles	Value	int	public
IDC_ELENSQUIG	m_iLenSquig	Value	int	public
IDC_CMAINTDRAW	m_bKeepDrawing	Value	BOOL	public

Tabelle 15.2: Steuerelement-Variablen.

2. Öffnen Sie die Quellcode-Datei SquigglePropPage.cpp.
3. Suchen Sie die Implementierung der Funktion DoDataExchange. Ändern Sie gemäß Listing 15.7 die DDX-Funktionen zu DDP-Funktionen und fügen Sie die mit den Variablen zu verknüpfenden Eigenschaftsnamen ein.

Listing 15.7: Die Funktion DoDataExchange.

```
 1: void CSquigglePropPage::DoDataExchange(CDataExchange* pDX)
 2: {
 3:   DDP_Text(pDX, IDC_ELENSQUIG, m_iLenSquig,
 4:                 _T("SquiggleLength"));
 5:   DDX_Text(pDX, IDC_ELENSQUIG, m_iLenSquig);
 6:
 7:   DDP_Text(pDX, IDC_ENBRSQUIG, m_iNbrSquiggles,
 8:                 _T("NumberSquiggles"));
 9:    DDX_Text(pDX, IDC_ENBRSQUIG, m_iNbrSquiggles);
10:
11:   DDP_Check(pDX, IDC_CMAINTDRAW, m_bKeepDrawing,
12:                  _T("KeepCurrentDrawing"));
13:   DDX_Check(pDX, IDC_CMAINTDRAW, m_bKeepDrawing);
14:
15:   DDP_PostProcessing(pDX);
16: }
```

In älteren Versionen von Visual C++ besaß der Assistent zum Hinzufügen von Member-

Variablen einen zusätzlichen Bereich, in dem Sie die Steuerelementklasse angeben konnten, mit der die Eigenschaftsseite verbunden werden sollte. Hierdurch fielen die Schritte 2 und 3 weg, da diese Aktionen automatisch für Sie ausgeführt wurden. In der aktuell vorliegenden Version des Assistenten wird die DDP_PostProcessing-Funktion an den Anfang der Funktion DoDataExchange geschrieben, dies ist ein Fehler und muß von Ihnen korrigiert werden.

Grundlegende Steuerelementfunktionen realisieren

Als grundlegende Funktionalität soll Ihr Steuerelement auf Mausklicks hin eine neue Zeichnung generieren. Um dieses Verhalten zu steuern, nehmen Sie eine zweite boolesche Variable in die Steuerelementklasse auf, damit die Funktion OnDraw weiß, dass ein Mausklick ausgelöst wurde. Am einfachsten lässt sich der Zeichenbereich in der Funktion OnDraw ermitteln, sodass hier auch die Zeichnungen generiert werden. Soll nun das Steuerelement jedesmal eine neue Zeichnung hervorbringen, wenn der Benutzer die Anwendung mit Ihrem Steuerelement vor eine andere Anwendung verschiebt? Wahrscheinlich nicht. Höchstwahrscheinlich wollen Sie mehr Einfluss auf das Verhalten des Steuerelements ausüben, dasssodass sich diese zweite boolesche Variable anbietet.

Nehmen Sie eine Member-Variable in die Steuerelementklasse (CSquiggleCtrl) auf. Legen Sie den Variablentyp als BOOL, den Variablennamen mit m_bGenNewDrawing und den Zugriff als private fest.

Bevor Sie den Code eingeben, der die verschiedenen Aufgaben realisiert, müssen Sie zunächst alle Member-Variablen in der Steuerelementklasse initialisieren. Dazu gehört die als Eigenschaft festgelegte Member-Variable m_keepCurrentDrawing und die Member- Variable, die Sie gerade hinzugefügt haben, m_bGenNewDrawing. Das Steuerelement soll eine neue Zeichnung aufs Geratewohl generieren und Sie wollen sicherlich nicht irgendwelche Zeichnungen verwalten, außer wenn die Containeranwendung explizit festlegt, dass eine Zeichnung zu verwalten ist. Die beiden Variablen setzen Sie entsprechend im Konstruktor der Steuerelementklasse, wie es Listing 15.8 zeigt.

Listing 15.8: Der Konstruktor von CSquiggleCtrl

```
1: CSquiggleCtrl::CSquiggleCtrl()
2: : m_bGenNewDrawing(TRUE)
3: , m_KeepCurrentDrawing(FALSE)
4: {
5:   InitializeIIDs(&IID_DSquiggle, &IID_DSquiggleEvents);
6:   // TODO: Daten der Steuerelementinstanz hier initialisieren.
7: }
```

Als Nächstes fügen Sie den Code hinzu, um die Squiggle-Zeichnungen zu generieren und anzuzeigen. Für diese Funktionalität bietet sich die Funktion OnDraw in der Steuerelementklasse an. Die Funktion wird immer dann aufgerufen, wenn sich das Steuerelement selbst neu zeichnen muss, ob es nun verborgen war oder ein anderes Ereignis über die Funktion Invalidate das Neuzeichnen des Steuerelements ausgelöst hat. In der Funktion OnDraw können Sie auch gleich ermitteln, ob eine neue Zeichnung zu generieren oder einfach die vorhandene Zeichnung neu darzustellen ist. Man sollte auch daran denken, dass man für das Zeichnen des gesamten Bereichs, den das Steuerelement einnimmt, verantwortlich ist. Das bedeutet, dass Sie den Hintergrund der Squiggle-Zeichnung erstellen müssen, da sonst die Squiggles auf dem erscheinen, was gerade in diesem Anzeigebereich vorhanden ist. (Wer weiß? Vielleicht ist es gerade der Effekt, den Sie erzielen wollen.)

Um diese Funktionalität in Ihrem Steuerelement zu realisieren, bearbeiten Sie die Funktion OnDraw in der Steuerelementklasse und fügen den Code aus Listing 15.9 hinzu.

Listing 15.9: Die Funktion CSquiggleCtrl.OnDraw

```
1: void  CSquiggleCtrl::OnDraw(
2:         CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
3: {
4:   // TODO: Folgenden Code durch eigene Zeichenfunktion ersetzen.
5:
6:   pdc->FillRect(rcBounds,
7:        CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
8:   // pdc->Ellipse(rcBounds);
```

```
 9:    // Müssen wir eine neue Zeichnung generieren?
10:    if (m_bGenNewDrawing)
11:    {
12:      // Zeichenbereich für die neue Zeichnung einrichten
13:      m_maDrawing.SetRect(rcBounds);
14:      // Alte Zeichnung löschen
15:      m_maDrawing.ClearDrawing();
16:      // Neue Zeichnung erstellen
17:      m_maDrawing.NewDrawing();
18:      // Steuerflag zurücksetzen
19:      m_bGenNewDrawing = FALSE;
20:    }
21:    // Squigglezeichnung zeichnen
22:    m_maDrawing.Draw(pdc);
23: }
```

Schließlich lösen Sie das Steuerelement aus, um eine neue Zeichnung zu generieren, sobald der Benutzer auf das Steuerelement klickt. Dafür ist eine Behandlungsroutine für das Ereignis OnClick des Steuerelements erforderlich. Zuerst nehmen Sie aber eine vordefinierte Methode in das Steuerelement auf, um sicherzustellen, dass es die Nachricht OnClick erhält. Folgen Sie diesen Schritten, um diese vordefinierte Methode aufzunehmen:

1. Markieren Sie den Knoten der _DSquiggle-Schnittstelle in der Klassenansicht.

2. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Methode hinzufügen aus dem Kontextmenü.
3. Nehmen Sie in die Steuerelementklasse eine neue Methode auf. Wählen Sie dafür die Methode DoClick aus der Dropdown-Liste der vordefinierten Methoden aus, wie es Abbildung 15.7 zeigt. Klicken Sie auf Fertig stellen, um die Methode in Ihr Steuerelement aufzunehmen.

Abbildung 15.7: Der Assistent zum Hinzufügen von Methoden

4. Markieren Sie die Klasse CSquiggleCtrl in der Klassenansicht.
5. Wählen Sie in der Eigenschaftenansicht den Modus Überschreibungen.
6. Scrollen Sie nach unten, bis Sie die Methode OnClick finden. Klicken Sie in den Wertebereich für diese

Methode und wählen Sie aus dem Kombinationsfeld <Hinzufügen> OnClick.
7. In die OnClick-Behandlungsroutine übernehmen Sie den Code aus Listing 15.10.

Listing 15.10: Die Funktion CSquiggleCtrl.OnClick

```
 1: void CSquiggleCtrl::OnClick(USHORT iButton)
 2: {
 3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
 4:   // und/oder rufen Sie die Basisklasse auf.
 5:
 6:   // Können wir eine neue Zeichnung generieren?
 7:   if (!m_KeepCurrentDrawing)
 8:   {
 9:     // Flag setzen, sodass neue Zeichnung generiert wird
10:     m_bGenNewDrawing = TRUE;
11:     // Steuerelement zum Auslösen der OnDraw-Funktion
12:     // deaktivieren
13:     Invalidate();
14:   }
15:   COleControl::OnClick(iButton);
16: }
```

Die Funktion OnClick in Listing 15.10 prüft zuerst, ob eine neue Zeichnung zu generieren oder die aktuelle Zeichnung zu verwalten ist. Beim Generieren einer neuen Zeichnung wird das Flag m_bGenNewDrawing auf TRUE gesetzt und das Steuerelement mit der Funktion Invalidate ungültig gemacht. Das löst den Aufruf der Funktion OnDraw aus.

Als Nächstes müssen Sie Code aufnehmen, um beim Schließen des Steuerelements aufzuräumen. Sie müssen die DestroyWindow-Funktion der Steuerelementklasse überladen, um die Funktion zum Löschen der Zeichnung und zum Freigeben des Speichers in der Klasse CModArt aufzurufen. Um diese Funktionalität hinzuzufügen:

1. markieren Sie in der Klassenansicht die Klasse CsquiggleCtrl,

2. wählen Sie in der Eigenschaftenansicht den Modus Überschreibungen und fügen Sie eine Überschreibungsfunktion für DestroyWindow ein,
3. nehmen Sie in die neue Funktion folgenden Code auf:

```
BOOL CSquiggleCtrl::DestroyWindow()
{
  // TODO: Fügen Sie hier Ihren spezialisierten Code ein, und/oder rufen Sie
  // Zeichnung löschen
  m_maDrawing.ClearDrawing();
  return COleControl::DestroyWindow();
}
```

Methoden hinzufügen

Denken Sie noch einmal an die Funktionalität, die das Steuerelement erhalten soll: Eine Funktion lädt eine Squiggle-Zeichnung, die mit dem Squiggle-Modul in der Version von Tag 14 erstellt wurde. Zu diesem Zweck nehmen Sie eine Methode in das Steuerelement auf, die die Containeranwendung aufrufen kann, um den Namen einer zu ladenden Datei zu übergeben. Der Anwendung haben Sie bereits eine Methode hinzugefügt und zwar eine vordefinierte Methode. Eine benutzerdefinierte Methode lässt sich fast in der gleichen Weise hinzufügen, man muss aber ein paar Angaben mehr im Methoden- Assistenten ausfüllen.

In der Methode zum Laden einer vorhandenen Zeichnung erzeugen Sie ein CFile-Objekt für den als

Parameter übergebenen Dateinamen. Der Konstruktor von CFile übernimmt den Dateinamen und das Flag CFile::modeRead, um dem Objekt mitzuteilen, dass die Datei schreibgeschützt zu öffnen ist. Nachdem Sie das CFile-Objekt erstellt haben, erzeugen Sie ein CArchive-Objekt, um die Datei zu lesen. Der Konstruktor von CArchive übernimmt das CFile-Objekt, das Sie gerade erzeugt haben, und das Flag CArchive::load, um dem Objekt mitzuteilen, dass die Datei zu laden ist. Jetzt können Sie das CArchive- Objekt an die Funktion Serialize des Zeichenobjekts übergeben und die Zeichnung lesen und laden lassen. Nachdem die Zeichnung geladen ist, zeigen Sie sie an, indem Sie das Steuerelement ungültig machen. Vorher setzen Sie noch das Flag m_bGenNewDrawing auf FALSE, damit die eben geladene Zeichnung nicht überschrieben wird.

Um diese Funktionalität in Ihr Steuerelement einzufügen, folgen Sie diesen Schritten:

1. Markieren Sie den Knoten der _DSquiggle-Schnittstelle in der Klassenansicht.

2. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Methode hinzufügen aus dem Kontextmenü.
3. Geben Sie den Namen der Methode als LoadDrawing an. Der interne Name wird automatisch auf der Basis des eingegebenen externen Namens erzeugt. Legen Sie dann den Rückgabetyp als VARIANT-BOOL fest, um die Containeranwendung darüber informieren zu können, ob das Steuerelement die Zeichnung laden konnte. Schließlich fügen Sie noch einen Parameter in die Parameterliste ein. Legen Sie den Namen als sFileName und den Typ als BSTR fest (der Typ CString ist nicht verfügbar; der Typ BSTR ist der Stringtyp der COM-Schnittstelle), wie es Abbildung 15.8 zeigt.

Abbildung 15.8: Der Assistent zum Hinzufügen von Methoden

4. Klicken Sie auf Fertig stellen, um die Methode in Ihr Steuerelement aufzunehmen.
5. Nehmen Sie den Code gemäß Listing 15.11 in die Methode auf.

Listing 15.11: Die Funktion CSquiggleCtrl.LoadDrawing

```
1: VARIANT_BOOL CSquiggleCtrl::LoadDrawing(LPCTSTR sFileName)
2: {
3:    AFX_MANAGE_STATE(AfxGetStaticModuleState());
4:
5:    // TODO: Fügen Sie hier Ihren Dispatchhandlercode ein.
6:    try
```

```
 7:   {
 8:     // Ein CFile-Objekt erzeugen
 9:     CFile fFile(sFileName, CFile::modeRead);
10:     // Ein CArchive-Objekt zum Laden der Datei erzeugen
11:     CArchive lArchive(&fFile, CArchive::load);
12:     // Datei laden
13:     m_maDrawing.Serialize(lArchive);
14:     // Sicherstellen, dass die geladene Zeichnung nicht
15:     // überschrieben wird
16:     m_bGenNewDrawing = FALSE;
17:     // Geladene Zeichnung zeichnen
18:     Invalidate();
19:   }
20:   catch (CFileException* perr)
21:   {
22:     // Fehler melden
23:     perr->ReportError();
24:     // Ausnahme löschen
25:     perr->Delete();
26:     return VARIANT_FALSE;
27:   }
28:   return VARIANT_TRUE;
29: }
```

Der Code in Listing 15.11 übergibt der Funktion einen LPCTSTR-Datentyp. Ein BSTR ist ein als Teil der OLE-2.0-Spezifikation erzeugter Stringtyp. Es handelt sich bei ihm um ein nullterminiertes Standard-Zeichenarray, gewöhnlich aus UNICODE-Zeichen, mit einer Besonderheit. Der BSTR-Zeiger zeigt zwar auf das erste Zeichen im Array, doch wenn Sie vom ersten Zeichen ein paar Bytes zurückgehen, finden Sie einen numerischen Wert, der die Länge des Strings angibt. Dieser Standard-Stringtyp wird in Visual Basic verwendet und ist der bevorzugte Stringtyp für COM-Schnittstellen. Unglücklicherweise arbeitet MFC nicht einfach mit BSTR-Strings, weshalb man sie erst in einen CString umwandeln muss, bevor sie sich verwenden lassen. Dem CString-Konstruktor, der BSTR in CString umwandeln kann, muss der String als w_char sowie die Länge des Strings übergeben werden. Der Assistent zum Hinzufügen von Methoden hat jedoch den Code für die Stringumwandlung für Sie eingefügt, sodass Sie einen Standard-LPCTSTR-Stringzeiger erhalten, den Sie direkt an den CFile-Klassenkonstruktor für die zu öffnende Datei übergeben können.

Ereignisse hinzufügen

Als letzten Schritt beim Erstellen des Steuerelements fügen Sie die Ereignisse hinzu, die das Steuerelement in der Containeranwendung auslöst. Beim Einsatz des Steuerelements kann der Entwickler Code vorsehen, der bei diesen Ereignissen ausgelöst wird. Die Ereignisse nehmen Sie über den Assistenten zum Hinzufügen von Eigenschaften auf. Wenn Ihr Steuerelement ein vordefiniertes Ereignis auslösen soll, klicken Sie mit der rechten Maustaste auf die Steuerelementklasse, wählen Hinzufügen / Ereignis hinzufügen aus dem Kontextmenü und ein vordefiniertes Ereignis aus der Dropdown- Liste der Ereignisse aus. Soll Ihr Steuerelement ein benutzerdefiniertes Ereignis erhalten, wählen Sie im Ereignis-Assistenten kein vordefiniertes Ereignis aus, sondern geben den Namen Ihres benutzerdefinierten Ereignisses ein. In die Parameterliste im unteren Teil des Ereignis-Assistenten tragen Sie die Parameter ein, die Sie mit dem Ereignis von Ihrem Steuerelement an die Containeranwendung übergeben möchten.

Für das Steuerelement im heutigen Beispiel fügen Sie ein benutzerdefiniertes Ereignis hinzu, um die Anwendung darüber zu informieren, dass die angegebene Datei mit der Zeichnung geladen wurde. Folgen Sie diesen Schritten:

1. Markieren Sie in der Klassenansicht die Klasse CSquiggleCtrl.

2. Klicken Sie mit der rechten Maustaste und wählen Sie Hinzufügen / Ereignis hinzufügen aus dem Kontextmenü.
3. Geben Sie den Namen FileLoaded für das benutzerdefinierte Ereignis ein. Das Dialogfeld Ereignis hinzufügen erzeugt automatisch einen internen Namen für das Ereignis, im Beispiel FileLoaded (siehe Abbildung 15.9). Dieser interne Name ist der Name der Funktion, die Sie in Ihrem Code aufrufen müssen, wenn Sie das Ereignis auslösen möchten.

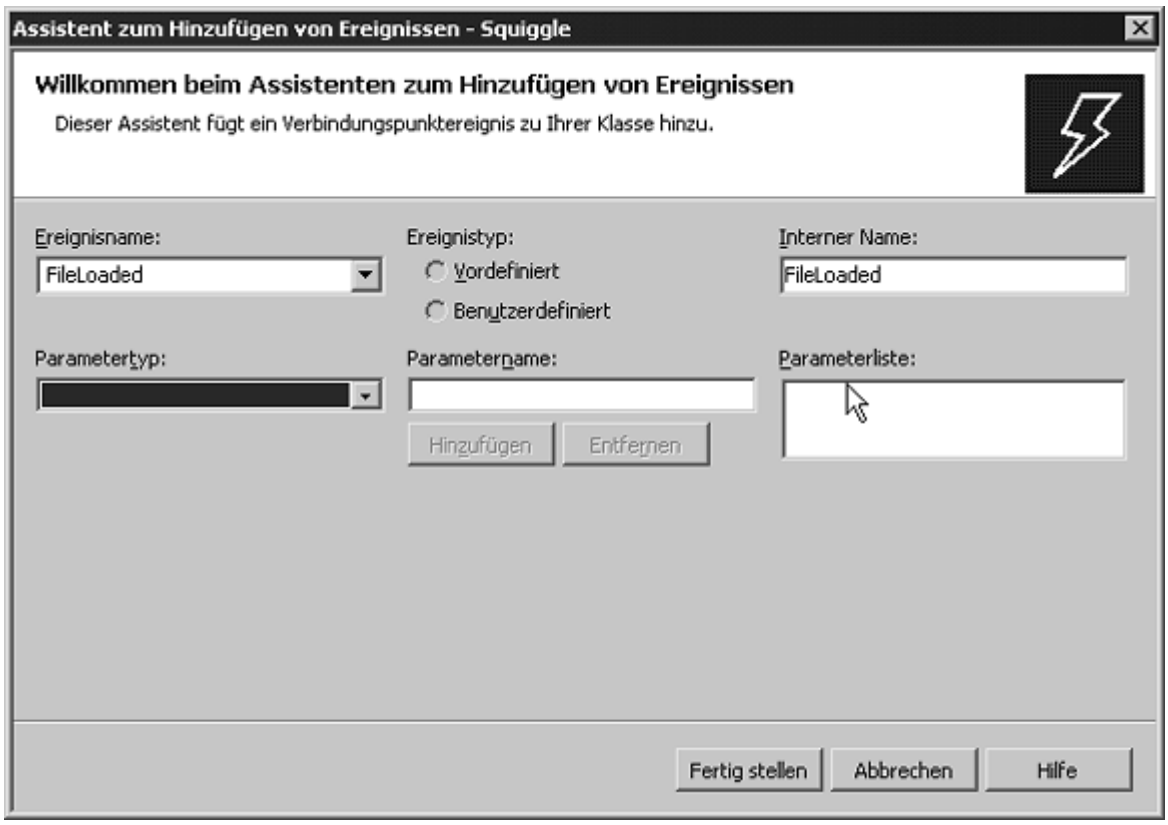


Abbildung 15.9: Der Assistent zum Hinzufügen von Ereignissen

4. Klicken Sie auf Fertig stellen, um das Ereignis hinzuzufügen.

Um ein vordefiniertes Ereignis hinzuzufügen, markieren Sie das Ereignis in der Dropdown-Liste der vordefinierten Ereignisse und klicken auf Fertig stellen.

Nachdem Sie nun das Ereignis in das Steuerelement eingefügt haben, passen Sie noch den Code an, um das Ereignis an den geeigneten Stellen auszulösen. Das Ereignis lösen Sie am Ende der Funktion LoadDrawing aus, vorausgesetzt, dass die Zeichnung korrekt geladen wurde. Bauen Sie die zusätzliche Funktionalität gemäß der fett gedruckten Zeilen von Listing 15.12 in die Funktion LoadDrawing ein.

Listing 15.12: Die modifizierte Funktion CSquiggleCtrl.LoadDrawing

```
 1: VARIANT_BOOL CSquiggleCtrl::LoadDrawing(LPCTSTR sFileName)
 2: {
 3:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
 4:
 5:   // TODO: Fügen Sie hier Ihren Dispatchhandlercode ein.
 6:   try
 7:   {
 8:     // Ein CFile-Objekt erzeugen
 9:     CFile fFile(sFileName, CFile::modeRead);
10:     // Ein CArchive-Objekt zum Laden der Datei erzeugen
11:     CArchive lArchive(&fFile, CArchive::load);
12:     // Datei laden
13:     m_maDrawing.Serialize(lArchive);
14:     // Sicherstellen, dass die geladene Zeichnung nicht
```

```
15:     // überschrieben wird
16:     m_bGenNewDrawing = FALSE;
17:     // Geladene Zeichnung zeichnen
18:     Invalidate();
19:     // Ereignis FileLoaded auslösen
20:     FileLoaded();
21:   }
22:   catch (CFileException* perr)
23:   {
24:     // Fehler melden
25:     perr->ReportError();
26:     // Ausnahme löschen
27:     perr->Delete();
28:     return VARIANT_FALSE;
29:   }
30:   return VARIANT_TRUE;
31: }
```

Das Steuerelement testen

Damit sind alle Vorbereitungen abgeschlossen. Sie können nun das Steuerelement kompilieren und testen. Sie brauchen aber nun nicht in den nächsten Computerladen zu rennen, um sich Visual Basic zuzulegen, sondern verfügen bereits über ein Werkzeug zum Testen von ActiveX-Steuerelementen. Im Menü Extras finden Sie den Befehl Test Container für ActiveX-Steuerelemente. Es handelt sich hierbei um ein Hilfsprogramm, das speziell dafür vorgesehen ist, selbst erstellte ActiveX-Steuerelemente auszuprobieren. Nachdem Sie das Steuerelement kompiliert haben, starten Sie den Testcontainer für ActiveX-Steuerelemente, um das Steuerelement auf Herz und Nieren zu prüfen.

Wenn Visual C++ das Steuerelement zwar kompilieren, nicht aber registrieren kann, müssen Sie es manuell registrieren. Wählen Sie Datei / Steuerelemente registrieren aus dem Menü des Testcontainers. So können Sie Ihr kompiliertes Steuerelement auffinden und registrieren.

Beim Start des Testcontainers sehen Sie einen leeren Bereich, in dem Ihr Steuerelement erscheinen wird. Um das Steuerelement in diesen Bereich einzufügen, wählen Sie Bearbeiten / Neues Steuerelement einfügen aus dem Menü des Testcontainers. Daraufhin öffnet sich das Dialogfeld Steuerelement einfügen (siehe Abbildung 15.10). Markieren Sie das Steuerelement in der Liste der verfügbaren Steuerelemente und klicken Sie auf OK, um das Steuerelement in den Containerbereich einzufügen.

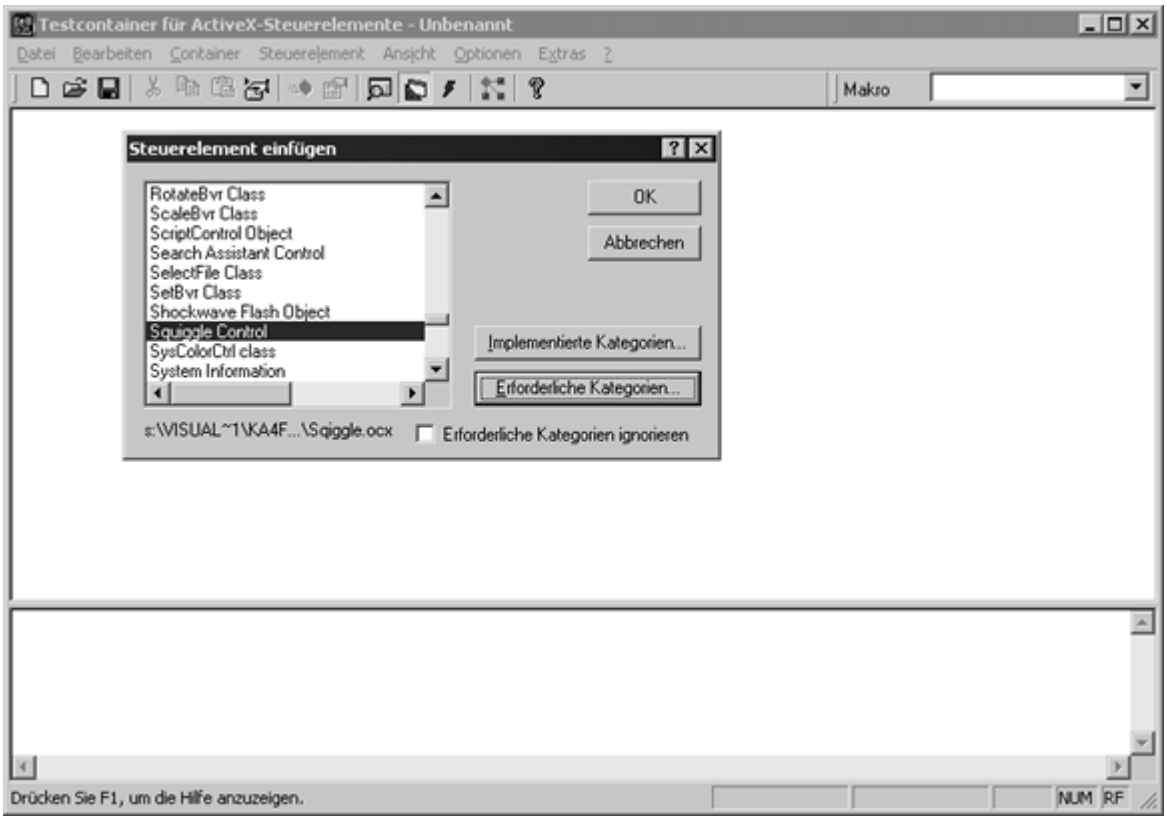


Abbildung 15.10: Das Dialogfeld Steuerelement einfügen

Nachdem Sie nun das Steuerelement in den Testcontainer geladen haben, können Sie es ausprobieren, in der Größe ändern, darauf klicken und prüfen, wann es eine neue Zeichnung generiert und wann es lediglich die vorhandene Zeichnung neu darstellt. Wenn Sie irgendwelche Ereignisse für das Steuerelement auslösen, sehen Sie das vom Steuerelement ausgelöste Ereignis im unteren Fensterbereich des Testcontainers, sodass Sie alle ausgelösten Ereignisse, die Sie für das Steuerelement vorgesehen haben, verfolgen können.

Wenn Sie bei markiertem Steuerelement den Befehl Bearbeiten / Eigenschaften wählen, erscheint die Eigenschaftsseite, die Sie für Ihr Steuerelement erstellt haben (siehe Abbildung 15.11). Hier können Sie die verschiedenen Eigenschaften des Steuerelements modifizieren, um sich davon zu überzeugen, dass sie korrekt arbeiten.

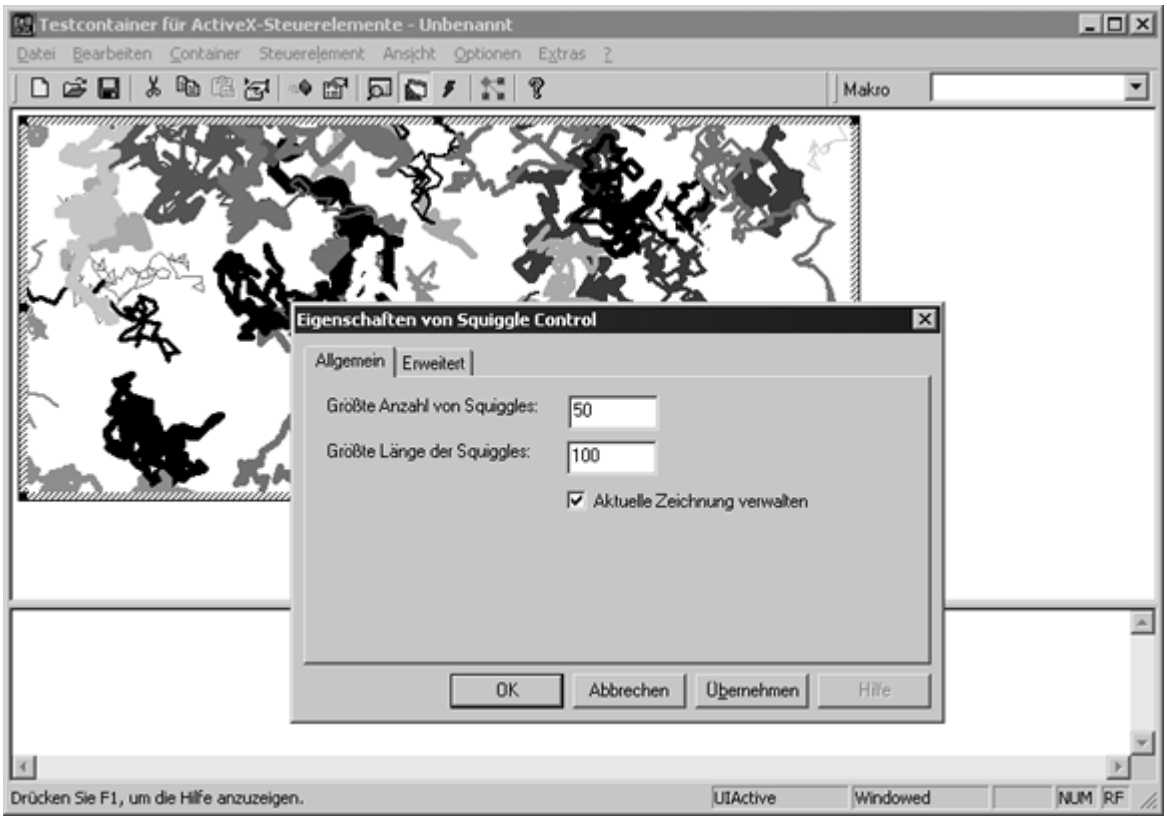


Abbildung 15.11: Die Eigenschaftsseite des Squiggle-Steuerelements

Um schließlich die dem Steuerelement hinzugefügten Methoden zu testen, wählen Sie Steuerelement / Methoden aufrufen. Daraufhin erscheint das Dialogfeld Methoden aufrufen, wie in Abbildung 15.12 zu sehen. Hier können Sie aus der Liste der für Ihr Steuerelement verfügbaren Methoden auswählen, die erforderlichen Parameter für die betreffende Methode eingeben und dann auf die Schaltfläche Aufrufen klicken, um die betreffende Methode zu starten. Dabei können Sie den Aufruf der Methode und die Reaktion des Steuerelements überwachen.

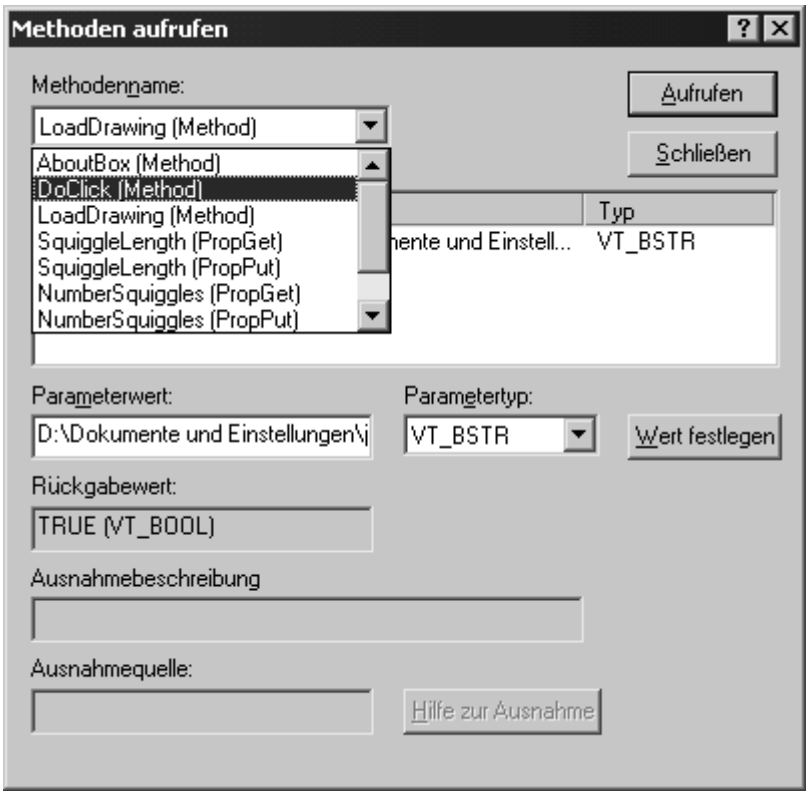


Abbildung 15.12: Das Dialogfeld Methoden aufrufen

15.3 Zusammenfassung

Heute haben Sie gelernt, wie man die Werkzeuge und Assistenten von Visual C++ einsetzt, um ActiveX-Steuerelemente mit geringem Aufwand von Ihrer Seite zu erstellen. Es wurde gezeigt, wie man das Gerüst des Steuerelementprojekts mit dem MFC-ActiveX- Steuerelement-Assistenten erstellt. Weiterhin haben Sie gesehen, wie man mit den verschiedenen Assistenten Eigenschaften, Methoden und Ereignisse in das Steuerelement aufnimmt. Sie haben eine Eigenschaftsseite für das Steuerelement angelegt und erfahren, wie man mit den verschiedenen Assistenten die Steuerelemente auf diesem Dialogfeld mit den Eigenschaften für das Steuerelement verbindet, ohne dass man dazu Code schreiben muss. Schließlich haben Sie das Steuerelement im Testcontainer für ActiveX- Steuerelemente ausprobiert und mit den Werkzeugen dieses Programms die gesamte Funktionalität ausgelöst.

15.4 Workshop

Fragen und Antworten

Frage:
Wie ändere ich das Symbol, das in der Toolbox für mein Steuerelement erscheint?

Antwort:
Öffnen Sie über die Registerkarte Ressourcenansicht des Arbeitsbereichs den Ordner Bitmap, der eine einzelne Bitmap enthalten sollte. Dieses Bild wird in der Toolbox für Ihr Steuerelement angezeigt, wenn Sie es in ein Visual C++- oder Visual Basic-Projekt einbinden. Diese Bitmap können Sie bearbeiten, sodass sie Ihr Steuerelement in gewünschter Weise repräsentiert.

Frage:
Warum weist mein Steuerelement ein Dialogfeld Info auf?

Antwort:
Wenn Sie ActiveX-Steuerelemente erstellen, die für andere Entwickler vorgesehen sind, möchten Sie sicherlich irgendwo einen Hinweis anbringen, dass Sie das Steuerelement geschrieben haben und Sie oder Ihr Brötchengeber das Copyright besitzen. Damit haben Sie eine rechtliche Kennzeichnung des Steuerelements geschaffen, sodass niemand Ihr Steuerelement ohne Ihre Zustimmung als seine eigene Kreation verkaufen darf.

Quiz

1. Welche drei Aspekte eines Steuerelements sind für die Containeranwendung sichtbar?

2. Warum müssen Sie eine Eigenschaftsseite für Ihr Steuerelement entwerfen?
3. Welche vier Arten von Eigenschaften kann ein Steuerelement besitzen?
4. Was passiert mit den Parametern, die an die Methoden eines Steuerelements übergeben werden?
5. Mit welchem Hilfsprogramm können Sie Ihre Steuerelemente testen?

Übungen

1. Nehmen Sie in Ihr Steuerelement eine Methode auf, um der Containeranwendung zu ermöglichen, das Generieren einer neuen Squiggle-Zeichnung auszulösen.

2. Nehmen Sie in Ihr Steuerelement eine Methode auf, um eine Squiggle-Zeichnung zu speichern. Verwenden Sie die Flags CFile::modeWrite und CArchive::store, wenn Sie die Objekte CFile und CArchive erstellen.

Tag 16

Funktionen für Webbrowser

Als Microsoft vor wenigen Jahren die Entscheidung getroffen hat, alle seine Anwendungen internetfähig zu machen, ging es nicht nur darum, daß man Word das Lesen und Schreiben von HTML-Seiten beibrachte. Das Internet sollte integraler Teil jeder Anwendung werden, auf die eine oder andere Art. Aus dem Blickwinkel der Entwicklungswerkzeuge betrachtet, handelt es sich natürlich nicht um eine wirklich praktische Integration, wenn man einem Editor die Doppelfunktion eines E-Mail-Clients einhaucht. Erleichtert man es allerdings den Benutzern von Entwicklungswerkzeugen, internetfähige Anwendungen zu erstellen, ist das schon eher ein praktisches Feature. Und das ist genau, was Microsoft umgesetzt hat.

Eine der Fähigkeiten, die Microsoft in seinen Werkzeugen zur Anwendungsentwicklung verfügbar gemacht hat, ist der Einsatz des Internet Explorers als integraler Teil einer beliebigen Anwendung. Das bedeutet, dass man den Internet Explorer und alle damit verbundenen Komponenten in eigene Anwendungen einbinden kann. Die Möglichkeiten gehen weit über die Bereitstellung der Webbrowser-Fähigkeiten hinaus. Ihre Anwendungen können auch Java-Applets aufnehmen und damit zusammenarbeiten. Man kann den Benutzern nicht nur eine, sondern zwei Makrosprachen bieten - VBScript und JScript (Microsofts Version von JavaScript). Sie können sogar mit dem im Browser dargestellten HTML-Dokument interagieren, es modifizieren und auf seinen Inhalt reagieren.

Heute lernen Sie, wie ...

- das ActiveX-Objektmodell des Internet Explorers die Integration all dieser Komponenten in Ihre Anwendungen gestattet,
- die Ansichtsklasse CHtmlView den größten Teil der Funktionalität des Internet Explorers in einer vorgefertigten Klasse verkapselt,
- man auf die verschiedenen COM-Schnittstellen zugreift, die im ActiveX- Objektmodell des Internet Explorers verfügbar sind,
- man die in Visual C++ integrierten Werkzeuge verwendet, um C++-Wrapper um die COM-Schnittstellen herum zu erstellen,
- man einen einfachen Webbrowser mithilfe der Klasse CHtmlView und dem Internet Explorer erstellt.

16.1 Das ActiveX-Modell des Internet Explorers

Als sich bei Microsoft der Gedanke durchsetzte, ActiveX in den Webbrowser Internet Explorer zu integrieren, wurde klar, dass man diesen technisch überholen musste, um den Einsatz von ActiveX-Steuerelementen zu unterstützen. Die Entwickler sahen sich an, was man tun musste und was im Rahmen des Möglichen lag, und entschieden sich dafür, den Internet Explorer zu mehr als zu einem Webbrowser zu machen.

Als Erstes hat Microsoft den eigentlichen Webbrowser von den ActiveX-Objekten, die die gesamte Arbeit verrichten, getrennt. Herausgekommen ist die Anwendung Internet Explorer, die etwas mehr als einen Dokument-Container für ActiveX darstellt, und das Internet Excplorer-HTML-Viewer-Steuerelement, das als ActiveX-Dokument-Server innerhalb der Anwendung läuft. Das bedeutet, dass die Anwendung Internet Explorer nicht nur Webseiten aufnehmen kann, sondern auch Word-Dokumente, Excel- Tabellenblätter, PowerPoint-Präsentationen und jedes beliebige ActiveX-Dokument, für das ein ActiveX-Dokument-Server auf demselben Computer installiert ist, wie es Abbildung 16.1 verdeutlicht.

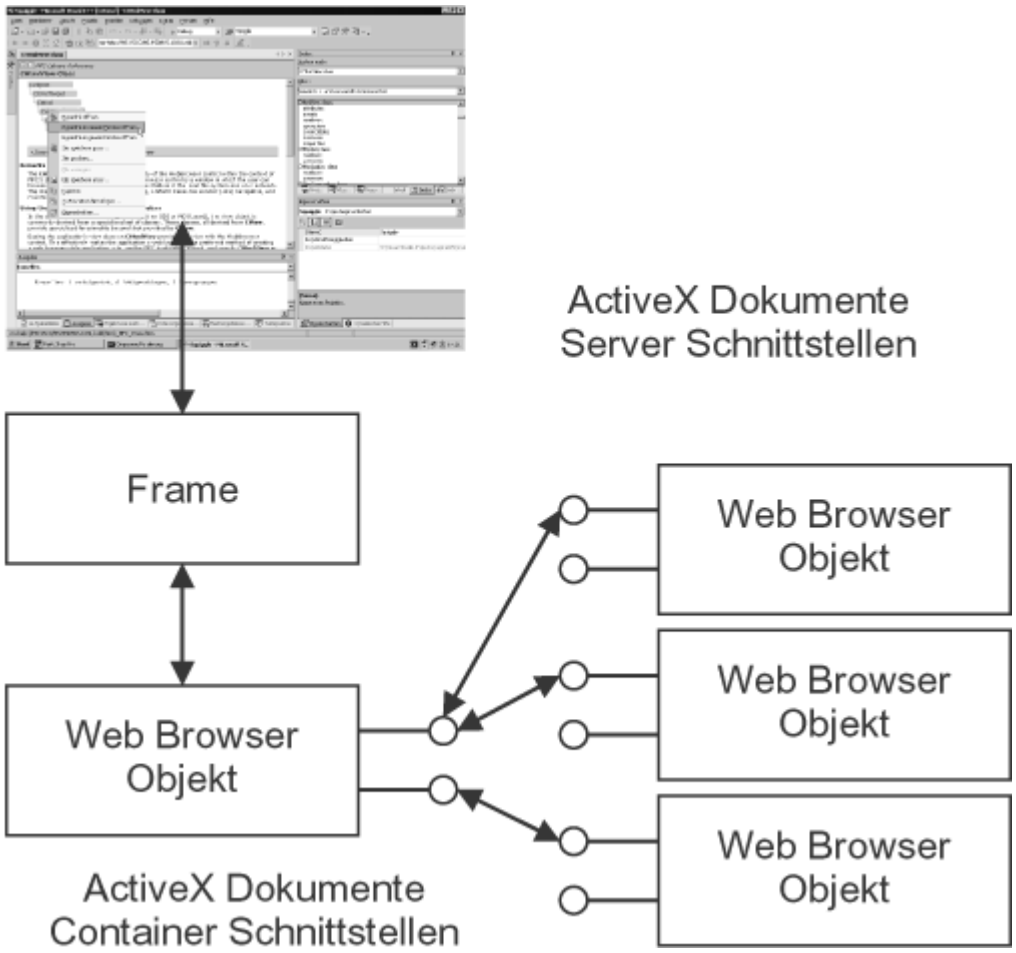


Abbildung 16.1: Das ActiveX-Dokumentmodell des Internet Explorers

Innerhalb der HTML-Viewer-Komponente hat Microsoft die Fähigkeit eingebaut, andere Steuerelemente aufzunehmen, einschließlich Scripting-Engines und ActiveX- Steuerelemente, wie es in Abbildung 16.2 dargestellt ist. Damit hat Microsoft eine Flexibilität erreicht, um dem Internet Explorer weitere Scriptsprachen hinzuzufügen, wenn diese gefordert und erstellt werden. Darüber hinaus kann der Internet Explorer beliebige ActiveX-Steuerelemente aufnehmen, die noch in den Schubläden der Entwickler liegen.

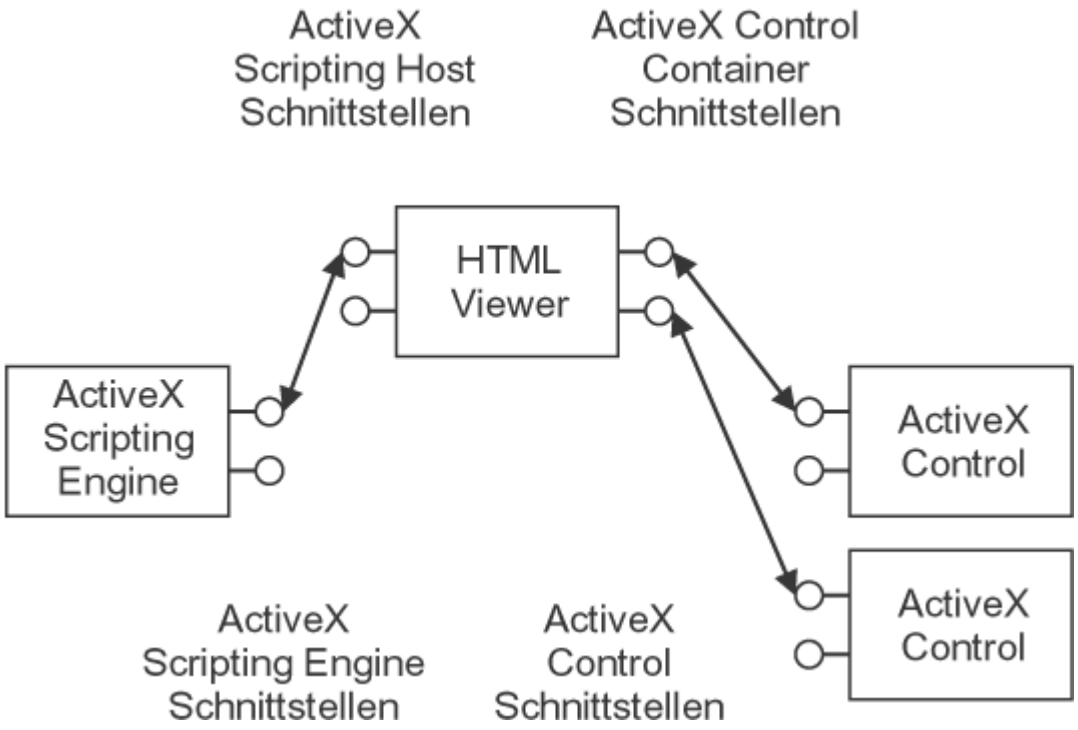


Abbildung 16.2: Das ActiveX-Objektmodell HTML Viewer des Internet Explorers

Durch dieses Konzept des Internet Explorers hat sich Microsoft nicht nur selbst eine Menge Flexibilität für zukünftige Erweiterungen der vom Internet Explorer unterstützten Funktionalität verschafft, sondern auch die gesamten Tätigkeiten des Internet Explorers jedem Entwickler verfügbar gemacht, der diese nutzen und in seine Anwendungen integrieren möchte.

16.2 Die Klasse CHtmlView

Um den Internet Explorer HTML-Viewer leicht in Visual C++-Anwendungen einbinden zu können, hat ihn Microsoft in die Klasse CHtmlView verpackt. Diese Klasse lässt sich als Basisklasse für die Ansichtsklassen in SDI- oder MDI-Anwendungen einsetzen. Man kann in einfacher Weise Anwendungen erstellen, die die Fähigkeiten eines Webbrowsers mitbringen.

Im Web navigieren

In der Klasse CHtmlView sind verschiedene Funktionen verfügbar, die sich auf die Navigation im Web beziehen. Dazu gehören Funktionen, die dem Browser die Startseite des Benutzers zurückgeben oder den Benutzer auf eine Suchseite des Internets bringen. Weiterhin gibt es Funktionen, über die der Benutzer zur vorhergehenden oder nächsten Seite gelangt oder sogar zu einer entfernten Webseite wechseln kann. Alle diese Funktionen sind Elemente der Klasse CHtmlView und demzufolge Member-Funktionen der Ansichtsklasse Ihrer Anwendung (wenn Sie die Klasse CHtmlView als Basisklasse für Ihre Ansichtsklasse verwenden). Die Navigationsfunktionen der Klasse CHtmlView fasst Tabelle 16.1 zusammen.

Die ersten vier Methoden übernehmen keine Argumente und führen genau die gleiche Funktion aus wie ihre Äquivalente auf der Symbolleiste des Internet Explorers. Von den Argumenten an die letzte Funktion ist nur der URL der anzuzeigenden Webseite erforderlich.

Methode	Beschreibung
GoBack	Bringt den Benutzer auf die vorhergehende Webseite
GoForward	Bringt den Benutzer zur nächsten Webseite (Setzt voraus, dass der Benutzer vorher mindestens eine Webseite zurückgegangen ist.)
GoHome	Bringt den Benutzer zur konfigurierten Startseite des Internet Explorer
GoSearch	Bringt den Benutzer zu einer Suchseite im Internet
Navigate	Bringt den Benutzer auf die Webseite, die in der URL-Variablen angegeben ist

Tabelle 16.1: Navigationsmethoden der Klasse CHtmlView

Den Browser steuern

Zusammen mit den Funktionen zur Navigation im Web verwenden Sie auch bestimmte Funktionen, mit denen Sie den Browser steuern. Zwei davon sind Refresh, die das HTML- Viewer-Steuerelement veranlasst, die aktuelle Webseite neu zu laden, und Stop, die einen laufenden Download anhält. Wie bei den meisten Navigationsfunktionen übernehmen diese Funktionen keine Argumente und arbeiten genau wie ihre äquivalenten Schaltflächen auf der Symbolleiste des Internet Explorers.

Den Browser-Status ermitteln

Eine weitere Kategorie von Funktionen, die in der Klasse CHtmlView verfügbar sind, haben informellen Charakter. Man kann diese Funktionen verwenden, um Informationen über den aktuellen Status des Browsers zu ermitteln. Wenn Sie zum Beispiel die aktuelle Webseite im Browser wissen wollen, rufen Sie die Funktion GetLocationURL auf, die einen CString mit dem URL zurückgibt. Wollen Sie bestimmen, ob der Browser gerade mit einem Download beschäftigt ist, rufen Sie die Funktion GetBusy auf. Diese Funktion gibt einen booleschen Wert zurück und zeigt damit an, ob der Browser beschäftigt ist oder nicht.

In der Klasse CHtmlView sind viele weitere Funktionen verfügbar. Einige davon arbeiten nur im Internet Explorer selbst und nicht im Browser-Steuerelement.

16.3 Interaktion mit COM-Schnittstellen

Es ist zwar nett, die Navigationsfähigkeiten mithilfe der Klasse CHtmlView einzufügen, doch die wirkliche Macht liegt in den COM-Schnittstellen, die im Objektmodell des Internet Explorers verfügbar sind. Bei der neuen Ausgabe von Visual C++ ist es einfacher denn je, auf diese COM-Schnittstellen zuzugreifen und mit ihnen zu arbeiten.

MFC-COM-Schnittstellen-Wrapper-Basisklassen

Die MFC-Klassenbibliothek enthält zwei als Basisklassen für COM-Schnittstellen verwendete Wrapper-Klassen. Diese Klassen bieten Zugriff auf die IDispatch-Schnittstelle und stellen unterstützende Dienste bereit, um die Schnittstellen MFC-freundlich zu machen.

Die Klasse COleDispatchDriver

Die Klasse COleDispatchDriver ist dafür gedacht, als Basisklasse für die Erstellung von Wrapper-Klassen für COM-IDispatch-Schnittstellen zu dienen. Diese Klasse bietet über die Member-Variable m_lpDispatch Zugriff auf die IDispatch-Schnittstelle. Diese Variable kann für den Aufruf der Funktion QueryInterface eingesetzt werden, um einen Zeiger auf eine beliebige Schnittstelle im COM-Objekt abzurufen.

Beim Erstellen von Wrapper-Klassen für Schnittstellen ist InvokeHelper die einzusetzende Schlüsselfunktion von COleDispatchDriver. Die Funktion InvokeHelper wird verwendet, um die IDispatch-Methode Invoke aufzurufen. Diese Funktion übernimmt eine Reihe von Parametern:

```
void InvokeHelper(DISPID dwDispID, WORD wFlags, VARTYPE atRet,
                  void* pvRet, const BYTE* pbParamInfo, . . .);
```

Der erste Parameter, dwDispID, ist die ID der aufzurufenden Methode oder Eigenschaft. Sie können den OLE/COM Objektkatalog (im Menü Extras) verwenden, um den numerischen Wert der Funktion oder Eigenschaft zu ermitteln, die Sie aufrufen möchten. Wenn Sie die Schnittstelle gefunden haben, durchsuchen Sie die Liste der Eigenschaften und Methoden, bis Sie gefunden haben, was Sie suchen, und verwenden dann die ID- Nummer unmittelbar darüber, wie es Abbildung 16.3 zeigt.

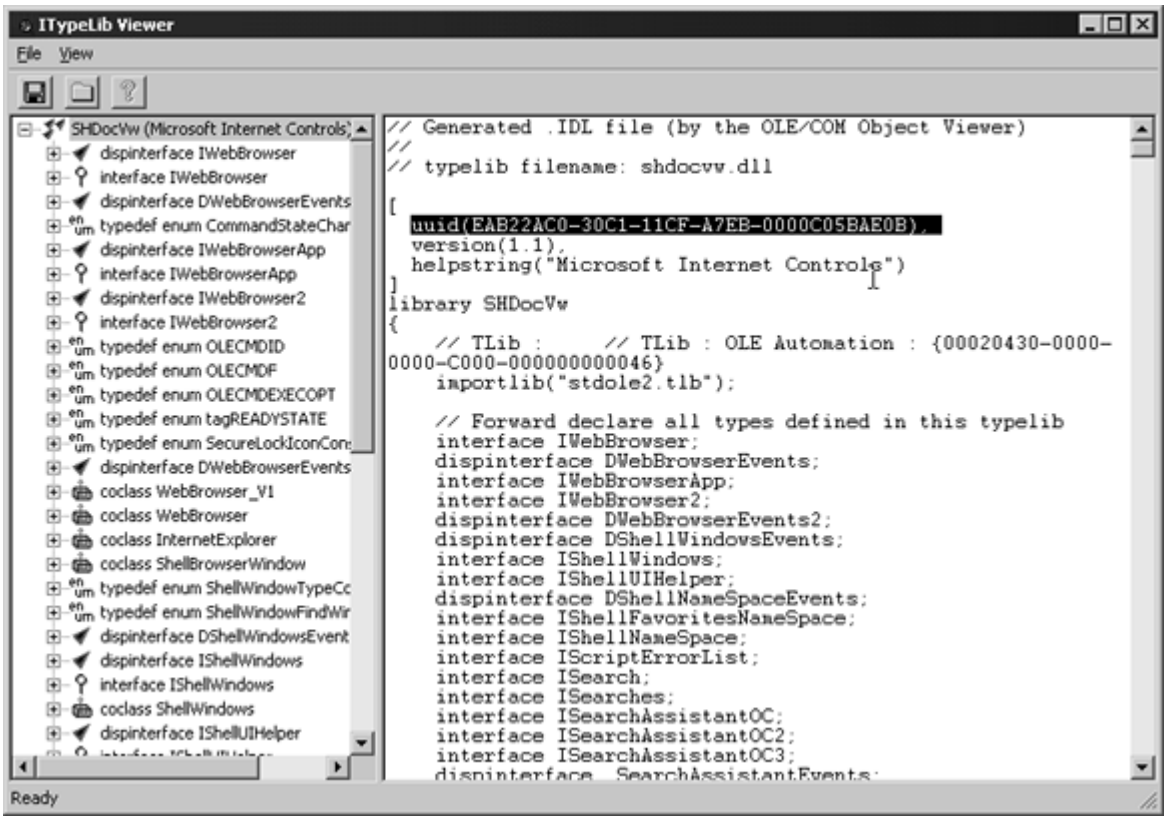


Abbildung 16.3: Mit dem Objektkatalog für OLE/COM können Sie IDs von Methoden und Eigenschaften ermitteln.

Der zweite Parameter, wFlags, gibt den Kontext des Methodenaufrufs an. Die möglichen Werte für diesen Parameter sind in Tabelle 16.2 aufgelistet.

Flag	Beschreibung
DISPATCH_METHOD	Ruft das IDispatch-Element als Methode auf.
DISPATCH_PROPERTYGET	Ruft das IDispatch-Element auf, um den Wert einer Eigenschaft abzufragen
DISPATCH_PROPERTYPUT	Ruft das IDispatch-Element auf, um den Wert einer Eigenschaft zu setzen
DISPATCH_PROPERTYPUTREF	Ruft das IDispatch-Element auf, um den Wert einer Eigenschaft durch Zuweisen einer Referenz zu setzen. Dieser Kontext lässt sich nur bei Eigenschaften verwenden, die eine Referenz auf ein Objekt akzeptiert.

Tabelle 16.2: Kontextflags von InvokeHelper

Der dritte Parameter, atRet, gibt den Rückgabetyp der Methode oder Eigenschaft an. Der Rückgabetyp kann aus den als out oder retval bezeichneten Parametern in der Schnittstellenspezifikation ermittelt werden, die im OLE/COM-Objektviewer zu sehen ist. Die möglichen Werte für diesen Parameter sind in Tabelle 16.3 aufgelistet.

Parameterwert	Datentyp
VT_EMPTY	void (kein Rückgabewert)
VT_I2	Short
VT_I4	Long
VT_R4	Float

VT_R8	Double
VT_CY	CURRENCY (Währung)
VT_DATE	DATE
VT_BSTR	BSTR
VT_DISPATCH	LPCISPATCH (Zeiger auf eine IDispatch-Schnittstelle)
VT_ERROR	SCODE
VT_BOOL	BOOL
VT_VARIANT	VARIANT
VT_UNKNOWN	LPUNKNOWN (Zeiger auf eine IUnknown-Schnittstelle)

Tabelle 16.3: Rückgabetypen von InvokeHelper

Der fünfte Parameter, pbParamInfo, ist ein Array von BYTEs, das die Datentypen der an die Methode oder Eigenschaft übergebenen Parameter angibt. Die möglichen Werte, die in diesem Array verwendet werden können, sind in Tabelle 16.4 aufgelistet. Sie können diesen Parameter folgendermaßen erzeugen:

```
static BYTE parms[] = VTS_BSTR;
```

Auf diesen Parameter folgen alle Parameter, die an die Methode oder Eigenschaft übergeben werden. Wenn die Methode oder Eigenschaft keine Parameter übernimmt, muss an dieser Stelle (nach dem Parameter pbParamInfo) NULL übergeben werden.

Parameterwert	**Datentyp**
VTS_I2	short
VTS_I4	long
VTS_R4	float
VTS_R8	double
VTS_COLOR	OLE_COLOR
VTS_CY	CURRENCY
VTS_DATE	DATE
VTS_BSTR	BSTR
VTS_DISPATCH	LPDISPATCH (Zeiger auf eine IDispatch-Schnittstelle)
VTS_FONT	IFontDispatch-Schnittstellenzeiger
VTS_HANDLE	HANDLE
VTS_SCODE	SCODE
VTS_BOOL	BOOL
VTS_VARIANT	const VARIANT
VTS_PVARIANT	VARIANT*
VTS_UNKNOWN	LPUNKNOWN (Zeiger auf eine IUnknown-Schnittstelle)
VTS_OPTEXCLUSIVE	OLE_OPTEXCLUSIVE
VTS_PICTURE	IPictureDisp-Schnittstellenzeiger
VTS_TRISTATE	OLE_TRISTATE
VTS_XPOS_PIXELS	OLE_XPOS_PIXELS
VTS_YPOS_PIXELS	OLE_YPOS_PIXELS
VTS_XSIZE_PIXELS	OLE_XSIZE_PIXELS

VTS_YSIZE_PIXELS	OLE_YSIZE_PIXELS
VTS_XPOS_HIMETRIC	OLE_XPOS_HIMETRIC
VTS_YPOS_HIMETRIC	OLE_YPOS_HIMETRIC
VTS_XSIZE_HIMETRIC	OLE_XSIZE_HIMETRIC
VTS_YSIZE_HIMETRIC	OLE_YSIZE_HIMETRIC

Tabelle 16.4: InvokeHelper-Parametertypen

Die typische Verwendung der Klasse InvokeHelper für die Erstellung einer Wrapper-Klasse könnte folgendermaßen aussehen:

```
CString CMyWrapper::get_designMode()
{
  CString result;
  InvokeHelper(0x3f6, DISPATCH_PROPERTYGET, VT_BSTR,
               (void*)&result, NULL);
  return result;
};
void CMyWrapper::put_designMode(CString newValue)
{
  static BYTE parms[] = VTS_BSTR ;
  InvokeHelper(0x3f6, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL,
               parms, newValue);
};
```

Die einfachste Möglichkeit, eine Instanz des Abkömmlings der Klasse COleDispatchDriver zu erzeugen, ist, dem Konstruktor einen Zeiger auf die IDispatch-Schnittstelle der COM- Schnittstelle zu übergeben, für die die Klasse als Wrapper dienen soll:

```
CMyWrapper* pWrap = new CMyWrapper(ppvDisp);
```

oder

```
CMyWrapper comWrap(ppvDisp);
```

Den Zeiger auf die Schnittstelle erhalten Sie mithilfe der Methode QueryInterface durch einen beliebigen Schnittstellenzeiger auf das COM-Objekt. Wenn Sie einen Zeiger auf eine IDispatch-Schnittstelle in der Klasse haben, können Sie mit seiner Hilfe einen IDispatch-Schnittstellenzeiger für eine beliebige andere Schnittstelle im COM-Objekt abrufen:

```
void* ppvDisp;

pWrap1->m_lpDispatch->QueryInterface(IID_IHTMLDocument, &ppvDisp);
```

Die Klasse CWnd

Wenn Sie mit ActiveX-Steuerelementen arbeiten, können Sie die Klasse CWnd als Basisklasse für manche Schnittstellen verwenden. Wenn sich die Wrapper-Klasse der Schnittstelle von CWnd ableitet, können Sie die abgeleitete Klasse als Variablentyp für das auf dem Anwendungsfenster platzierte Steuerelement verwenden. Wenn CWnd als Basisklasse für eine COM-Schnittstelle verwendet wird, besitzt es eine Member-Variable namens m_pOuterUnknown, bei der es sich um einen Schnittstellenzeiger handelt, der für den Aufruf der Methode QueryInterface verwendet werden kann, mit der man einen Zeiger auf eine andere Schnittstelle im Steuerelement abrufen kann. Wenn das Steuerelement keine sichtbare Schnittstelle besitzt, kann CWnd nicht als seine Basisklasse verwendet werden.

Wrapper-Klassen für Schnittstellen erzeugen

In älteren Ausgaben von Visual C++ mussten Sie die ganze Arbeit alleine erledigen, wenn Sie eine Wrapper-Klasse für eine COM-Schnittstelle erstellen wollten. Dazu mussten Sie die Schnittstelle im OLE/COM-Objektviewer untersuchen und alle Methoden und Eigenschaftsmethoden entweder mit der Klasse COleDispatchDriver oder mit CWnd erstellen. In der neuen Version von Visual C++ können Assistenten die Wrapper-Klassen für Sie erstellen. Sie müssen nur das ActiveX-Steuerelement oder die COM-Schnittstelle angeben, die Schnittstellen auswählen, für die Wrapper-Klassen erstellt werden sollen, und dem Assistenten den Rest überlassen.

Schnittstellen-Wrapper-Klassen für ActiveX-Steuerelemente

Bei der Verwendung des Dialogfelds Klasse hinzufügen sind Ihnen möglicherweise zwei Vorlagen für die Erstellung einer neuen Klasse basierend auf einem ActiveX- Steuerelement oder einer TypeLib aufgefallen. Diese beiden Vorlagen für neue Klassen öffnen Assistenten, die Wrapper-Klassen für Steuerelement- und Objektschnittstellen erzeugen. Wenn Sie MFC-Klasse aus ActiveX-Steuerelement auswählen, wird der Assistent zum Hinzufügen von Klassen aus ActiveX-Steuerelementen gestartet, der in Abbildung 16.4 gezeigt ist.

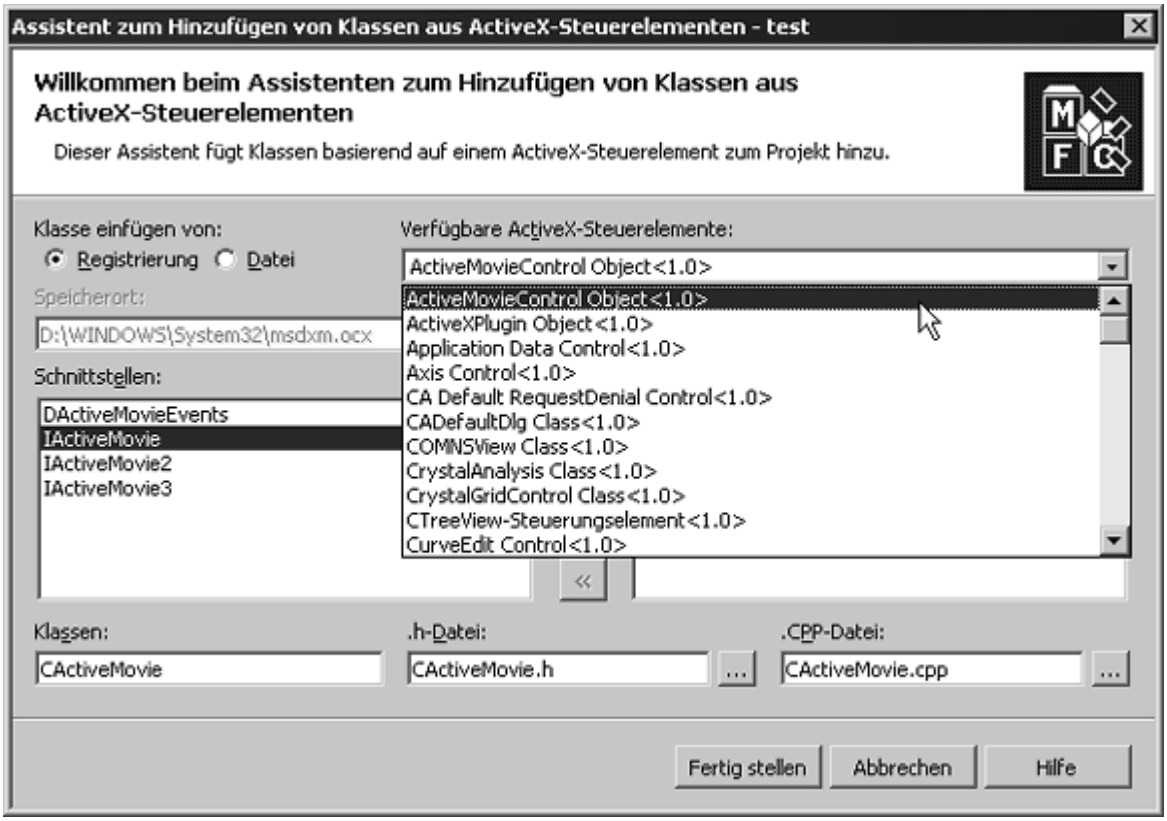


Abbildung 16.4: Der Assistent zum Hinzufügen von Klassen aus ActiveX-Steuerelementen

In diesem Assistenten können Sie angeben, ob Sie ein bereits auf Ihrem System registriertes Steuerelement auswählen oder die OCX-Datei des Steuerelements angeben möchten. Wenn Sie das Steuerelement aus den registrierten Steuerelementen auswählen wollen, können Sie das in einem Kombinationsfeld tun, das alle derzeit auf Ihrem System registrierten ActiveX-Steuerelemente enthält.

Nachdem Sie ein Steuerelement ausgewählt haben, sehen Sie auf der linken Seite des Assistents eine Liste der verfügbaren Schnittstellen. Sie müssen die Schnittstellen markieren, für die Sie Wrapper-Klassen erstellen möchten, und auf die Schaltfläche mit der spitzen Klammer nach rechts (>) klicken, um die Schnittstellen zur Liste der zu erzeugenden Wrapper-Klassen hinzuzufügen. Die Namen der erzeugten Klassen werden ebenso wie die Dateinamen automatisch aus dem Namen der Schnittstelle generiert. Wenn Sie eine der Klassen oder Dateien umbenennen möchten, bearbeiten Sie sie in den Eingabefeldern am unteren Ende des Assistents. Nachdem Sie alle Schnittstellen ausgewählt haben, für die Wrapper-Klassen erstellt werden, klicken Sie auf die Schaltfläche Fertig stellen, um die Klassen zu erzeugen, die Ihrem aktuellen Projekt hinzugefügt werden sollen. Abhängig von der ausgewählten Schnittstelle leitet sich die erzeugte Wrapper-Klasse entweder von CWnd oder von COleDispatchDriver ab.

Schnittstellen-Wrapper-Klassen für COM-Objekte

Wenn Sie eine Wrapper-Klasse für ein COM-Objekt erstellen wollen, wählen Sie die Vorlage MFC-Klasse aus der Typenbibliothek. Der Assistent zum Hinzufügen von Klassen aus der Typenbibliothek wird geöffnet. Dieser Assistent funktioniert grundsätzlich genauso wie der Assistent zum Hinzufügen von Klassen aus ActiveX-Steuerelementen, mit dem Unterschied, dass es am unteren Ende des Assistenten ein Eingabefeld für den Dateinamen der Wrapper-Klasse gibt. Der Grund dafür ist, dass sich die für COM- Steuerelemente aus der TypeLibs erstellten Wrapper-Klassen alle von der Klasse COleDispatchDriver ableiten und als Inline-Funktionen innerhalb der Klassendeklaration implementiert sind. Da alle Methoden in der Klassendeklaration deklariert werden, erstellt der Assistent nur eine Header-Datei für die Wrapper-Klasse.

Die IHTMLDocument-Schnittstelle aus der CHTMLView- Klasse abfragen

Durch geschickten Einsatz der Werkzeuge, die Visual C++ bereitstellt, können Sie problemlos mit ActiveX-Steuerelementen und COM-Objekten arbeiten. Sie müssen das Steuerelement oder Objekt nur auswählen, die passenden Wrapper-Klassen erzeugen und los geht's. Jedenfalls fast. Sie müssen sich nämlich noch diesen ersten Schnittstellenzeiger aneignen, von dem aus Sie einen Zeiger auf die IDispatch-Schnittstelle einer der anderen Schnittstellen im Objekt oder Steuerelement abrufen können.

Wenn Sie ein ActiveX-Steuerelement in einem Dialogfeld verwenden, können Sie mit dem Variable hinzufügen Assistent eine Steuervariable deklarieren. Es wird eine Standard- Wrapper-Klasse erzeugt, die auf CWnd basiert und nicht viel Funktionalität bereitstellt. Was sie allerdings bereitstellt, ist die Member-Variable m_pOuterUnknown, bei der es sich um einen Zeiger auf die IUnknown-Schnittstelle handelt. Sie kann verwendet werden, um einen Zeiger auf eine beliebige andere Schnittstelle abzurufen.

Wenn Sie mit nicht sichtbaren ActiveX-Steuerelementen oder anderen COM-Objekten arbeiten, können Sie einen Schnittstellenzeiger abrufen, indem Sie eine Instanz der Schnittstelle mit dem Objektnamen erzeugen und sie mit dem Namen der Schnittstelle initialisieren:

```
IMyObjectPtr pMO(L"MyObject.MyInterface.1");
```

Für diese Vorgehensweise benötigen Sie die #import-Direktive, um die Typenbibliothek für das COM-Objekt zu importieren, bevor der Schnittstellenzeiger erzeugt wird. Sie müssen außerdem die COM-Umgebung durch einen Aufruf der API-Funktion CoInitialize initialisiert haben.

In der Klasse CHTMLView ist das alles stark vereinfacht. Die Ansichtsklasse besitzt eine Member-Funktion namens GetHTMLDocument, die einen Zeiger auf die IDispatch- Schnittstelle des IHTMLDocument-Objekts zurückgibt. Über diesen Schnittstellenzeiger können Sie Schnittstellenzeiger auf jede beliebige andere COM-Objektschnittstelle innerhalb des Objektmodells des Internet Explorers abrufen.

16.4 Eine Webbrowser-Anwendung erstellen

Als Beispiel, wie man die Webbrowser-Komponente des Internet Explorers in eigene Anwendungen einbindet, erstellen Sie heute eine einfache Webbrowser-Anwendung. Das Ganze baut auf einer SDI-Anwendung auf, die die Klasse CHtmlView als Basisklasse der Ansichtsklasse verwendet. Für die Navigation fügen Sie ein Menü mit den Befehlen Zurück und Vorwärts hinzu. Weiterhin erhält die Anwendung ein Dialogfeld, um dem Benutzer die Eingabe eines URL zu ermöglichen. Damit navigieren Sie den Browser zur angegebenen Webseite.

Das Anwendungsgerüst erstellen

Für eine Webbrowser-Anwendung können Sie auf ein normales SDI- oder MDI- Anwendungsgerüst zurückgreifen. Der Internet Explorer muss auf dem Computer installiert sein, auf dem Ihre Anwendung läuft. Bei Ihrem Entwicklungscomputer ist das kein Problem, weil die Installation von Visual C++ ohnehin darauf dringt, dass Sie die neueste Version des Internet Explorers installieren. Bei jedem anderen Computer, auf dem Ihre Anwendung laufen soll, müssen Sie sich allerdings zuerst davon überzeugen, ob der Internet Explorer installiert ist. Andernfalls müssen Sie die Installation selbst vornehmen.

Um das Gerüst der heutigen Beispielanwendung zu erstellen, folgen Sie diesen Schritten:

1. Legen Sie ein neues MFC-Anwendungs-Projekt an. Nennen Sie das Projekt WebBrowse und klicken Sie auf OK, um den MFC-Anwendungs-Assistenten zu starten.

2. Im Bereich Anwendungstyp des MFC-Anwendungs-Assistenten wählen Sie die Option Einfaches Dokument.
3. Im Bereich Benutzeroberflächenfeatures des MFC-Anwendungs-Assistenten wählen Sie unter Toolbars die Optionen Andockbar (Standard) und Browser- Stil.
4. Im Bereich Erstellte Klassen des MFC-Anwendungs-Assistenten legen Sie die Klasse CHtmlView als Basisklasse für die Klasse CWebBrowseView fest.
5. Klicken Sie auf Fertig stellen, um das Anwendungsgerüst zu erzeugen.

Wenn Sie dieses Gerüst der Anwendung kompilieren und starten sowie eine Verbindung zum Internet herstellen, haben Sie bereits einen funktionsfähigen Webbrowser vor sich, wie es Abbildung 16.5 zeigt. Allerdings lässt sich noch nicht festlegen, wohin Sie der Browser bringen soll, es sei denn, Sie klicken auf die Links in den angezeigten Webseiten.

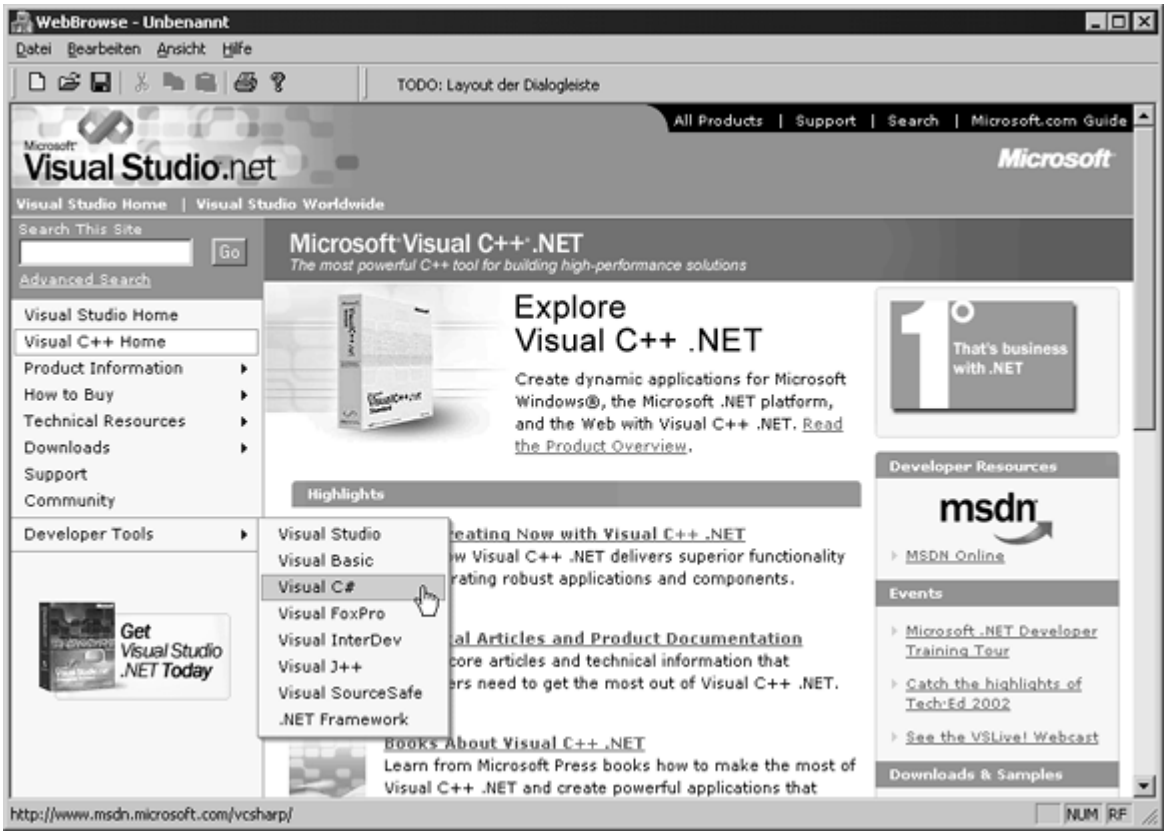


Abbildung 16.5: Die anfängliche Webbrowser-Anwendung

Funktionen zum Navigieren hinzufügen

Mit einem funktionsfähigen Browser allein ist es noch nicht getan. Man muss auch steuern können, wohin der Browser navigieren soll. Es ist ein Eingabefeld erforderlich, in das der Benutzer einen URL eingeben kann. Wenn Sie sich die Symbolleiste der laufenden Anwendung ansehen, finden Sie einen Platz, wo sich dieses Steuerelement unterbringen lässt.

Einen URL spezifizieren

Wenn Sie die Anwendung ausführen, haben Sie wahrscheinlich schon an dem Text in der zweiten Symbolleiste bemerkt, dass an dieser Stelle das Layout der Dialogleiste unterzubringen ist. Die Dialogleiste unterscheidet sich von dem, was Sie bisher kennen gelernt haben. Es handelt sich um eine Symbolleiste, auf der normale Steuerelemente wie bei einem Dialogfeld enthalten sind. Die Leiste entwerfen Sie sogar mit dem Dialog- Designer. Wenn Sie auf der Registerkarte Ressourcen nach der Dialogleiste suchen, finden Sie sie nicht im Ordner Toolbar, sondern im Ordner Dialog.

Öffnen Sie den Ordner Dialog und doppelklicken Sie auf das Dialogfeld IDR_MAINFRAME, um es im Dialog-Designer zu öffnen. Es zeigt sich, dass dieses Dialogfeld die zweite Symbolleiste in Ihrer Anwendung verkörpert. Auf dieser Symbolleiste lassen sich Eingabefelder, Schaltflächen, Kombinationsfelder und Kontrollkästchen platzieren - praktisch jedes beliebige Steuerelement, das Sie auch in einem normalen Dialogfeld verwenden können.

1. Modifizieren Sie das bereits auf der Dialogleiste vorhandene Textfeld und fügen Sie ein Eingabefeld hinzu, wie es Abbildung 16.6 zeigt.

2. Legen Sie für dieses Beispiel die ID des Eingabefelds mit IDC_EADDRESS fest.

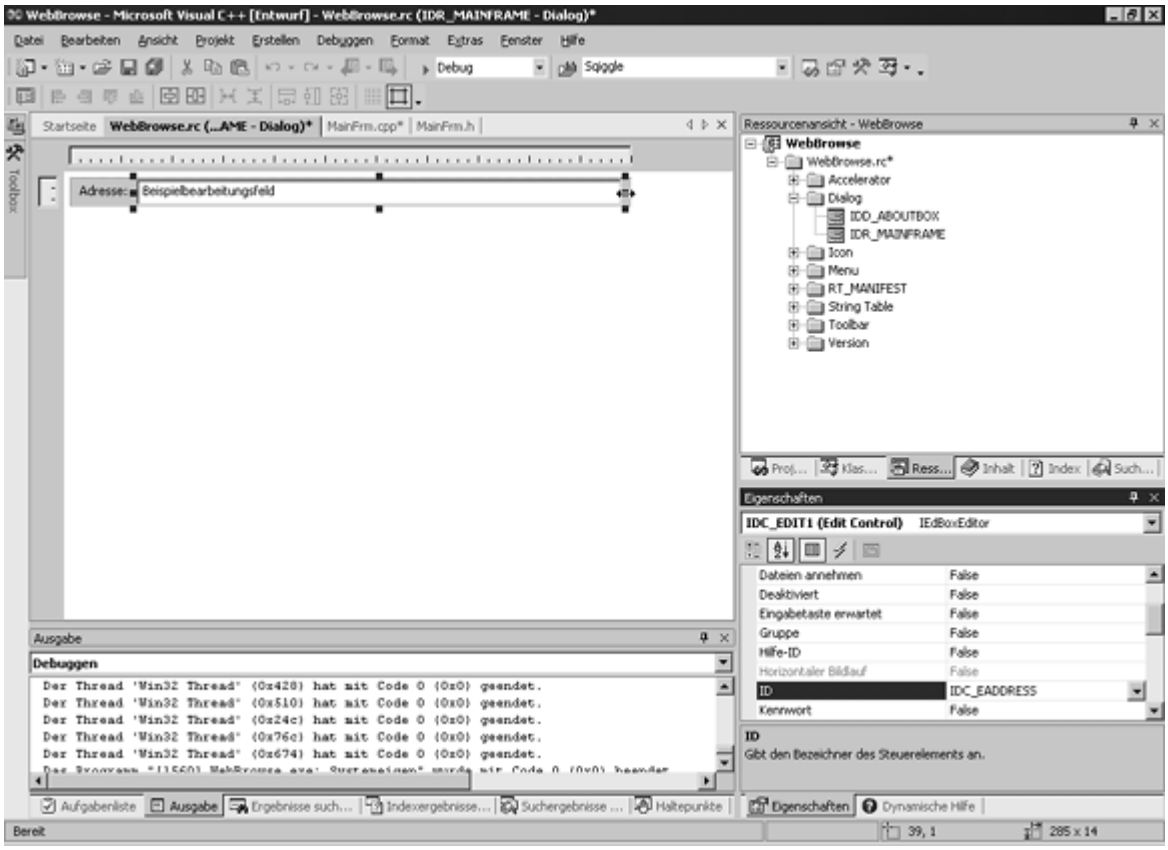


Abbildung 16.6: Das Layout der Dialogleiste

Bevor Sie den Klassen-Assistenten öffnen, um Variablen und Behandlungsroutinen für die Dialogleiste hinzuzufügen, sollten Sie wissen, dass die Dialogleiste ihre Ereignisse automatisch an die Hauptrahmenklasse Ihrer Anwendung schickt. Sie müssen keine Dialogfeldklasse für diese Symbolleiste erstellen, da Sie alle Ereignisse über den Rahmen abbilden und von dort aus an die Ansichts- oder Dokumentklassen weiterleiten können.

Für die heutige Beispielanwendung brauchen Sie nicht einmal den Ereignis-Handler- Assistenten zu bemühen, um irgendwelche Behandlungsroutinen für die Dialogleiste aufzunehmen. Sie müssen eine Aktion auslösen, wenn der Benutzer den URL vollständig in das Eingabefeld eingegeben hat. Von den im Klassen-Assistenten verfügbaren Ereignissen kommt am ehesten EN_CHANGED in Frage, das jeden vom Benutzer eingegebenen Buchstaben meldet. Wir brauchen aber ein Ereignis, das ausgelöst wird, wenn der Benutzer die (Enter)-Taste drückt. Zum Glück wird die Befehls-ID IDOK an die Rahmenklasse gesendet, wenn der Benutzer das Eingabefeld in der Dialogleiste ausfüllt und die (Enter)-Taste drückt. Man kann in diesem Fall eine Behandlungsroutine in die Nachrichtenzuordnungstabelle einfügen, um eine Funktion für den Befehl IDOK aufzurufen.

In der Behandlungsroutine des Befehls müssen Sie den Fenstertext aus dem Eingabefeld in der Dialogleiste ermitteln. Diesen String können Sie an die Funktion Navigate in der Ansichtsklasse übergeben und damit den Browser veranlassen, zu der vom Benutzer angegebenen Seite zu gehen.

Um diese Funktionalität in Ihre Anwendung einzufügen:

1. fügen Sie eine neue Member-Funktion in die Klasse CMainFrame ein,

2. legen Sie den Funktionstyp als void, den Funktionsnamen als OnNewAddress und den Zugriff als public fest. In die Funktion schreiben Sie den Code aus Listing 16.1.

Listing 16.1: Die Funktion CMainFrame.OnNewAddress

```
 1: void CMainFrame::OnNewAddress(void)
 2: {
 3:   CString strAddress;
 4:
 5:   // Neuen URL ermitteln
 6:   m_wndDlgBar.GetDlgItem( IDC_EADDRESS
 7:                           )->GetWindowText(strAddress);
 8:   // Zum neuen URL navigieren
 9:   ((CWebBrowseView*)GetActiveView())->Navigate(strAddress);
10: }
```

Die Funktion fragt mit der Funktion GetWindowText den Text im Eingabefeld ab und schreibt ihn in die Variabl

In der nächsten Zeile findet die Typumwandlung des aus der Funktion GetActiveView zurückgegebenen Zeigers auf eine CView-Klasse in einen Zeiger auf CWebBrowseView statt. Damit kann man die Funktion Navigate auf der Ansichtsklasse aufrufen und den URL übergeben, welchen der Benutzer im Eingabefeld spezifiziert hat.

Nachdem Sie nun den vom Benutzer eingegebenen URL übernehmen und die Browser- Komponente anweisen können, zu dieser Webseite zu gehen, wie lösen Sie dann diese Funktion aus? Dazu müssen Sie manuell einen Eintrag in die Nachrichtenzuordnungstabelle einfügen, da der Klassen-Assistent dazu nicht in der Lage ist. Fügen Sie in der Nachrichtenzuordnungstabelle das Makro ON_COMMAND ein, um damit den Befehl IDOK und Ihre neue Funktion als aufzurufenden Handler zu spezifizieren, wie es Listing 16.2 zeigt.

Listing 16.2: Die Nachrichtenzuordnungstabelle von CMainFrame

```
1: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
2:     ON_WM_CREATE()
3:     ON_COMMAND(IDOK, OnNewAddress)
4: END_MESSAGE_MAP()
```

Bevor Sie Ihre Anwendung kompilieren können, müssen Sie im Quellcode des Hauptrahmens #include-Direktiven einfügen, um die Header-Dateien für die Dokument- und Ansichtsklassen einzubinden, wie es Listing 16.3 zeigt.

Listing 16.3: Die #include-Direktiven von CMainFrame

```
 1: // MainFrm.cpp : Implementierung der Klasse CMainFrame
 2: //
 3:
 4: #include "stdafx.h"
 5: #include "WebBrowse.h"
 6:
 7: #include "MainFrm.h"
 8: #include "WebBrowseDoc.h"
 9: #include "WebBrowseView.h"
10:
11: #ifdef _DEBUG
12: #define new DEBUG_NEW
13: #endif
```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie einen URL in das Eingabefeld auf der Symbolleiste eintragen und die (Enter)-Taste drücken und Ihre Anwendung sollte zur angegebenen Webseite gehen, wie es Abbildung 16.7 zeigt.

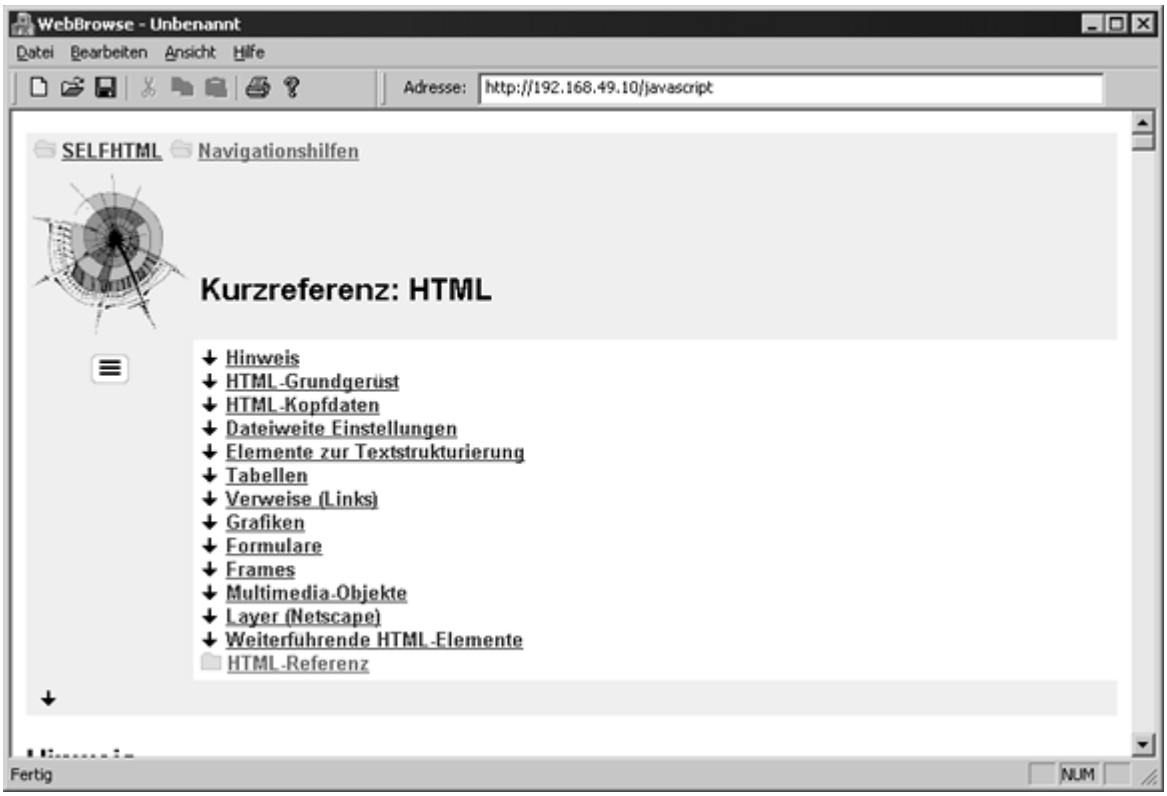


Abbildung 16.7: Zu einem bestimmten URL gehen

Den aktuellen URL anzeigen

Wenn Sie durch das Web surfen, folgen Sie oftmals Verweisen auf Webseiten, die zu anderen Websites führen. In diesem Fall wissen Sie nicht, auf welche Website Sie zugreifen, wenn Ihr Browser nicht den URL in das Adressfeld stellt und damit Ihren aktuellen Standort angibt sowie die Möglichkeit bietet, den URL zu kopieren oder zu modifizieren, um eine andere Seite auf derselben Site zu suchen.

Das Abfragen des aktuellen URL vom Browser ist sehr einfach: Man ruft die Funktion GetLocationURL auf und übergibt das Ergebnis an die Dialogleiste. Das Problem besteht in dem Zeitpunkt, zu dem man den URL holt. Es stellt sich heraus, dass sich manche Ereignisfunktionen in der Klasse CHtmlView in Ihrer Klasse überschreiben lassen. Diese Funktionen werden bei verschiedenen Ereignissen ausgelöst, die das Browser-Steuerelement initiiert. Es gibt Ereignisfunktionen für das Starten der Navigation, den Beginn eines Downloads, die Überwachung des Download-Fortschritts und - am wichtigsten für unsere Erfordernisse - die Anzeige, dass der Download abgeschlossen ist.

Folgen Sie diesen Schritten, um die Behandlungsroutine für einen abgeschlossenen Download in Ihre Anwendung einzubauen:

1. Markieren Sie die Klasse CWebBrowseView in der Klassenansicht.

2. Wählen Sie im Eigenschaftenfenster den Modus Überschreibungen aus.
3. Scrollen Sie nach unten, um die Funktion OnDocumentComplete zu finden.
4. Klicken Sie in die Wertseite und wählen Sie <Hinzufügen> OnDocumentComplete aus dem Kombinationsfeld, um die Überschreibungs- Funktion zu erzeugen.
5. In die Funktion schreiben Sie den Code aus Listing 16.4.

Listing 16.4: Die Funktion CWebBrowseView.OnDocumentComplete

```
1: void CWebBrowseView::OnDocumentComplete(LPCTSTR lpszURL)
2: {
3:    // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
4:    // und/oder rufen Sie die Basisklasse auf.
5:    // Neuen URL an Adressleiste übergeben
```

```
6:    ((CMainFrame*)GetParentFrame())->SetAddress(lpszURL);
7:
8:    CHtmlView::OnDocumentComplete(lpszURL);
9: }
```

In dieser Funktion fällt auf, dass man die Funktion GetLocationURL überhaupt nicht aufrufen muss. Der heruntergeladene URL wird als Argument an die Funktion übergeben. Damit können Sie den URL an den Rahmen weiterreichen, wo Sie eine andere Funktion hinzufügen, um das Eingabefeld auf der Dialogleiste mit dem URL zu füllen.

Um die Funktion einzufügen, die die Dialogleiste mit der neuen URL ausfüllt:

1. Nehmen Sie eine Member-Funktion in die Hauptrahmenklasse CMainFrame auf.

2. Legen Sie den Funktionstyp als void und den Funktionsnamen als SetAddress fest und fügen Sie einen Parameter vom Typ LPCTSTR namens lpszURL ein. Legen Sie den Zugriff als public fest.
3. In die Funktion übernehmen Sie den Code aus Listing 16.5.

Listing 16.5: Die Funktion CMainFrame.SetAddress

```
1: void CMainFrame::SetAddress(LPCTSTR lpszURL)
2: {
3:    // Neue URL in das Adresseingabefeld eintragen
4:    m_wndDlgBar.GetDlgItem(IDC_EADDRESS)->SetWindowText(lpszURL);
5: }
```

Hier haben Sie den entgegengesetzten Weg eingeschlagen, mit dem Sie den Text aus dem Eingabefeld geholt haben. Die Funktion SetAddress ändert den Text im Eingabefeld mit der Funktion SetWindowText auf den übergebenen URL.

4. Fügen Sie gemäß Listing 16.6 eine #include-Direktive in die Klasse CWebBrowseView ein, um den Header der Hauptrahmenklasse einzubinden.

Listing 16.6: Die #include-Direktiven von CWebBrowseView

```
 1: // WebBrowseView.cpp : Implementierung der Klasse CWebBrowseView
 2: //
 3:
 4: #include "stdafx.h"
 5: #include "WebBrowse.h"
 6:
 7: #include "WebBrowseDoc.h"
 8: #include "WebBrowseView.h"
 9: #include "MainFrm.h"
10:
11: #ifdef _DEBUG
12: #define new DEBUG_NEW
13: #endif
```

Wenn Sie die Anwendung ausführen, sollte die URL-Adresse in der Dialogleiste die momentan angezeigte Webseite widerspiegeln.

Zurück und Vorwärts

Nachdem Sie einen URL in die Dialogleiste eingeben und Ihre Anwendung veranlassen können, zu dieser Website zu gehen, und auch die Adresse aller betrachteten Websites zu sehen sind, wäre es schön, wenn man wieder an die Stelle zurückkehren könnte, von der man gekommen ist. Dazu ruft man einfach die Funktionen GoBack und GoForward in der Ansichtsklasse der Anwendung auf. Die Funktionen lassen sich über Menübefehle aktivieren, was Ihnen außerdem ermöglicht, dieselben Aufrufe über Schaltflächen der Symbolleiste zu realisieren.

Um diese Funktionalität zu implementieren:

1. Öffnen Sie das Hauptmenü im Menü-Designer.

2. Aus der Menüleiste können Sie das Menü Bearbeiten mit allen zugehörigen Menübefehlen löschen, da sie für die heutige Beispielanwendung keine Rolle spielen.
3. Klicken Sie in den leeren Menüeintrag auf der Leiste und tragen Sie als Titel &Wechseln zu ein.
4. Markieren Sie das neue Menü und ziehen Sie es an die Position links des Menüs Hilfe. In diesem Menü sind alle Navigationsfunktionen zusammengefasst.
5. Fügen Sie zwei Menübefehle hinzu, einen für die Funktion GoForward und einen für die Funktion GoBack. Legen Sie die Eigenschaften für die beiden Menübefehle gemäß Tabelle 16.5 fest.

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_NAVIGATE_BACK
	Beschriftung	&Zurück\tStrg + B
	Eingabeaufforderung	Wechselt zum vorherigen Objekt\nZurück
Menübefehl	ID	IDM_NAVIGATE_NEXT
	Beschriftung	&Vorwärts\tStrg + N
	Eingabeaufforderung	Wechselt zum nächsten Objekt\nVorwärts

Tabelle 16.5: Eigenschaftseinstellungen des Menüs Wechseln zu

Nachdem Sie die Menübefehle hinzugefügt haben, können Sie mit dem Ereignis-Handler-Assistenten die Funktionen in die Ansichtsklasse für beide Menüereignisse einfügen.

6. Für die Menü-ID IDM_NAVIGATE_BACK fügen Sie eine Behandlungsfunktion für die Nachricht COMMAND hinzu und schreiben den Code aus Listing 16.7 in die Funktion.

Listing 16.7: Die Funktion CWebBrowseView.OnNavigateBack

```
1: void CWebBrowseView::OnNavigateBack()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Zur vorherigen Seite gehen
5:    GoBack();
6: }
```

7. Fügen Sie eine Behandlungsfunktion für die Objekt-ID IDM_NAVIGATE_NEXT für die Nachricht COMMAND ein. In diese Funktion nehmen Sie den Code aus Listing 16.8 auf.

Listing 16.8: Die Funktion CWebBrowseView.OnNavigateNext

```
1: void CWebBrowseView::OnNavigateNext()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Zur nächsten Seite gehen
5:    GoForward();
6: }
```

Nun können Sie Ihre Anwendung ausführen und - egal, wohin Sie gerade gesurft sind - über die Menüs zu der

Wenn Sie die Accelerator-Tabelle im Ressourcenbaum öffnen, finden Sie bereits ein ganzes Bündel von Zugriffstasten, die mit Menü-IDs verbunden sind. Jede dieser Zugriffstasten besteht aus einer ID und einer

Tastenkombination. Wenn Sie an einer beliebigen Stelle in der Accelerator-Tabelle mit der rechten Maustaste klicken, finden Sie im Kontextmenü den Befehl, um eine neue Zugriffstaste zur Tabelle hinzuzufügen. Wenn Sie diesen Befehl wählen, wird am unteren Ende der Liste eine neue Zeile angehängt, in die Sie die Angaben für die Zugriffstaste eintragen können. Als Erstes ist in der linken Spalte die Menü-ID anzugeben, mit der die Zugriffstaste verbunden wird. (Wie bei Symbolleisten-Schaltflächen sind Zugriffstasten mit Menübefehlen gekoppelt.) In der dritten Spalte geben Sie die Taste an, die als Zugriffstaste wirksam ist, oder Sie wählen eine Taste aus der Dropdown-Liste aus.

In der zweiten Spalte können Sie die Zusatztaste festlegen. Dabei handelt es sich um Tasten, die in Verbindung mit der bereits festgelegten Zugriffstaste zu drücken sind, um die Funktion auszulösen.

Es empfiehlt sich, die Tasten (Strg) oder (Alt) als eine der Zusatztasten für alle Zugriffstasten zu verwenden, die mit Standardtasten arbeiten. Wenn Sie nicht eine dieser beiden Tasten als Teil der Zugriffstaste spezifizieren, kann die Anwendung nicht unterscheiden, ob der Benutzer Informationen eingeben oder die mit einer Zugriffstaste verbundene Funktion auslösen möchte.

Um Zugriffstasten für die Menüs Zurück und Vorwärts in Ihre Anwendung aufzunehmen:

1. fügen Sie eine neue Zugriffstaste ein und legen die ID mit IDM_NAVIGATE_BACK, die Taste mit B und die Zusatztaste mit Strg fest,

2. fügen Sie eine zweite Zugriffstaste ein und geben Sie IDM_NAVIGATE_NEXT als ID, N als Taste und Strg als Zusatztaste an.

Wenn Sie die Anwendung ausführen, können Sie mit der Tastenkombination (Strg)+(B) zur vorherigen Seite und mit der Tastenkombination (Strg)+(N) zur nächsten Seite gehen. Damit sich die Anwendung wie die meisten verfügbaren Webbrowser verhält, fügen Sie für die beiden Menübefehle noch Symbolleisten-Schaltflächen hinzu, die mit einem Pfeil nach links bzw. rechts versehen sind.

Haben Sie was gemerkt? Die Tastenkombination (Strg)+(N) führt uns zur festeigstellten Startseite und nicht zur nächsten Seite. Warum? Wenn Sie etwas weiter oben in der Accelerator-Tabelle schauen werden Sie erkennen, daß die Tastenkombination (Strg)+(N) bereits für die ID ID_FILE_NEW verwendet worden ist. Da wir diese Funktionalität nicht benötigen löschen Sie diese Zeile und kompilieren erneut.

Den Browser steuern

Beim Browsen gelangt man oft auf eine Webseite, bei der man das Herunterladen gar nicht abwarten will. Die Übertragung lässt sich auch mittendrin anhalten. Es kann sein, dass Sie den falschen URL eingegeben haben oder Ihnen der Download zu lange dauert. Es spielt keine Rolle, warum Sie den Download anhalten wollen. Es genügt, dass Sie es wollen. Aus diesem Grund verfügt die Klasse CHtmlView über die Funktion Stop. Diese Funktion bricht den momentan laufenden Download ab. Um die Funktionalität in die Beispielanwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie dem Menü Ansicht mit dem Menü-Designer einen neuen Menübefehl hinzu. Legen Sie die Eigenschaften des Menübefehls gemäß Tabelle 16.6 fest.

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_NAVIGATE_STOPP
	Beschriftung	A&bbrechen
	Eingabeaufforderung	Bricht die aktuelle Übertragung ab\nAbbrechen

Tabelle 16.6: Eigenschaften des Menübefehls Abbrechen

2. Mit dem Ereignis-Handler-Assistenten nehmen Sie eine Behandlungsroutine in die Klasse CWebBrowseView für diese Menü-ID und für die Nachricht COMMAND auf. In die Funktion schreiben Sie den Code aus Listing 16.9.

Listing 16.9: Die Funktion CWebBrowseView.OnNavigateStop

```
1: void CWebBrowseView::OnNavigateStop()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Aktuellen Download abbrechen
5:    Stop();
6: }
```

Wenn Sie die Anwendung ausführen, können Sie mit diesem Menübefehl jegliches Herunterladen einer Webseite stoppen, auf die Sie nicht warten wollen. Es wäre allerdings bequemer, wenn man das über eine Schaltfläche auf der Symbolleiste vornehmen könnte.

Die meisten Browser verfügen außerdem über die Möglichkeit, die aktuelle Seite neu zu laden. Diese Funktion ist vor allem für Webseiten praktisch, die dynamische Elemente enthalten, die sich bei jedem Herunterladen der Seite ändern. Sie ist auch hilfreich für Webseiten, die Ihr Browser im Cache abgelegt hat, sodass er die neueste Version der Seite vielleicht gar nicht abruft. Man muss den Browser dazu zwingen können, die Seite erneut zu laden und nicht einfach die Cache-Version anzuzeigen (insbesondere bei einer Webseite, die Sie gerade entwerfen). Diese Fähigkeit realisiert die Browser-Funktion Refresh. Ein Aufruf dieser Funktion bedeutet, dass die aktuelle Seite erneut geladen wird.

In die Beispielanwendung können Sie diese Funktionalität mit einem weiteren Menübefehl im Menü Wechseln zu realisieren. Legen Sie die Eigenschaften für den neuen Menübefehl gemäß Tabelle 16.7 fest. Zwischen den ursprünglich vorhandenen und den neuen Menübefehlen können Sie auch eine Trennlinie einfügen, sodass Ihr Menü wie in Abbildung 16.8 aussieht.

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_NAVIGATE_REFRESH
	Beschriftung	&Aktualisieren
	Eingabeaufforderung	Aktualisiert den Inhalt der aktuellen Seite\nAktualisieren

Tabelle 16.7: Eigenschaften des Menübefehls Aktualisieren

Abbildung 16.8: Das modifizierte Menü Wechseln zu

Nachdem Sie den Menübefehl hinzugefügt haben, nehmen Sie mit dem Klassen- Assistenten eine Behandlungsroutine in die Klasse CWebBrowseView für die Nachricht COMMAND und für diesen Menübefehl auf. In die Funktion schreiben Sie den Code aus Listing 16.10.

Listing 16.10: Die Funktion CWebBrowseView.OnNavigateRefresh

```
1: void CWebBrowseView::OnNavigateRefresh()
2: {
3:   // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:   // Aktuelle Seite neu laden
5:   Refresh();
6: }
```

Diese Funktionalität können Sie nun testen, indem Sie eine Website suchen, die ihren Inhalt permanent aktualisiert, wenn Sie den Browser aktualisieren (z.B. Börsenkurse oder Nachrichtenticker). Wie bei den übrigen Menüfunktionen, die Sie in die Beispielanwendung aufgenommen haben, sollten Sie auch für diese eine Schaltfläche auf der Symbolleiste vorsehen.

Der Seitentitel in der Titelleiste

Für das letzte Element, das Sie in Ihre Anwendung aufnehmen, müssen Sie eine Wrapper- Klasse für die IHTMLDocument2-Schnittstelle erzeugen. Sie verwenden die Eigenschaft Title und die Eigenschaftsfunktion get_title, um den Titel der Seite abzufragen und in die Titelleiste Ihrer Anwendung zu setzen. Um diese Funktionalität in Ihre Anwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Klasse in Ihre Anwendung ein.

2. Wählen Sie im Dialogfeld Klasse hinzufügen die Option MFC-Klasse aus der Typenbibliothek als Vorlage für die zu erstellende Klasse.
3. Wählen Sie im Assistenten zum Hinzufügen von Klassen aus der Typenbibliothek die Microsoft HTML Object Library <4.0> aus dem Kombinationsfeld mit den verfügbaren Typenbibliotheken.
4. Scrollen Sie in der Liste verfügbarer Schnittstellen nach unten, um die IHTMLDocument2-Schnittstelle zu finden. Markieren Sie sie und fügen Sie sie zur Liste Generierte Klassen hinzu, indem Sie auf die Schaltfläche mit der spitzen Klammer nach rechts (>) klicken. Klicken Sie auf Fertig stellen, um die Klasse zu erzeugen.
5. Bearbeiten Sie in der Klasse CWebBrowseView die Funktion OnDocumentComplete und nehmen Sie den fett gedruckten Code aus Listing 16.11 auf.

Listing 16.11: Die modifizierte Funktion CWebBrowseView.OnDocumentComplete

```
 1: void CWebBrowseView::OnDocumentComplete(LPCTSTR lpszURL)
 2: {
 3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
 4:   // und/oder rufen Sie die Basisklasse auf.
 5:   // Neue URL an die Adressleiste übergeben
 6:   ((CMainFrame*)GetParentFrame())->SetAddress(lpszURL);
 7:
 8:   // Eine Instanz der IHTMLDocument2-Schnittstelle holen
 9:   CHTMLDocument2* pHDoc = new CHTMLDocument2(GetHtmlDocument());
10:   // Seitentitel ermitteln
11:   CString strTitle = (CString)pHDoc->get_title();
12:   // Zeiger auf die Dokumentklasse holen
13:   CWebBrowseDoc* pDoc = GetDocument();
14:   // Seitentitel setzen
15:   pDoc->SetTitle(strTitle);
16:   // Aufräumen
17:   delete pHDoc;
18:
19:   CHtmlView::OnDocumentComplete(lpszURL);
20: }
```

6. Bearbeiten Sie die #include-Direktiven in der Klasse CWebBrowswView und fügen Sie gemäß Listing 16.12 eine Include-Anweisung für den Header der Klasse CHtmlDocument2 ein.

Listing 16.12: Die modifizierten #include-Direktiven von CWebBrowseView

```
1: // WebBrowseView.cpp : Implementierung der Klasse CWebBrowseView
2: //
```

```
 3:
 4: #include "stdafx.h"
 5: #include "WebBrowse.h"
 6:
 7: #include "WebBrowseDoc.h"
 8: #include "WebBrowseView.h"
 9: #include "MainFrm.h"
10: #include "CHtmlDocument2.h"
11:
12: #ifdef _DEBUG
13: #define new DEBUG_NEW
14: #endif
```

Es ist möglich, daß Sie beim Kompilieren einen Fehler in der Datei CHTMLDocument2.h erhalten:

```
error C2440:  'Initialisierung' : 'int' kann nicht in 'BYTE []'
                                  konvertiert werden
```

Dieser Fehler liegt möglicherweise an Visual C++ selber, welches diese Datei über den Klassenassistenten erstellt hat. Die beiden betroffenen Codezeile mit dem Fehler sehen so aus:

```
static BYTE parms[] = VTS_NONE ;
```

Ändern Sie diese beiden Zeilen wie folgt ab und Ihre Anwedung wird ohne Fehler kompilieren:

```
static BYTE parms[] = VTS_VARIANT ;
```

Nun sollten Sie Ihre Anwendung kompilieren und ausführen können und sehen, dass der Titel der Webseite in die Titelleiste Ihrer Anwendung eingefügt wird, wenn Sie von Seite zu Seite navigieren.

16.5 Zusammenfassung

Heute haben Sie gelernt, wie Microsoft den Webbrowser Internet Explorer als eine Reihe von ActiveX-Komponenten konzipiert hat, die sich auch in anderen Anwendungen nutzen lassen. Es wurde gezeigt, wie Microsoft den Browser in der Klasse CHtmlView abgekapselt hat. Diese Klasse kann man in SDI- und MDI-Anwendungen nutzen, um die Funktionalität eines Browsers in nahezu jeder Anwendung umsetzen zu können. Sie haben gesehen, wie man die Dialogleiste verwendet, um Steuerelemente auf einer Symbolleiste unterzubringen, und wie sich die Ereignisse für diese Steuerelemente in der Rahmenklasse der Anwendung behandeln lassen. Schließlich haben Sie gelernt, wie man Menüs in die Anwendung aufnimmt, um die verschiedenen Funktionen des Webbrowsers aufzurufen, damit das Surferlebnis komplett wird. Zuletzt haben Sie gelernt, wie Sie mit den in Visual C++ integrierten Assistenten Wrapper-Klassen für ActiveX-Steuerelemente und COM-Objekte erstellen können.

16.6 Workshop

Fragen und Antworten

Frage:
Warum gehört die Seitenansicht nicht zu den Standardmenüs, wenn ich als Basisklasse CHtmlView für meine Ansichtsklasse wähle?

Antwort:
Die Druckfunktionen für die Klasse CHtmlView übernimmt der Browser und nicht die Ansichtsklasse. Die

Seitenansicht ist nicht vorhanden, da sie der Browser nicht unterstützt.

Frage:
Wie kann ich den HTML-Quellcode vom Browser beziehen, damit ich ihn ansehen oder bearbeiten kann?

Antwort:
Die Klasse CHtmlView verfügt über die Member-Funktion GetSource, die eine CString-Variable als einziges Argument übernimmt. In der CString-Variable wird der HTML-Quelltext abgespeichert.

Quiz

1. Was verkapselt die Klasse CHtmlView für den Einsatz in Visual C++-Anwendungen?

2. Wie kann man den URL für die aktuelle Webseite aus der Klasse CHtmlView erhalten?
3. Welcher Befehl wird für die Rahmenklasse ausgelöst, wenn der Benutzer die (Enter)- Taste im Eingabefeld der Dialogleiste drückt?
4. Welche Funktionen können Sie aufrufen, um den Browser zur vorherigen und zur nächsten Webseite zu navigieren?
5. Wie kann man einen laufenden Download anhalten?

Übungen

1. Fügen Sie die Funktion GoSearch in das Menü und die Symbolleiste ein.

2. Nehmen Sie die Funktion GoHome in das Menü und die Symbolleiste auf.
3. Deaktivieren Sie die Schaltfläche Abbrechen und den zugehörigen Menübefehl, wenn die Anwendung keine Webseite herunterlädt.

Tag 17

Multitasking

Manchmal ist es angebracht, Anwendungen mehrere Dinge gleichzeitig ausführen zu lassen. Die Anwendung könnte eine Sicherungsdatei schreiben oder im Hintergrund drucken, während der Benutzer am selben Dokument weiterarbeitet. Vielleicht führt Ihre Anwendung Berechnungen aus, während der Benutzer neue Daten eingibt, oder zeichnet mehrere Bilder gleichzeitig. Es gibt viele verschiedene Gründe, warum man diese Fähigkeit, das so genannte Multitasking (Task - Aufgabe), in Anwendungen realisieren möchte. Windows stellt verschiedene Einrichtungen bereit, um derartige Fähigkeiten in eine Anwendung einbauen zu können.

Heute lernen Sie, wie ...

- sich Tasks ausführen lassen, während eine Anwendung im Leerlauf arbeitet,
- man Tasks unabhängig von der übrigen Anwendung starten kann,
- man den Zugriff auf die Ressourcen koordiniert, die von mehreren unabhängigen Tasks gemeinsam genutzt werden,
- man unabhängig laufende Tasks startet und stoppt.

17.1 Was ist Multitasking?

Zum Verständnis des Multitaskings muss man den generellen Unterschied zwischen Task (auch als Prozess bezeichnet) und Thread (engl. Faden) verstehen. Das Betriebssystem verwaltet die verschiedenen Prozesse, die auf dem System vorhanden sind, und teilt jedem Prozess beim Start Speicherbereiche für den Programmcode und die Daten zu. Da gewöhnlich auf den Programmcode nur lesend zugegriffen wird, kann es sein, dass ein Programm, welches mehrfach aufgerufen wurde, nur einmal den Programmcode geladen bekommt.

Das Betriebssystem sorgt für die Zuteilung des Hauptspeichers auf die Prozesse und gibt diesen nach Ende eines Prozesses wieder frei. Gleichzeitig verhindert es den Zugriff eines Prozesses auf einen Speicherbereich, der einem anderen Prozess zugeordnet ist. Jeder Prozessor eines Systems bearbeitet zu einer bestimmten Zeit auch nur einen Prozess, durch geeignete Verfahren sorgt das Betriebssystem für den Wechsel zwischen den verschiedenen Prozessen.

Die in Windows 3.x realisierte Form des Multitaskings bezeichnet man als kooperatives Multitasking. Der Schlüssel zu dieser Art des Multitaskings liegt darin, dass jede einzelne Anwendung darüber entscheidet, wann sie den Prozessor für eine andere Anwendung freigibt, damit diese die anstehenden Arbeiten erledigen kann. Im Ergebnis war Windows 3.x anfällig für eine sich schlecht verhaltende Anwendung, die andere Anwendungen gefangen hält, während sie einen länger dauernden Prozess ausführt oder sich sogar selbst in irgendeiner Schleife verfängt.

Seit Windows NT/95 hat sich die Natur des Betriebssystems geändert. Das neue präemptive Multitasking (preemptive) hat das kooperative Multitasking abgelöst. Beim präemptiven Multitasking entscheidet das Betriebssystem, wann es der einen Anwendung den verwendeten Prozessor entzieht und einer anderen Anwendung, die bereits darauf wartet, zuteilt. Es spielt keine Rolle, ob die Anwendung, die momentan den Prozessor für sich in Anspruch nimmt, bereit

ist, den Prozessor abzugeben. Das Betriebssystem entzieht der Anwendung den Prozessor, ohne die Anwendung dazu um Erlaubnis zu fragen. Auf diese Weise trägt das Betriebssystem dazu bei, dass mehrere Anwendungen rechenintensive Aufgaben erledigen können und trotzdem alle Anwendungen in ihren eigenen Verarbeitungen vorwärts kommen. Indem man diese Fähigkeit dem Betriebssystem überträgt, verhindert man, dass eine einzige Anwendung alle anderen Anwendungen blockiert, während sie den Prozessor beansprucht.

Der Wechsel zwischen den verschiedenen laufenden Prozessen ist eine relativ umständliche Tätigkeit. Aus diesem Grund hat man Threads eingeführt als Unterteilungsmöglichkeit einzelner Prozesse. Alle Threads eines einzigen Prozesses teilen sich gemeinsam den gesamten Datenbereich ihres Prozesses, deshalb werden Threads auch manchmal als leichtgewichtige (ligthweight) Prozesse bezeichnet. Es ist Aufgabe des Programmierers, dafür Sorge zu tragen, dass die verschiedenen Threads einer Multithreading-Anwendung sich gegenseitig über die Verwendung des Speichers einigen. In den Tagen von Windows 3.x waren alle Windows-Anwendungen Singlethreading-Anwendungen, die zu einem beliebigen Zeitpunkt nur einen Ausführungspfad aufwiesen. Der Vorteil einer Multithread-Anwendung ist, dass die Threads mit den gemeinsam genutzten Daten arbeiten können. Während ein Thread beispielsweise noch Daten lädt, kann ein anderer Thread bereits beginnen, die Daten anzuzeigen.

Bei der 16/32-Bit-Struktur von Windows 95/98/ME ist es trotzdem noch möglich, dass eine sich schlecht verhaltende 16-Bit-Anwendung das System blockiert, da ein großer Teil von 16-Bit-Code weiterhin zum Kern des Betriebssystems gehört. Der 16-Bit-Code von Windows 95/98/ME stellt immer noch eine kooperative Multitasking-Umgebung dar, sodass nur jeweils eine Anwendung 16-Bit-Code zu einem bestimmten Zeitpunkt ausführen kann. Da alle USER-Funktionen und ein guter Teil der GDI-Funktionen auf die 16-Bit-Versionen zurückgreifen, kann eine einzige 16-Bit-Anwendung das gesamte System blockieren.

Da in Windows NT alle 16-Bit-Anwendungen in einem gemeinsam genutzten Speicherraum laufen, kann eine sich schlecht verhaltende Anwendung zwar alle 16-Bit-Anwendungen blockieren, was aber ohne Auswirkung auf irgendwelche 32-Bit-Anwendungen bleibt.

Mehrere Aufgaben gleichzeitig ausführen

Zusätzlich zur Fähigkeit, mehrere Anwendungen simultan ausführen zu können, kommt die Fähigkeit, dass eine einzelne Anwendung mehrere Threads zu ein und demselben Zeitpunkt ausführt. Ein Thread (Programmfaden) ist für eine Anwendung das, was eine Anwendung für das Betriebssystem darstellt. Wenn eine Anwendung mehrere Threads laufen hat, führt sie im Prinzip mehrere Anwendungen innerhalb der Anwendung als Ganzes aus und kann damit mehrere Dinge gleichzeitig erledigen. Beispielsweise prüft Microsoft Word die Rechtschreibung zum gleichen Zeitpunkt, zu dem man Eingaben in das Dokument vornimmt.

Threads zur Leerlaufverarbeitung

Einer der einfachsten Wege, um die Anwendung mehrere Verarbeitungen gleichzeitig ausführen zu lassen, ist die Realisierung der Leerlaufverarbeitung. Eine Leerlaufverarbeitung kommt zum Zuge, wenn sich eine Anwendung im Leerlaufzustand befindet. Im wahrsten Sinne des Wortes wird eine Funktion in der Anwendungsklasse aufgerufen, wenn keine Nachrichten in der Nachrichtenwarteschlange der Anwendung stehen. Der Gedanke hinter dieser Funktion ist der, dass eine Anwendung im Leerlaufzustand Aufgaben wie Speicherbereinigung (die so genannte Garbage Collection) oder Schreiben in einen Drucker-Spooler ausführen kann.

Die Funktion OnIdle ist ein Überbleibsel aus den Tagen von Windows 3.x. Sie gehört zur Klasse CWinApp, von der Ihre Anwendung abgeleitet ist. Per Vorgabe fügt der MFC- Anwendungs-Assistent keinerlei Verarbeitungen in diese Funktion ein. Wenn Sie demzufolge die Funktion in Ihrer Anwendung nutzen möchten, müssen Sie sie in Ihre Anwendungsklasse mit den Überschreibungen für Klasseneigenschaften aufnehmen. (OnIdle ist eine der verfügbaren Nachrichten für die Anwendungsklasse in Ihren Anwendungen.)

Die Funktion OnIdle übernimmt ein Argument, das die Anzahl der Aufrufe dieser Funktion angibt, die seit der letzten von der Anwendung verarbeiteten Nachricht stattgefunden haben. Anhand dieses Wertes können Sie ermitteln, wie lange die Anwendung im Leerlauf gearbeitet hat und wann eine Aktion auszulösen ist, falls die Anwendung eine bestimmte Zeitbegrenzung im Leerlauf überschritten hat.

Einer der wichtigsten Punkte bei der OnIdle-Verarbeitung in Ihren Anwendungen ist der, dass der Umfang der Funktionalität klein sein und die Steuerung schnell an den Benutzer zurückgehen muss. Wenn eine Anwendung eine OnIdle-Verarbeitung durchführt, kann der Benutzer nicht mit der Anwendung weiterarbeiten, bis die OnIdle-Verarbeitung abgeschlossen ist und der Benutzer die Steuerung wiedererlangt. Wenn Sie eine längere, ausgedehnte Aufgabe in der Funktion OnIdle ausführen müssen, sollten Sie sie in viele kleine und schnelle Fragmente aufteilen, damit der Benutzer die Steuerung zwischenzeitlich zurückerhält. Sobald die Nachrichtenwarteschlange wieder leer ist, können Sie Ihre OnIdle-Verarbeitung fortsetzen. Das bedeutet, dass Sie auch den Fortschritt der Anwendung in der OnIdle-Verarbeitung verfolgen müssen, damit die Anwendung beim nächsten Aufruf der Funktion OnIdle die Verarbeitung an der Stelle wieder aufnehmen kann, wo sie verlassen wurde.

Unabhängige Threads aufspannen

Wenn Sie tatsächlich eine längere Hintergrundverarbeitung ausführen müssen, die der Benutzer nicht unterbrechen soll, müssen Sie einen unabhängigen Thread erzeugen. Ein Thread verhält sich wie eine andere Anwendung, die innerhalb Ihrer Anwendung läuft. Der Thread muss nicht darauf warten, dass die Anwendung in den Leerlauf gelangt, um seine Aufgaben auszuführen, und er bewirkt auch nicht, dass der Benutzer warten muss, bis der Thread die Steuerung zwischenzeitlich abgibt.

Die beiden Methoden zum Erzeugen eines unabhängigen Threads verwenden die gleiche Funktion, um den Thread zu erzeugen und zu starten. Um einen unabhängigen Thread zu erzeugen und zu starten, ruft man die Funktion AfxBeginThread auf. Dabei kann man wählen, ob man ihr eine aufzurufende Funktion für die Ausführung der Thread- Verarbeitung oder einen Zeiger auf die Laufzeitklasse eines Objekts, das von der Klasse CWinThread abgeleitet ist, übergibt. Beide Versionen der Funktion liefern einen Zeiger auf ein CWinThread-Objekt zurück, das als unabhängiger Thread läuft.

In der ersten Version der Funktion AfxBeginThread ist das erste Argument ein Zeiger auf die Hauptfunktion, um den Thread zu starten. Diese Funktion ist das Äquivalent der Funktion main in einem C/C++-Programm. Sie steuert die Ausführung auf höchster Ebene für den Thread. Diese Funktion ist als UINT-Funktion mit einem einzelnen LPVOID- Argument zu definieren:

```
UINT MyThreadFunction(LPVOID pParam);
```

Diese Version von AfxBeginThread erfordert auch ein zweites Argument, das als einziges Argument an die Hauptfunktion des Threads weitergereicht wird. Dieses Argument kann ein Zeiger auf eine Struktur sein, die alle Informationen enthält, die der Thread für die korrekte Ausführung seiner Aufgabe kennen muss.

Das erste Argument an die zweite Version der Funktion AfxBeginThread ist ein Zeiger auf die Laufzeitklasse eines Objekts, das von der Klasse CWinThread abgeleitet ist. Einen Zeiger auf die Laufzeitklasse Ihrer CWinThread-Klasse können Sie mit dem Makro RUNTIME_CLASS ermitteln, wobei Sie Ihre Klasse als einziges Argument übergeben.

Nach diesen ersten Argumenten sind bei beiden Versionen die restlichen Argumente an die Funktion AfxBeginThread gleich und insgesamt optional. Das erste dieser Argumente bezeichnet die gewünschte Priorität des Threads, wobei der Wert THREAD_PRIORITY_NORMAL vorgegeben ist. Tabelle 17.1 listet die verfügbaren Thread-Prioritäten auf.

Priorität	**Beschreibung**
0	Der Thread erbt die Thread-Priorität der Anwendung, die den Thread erzeugt.
THREAD_PRIORITY_NORMAL	Eine normale (Standard-) Priorität
THREAD_PRIORITY_ABOVE_NORMAL	1 Punkt über der normalen Priorität
THREAD_PRIORITY_BELOW_NORMAL	1 Punkt unterhalb der normalen Priorität
THREAD_PRIORITY_HIGHEST	2 Punkte über der normalen Priorität

THREAD_PRIORITY_LOWEST	2 Punkte unterhalb der normalen Priorität
THREAD_PRIORITY_IDLE	Prioritätsebene 1 für die meisten Threads (alle Threads ohne Echtzeitverarbeitung)
THREAD_PRIORITY_TIME_CRITICAL	Prioritätsebene 15 für die meisten Threads (alle Threads ohne Echtzeitverarbeitung)

Tabelle 17.1: Prioritäten von Threads

Die Thread-Priorität steuert, wie viel CPU-Zeit der Thread in Relation zu den anderen Threads und Prozessen, die auf dem Computer laufen, erhält. Wenn ein Thread keinerlei Verarbeitungen ausführt, die schnell fertig zu stellen sind, dann sollten Sie dem Thread beim Erstellen eine niedrigere Priorität geben. Es empfiehlt sich nicht, einem Thread eine Priorität höher als normal zu geben, solange es nicht unbedingt notwendig ist, dass der Thread seine Verarbeitung schneller als andere Prozesse auf dem Computer ausführen muss. Je höher die Priorität eines Threads ist, desto mehr CPU-Zeit erhält er und desto weniger CPU-Zeit steht für alle anderen Prozesse und Threads auf dem Computer zur Verfügung.

Das nächste Argument an die Funktion AfxBeginThread ist die Stack-Größe, die für den neuen Thread bereitzustellen ist. Jeder Thread hat einen eigenen Stack, auf dem Parameter an Funktionen übergeben und lokale Variablen erzeugt werden. Der Standardwert für dieses Argument ist 0. Damit erhält der Thread die gleiche Stack-Größe wie die Hauptanwendung.

Als nächstes Argument an die Funktion AfxBeginThread übergibt man das Flag zum Erzeugen des Threads. Dieses Flag kann einen von zwei Werten enthalten. Es steuert den Start des Threads. Wenn man für dieses Argument den Wert CREATE_SUSPENDED übergibt, wird der Thread in einem angehaltenen Zustand erzeugt. Der Thread startet erst dann, wenn die Funktion ResumeThread für den Thread aufgerufen wird. Übergibt man für dieses Argument den Wert 0, d.h. den Standardwert, beginnt der Thread die Ausführung in dem Moment, zu dem er erzeugt wird.

Das letzte Argument an die Funktion AfxBeginThread ist ein Zeiger auf die Sicherheitsattribute für den Thread. Der Standardwert für dieses Argument ist NULL. Damit wird der Thread mit dem gleichen Sicherheitsprofil wie die Anwendung erzeugt. Solange Sie keine Anwendungen für Windows NT/2000/XP erstellen und einen Thread mit einem bestimmten Sicherheitsprofil bereitstellen müssen, sollten Sie für dieses Argument immer den Standardwert einsetzen.

Strukturen erstellen

Stellen Sie sich eine Anwendung vor, die zwei Threads ausführt, wobei jeder seinen eigenen Satz von Variablen zur gleichen Zeit abarbeitet. Stellen wir uns weiterhin vor, dass die Anwendung ein globales Objektfeld verwendet, um diese Variablen aufzunehmen. Wenn die Methode zur Reservierung und Größenänderung des Arrays die aktuelle Größe prüft und eine Position am Ende des Feldes anfügt, bauen Ihre beiden Threads vielleicht ein Feld auf, das etwa gemäß Abbildung 17.1 gefüllt ist. Hier sind die durch den ersten Thread gefüllten Positionen mit denjenigen vermischt, die der zweite Thread erzeugt hat. Die beiden Threads können dabei leicht durcheinander kommen, wenn sie Werte aus dem Feld abrufen und weiterverarbeiten wollen, da jeder Thread mit hoher Wahrscheinlichkeit einen Wert abruft, der eigentlich zum anderen Thread gehört. Das führt dazu, dass die Threads auf den falschen Daten arbeiten und falsche Ergebnisse zurückliefern.

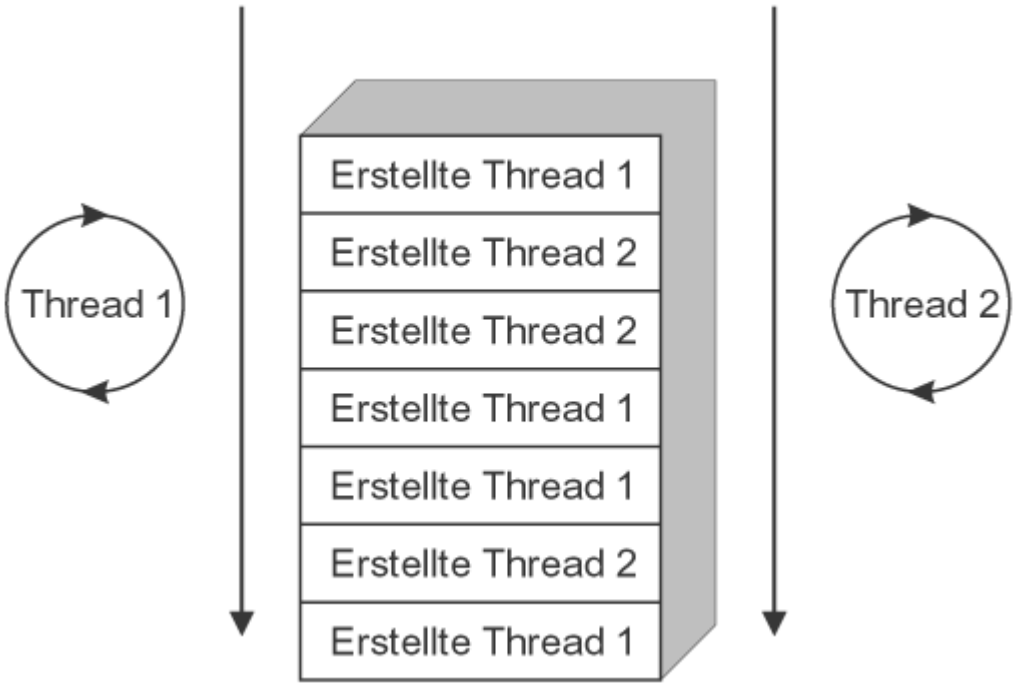


Abbildung 17.1: Zwei Threads, die ein gemeinsames Datenfeld füllen

Wenn die Anwendung die Felder als lokal und nicht als global erstellt, kann sie den Zugriff auf die Felder nur auf den Thread einschränken, der das Feld erstellt. Im Beispiel von Abbildung 17.2 tritt keine Vermischung von Daten aus mehreren Threads auf. Wenn Sie mit dieser Lösung bei Feldern und anderen Speicherstrukturen arbeiten, kann jeder Thread seine Aufgaben erledigen und die Ergebnisse an den Client zurückgeben. Dabei ist sichergestellt, dass es sich um die richtigen Ergebnisse handelt, da die Berechnungen auf eigenständigen Daten ablaufen.

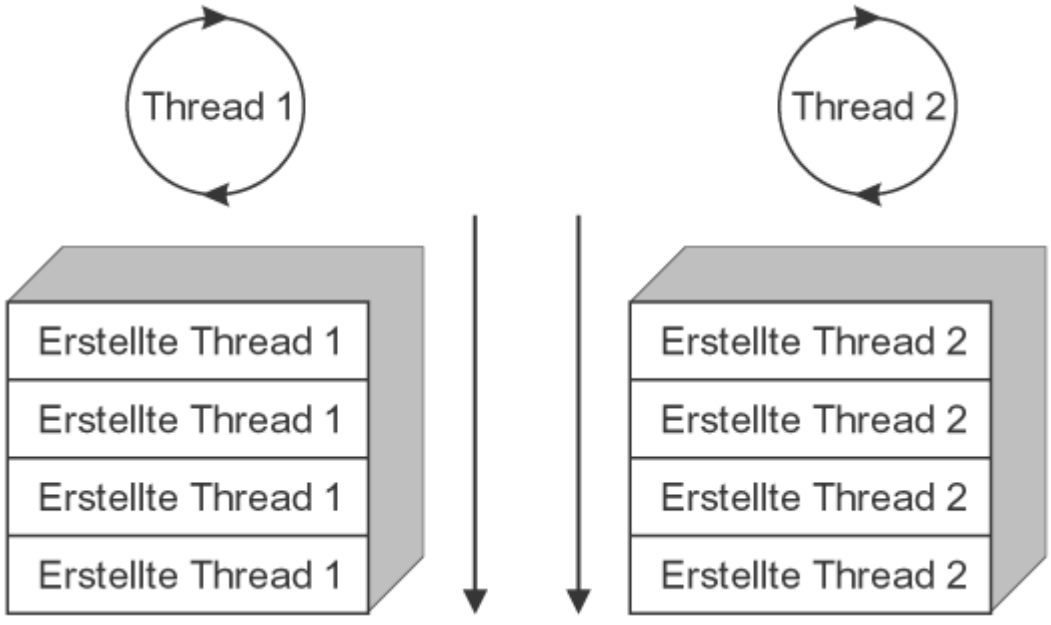


Abbildung 17.2: Zwei Threads, die lokal angelegte Datenfelder füllen

Zugriff auf gemeinsam genutzte Ressourcen verwalten

Nicht alle Variablen lassen sich in einer lokalen Form bereitstellen und es ist oft erforderlich, bestimmte Ressourcen zwischen allen Threads, die in Ihren Anwendungen laufen, gemeinsam zu nutzen. Diese gemeinsame Nutzung führt zu einem Problem bei Anwendungen mit mehreren Threads. Nehmen wir an, dass drei Threads mit einem einzigen Zähler arbeiten, der eindeutige Zahlen erzeugt. Da man nicht weiß, wann die Steuerung des Prozessors von einem Thread zum nächsten übergeht, kann die Anwendung

doppelte »eindeutige« Werte erzeugen, wie es Abbildung 17.3 zeigt.

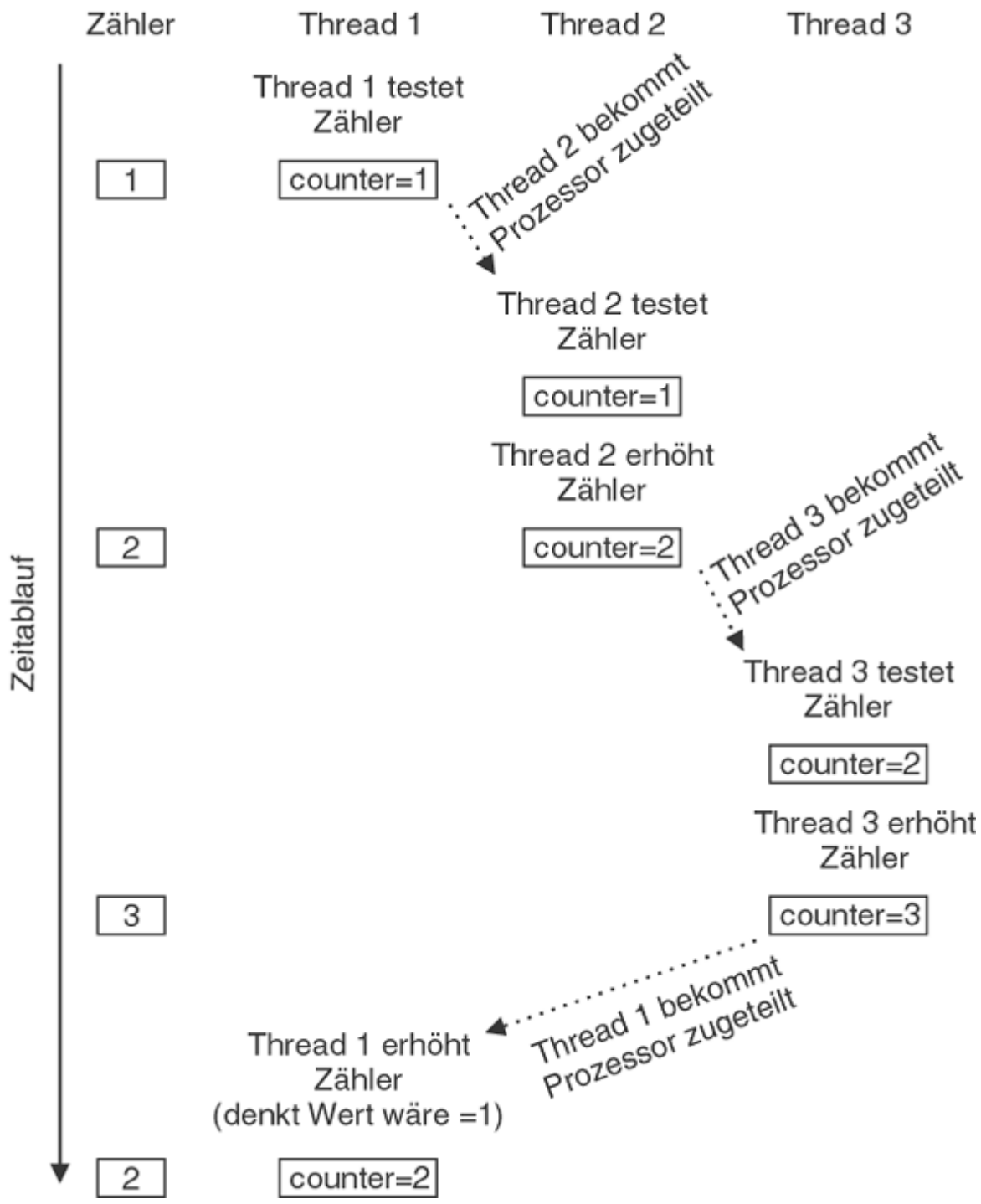


Abbildung 17.3: Drei Threads, die ein und denselben Zähler gemeinsam nutzen

Wie man sieht, funktioniert diese gemeinsame Nutzung in einer Anwendung mit mehreren Threads nicht besonders gut. Man braucht eine Möglichkeit, um den Zugriff auf eine gemeinsam genutzte Ressource zu einem bestimmten Zeitpunkt auf nur einen Thread einzuschränken. In der Tat existieren vier Mechanismen, mit denen man den Zugriff auf gemeinsame Ressourcen begrenzen und die Verarbeitung zwischen Threads synchronisieren kann. Diese Mechanismen arbeiten nach verschiedenen Methoden. Welche davon geeignet ist, hängt von den konkreten Umständen ab:

- Kritische Abschnitte
- Mutexe
- Semaphore
- Ereignisse

Kritische Abschnitte

> *Ein kritischer Abschnitt ist ein Mechanismus, der den Zugriff auf eine bestimmte Ressource bei einem einzelnen Thread innerhalb einer Anwendung beschränkt. Ein Thread tritt in den kritischen Abschnitt ein, bevor er mit der angegebenen gemeinsam genutzten Ressource arbeiten muss, und lässtverlässt den kritischen Abschnitt, nachdem der Zugriff auf die Ressource abgeschlossen ist. Wenn ein anderer Thread versucht, in den kritischen Abschnitt einzutreten, bevor der erste Thread den kritischen Abschnitt verlassen hat, wird der zweite Thread blockiert und erhält keine Prozessorzeit, bis der erste Thread den kritischen Abschnitt verlässt und damit dem zweiten Thread den Eintritt ermöglicht. Mit kritischen Abschnitten markiert man Code-Bereiche, die nur ein Thread zu einem bestimmten Zeitpunkt ausführen sollte. Damit verhindert man nicht, dass der Prozessor von diesem Thread zu einem anderen umschaltet. Es wird nur unterbunden, dass zwei oder mehr Threads in denselben Code-Abschnitt eintreten.*

Wenn Sie einen kritischen Abschnitt mit dem Zähler gemäß Abbildung 17.3 verwenden, können Sie erzwingen, dass jeder Thread in einen kritischen Abschnitt eintritt, bevor er den aktuellen Wert des Zählers prüft. Wenn der Thread den kritischen Abschnitt erst dann verlässt, wenn er den Zähler inkrementiert und aktualisiert hat, ist garantiert, dass - unabhängig davon, wie viele Threads ausgeführt werden und in welcher Reihenfolge die Ausführung erfolgt - wirklich eindeutige Zahlen generiert werden, wie es aus Abbildung 17.4 hervorgeht.

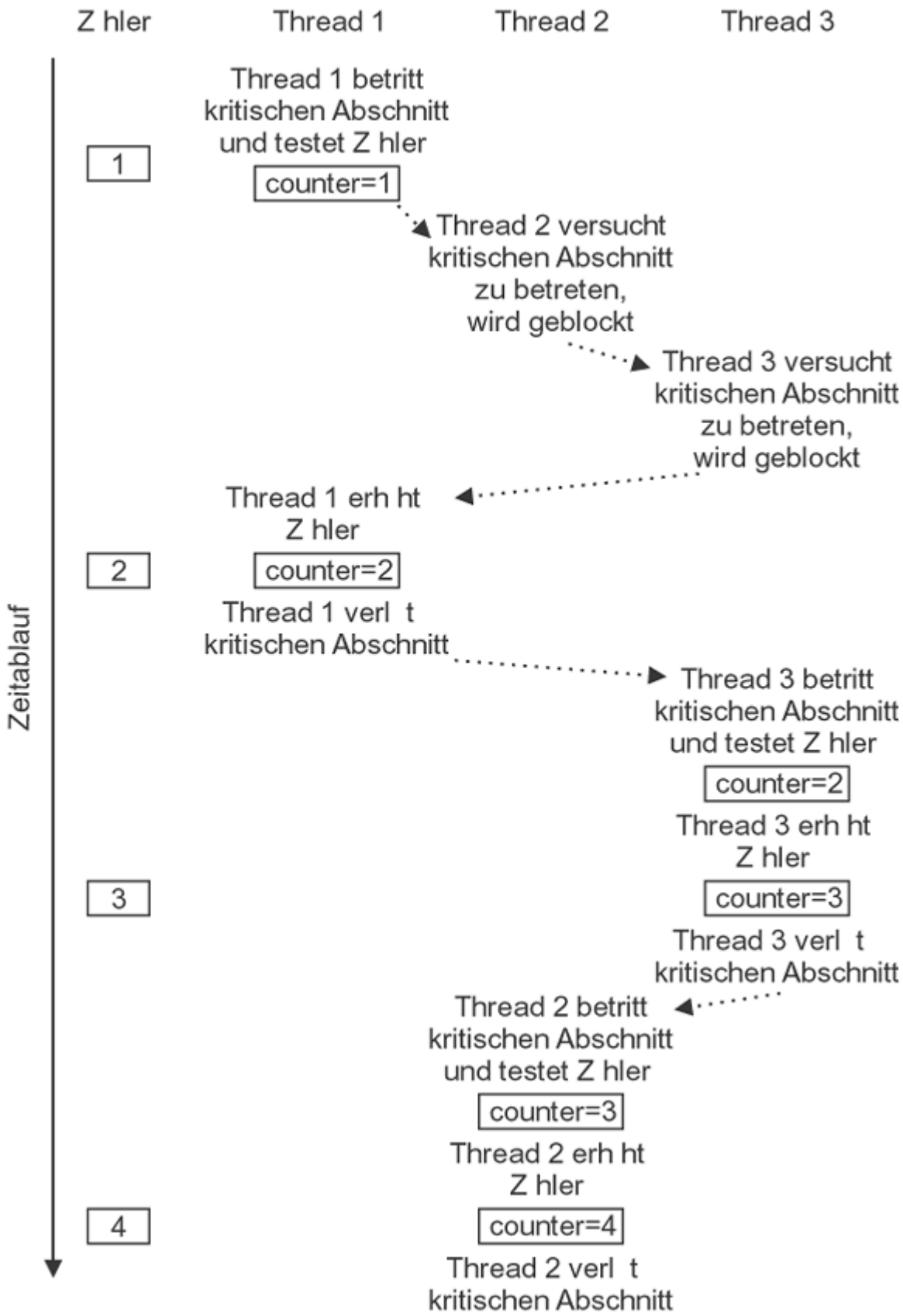


Abbildung 17.4: Drei Threads, die denselben Zähler verwenden, wobei der Zähler durch einen kritischen Abschnitt geschützt ist

Wenn Sie ein Objekt mit einem kritischen Abschnitt in Ihrer Anwendung benötigen, erzeugen Sie eine Instanz der Klasse CCriticalSection. Dieses Objekt enthält die beiden Methoden Lock und Unlock, mit denen Sie die Steuerung des kritischen Abschnitts an sich nehmen bzw. freigeben.

Eine weitere Option, die sich zu untersuchen lohnt, ist die Verwendung der Klasse CSingleLock, mit der kritische Abschnitte gesperrt und entsperrt werden. CSingleLock kann mit allen heute besprochenen Synchronisationsobjekten verwendet werden, um den Zugriff auf Objekte zu steuern. Eine weitere Synchronisationsklasse, die Sie verwenden können, ist CMultiLock, die genau wie CSingleLock arbeitet, allerdings nicht mit kritischen Abschnitten verwendet werden kann (mit allen anderen Synchronisationsobjekten dagegen schon).

Mutexe

Mutexe (als Abkürzung von Mutual Exclusive - gegenseitig ausschließend) arbeiten grundsätzlich in der gleichen Weise wie kritische Abschnitte. Allerdings setzt man Mutexe ein, wenn man Ressourcen zwischen mehreren Anwendungen (mehreren Prozessen) gemeinsam nutzen will. Mit einem Mutex lässt sich garantieren, dass keine zwei Threads, die in einer beliebigen Anzahl von Anwendungen laufen, auf dieselbe Ressource zum selben Zeitpunkt zugreifen.

Auf Grund ihrer systemweiten Verfügbarkeit bringen Mutexe wesentlich mehr Overhead mit als kritische Abschnitte. Die Lebensdauer eines Mutexes endet nicht, wenn die Anwendung, die ihn erzeugt hat, beendet wird. Der Mutex kann weiterhin von anderen Anwendungen verwendet werden. Das Betriebssystem muss demnach verfolgen, welche Anwendungen einen Mutex verwenden, und es muss den Mutex zerstören, sobald er nicht mehr benötigt wird. Im Gegensatz dazu bringen kritische Abschnitte nur einen geringen Verwaltungsaufwand mit sich, da sie nicht außerhalb der Anwendung existieren, die sie erzeugt und verwendet. Nachdem die Anwendung beendet wird, ist der kritische Abschnitt verschwunden.

Wenn Sie einen Mutex in Ihren Anwendungen verwenden müssen, erzeugen Sie eine Instanz der Klasse CMutex. Der Konstruktor der Klasse CMutex weist drei verfügbare Argumente auf:

- Ein boolescher Wert, der festlegt, ob der Thread, der das CMutex-Objekt erzeugt, der anfängliche Besitzer des Mutex ist. In diesem Fall muss dieser Thread den Mutex freigeben, bevor irgendwelche anderen Threads darauf zugreifen können.
- Der Name für den Mutex. Alle Anwendungen, die gemeinsam auf den Mutex zurückgreifen müssen, können ihn nach diesem textuellen Namen identifizieren.
- Ein Zeiger auf die Sicherheitsattribute für das Mutex-Objekt. Übergibt man für diesen Zeiger den Wert NULL, verwendet das Mutex-Objekt die Sicherheitsattribute des Threads, der den Mutex erzeugt hat.

Nachdem Sie ein CMutex-Objekt erzeugt haben, können Sie es mit den Elementfunktionen Lock und Unlock sperren bzw. freigeben. Damit haben Sie die Möglichkeit, den Zugriff auf gemeinsam genutzte Ressourcen zwischen mehreren Threads in mehreren Anwendungen zu steuern.

Semaphoren

Die Arbeitsweise von Semaphoren (ursprünglich mal ein Gerät zur Nachrichtenübermittlung mit Flaggensignalen) unterscheidet sich grundsätzlich von kritischen Abschnitten und Mutexen. Semaphoren setzt man bei Ressourcen ein, die nicht auf einen einzelnen Thread zu einem Zeitpunkt beschränkt sind - eine Ressource, die auf eine bestimmte Anzahl von Threads beschränkt ist. Vom Prinzip her ist ein Semaphor eine Art Zähler, den die Threads inkrementieren oder dekrementieren können. Der Trick bei Semaphoren besteht darin, dass sie nicht kleiner als Null werden können. Wenn demzufolge ein Thread versucht, einen Semaphor zu dekrementieren, der bereits Null ist, wird dieser Thread blockiert, bis ein anderer Thread den Semaphor inkrementiert.

Nehmen wir an, dass eine Warteschlange von mehreren Threads gefüllt wird und ein Thread die Elemente aus der Warteschlange entfernt und weiterverarbeitet. Wenn die Warteschlange leer ist, hat der Thread, der Elemente entfernt und weiterverarbeitet, nichts zu tun. Dieser Thread kann in eine Leerlaufschleife eintreten, in der die Warteschlange ständig daraufhin untersucht wird, ob sich irgend etwas darin befindet. Das Problem bei diesem Szenario besteht darin, dass der Thread Verarbeitungszeit verbraucht (busy wait), um eigentlich überhaupt nichts zu tun. Diese Prozessorzeiten könnte man für andere Threads vergeben, die wirklich etwas ausführen müssen. Wenn Sie eine Warteschlange mit Semaphoren steuern, kann jeder Thread, der Elemente in die Warteschlange stellt, den Semaphor für jedes platzierte Element inkrementieren, und der Thread, der die Elemente aus der Warteschlange entfernt, kann den Semaphor dekrementieren, bevor er ein Element aus der Warteschlange nimmt. Wenn die Warteschlange leer ist, hat der Semaphor den Wert Null, und der Thread, der Elemente entfernt, wird beim Aufruf zum Dekrementieren der Warteschlange blockiert. Der Thread verbraucht damit keine Prozessorzeit, bis einer der anderen Threads den Semaphor inkrementiert, um anzuzeigen, dass er ein Element in die Warteschlange gestellt hat. Zu diesem Zeitpunkt wird die Blockierung für den Thread, der Elemente entfernt, unverzüglich aufgehoben, und der Thread kann das eben platzierte Element aus der Warteschlange nehmen und weiterverarbeiten, wie es Abbildung 17.5 zeigt.

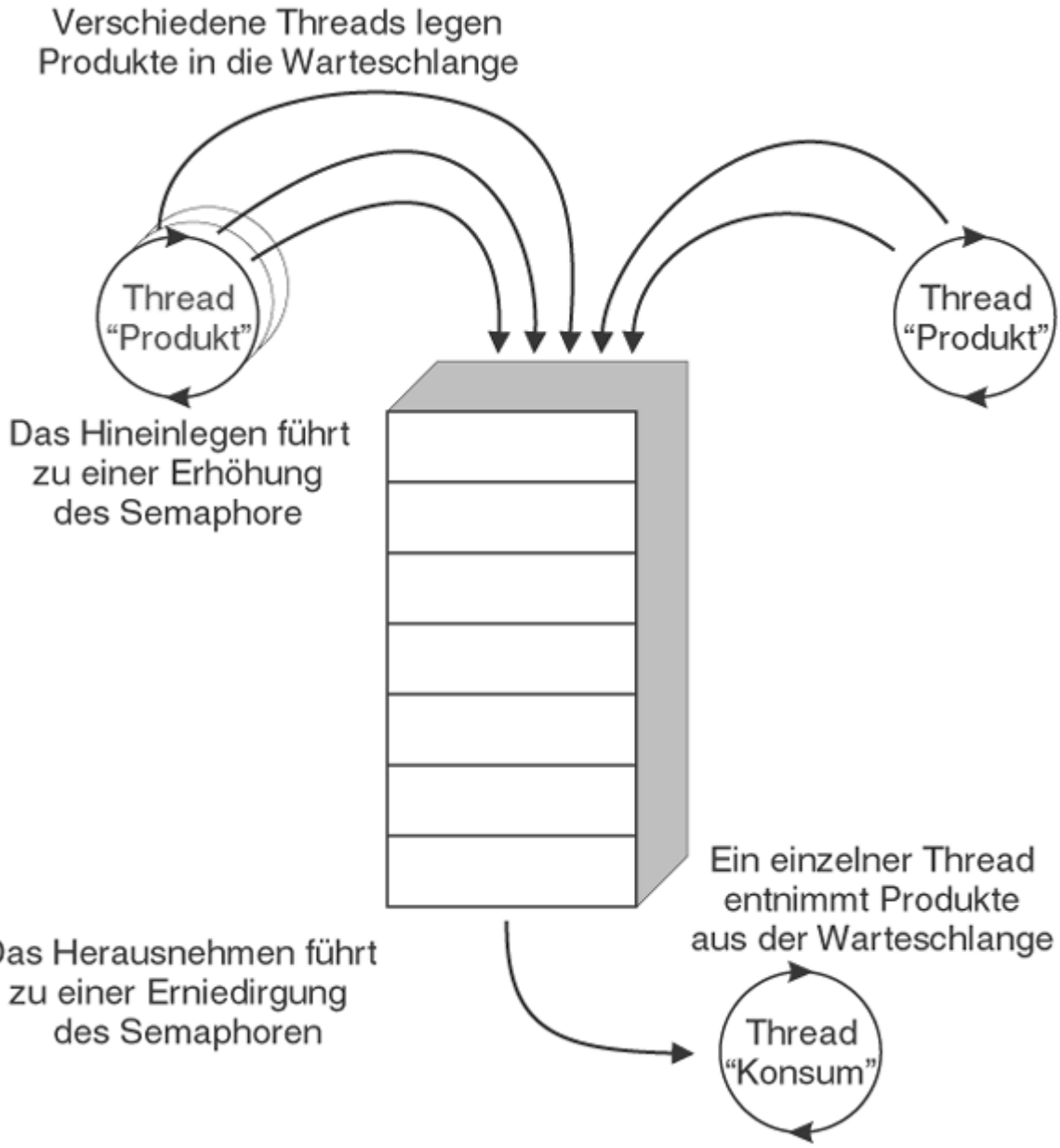


Abbildung 17.5: Mehrere Threads, die Objekte in eine Warteschlange stellen

Wenn Sie einen Semaphor in Ihrer Anwendung einsetzen müssen, können Sie eine Instanz der Klasse CSemaphore erzeugen. Diese Klasse hat vier Argumente, die man an den Konstruktor der Klasse übergeben kann:

- Die ersten beiden Argumente legen den anfänglichen und den maximalen Zählerstand für den Semaphor fest. Mit diesen beiden Argumenten lässt sich steuern, wie viele Threads und Prozesse gleichzeitig auf eine gemeinsam genutzte Ressource zugreifen können.
- Der Name für den Semaphor. Dieser Name identifiziert den Semaphor gegenüber allen Anwendungen, die im System laufen, analog zur Klasse CMutex.
- Die Klasse verwendet außerdem einen Zeiger auf die Sicherheitsattribute für den Semaphor.

Beim CSemaphore-Objekt kann man mit den Elementfunktionen Lock und Unlock die Steuerung des Semaphors erlangen oder freigeben. Ruft man die Funktion Lock auf, wenn der Verwendungszähler des Semaphors größer als Null ist, wird der Verwendungszähler dekrementiert und Ihr Programm kann weiterlaufen. Ist der Verwendungszähler bereits Null, dann wartet die Funktion Lock, bis der Verwendungszähler inkrementiert wird, damit Ihr Prozess den Zugriff auf die gemeinsam genutzte Ressource erlangen kann. Wenn Sie die Funktion Unlock aufrufen, wird der Verwendungszähler des Semaphors inkrementiert.

Ereignisse

In dem Maße wie die Synchronisierungsmechanismen für Threads dafür ausgelegt sind, den Zugriff auf begrenzte Ressourcen zu steuern, sind sie auch dafür vorgesehen, unnötige Prozessorzeit in Threads zu unterbinden. Je mehr Threads gleichzeitig laufen, desto langsamer führt jeder einzelne Thread seine Aufgaben aus. Wenn ein Thread also nichts zu tun hat, blockieren Sie ihn und belassen ihn im Leerlauf. Damit erhalten andere Threads mehr Prozessorzeit und laufen deshalb schneller, bis die Bedingungen erfüllt sind und der im Leerlauf arbeitende Thread etwas zu tun bekommt.

Aus diesem Grund verwendet man Ereignisse - um Threads den Leerlauf zu ermöglichen, bis die Bedingungen erfüllt sind und sie etwas zu tun bekommen. Ereignisse sind dem Namen nach mit den Ereignissen verwandt, die die meisten Windows-Anwendungen treiben, nur ist das Ganze etwas komplizierter. Die Ereignisse zur Synchronisierung von Threads arbeiten nicht nach den Mechanismen der Ereigniswarteschlangen und -behandlung. Statt eine Nummer zu erhalten und dann darauf zu warten, dass diese Nummer an die Behandlungsroutine von Windows übergeben wird, sind Ereignisse der Thread-Synchronisierung eigentliche Objekte, die sich im Speicher befinden. Jeder Thread, der auf ein Ereignis warten muss, teilt dem Ereignis mit, dass er auf dessen Auslösung wartet, und nimmt dann einen Ruhezustand ein. Wird das Ereignis ausgelöst, geht ein Signal an jeden Thread, der dem Ereignis seinen Wartezustand bekanntgegeben hat. Die Threads nehmen die Verarbeitung genau an dem Punkt auf, wo sie dem Ereignis die Warteabsicht mitgeteilt haben.

Wenn Sie in Ihrer Anwendung mit Ereignissen arbeiten, können Sie ein CEvent-Objekt einsetzen. Das CEvent-Objekt ist zu erzeugen, wenn Sie auf das Ereignis zugreifen und darauf warten müssen. Sobald der CEvent-Konstruktor zurückkehrt, ist das Ereignis aufgetreten und Ihr Thread kann seine Arbeit weiter fortsetzen.

Der Konstruktor für die CEvent-Klasse kann vier Argumente übernehmen. Das erste Argument ist ein boolesches Flag. Es gibt an, ob der Thread das von ihm erzeugte Ereignis anfänglich besitzt. Diesen Wert sollte man auf TRUE setzen, wenn der das CEvent-Objekt erzeugende Thread auch ermittelt, wann das Ereignis auftritt.

Das zweite Argument an den CEvent-Konstruktor legt fest, ob es sich um ein automatisches oder manuelles Ereignis handelt. Ein manuelles Ereignis verbleibt im signalisierten oder nicht signalisierten Zustand, bis es der Thread, der das Ereignisobjekt besitzt, explizit in den anderen Zustand versetzt. Ein automatisches Ereignis verbleibt die meiste Zeit im nicht signalisierten Zustand. Wurde es in den signalisierten Zustand gesetzt und wird mindestens ein Thread freigegeben und setzt seine Arbeit im Ausführungspfad fort, kehrt das Ereignis in den nicht signalisierten Zustand zurück.

Das dritte Argument an den Ereigniskonstruktor ist der Name für das Ereignis. Alle Threads, die auf das Ereignis zugreifen müssen, identifizieren es anhand dieses Namens. Das vierte und letzte Argument ist ein Zeiger auf die Sicherheitsattribute für das Ereignisobjekt.

Die Klasse CEvent verfügt über mehrere Member-Funktionen, mit denen Sie den Zustand des Ereignisses steuern können. Tabelle 17.2 listet diese Funktionen auf.

Funktion	Beschreibung

SetEvent	Versetzt das Ereignis in den signalisierten Zustand
PulseEvent	Versetzt das Ereignis in den signalisierten Zustand und dann wieder zurück in den nicht signalisierten
ResetEvent	Versetzt das Ereignis in den nicht signalisierten Zustand
Unlock	Gibt das Ereignisobjekt frei

Tabelle 17.2: Member-Funktionen der Klasse CEvent

17.2 Eine Multitasking-Anwendung erstellen

Um zu zeigen, wie Sie eigene Multitasking-Anwendungen erstellen können, erzeugen Sie eine Anwendung mit vier sich drehenden Farbfächern, von denen jeder in seinem eigenen Thread läuft. Zwei der Fächer verwenden die Funktion OnIdle, während die beiden anderen als unabhängige Threads laufen. Mit dieser Einrichtung können Sie die Unterschiede zwischen beiden Typen des Threadings studieren und lernen, wie man jeden Typ einsetzt. Das Anwendungsfenster erhält vier Kontrollkästchen, um jeden Thread zu starten und anzuhalten. Auf diese Weise lässt sich demonstrieren, wie viel Verarbeitungsleistung das System erbringen muss, wenn jeder Thread allein oder in Verbindung mit den anderen läuft.

Das Anwendungsgerüst

Für die heute zu erstellende Anwendung ist das Gerüst einer SDI-Anwendung erforderlich, bei dem die Ansichtsklasse von der Klasse CFormView abgeleitet ist, sodass Sie mit dem Dialog-Designer das Layout der wenigen Steuerelemente im Fenster gestalten können. In der Dokumentklasse sind die Fächer und die unabhängigen Threads realisiert. Die Ansicht enthält die Kontrollkästchen und Variablen, die die Aktivität oder den Leerlauf der Threads steuern.

Um mit Ihrer Anwendung zu beginnen, folgen Sie diesen Schritten:

1. Erzeugen Sie ein neues MFC-Anwendungs-Visual-C++-Projekt und nennen Sie es Multitasking.

2. Im Bereich Anwendungstyp wählen Sie die Option Einfaches Dokument als Anwendungstyp.
3. Im Bereich Erstellte Klassen legen Sie die Basisklasse als CFormView fest und klicken Sie auf Fertig stellen. Der MFC-Anwendungs-Assistent erstellt das Anwendungsgerüst.
4. Löschen Sie das Textfeld aus dem Hauptfenster der Anwendung und fügen Sie vier Kontrollkästchen etwa in der oberen linken Ecke jedes Quadranten hinzu, wie es Abbildung 17.6 zeigt. Legen Sie die Eigenschaften der Kontrollkästchen gemäß Tabelle 17.3 fest.

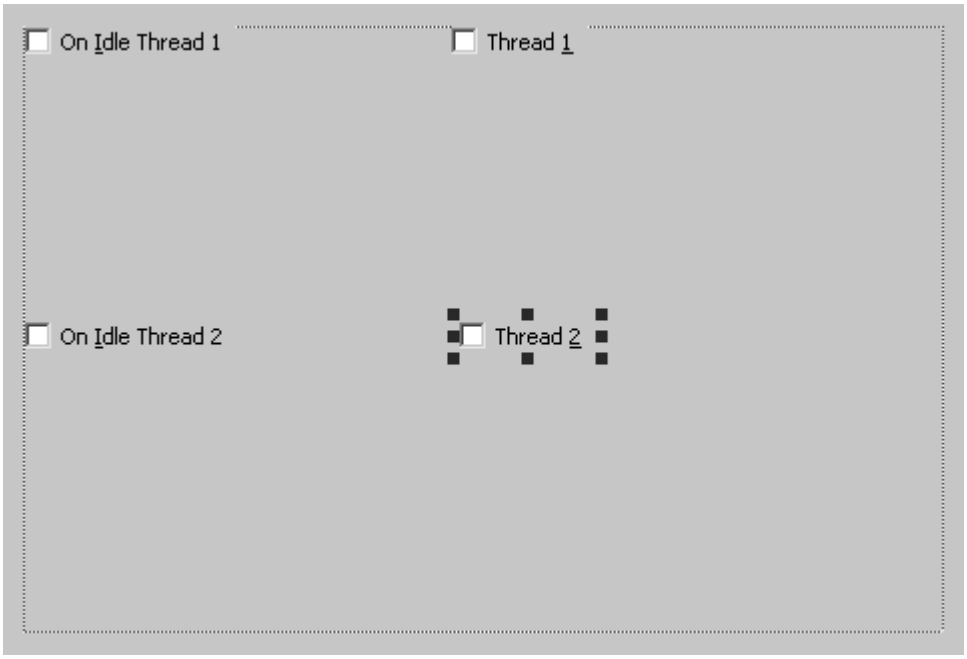


Abbildung 17.6: Der Entwurf des Hauptfensters

Objekt	Eigenschaft	Einstellung
Kontrollkästchen	ID Beschriftung	IDC_CBONIDLE1 On &Idle Thread 1
Kontrollkästchen	ID Beschriftung	IDC_CBTHREAD1 Thread &1
Kontrollkästchen	ID Beschriftung	IDC_CBONIDLE2 On &Idle Thread 2
Kontrollkästchen	ID Beschriftung	IDC_CBTHREAD2 Thread &2

Tabelle 17.3: Eigenschaften der Steuerelemente

5. Fügen Sie für jedes Kontrollkästchen eine Variable hinzu. Geben Sie allen Variablen den Typ BOOL und vergeben Sie folgende Namen:

Objekt	Name	Kategorie	Typ	Zugriff
IDC_CBONIDLE1	m_bOnIdle1	Value	BOOL	public
IDC_CBONIDLE2	m_bOnIdle2	Value	BOOL	public
IDC_CBTHREAD1	m_bThread1	Value	BOOL	public
IDC_CBTHREAD2	m_bThread2	Value	BOOL	public

Fächer entwerfen

Bevor Sie Threads in Ihre Anwendung einfügen können, erstellen Sie den drehenden Farbfächer, auf dem die Threads operieren. Da sich alle vier Farbfächer unabhängig voneinander drehen, ist es sinnvoll, die gesamte

Funktionalität in einer einzigen Klasse zu verkapseln. Diese Klasse verfolgt, welche Farbe gezeichnet wird, wo im Fächer die nächste Linie zu zeichnen ist, welche Größe der Fächer hat und wo sich der Fächer im Anwendungsfenster befindet. Außerdem braucht die Klasse einen Zeiger auf die Ansichtsklasse, um den Gerätekontext zu holen und sich selbst darin zu zeichnen. Für die unabhängigen Fächer benötigt die Klasse einen Zeiger auf das Flag, das die Drehung der Fächer steuert.

Für die Fächerklasse folgen Sie diesen Schritten:

1. Erzeugen Sie eine neue allgemeine Klasse, die von der Basisklasse CObject abgeleitet ist.

2. Geben Sie der neuen Klasse einen Namen, die beschreibt, was sie tut. In diesem Beispiel nennen Sie die Klasse CSpinner.

Die Fächervariablen festlegen

Der neu erstellten Klasse für Ihr Fächerobjekt fügen Sie nun einige Variablen hinzu. Gemäß der objektorientierten Richtlinien legen Sie diese Variablen als »private« fest und nehmen Methoden in die Klasse auf, um die Werte der Variablen zu setzen und abzurufen.

Die Variablen sind für die folgenden Werte vorgesehen:

- aktuelle Farbe
- aktuelle Position in der Drehung des Farbfächers
- Größe des Farbfächers
- Position des Farbfächers im Anwendungsfenster
- die Farbtabelle, aus der die Farben für das Zeichnen im Farbfächer ausgewählt werden
- ein Zeiger auf das Ansichtsobjekt, damit der Fächer den Gerätekontext für das Zeichnen im Fenster holen kann
- ein Zeiger auf die Variable des Kontrollkästchens, das festlegt, ob der jeweilige Thread laufen soll

Tabelle 17.4 gibt die Namen und Datentypen dieser Variablen an, die Sie alle in die Klasse CSpinner einfügen. Kennzeichnen Sie alle als private.

Name	Typ	Beschreibung
m_iColor	Int	Die aktuelle Farbe aus der Farbtabelle
m_iMinute	Int	Die Position bei der Drehung um die Achse
m_iRadius	Int	Der Radius (die Größe) des Fächers
m_ptCenter	CPoint	Der Mittelpunkt des Fächers
m_crColors[8]	static COLORREF	Die Farbtabelle mit allen Farben, die im Farbfächer zu zeichnen sind
m_pViewWnd	CWnd*	Ein Zeiger auf das Ansichtsobjekt
m_pbContinue	BOOL*	Ein Zeiger auf die Variable des Kontrollkästchens, die festlegt, ob dieser Thread laufen soll

Tabelle 17.4: Die Variablen der Klasse CSpinner

Wenn Sie alle erforderlichen Variablen hinzugefügt haben, müssen Sie sicherstellen, dass Ihre Klasse die Variablen entweder initialisiert oder ein geeignetes Instrument bereitstellt, um die Werte der Variablen zu setzen und abzurufen. Die Integer-Variablen können den Anfangswert 0 erhalten. Die Zeiger sollten Sie mit NULL initialisieren. Die Initialisierungen lassen sich insgesamt im Konstruktor der Klasse erledigen, wie es Listing 17.1 zeigt.

Listing 17.1: Der Konstruktor von CSpinner

```
1: CSpinner::CSpinner(void)
```

```
2: :  m_iColor(0)
3: , m_iMinute(0)
4: , m_iRadius(0)
5: , m_ptCenter(0)
6: , m_pViewWnd(NULL)
7: , m_pbContinue(NULL)
8: {
9: }
```

Für diejenigen Variablen, die Sie setzen und abrufen müssen, ist Ihre Fächerklasse einfach genug, damit Sie alle diesbezüglichen Funktionen als Inline-Funktionen in der Deklaration der Klasse schreiben können. Farbe und Position werden automatisch durch das Fächerobjekt berechnet, sodass für diese beiden Variablen keine Funktionen für das Setzen erforderlich sind. Die übrigen Variablen (nicht mitgezählt die Farbtabelle) müssen sich aber setzen lassen. Die einzigen Variablen, die Sie aus dem Fächerobjekt abrufen müssen, sind die Zeiger auf die Ansichtsklasse und die Variable des Kontrollkästchens.

Die betreffenden Funktionen können Sie in die Deklaration der Klasse CSpinner aufnehmen, indem Sie die Header-Datei öffnen und die Inline-Funktionen gemäß Listing 17.2 einfügen.

Listing 17.2: Die Klassendeklaration von CSpinner

```
 1: class CSpinner :
 2:     public CObject
 3: {
 4: public:
 5:     CSpinner(void);
 6:     ~CSpinner(void);
 7:     BOOL* GetContinue() { return m_pbContinue;}
 8:     void SetContinue(BOOL* pbContinue)
 9:                                 { m_pbContinue =  pbContinue;}
10:     CWnd* GetViewWnd() { return m_pViewWnd;}
11:     void SetViewWnd(CWnd* pWnd) { m_pViewWnd = pWnd;}
12:     void SetLength(int iLength) { m_iRadius = iLength;}
13:     void SetPoint(CPoint ptPoint) { m_ptCenter = ptPoint;}
14:
15: private:
16:     // Die aktuelle Farbe aus der Farbtabelle
17:     int m_iColor;
18:     // Die Position in der Drehung des Fächers
19:     int m_iMinute;
20:     // Der Radius (die Größe) des Fächers
21:     int m_iRadius;
22:     // Der Mittelpunkt des Fächers
23:     CPoint m_ptCenter;
24:     // Die Farbtabelle mit allen im Fächer zu zeichnenden Farben
25:     static COLORREF m_crColors[8];
26:     // Ein Zeiger auf das Ansichtsobjekt
27:     CWnd* m_pViewWnd;
28:     // Ein Zeiger auf die Kontrollkästchenvariable, die angibt,
29:     // ob der Thread laufen soll
30:     BOOL* m_pbContinue;
31: };
```

Nachdem Sie nun alle Unterstützungsfunktionen für das Setzen und Abrufen der erforderlichen Variablen hinzugefügt haben, müssen Sie die Farbtabelle deklarieren und füllen. Das entspricht in etwa der Definition für die Farbtabelle, die Sie in die Anwendung von Tag 10, »SDI- und MDI-Anwendungen«, aufgenommen haben. Die Farbtabelle besteht aus acht RGB-Werten, wobei jeder Wert entweder 0 oder 255 sein kann und jede Kombination dieser beiden Einstellungen möglich ist.

Der beste Platz für diese Tabellendeklaration ist in der Quellcode-Datei des Fächers unmittelbar vor dem Konstruktor der Klasse, wie es Listing 17.3 angibt.

Listing 17.3: Die Farbtabelle von CSpinner

```
 1: #include "StdAfx.h"
 2: #include "spinner.h"
 3:
 4: COLORREF CSpinner::m_crColors[8] = {
 5:     RGB(   0,   0,   0),    // Schwarz
 6:     RGB(   0,   0, 255),    // Blau
 7:     RGB(   0, 255,   0),    // Grün
 8:     RGB(   0, 255, 255),    // Cyan
 9:     RGB( 255,   0,   0),    // Rot
10:     RGB( 255,   0, 255),    // Magenta
11:     RGB( 255, 255,   0),    // Gelb
12:     RGB( 255, 255, 255)     // Weiß
13: };
14:
15: CSpinner::CSpinner(void)
16: : m_crColor(0)
17: . ..
```

Den Fächer zeichnen

Nun kommt der angenehme Teil: das Fächerobjekt tatsächlich in Drehung versetzen. Um dies zu erreichen, berechnen Sie die neue Position für den Start- und Endpunkt jeder Linie, setzen den Ursprungspunkt des Viewports, wählen die Zeichenfarbe und erzeugen einen Zeichenstift in dieser Farbe. Anschließend zeichnen Sie die Linie vom Startpunkt zum Endpunkt. Danach stellen Sie den ursprünglichen Stift - der vor dem Zeichnen der Linie gültig war - wieder her. Vor dem Verlassen der Funktion berechnen Sie noch die Position der nächsten Linie.

Um die Funktionalität in das Fächerobjekt aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie eine Member-Funktion in die Klasse CSpinner ein. Legen Sie den Typ als void, den Namen mit Draw und den Zugriff als public fest.

2. In die Funktion schreiben Sie den Code aus Listing 17.4.

Listing 17.4: Die Funktion CSpinner.Draw

```
 1: void CSpinner::Draw(void)
 2: {
 3:   // Zeiger auf den Gerätekontext holen
 4:   CDC *pDC = m_pViewWnd->GetDC();
 5:   // Abbildungsmodus setzen
 6:   pDC->SetMapMode (MM_LOENGLISH);
 7:   // Fächermittelpunkt kopieren
 8:   CPoint org = m_ptCenter;
 9:   CPoint ptStartPoint;
10:   // Anfangspunkt setzen
11:   ptStartPoint.x = (m_iRadius / 2);
12:   ptStartPoint.y = (m_iRadius / 2);
13:   // Ursprungspunkt setzen
14:   org.x = m_ptCenter.x + (m_iRadius / 2);
15:   org.y = m_ptCenter.y + m_iRadius;
16:   // Ursprungspunkt des Viewport setzen
17:   pDC->SetViewportOrg(org.x, org.y);
18:
19:   CPoint ptEndPoint;
20:   // Winkel für die nächste Linie berechnen
21:   const double pi = 3.1415926535;
22:   double nRadians = (double) (m_iMinute) * pi / 30;
23:   // Endpunkt der Linie setzen
24:   ptEndPoint.x = (int) (m_iRadius * sin(nRadians));
25:   ptEndPoint.y = (int) (m_iRadius * cos(nRadians));
26:
27:   // Zu verwendenden Stift erzeugen
```

```
28:   CPen pen(PS_SOLID, 0, m_crColors[m_iColor]);
29:   // Stift auswählen
30:   CPen* pOldPen = pDC->SelectObject(&pen);
31:   // Zum Anfangspunkt gehen
32:   pDC->MoveTo (ptEndPoint);
33:   // Linie zum Endpunkt zeichnen
34:   pDC->LineTo (ptStartPoint);
35:
36:   // Vorherigen Stift wieder auswählen
37:   pDC->SelectObject(&pOldPen);
38:
39:   // Gerätekontext freigeben
40:   m_pViewWnd->ReleaseDC(pDC);
41:
42:   // Minute inkrementieren
43:   if (++m_iMinute == 60)
44:   {
45:     // Wurde vollständiger Kreis beschrieben, auf 0 zurücksetzen
46:     m_iMinute = 0;
47:     // Farbe inkrementieren
48:     if (++m_iColor == 8)
49:       // Wurden alle Farben durchlaufen, von vorn beginnen
50:       m_iColor = 0;
51:   }
52: }
```

Das ist eine ganze Menge Code! Was bewirkt er? Um zu verstehen, was diese Funktion realisiert und wie der sich drehende Farbfächer in das Fenster gelangt, sehen wir uns den Code näher an.

Damit die verschiedenen Threads den Fächer effektiv nutzen können, zeichnet die Funktion bei jedem Aufruf nur jeweils eine Linie. Für einen vollständigen Kreis ist diese Funktion 60-mal aufzurufen, einmal für jeden Minutenzeiger bei Drehung im Uhrzeigersinn. Nach jeder kompletten Umdrehung schaltet der Fächer zur nächsten Farbe in der Farbtabelle.

Bevor man überhaupt in das Fenster zeichnen kann, muss man zunächst den Gerätekontext des Fensters holen. Dazu ruft man die Funktion GetDC auf dem Zeiger des Ansichtsobjekts auf:

```
CDC *pDC = m_pViewWnd->GetDC();
```

Die Funktion liefert einen Zeiger auf ein CDC-Objekt zurück, d.h. auf eine MFC-Klasse, die den Gerätekontext verkapselt.

Wenn Sie über einen Zeiger auf den Gerätekontext verfügen, können Sie die Member-Funktion SetMapMode aufrufen, um den Abbildungsmodus festzulegen:

```
pDC->SetMapMode (MM_LOENGLISH);
```

Der Abbildungsmodus bestimmt, wie die x- und y-Koordinaten in Positionen auf dem Bildschirm übersetzt werden. Der Modus MM_LOENGLISH bildet eine logische Einheit auf 0,01 Zoll auf dem Bildschirm ab. Es sind verschiedene Abbildungsmodi verfügbar, die jeweils logische Einheiten in bestimmte Maßeinheiten auf dem Bildschirm konvertieren.

Jetzt wenden Sie sich den Vorbereitungen zu, um die aktuelle Linie für den Farbfächer zu zeichnen. Zuerst berechnen Sie den Anfangspunkt für die zu zeichnende Linie. Dieser Punkt ist für alle vom Fächerobjekt gezeichneten Linien gleich. Anschließend berechnen Sie die Position

des Viewports. Der Viewport dient als Ausgangspunkt für das Koordinatensystem, in dem die Zeichnung liegt.

Der Viewport ist der rechteckige Bereich eines Formulars, das im Container des Formulars angezeigt wird - mit anderen Worten ein Grafikfenster.

Der Anfangspunkt der zu zeichnenden Linie wird für eine Position außerhalb des Mittelpunktes berechnet. Wenn der Ursprung für die Linien genau im Mittelpunkt des Fächers liegen soll, setzen Sie sowohl die x- als auch die y-Koordinate des Anfangspunktes auf 0.

Nachdem der Ursprung des Viewports berechnet ist, setzen Sie den Viewport mit der Funktion SetViewportOrg:

```
pDC->SetViewportOrg(org.x, org.y);
```

Der Zeichenbereich und der Startpunkt für die zu zeichnende Linie sind nun definiert. Als Nächstes ist der Endpunkt der Linie zu bestimmen. Diese Berechnung steht in den folgenden vier Code-Zeilen:

```
const double pi = 3.1415926535;
double nRadians = (double) (m_iMinute) * pi / 30
 ptEndPoint.x = (int) (m_iRadius * sin(nRadians));
ptEndPoint.y = (int) (m_iRadius * cos(nRadians));
```

Die erste Zeile konvertiert die Minuten (wie auf der guten alten Analoguhr, nicht zu verwechseln mit Bogenminuten) in Grad (gemessen in Radians). Das Ergebnis lässt sich dann in die Sinus- und Kosinusfunktionen einsetzen, um die x- und y-Koordinaten für das Zeichnen eines Kreises festzulegen. Daraus ergibt sich der Endpunkt der zu zeichnenden Linie.

Nunmehr verfügen Sie über den Anfangs- und Endpunkt der Linie und können einen Stift erzeugen, der die Linie zeichnet:

```
CPen pen(PS_SOLID, 0, m_crColors[m_iColor]);
```

Diese Anweisung legt fest, dass der Stift durchgehend und dünn sein soll und die aktuelle Farbe aus der Farbtabelle auszuwählen ist. Nachdem der zu verwendende Stift erzeugt wurde, selektieren Sie ihn zum Zeichnen. Dabei übernehmen Sie den aktuellen Stift als Rückgabewert vom Gerätekontextobjekt:

```
CPen* pOldPen = pDC->SelectObject(&pen);
```

Jetzt lässt sich die Linie zeichnen. Dabei kommen die Funktionen MoveTo und LineTo zum Einsatz, die Ihnen mittlerweile vertraut sein dürften. Nach dem Zeichnen der Linie geben Sie den Gerätekontext wieder frei, damit kein Ressourcenmangel in der Anwendung auftritt:

```
m_pViewWnd->ReleaseDC(pDC);
```

Damit ist die Linie fertig gestellt. Jetzt inkrementieren Sie noch den Minutenzähler. Falls der Kreis geschlossen ist, setzen Sie den Zähler auf 0 zurück. Nach jedem vollständigen Kreis inkrementieren Sie den Farbenzähler und setzen ihn nach der achten Farbe wieder auf 0.

Damit Sie in dieser Funktion auf die trigonometrischen Funktionen (sin und cos) zurückgreifen können, binden Sie die Header-Datei math.h in die Quelldatei der Fächerklasse ein. Gehen Sie dazu an den Beginn der Quelldatei und nehmen Sie die #include-Zeile mit der Header-Datei math.h auf:

```
#include "StdAfx.h"
#include <math.h>
#include "Spinner.h"
```

Die Fächer unterstützen

Sie haben nun die Fächerklasse erzeugt, um den sich drehenden Farbfächer im Fenster zu zeichnen. Für die Fächer sind noch verschiedene Unterstützungsroutinen erforderlich. Man kann zwar der Dokumentklasse ein Array hinzufügen, das die vier Fächer aufnimmt, trotzdem muss man noch berechnen, wo jeder Fächer im Anwendungsfenster erscheinen soll. Außerdem sind die Variablen für jeden Fächer festzulegen.

Diesen Code bringen Sie in der Dokumentklasse unter, wobei Sie mit dem Array der Fächer beginnen. Fügen Sie der Dokumentklasse (im Beispiel CMultitaskingDoc) eine Member-Variable hinzu, für die Sie den Typ mit CSpinner, den Namen als m_cSpin[4] und den Zugriff als private festlegen.

Die Fächerpositionen berechnen

Zu den vorbereitenden Maßnahmen in der Initialisierungsphase der Anwendung gehört es, die Positionen aller vier Fächer zu bestimmen. Das Fenster ist durch die Kontrollkästchen, die die Fächer-Threads ein- und ausschalten, in vier etwa gleich große Quadranten unterteilt, sodass es naheliegt, den Fensterbereich in vier Rechtecke aufzuteilen und in jedem Rechteck einen Fächer unterzubringen.

Die Lage der einzelnen Fächer lässt sich am einfachsten berechnen, indem man eine Funktion erzeugt, die die Position für einen Fächer berechnet und ihn im Quadranten der jeweiligen Fächernummer platziert. Übergibt man der Funktion einen Zeiger auf das Fächerobjekt, kann sie dieses direkt an der Position aktualisieren.

Um die Funktionalität in Ihre Anwendung einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Funktionstyp als void und den Namen als CalcPoint fest und fügen Sie zwei Parameter ein: einen Parameter von Typ int namens iID und einen vom Typ CSpinner* namens pSpin. Legen Sie den Zugriff mit private fest.
3. Nehmen Sie in die Funktion den Code aus Listing 17.5 auf.

Listing 17.5: Die Funktion CMultitaskingDoc.CalcPoint

```
 1: void CMultitaskingDoc::CalcPoint(int iID, CSpinner* pSpin)
 2: {
 3:   RECT rWndRect;
 4:   CPoint ptPos;
 5:   int iLength;
 6:   CMultitaskingView *pWnd;
 7:
 8:   // Einen Zeiger auf das Ansichtsfenster holen
 9:   pWnd = (CMultitaskingView*)pSpin->GetViewWnd();
10:   // Rechteck des Anzeigebereichs ermitteln
11:   pWnd->GetClientRect(&rWndRect);
12:   // Größe der Fächer berechnen
13:   iLength = rWndRect.right / 6;
```

```
14:   // Welchen Fächer platzieren wir?
15:   switch (iID)
16:   {
17:     case 0:    // Ersten Fächer positionieren
18:       ptPos.x = (rWndRect.right / 4) - iLength;
19:       ptPos.y = (rWndRect.bottom / 4) - iLength;
20:       break;
21:     case 1:    // Zweiten Fächer positionieren
22:       ptPos.x = ((rWndRect.right / 4) * 3) - iLength;
23:       ptPos.y = (rWndRect.bottom / 4) - iLength;
24:       break;
25:     case 2:    // Dritten Fächer positionieren
26:       ptPos.x = (rWndRect.right / 4) - iLength;
27:       ptPos.y = ((rWndRect.bottom / 4) * 3) -
28:                  (long)(iLength * 1.25);
29:       break;
30:     case 3:    // Vierten Fächer positionieren
31:       ptPos.x = ((rWndRect.right / 4) * 3) - iLength;
32:       ptPos.y = ((rWndRect.bottom / 4) * 3) -
33:                  (long)(iLength * 1.25);
34:       break;
35:   }
36:   // Größe des Fächers setzen
37:   pSpin->SetLength(iLength);
38:   // Position des Fächers setzen
39:   pSpin->SetPoint(ptPos);
40: }
```

Diese Funktion ermittelt zuerst den Zeiger auf das Ansichtsfenster vom Fächerobjekt durch den Aufruf der Funktion GetViewWnd:

```
pWnd = (CMultitaskingView*)pSpin->GetViewWnd();
```

Indem man den Zeiger direkt vom Fächerobjekt ermittelt, spart man ein paar Schritte ein, um zu diesen Informationen zu gelangen.

Nachdem der Zeiger auf das Ansichtsobjekt vorhanden ist, kann man die Funktion GetClientRect des Fensters aufrufen, um die Größe des verfügbaren Zeichenbereichs zu ermitteln:

```
pWnd->GetClientRect(&rWndRect);
```

Aus der Größe des verfügbaren Zeichenbereichs lässt sich eine sinnvolle Größe für den Farbfächer berechnen, indem man die Länge des Zeichenbereichs durch 6 teilt:

```
iLength = rWndRect.right / 6;
```

Diese Berechnung ist nur überschlägig, da für die exakte Größe des Fächers sowohl Breite als auch Höhe des Fensters beachtet werden müssen. Sofern Sie beim Design des Fensters darauf achten, dass das Fenster etwa eineinhalb mal so breit wie hoch ist, »wird's scho' gehen«.

Teilt man die Breite des Zeichenbereiches durch 4, erhält man eine Position in der Mitte des oberen linken Quadranten. Von diesem Punkt zieht man die Größe des Kreises ab und man erhält die obere linke Ecke des Zeichenbereichs für den ersten Fächer:

```
ptPos.x = (rWndRect.right / 4) - iLength;
```

```
ptPos.y = (rWndRect.bottom / 4) - iLength;
```

Von diesem Punkt ausgehend erhält man die anderen Positionen, indem man den Mittelpunkt des Quadranten mit 3 multipliziert, um ihn in die Mitte des rechts daneben oder darunter liegenden Quadranten zu verschieben. Die Positionen der anderen drei Fächer lassen sich analog berechnen.

Nachdem die Länge und die Lage für den Fächer feststehen, übergibt man mit den Funktionen SetLength und SetPoint die ermittelten Werte an den Fächer, für den sie berechnet wurden:

```
pSpin->SetLength(iLength);
pSpin->SetPoint(ptPos);
```

Die Fächer initialisieren

Da Sie die Funktion in Listing 17.5, mit der Sie die Lage eines jeden Fächers im Fenster berechnen, geschrieben haben, um bei einem Aufruf der Funktion nur mit jeweils einem Fächer zu arbeiten, brauchen Sie eine Routine, die jeden Fächer initialisiert und dabei die Funktion CalcPoint einmal für jeden Fächer aufruft. Diese Funktion ist erforderlich, um einen Zeiger auf das Ansichtsobjekt zu ermitteln und den Zeiger dann an den Fächer weiterzureichen. Außerdem sind Zeiger auf die Variablen der Kontrollkästchen für die Fächer zu ermitteln, die von den unabhängig laufenden Threads verwendet werden. Der Code durchläuft dazu eine Schleife durch das Fächerfeld, legt dabei die Zeiger für jeden Fächer fest und übergibt dann den Fächer an die obige Funktion.

Um diese Funktion für Ihre Anwendung zu erstellen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Funktion in die Dokumentklasse ein.

2. Legen Sie den Typ als void und den Namen mit InitSpinners fest. Als Zugriffsstatus wählen Sie die Option private, da die Funktion nur einmal aufzurufen ist, wenn die Anwendung startet.
3. Nehmen Sie in die neue Funktion den Code aus Listing 17.6 auf.

Listing 17.6: Die Funktion CMultitaskingDoc.InitSpinners

```
 1: void CMultitaskingDoc::InitSpinners(void)
 2: {
 3:   int i;
 4:
 5:   // Position der Ansicht ermitteln
 6:   POSITION pos = GetFirstViewPosition();
 7:   // Ist die Position gültig?
 8:   if (pos)
 9:   {
10:     // Zeiger auf die Ansicht holen
11:     CView* pView = GetNextView(pos);
12:
13:     // Schleife durch die Fächer
14:     for (i = 0; i < 4; i++)
15:     {
16:       // Zeiger auf die Ansicht setzen
17:       m_cSpin[i].SetViewWnd(pView);
18:       // Zeiger auf den Anzeiger zum Fortfahren setzen
19:       m_cSpin[i].SetContinue(NULL);
20:       switch (i)
21:       {
22:         case 1:
23:           // Zeiger auf den ersten Anzeiger zum Fortfahren
24:           // setzen
25:           m_cSpin[i].SetContinue(
26:                     &((CMultitaskingView*)pView)->m_bThread1);
27:           break;
28:         case 3:
```

```
29:           // Zeiger auf den zweiten Anzeiger zum Fortfahren
30:           // setzen
31:           m_cSpin[i].SetContinue(
32:                     &((CMultitaskingView*)pView)->m_bThread2);
33:           break;
34:       }
35:       // Position des Fächers berechnen
36:       CalcPoint(i, &m_cSpin[i]);
37:     }
38:   }
39: }
```

Die Funktion durchläuft zuerst die Schritte, um einen Zeiger auf die Ansichtsklasse des Dokuments zu ermitteln, wie Sie es zu Beginn von Tag 10 kennen gelernt haben. Nachdem Sie einen gültigen Zeiger auf die Ansicht besitzen, starten Sie eine Schleife, um alle Fächer im Feld zu initialisieren. Sie rufen die Fächerfunktion SetViewWnd auf, um den Zeiger des Fächers auf das Ansichtsfenster zu setzen, und initialisieren dann den Zeiger auf die Variable für das Kontrollkästchen des Fächers mit NULL für alle Fächer. Gehört der Fächer zu den beiden, die unabhängige Threads verwenden, übergeben Sie einen Zeiger auf die entsprechende Variable des Kontrollkästchens. Nachdem das alles eingerichtet ist, rufen Sie die Funktion CalcPoint auf, die Sie gerade erstellt haben, um die Position des Fächers im Ansichtsfenster zu berechnen.

Auch wenn Sie schon mehrere Beispiele von Zeigern gesehen haben, verdient die Art und Weise der Übergabe eines Zeigers auf die Variable des Kontrollkästchens für den Fächer eine genauere Betrachtung:

```
m_cSpin[i].SetContinue(&((CMultitaskingView*)pView)->m_bThread1);
```

In dieser Anweisung nehmen Sie den Zeiger auf das Ansichtsobjekt pView, das einen Zeiger auf ein CView-Objekt darstellt, und führen eine Typumwandlung in einen Zeiger auf die spezielle Ansichtsklasse durch, die Sie in Ihrer Anwendung erstellt haben:

```
(CMultitaskingView*)pView
```

Da Sie nun den Zeiger auf das Ansichtsobjekt als CMultitaskingView-Objekt behandeln, können Sie zur Variablen m_bThread1 des Kontrollkästchens gelangen, die ein öffentliches Element der Klasse CMultitaskingView ist:

```
((CMultitaskingView*)pView)->m_bThread1
```

Nunmehr haben Sie Zugriff auf die Variable m_bThread1 und können die Adresse dieser Variablen ermitteln, indem Sie ein kaufmännisches Und-Zeichen vor die gesamte Anweisung stellen:

```
&((CMultitaskingView*)pView)->m_bThread1
```

Wenn Sie diese Adresse für die Variable m_bThread1 an die Funktion SetContinue übergeben, liefern Sie praktisch einen Zeiger auf die Variable m_bThread1. Damit kann man schließlich den Zeiger auf diese Variable, die das Fächerobjekt enthält, setzen.

Sie haben nun die Routinen erstellt, um alle Fächer zu initialisieren. Jetzt müssen Sie gewährleisten, dass diese Routine beim Start der Anwendung aufgerufen wird. Es liegt auf der Hand, dass man diese Logik in der Funktion OnNewDocument in der Dokumentklasse unterbringt. Diese Funktion wird aufgerufen, wenn die Anwendung startet. Somit ist es logisch, die Initialisierung der Fächerobjekte von dieser Stelle auszulösen.

Der Code, den Sie in die Funktion OnNewDocument in der Dokumentklasse einfügen, ist in Listing 17.7. wiedergegeben.

Listing 17.7: Die Funktion CMultitaskingDoc.OnNewDocument

```
 1: BOOL CMultitaskingDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Fächer initialisieren
 9:   InitSpinners();
10:
11:   return TRUE;
12: }
```

Den Fächer drehen

Eines der letzten Dinge, die Sie fürs Erste in die Dokumentklasse aufnehmen, ist ein Instrument, das den Aufruf der Funktion Draw für einen bestimmten Fächer von außerhalb der Dokumentklasse erlaubt. Da das Fächerfeld als private Variable deklariert ist, können Objekte von außerhalb nicht auf die Fächer zugreifen. Daher sind Mechanismen für derartige Zugriffe vorzusehen. Folgen Sie diesen Schritten:

1. Mit einer neuen Member-Funktion in der Dokumentklasse kann man diesen Zugriff bereitstellen.

2. Legen Sie für die Funktion den Typ void fest, spezifizieren Sie den Funktionsnamen als DoSpin und fügen Sie einen einzelnen Parameter vom Typ int namens iIndex ein. Wählen Sie als Zugriffsstatus die Option public.
3. In die Funktion schreiben Sie folgenden Code, um den eigentlichen Aufruf des angegebenen Fächers zu realisieren.

```
void CMultitaskingDoc::DoSpin(int iIndex)
{
  // Fächer drehen
  m_cSpin[iIndex].Draw();
}
```

Die OnIdle-Tasks hinzufügen

Nachdem Sie nun die unterstützende Funktionalität unter Dach und Fach haben, ist es an der Zeit, Ihre Aufmerksamkeit den verschiedenen Threads zu widmen, die für die Drehung der einzelnen Fächer verantwortlich sind. Als Erstes fügen Sie die Threads hinzu, die im Leerlauf der Anwendung ausgeführt werden. Für die zwei Kontrollkästchen sehen Sie eine Behandlungsroutine für das Klickereignis vor, sodass die Variablen für diese beiden Kontrollkästchen mit dem Fenster synchron bleiben. Weiterhin schreiben Sie Code in die OnIdle-Funktion der Anwendung, um diese beiden Fächer zu starten, wenn sich die Anwendung im Leerlauf befindet und die Kontrollkästchen für diese beiden Fächer-Threads eingeschaltet sind.

Der Begriff Thread in den vorstehenden Erläuterungen ist etwas irreführend. Die Funktionalität,

die Sie in die OnIdle-Funktion aufnehmen, läuft im Haupt-Thread der Anwendung ab. Die gesamte OnIdle-Verarbeitung, die Sie der Beispielanwendung hinzufügen, läuft nicht als unabhängiger Thread, sondern besteht eigentlich nur aus Funktionen, die sich über den Haupt-Thread aufrufen lassen.

Die OnIdle-Tasks starten und anhalten

Die Funktion OnIdle prüft die Werte der beiden Variablen für die Kontrollkästchen, die festlegen, ob der jeweilige Task laufen soll. Die Anwendung muss also nur dafür Sorge tragen, dass die Variablen im Ansichtsobjekt mit den Steuerelementen im Fenster synchronisiert werden, wenn man auf die Kontrollkästchen klickt. Dazu ist lediglich die Funktion UpdateData aufzurufen. Um die OnIdle-Tasks starten und anhalten zu können, muss man nur eine einzige Behandlungsroutine für beide On Idle Thread-Kontrollkästchen hinzufügen und dann die Funktion UpdateData in dieser Behandlungsroutine aufrufen.

Um die Funktionalität in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Öffnen Sie den Hauptfensterdialog im Fenster-Designer.

2. Markieren Sie eines der On Idle-Kontrollkästchen und fügen Sie eine Behandlungsroutine für das Ereignis BN_CLICKED hinzu.
3. Fügen Sie folgenden Code in die Funktion ein.

```
void CMultiaskingView::OnBnClickedCbonidle1()
{
  // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die Benachrichti
  // Variablen mit dem Dialogfeld synchronisieren
  UpdateData(TRUE);
}
```

4. Wiederholen Sie die Schritte 1-3 für die restlichen On Idle-Kontrollkästchen.

Die OnIdle-Threads erstellen

Wenn Sie den Quellcode der Anwendungsklasse (CMultitaskingApp) untersuchen, fällt Ihnen sicherlich auf, dass es hier keine OnIdle-Funktion gibt. Die ganze Funktionalität, die die OnIdle-Funktion per Vorgabe ausführen muss, befindet sich in der Basisklasse der Anwendungsklasse, die für Ihr Projekt erzeugt wurde. Der einzige Grund, eine OnIdle- Funktion in Ihre Anwendungsklasse aufzunehmen, besteht darin, dass Ihre Anwendung eine bestimmte Funktionalität während dieses Ereignisses realisieren muss. Im Ergebnis müssen Sie diese Behandlungsroutine explizit in Ihre Anwendung mithilfe der Überschreibungen von Klasseneigenschaften einbinden.

Was muss die OnIdle-Funktion tun, nachdem Sie sie in Ihre Anwendungsklasse aufgenommen haben?

- Es ist ein Zeiger auf die Ansicht zu ermitteln, damit sich der Zustand der Variablen für die Kontrollkästchen prüfen lässt.
- Die Funktion muss einen Zeiger auf die Dokumentklasse holen, damit der Aufruf der Funktion DoSpin möglich ist, um das entsprechende Fächerobjekt auszulösen.

Der Schlüssel für beide Aktionen liegt darin, Zeiger auf jedes dieser Objekte zu ermitteln. Wenn Sie sich damit beschäftigen, auf welche Weise diese Zeiger zu erhalten sind, stellen Sie fest, dass die angegebene Reihenfolge umzukehren ist. Sie müssen einen Zeiger auf das Dokumentobjekt holen, damit Sie einen Zeiger auf die Ansicht erhalten. Um jedoch einen Zeiger auf das Dokument zu erhalten, muss man durch die Dokumentvorlage gehen und einen Zeiger auf die Vorlage holen. Jeder dieser Schritte erfordert die gleiche Folge von Ereignissen - zuerst die Position des ersten Objekts holen und dann einen Zeiger auf das Objekt auf dieser Position ermitteln. Es ist also folgendes zu erledigen: die Position der ersten Dokumentvorlage ermitteln und dann einen Zeiger auf die Dokumentvorlage an dieser Position holen. Als Nächstes nehmen Sie die Dokumentvorlage, um die Position des ersten Dokuments zu ermitteln, und dann verwenden Sie sie, um einen Zeiger auf das Dokument an dieser ersten Position zu holen. Schließlich verwenden Sie das Dokument, um die Position der ersten Ansicht zu ermitteln, und verwenden es dann erneut , um einen Zeiger auf die Ansicht an der angegebenen Position zu holen. Nachdem Sie einen Zeiger auf die Ansicht besitzen, können Sie den Wert der Kontrollkästchen testen und den betreffenden Fächer aufrufen.

Um die Funktionalität in Ihre Anwendung einzubauen, folgen Sie diesen Schritten:

1. Nehmen Sie mit dem Modus Überschreibungen im Eigenschaftenfenster eine Funktion für die Nachricht OnIdle in die Anwendungsklasse (in diesem Fall CMultitaskingApp) auf.

2. Fügen Sie den Code von Listing 17.8 ein.

Listing 17.8: Die Funktion CMultitaskingApp.OnIdle

```
 1: BOOL CMultitaskingApp::OnIdle(LONG lCount)
 2: {
 3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
 4:   // und/oder rufen Sie die Basisklasse auf.
 5:   // Position der ersten Dokumentvorlage ermitteln
 6:   POSITION pos = GetFirstDocTemplatePosition();
 7:   // Ist die Position gültig?
 8:   if (pos)
 9:   {
10:     // Zeiger auf die Dokumentvorlage holen
11:     CDocTemplate* pDocTemp = GetNextDocTemplate(pos);
12:     // Ist der Zeiger gültig?
13:     if (pDocTemp)
14:     {
15:       // Position des ersten Dokuments ermitteln
16:       POSITION dPos = pDocTemp->GetFirstDocPosition();
17:       // Ist die Position gültig?
18:       if (dPos)
19:       {
20:         // Zeiger auf das Dokument holen
21:         CMultitaskingDoc* pDocWnd =
22:                 (CMultitaskingDoc*)pDocTemp->GetNextDoc(dPos);
23:
24:         // Ist der Zeiger gültig?
25:         if (pDocWnd)
26:         {
27:           // Position der Ansicht ermitteln
28:           POSITION vPos = pDocWnd->GetFirstViewPosition();
29:           // Ist die Position gültig?
30:           if (vPos)
31:           {
32:             // Zeiger auf die Ansicht holen
33:             CMultitaskingView* pView =
34:                 (CMultitaskingView*)pDocWnd->GetNextView(vPos);
35:
36:             // Ist der Zeiger gültig?
37:             if (pView)
38:             {
39:               // Ersten Idle-Thread drehen?
40:               if (pView->m_bOnIdle1)
41:                 // Ersten Idle-Thread drehen
42:                 pDocWnd->DoSpin(0);
43:               // Zweiten Idle-Thread drehen?
44:               if (pView->m_bOnIdle2)
45:                 // Zweiten Idle-Thread drehen
46:                 pDocWnd->DoSpin(2);
47:             }
48:           }
49:         }
50:       }
51:     }
52:   }
53:
54:   // Idle-Verarbeitung des Vorfahren aufrufen
```

```
55:    return CWinApp::OnIdle(lCount);
56: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie die Kontrollkästchen On Idle Thread einschalten und Farbfächer wie in Abbildung 17.7 zeichnen können, solange Sie die Maus bewegen. In dem Moment aber, in dem Sie die Anwendung im Leerlauf - ohne Mausbewegung oder irgend etwas anderes - arbeiten lassen, dreht sich der Fächer nicht mehr.

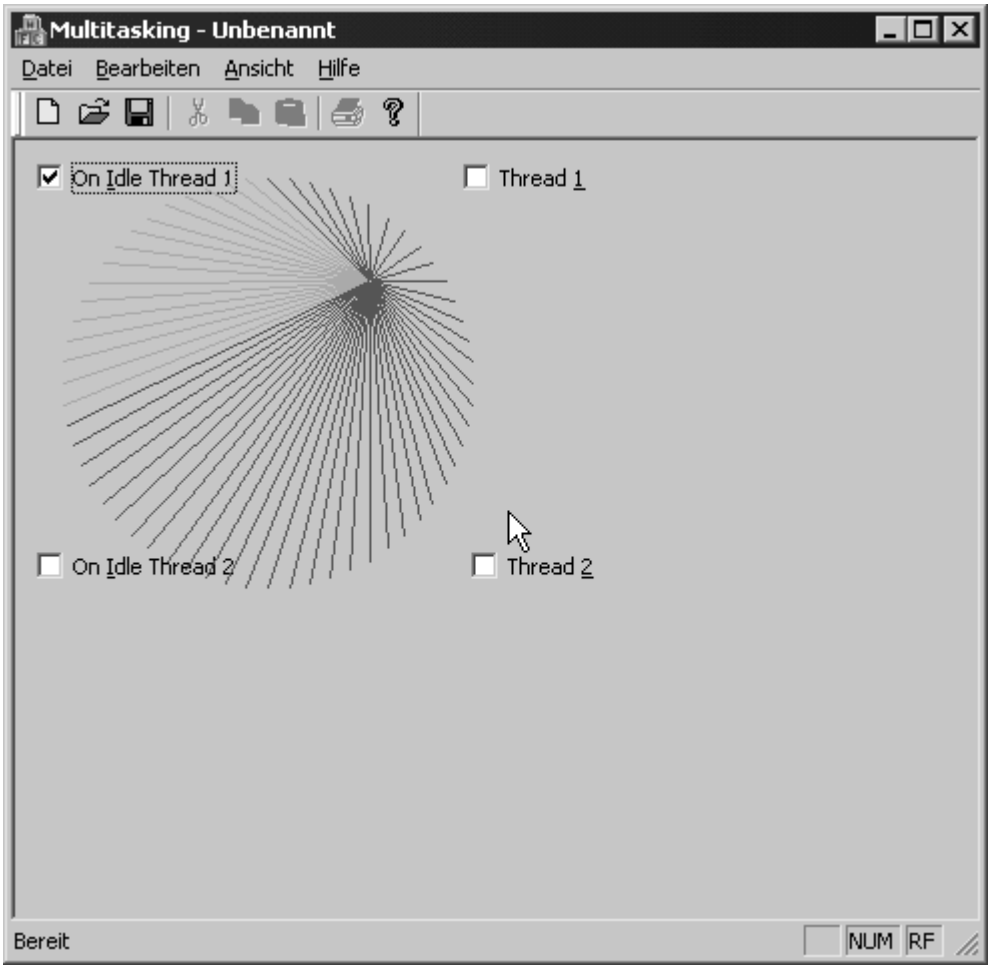


Abbildung 17.7: Ein OnIdle-Thread, der einen Farbfächer zeichnet

Fortlaufend arbeitende OnIdle-Tasks

Es ist nicht gerade sehr praktisch, die Maus in Bewegung zu halten, damit die Anwendung die Aufgaben, die eigentlich im Leerlauf stattfinden sollen, fortlaufend ausführt. Es muss eine Möglichkeit geben, die Funktion OnIdle aufzurufen, solange sich die Anwendung im Leerlauf befindet. Natürlich gibt es die. In der letzten Zeile von Listing 17.8 ist zu sehen, dass die Funktion OnIdle den Ergebniswert der OnIdle-Funktion aus der Basisklasse zurückgibt. Und diese Funktion liefert den Wert FALSE, sobald keine Leerlaufaktivitäten mehr auszuführen sind.

Man muss also erreichen, dass die Funktion OnIdle immer den Wert TRUE zurückliefert. Damit wird die Funktion OnIdle fortlaufend aufgerufen, wann immer die Anwendung im Leerlauf arbeitet. Wenn Sie den Aufruf der OnIdle-Funktion der Basisklasse - wie in Listing 17.9 angegeben - in den ersten Teil der Funktion verschieben und dann TRUE zurückgeben, dreht sich der Fächer ununterbrochen, unabhängig davon, wie lange die Anwendung im Leerlauf arbeitet.

Listing 17.9: Die modifizierte Funktion CMultitaskingApp.OnIdle

```
1: BOOL CMultitaskingApp::OnIdle(LONG lCount)
2: {
3:    // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
```

```
 4:    // und/oder rufen Sie die Basisklasse auf.
 5:
 6:    // Idle-Verarbeitung des Vorfahren aufrufen
 7:    CWinApp::OnIdle(lCount);
 8:
 9:    // Position der ersten Dokumentvorlage ermitteln
10:    POSITION pos = GetFirstDocTemplatePosition();
11:    // Ist die Position gültig?
12:    if (pos)
13:    {
14: . . .
15:    }
16:    return TRUE;
17: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie die OnIdle-Tasks einschalten und eine fortlaufende Drehung beobachten, selbst wenn Sie die Maus nicht bewegen. Falls Sie jedoch eines der Menüs aktivieren oder das Info-Dialogfeld öffnen, werden beide Tasks vollständig gestoppt, wie es Abbildung 17.8 zeigt. Der Grund dafür liegt darin, dass geöffnete Menüs und alle geöffneten modalen Dialogfelder den Aufruf der OnIdle-Funktion unterbinden. Eine der Beschränkungen der OnIdle-Verarbeitung ist es, dass bestimmte Funktionen der Anwendung das weitere Laufen der Tasks verhindern.

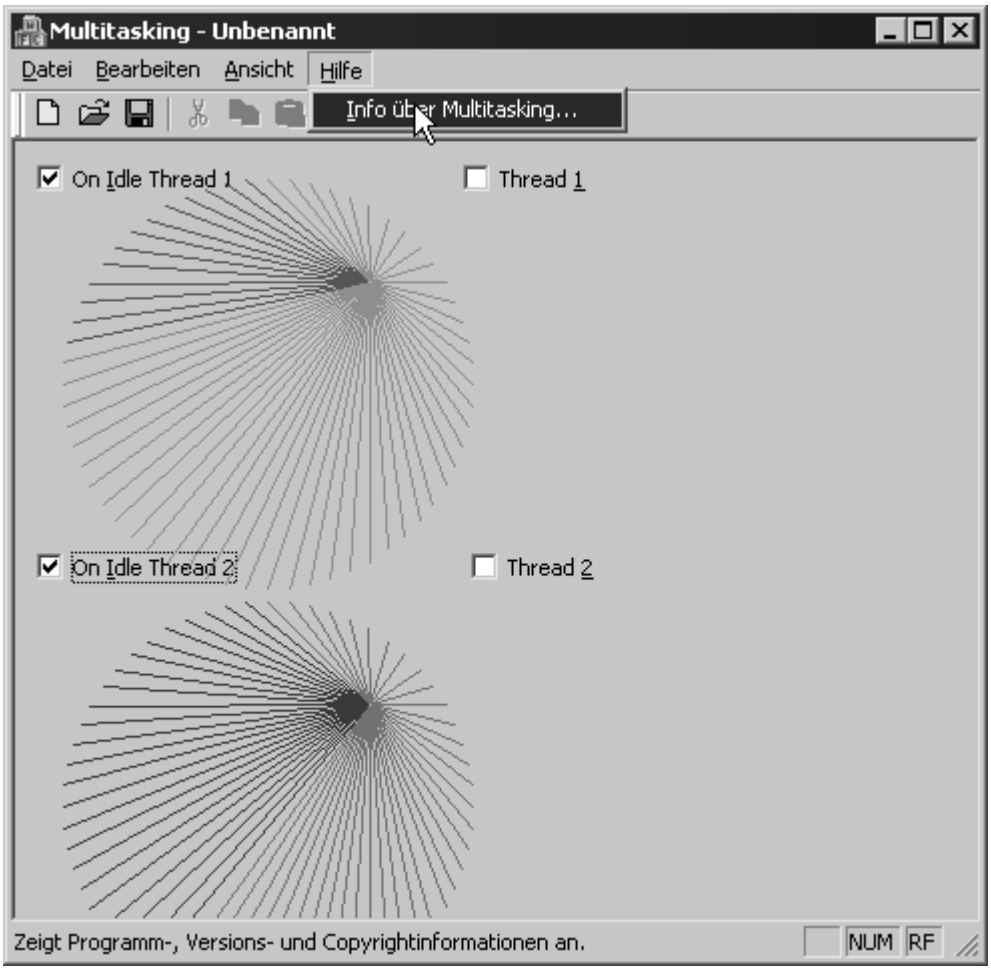


Abbildung 17.8: Ein OnIdle-Thread, der durch das Menü angehalten wird

Unabhängige Threads hinzufügen

Nachdem Sie sich damit bekannt gemacht haben, was alles zu einem OnIdle-Task gehört, geht es nun darum, unabhängige Threads in einer Anwendung zu realisieren. Als Erstes fügen Sie eine Hauptfunktion für die Threads hinzu. Weiterhin ist der Code zum Starten und Anhalten der Threads aufzunehmen. Schließlich kommt noch der Code für die Kontrollkästchen der unabhängigen Threads hinzu, um diese starten und

stoppen zu können.

Die Hauptfunktion der Threads

Bevor Sie irgendeinen unabhängigen Thread starten können, muss dieser wissen, was zu tun ist. Dazu erzeugen Sie eine Thread-Hauptfunktion, die der Thread ausführt, wenn er startet. Er endet, sobald die Funktion endet. Demzufolge muss diese Funktion als Hauptsteuerung des Threads agieren und ihn solange laufen lassen, wie es irgendetwas für ihn zu tun gibt. Ist die Arbeit des Threads erledigt, muss die Funktion zurückkehren.

Wenn man eine Funktion als Hauptfunktion für einen Thread erzeugt, kann man ihr einen einzelnen Parameter übergeben. Dieser Parameter ist ein Zeiger auf Informationen, die der Thread für die Ausführung seiner Aufgaben benötigt. Bei der Anwendung, die Sie in diesem Kapitel erstellen, kann der Parameter ein Zeiger auf den Fächer sein, auf dem der Thread operiert. Alles andere, was dieser benötigt, lässt sich aus dem Fächerobjekt herausziehen.

Nachdem der Thread einen Zeiger auf seinen Fächer hat, kann er einen Zeiger auf die Variable für das Kontrollkästchen ermitteln, das Auskunft darüber gibt, ob die Drehung weiterlaufen oder er sich selbst anhalten soll. Solange die Variable TRUE ist, soll der Thread die Drehung weiter ausführen. Wenn er dann abgeschlossen ist, sollte er die Funktion AfxEndThread aufrufen, die ihn beendet und aufräumt. Die Funktion AfxEndThread übernimmt zwei Parameter: Der erste ist der Exitcode des Threads und der zweite ein boolescher Wert, der angibt, ob die von der Klasse CWndThread beanspruchten Ressourcen freizugeben sind.

Um die Funktion in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie der Dokumentklasse in Ihrer Anwendung eine neue Member-Funktion hinzu.

2. Legen Sie den Typ der Funktion mit UINT und den Funktionsnamen mit ThreadFunc fest und fügen Sie einen Parameter von Typ LPVOID namens pParam ein. Aktivieren Sie das Kontrollkästchen static und legen Sie den Zugriff als private fest.
3. In die Funktion übernehmen Sie den Code aus Listing 17.10.

Listing 17.10: Die Funktion CMultitaskingDoc.ThreadFunc

```
 1: UINT CMultitaskingDoc::ThreadFunc(LPVOID pParam)
 2: {
 3:   // Argument in Zeiger auf den Fächer dieses Threads umwandeln
 4:   CSpinner* pSpin = (CSpinner*)pParam;
 5:   // Zeiger auf das Fortfahren-Flag holen
 6:   BOOL* pbContinue = pSpin->GetContinue();
 7:
 8:   // Schleife, solange Flag wahr ist
 9:   while (*pbContinue)
10:     // Fächer drehen
11:     pSpin->Draw();
12:   // Thread beenden
13:   AfxEndThread(0, TRUE);
14:   return 0;
15: }
```

Die Threads starten und anhalten

Mit der Funktion ThreadFunc können Sie nun die unabhängigen Threads aufrufen. Jetzt brauchen Sie noch eine Möglichkeit, die Threads zu steuern, d h. zu starten und anzuhalten. Dafür müssen Sie eine Reihe von Zeigern für CWinThread-Objekte, die die Threads verkapseln, speichern können. Die Zeiger fügen Sie als Variablen in das Dokumentobjekt ein und verwenden sie, um den Rückgabewert der Funktion AfxBeginThread, mit der Sie beide Threads starten, aufzunehmen.

Um die Variablen in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie eine neue Member-Variable in die Dokumentklasse ein.

2. Legen Sie den Variablentyp mit CWinThread*, den Variablennamen als m_pSpinThread[2] und den Zugriff auf die Variable mit private fest. Auf diese Weise erhalten Sie ein zweielementiges Array für die Aufnahme der oben genannten Variablen. Sie müssen die Header-Datei bearbeiten und die Anzahl der Arraypositionen der Variablen hinzuzufügen.

Sie verfügen jetzt über einen Platz, an dem Sie die Zeiger auf die beiden Threads speichern können. Als Nächstes schreiben Sie den Code, um die Threads zu starten. Für beide Threads können Sie ein und dieselbe Funktion vorsehen. Läuft der Thread noch nicht, starten Sie ihn, ansonsten warten Sie, bis er sich selbst anhält. Die Funktion muss wissen, auf welchen Thread sich der Aufruf bezieht und ob dieser zu starten oder anzuhalten ist. Um die Funktionalität hinzuzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie eine neue Member-Funktion in die Dokumentklasse auf.

2. Legen Sie den Funktionstyp mit void und den Funktionsnamen mit SuspendSpinner fest und fügen Sie zwei Parameter ein: einen vom Typ int namens iIndex und den zweiten vom Typ BOOL namens bRun. Legen Sie den Zugriff auf die Funktion mit public fest.
3. In die Funktion schreiben Sie den Code gemäß Listing 17.11.

Listing 17.11: Die Funktion CMultitaskingDoc.SuspendSpinner

```
 1: void CMultitaskingDoc::SuspendSpinner(int iIndex, BOOL bRun)
 2: {
 3:   // Wenn der Thread angehalten wird
 4:   if (!bRun)
 5:   {
 6:     // Ist der Zeiger auf den Thread gültig?
 7:     if (m_pSpinThread[iIndex])
 8:     {
 9:       // Handle für den Thread holen
10:       HANDLE hThread = m_pSpinThread[iIndex]->m_hThread;
11:       // Auf Beenden des Threads warten
12:       ::WaitForSingleObject (hThread, INFINITE);
13:     }
14:   }
15:   else    // Thread läuft
16:   {
17:     int iSpnr;
18:     // Welcher Fächer wird verwendet?
19:     switch (iIndex)
20:     {
21:       case 0:
22:         iSpnr = 1;
23:         break;
24:       case 1:
25:         iSpnr = 3;
26:         break;
27:     }
28:     // Thread starten, Zeiger auf Fächer übergeben
29:     m_pSpinThread[iIndex] = AfxBeginThread(ThreadFunc,
30:                             (LPVOID)&m_cSpin[iSpnr]);
31:   }
32: }
```

Als Erstes prüft die Funktion, ob der Thread zu starten oder anzuhalten ist. Wird der Thread gestoppt, prüft die Funktion als Nächstes, ob der Zeiger auf den Thread gültig ist. Bei einem

gültigen Zeiger rufen Sie den Handle für den Thread ab, indem Sie den Wert der Handle-Eigenschaft der Klasse CWinThread lesen:

```
HANDLE hThread = m_pSpinThread[nIndex]->m_hThread;
```

Mit diesem Handle rufen Sie die Funktion WaitForSingleObjekt auf, um darauf zu warten, dass sich der Thread selbst stoppt.

```
::WaitForSingleObject (hThread, INFINITE);
```

Die Windows-API-Funktion WaitForSingleObject teilt dem Betriebssystem mit, dass Sie darauf warten, ob der per Handle bezeichnete Thread anhält. Das zweite Argument an diese Funktion gibt an, wie lange Sie warten wollen. Der Wert INFINITE bedeutet, dass Sie unbegrenzt lange warten wollen, bis dieser Thread stoppt. Wenn Sie eine Zeitüberschreitung spezifizieren und der Thread nicht innerhalb dieser Zeit anhält, liefert die Funktion einen Wert zurück, der angibt, ob er gestoppt hat. Da hier INFINITE als Zeitüberschreitung festgelegt ist, brauchen Sie sich nicht um den Rückgabewert der Funktion zu kümmern, da die Funktion erst dann zurückkehrt, wenn der Thread stoppt.

Wenn der Thread noch nicht läuft, bestimmt die Funktion den zu verwendenden Fächer und startet dann den Thread durch Aufruf der Funktion AfxBeginThread.

```
m_pSpinThread[nIndex] = AfxBeginThread(ThreadFunc,
                        (LPVOID)&m_cSpin[iSpnr]);
```

Als erstes Argument übergeben Sie die für den Thread aufzurufende Hauptfunktion, als zweites Argument die Adresse des Fächers, den dieser verwenden soll.

Die Threads vom Ansichtsobjekt auslösen

Sie verfügen nun über die Mechanismen, um unabhängige Threads zu starten und zu stoppen. Jetzt müssen Sie sie noch über die Kontrollkästchen im Fenster starten und anhalten können. Sind beide Kontrollkästchen eingeschaltet, starten Sie beide Threads. Wenn die Kontrollkästchen ausgeschaltet sind, müssen beide Threads gestoppt werden. Die zweite Aufgabe ist leicht: Solange die Variable, die an das Kontrollkästchen gebunden ist, mit dem Steuerelement synchronisiert ist, stoppt sich der Thread selbst, sobald Sie das Kontrollkästchen ausschalten. Wenn das Kontrollkästchen jedoch eingeschaltet ist, müssen Sie die eben erstellte Funktion des Dokumentobjekts (zum Starten der Threads) aufrufen.

Um die Funktionalität für das erste der beiden Thread-Kontrollkästchen zu realisieren, folgen Sie diesen Schritten:

1. Öffnen Sie den Hauptfensterdialog im Fenster-Designer.

2. Fügen Sie eine Behandlungsroutine für das Ereignis BN_CLICKED des Kontrollkästchens für den ersten Thread hinzu.
3. Übernehmen den Code aus Listing 17.12 in die Funktion.

Listing 17.12: Die Funktion CMultitaskingView.OnBnClickedCbthread1

```
 1: void CMultitaskingView::OnBnClickedCbthread1()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen mit dem Dialogfeld synchronisieren
 6:   UpdateData(TRUE);
 7:   // Zeiger auf das Dokument holen
 8:   CMultitaskingDoc* pDocWnd = GetDocument();
 9:   // Ist der Zeiger gültig?
10:   ASSERT_VALID(pDocWnd);
11:
12:   // Fächerthread anhalten oder starten
```

```
13:   pDocWnd->SuspendSpinner(0, m_bThread1);
14: }
```

Die Funktion ruft zuerst UpdateData auf, um die Variablen mit den Steuerelementen im Fenster zu synchronisieren. Als Nächstes wird ein Zeiger auf das Dokument abgerufen. Mit dem gültigen Zeiger rufen Sie die Funktion SuspendSpinner des Dokuments auf, wobei Sie den ersten Thread spezifizieren. Außerdem übergeben Sie den aktuellen Wert der mit dem Kontrollkästchen verbundenen Variablen, damit die Funktion weiß, ob der Thread zu starten oder zu stoppen ist.

Um die gleiche Funktionalität für das andere Thread-Kontrollkästchen zu realisieren, führen Sie die gleichen Schritte aus, fügen aber den Code aus Listing 17.13 ein.

Listing 17.13: Die Funktion CMultitaskingView.OnBnClickedCbthread2

```
 1: void CMultitaskingView::OnBnClickedCbthread2()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen mit dem Dialogfeld synchronisieren
 6:   UpdateData(TRUE);
 7:   // Zeiger auf das Dokument holen
 8:   CMultitaskingDoc* pDocWnd = GetDocument();
 9:   // Ist der Zeiger gültig?
10:   ASSERT_VALID(pDocWnd);
11:
12:   // Fächerthread starten oder anhalten
13:   pDocWnd->SuspendSpinner(1, m_bThread2);
14: }
```

Nachdem Sie nun die unabhängigen Threads starten und stoppen können, kompilieren und starten Sie die Anwendung. Überzeugen Sie sich davon, dass sich die unabhängigen Threads genau wie die OnIdle-Tasks über die betreffenden Kontrollkästchen starten und stoppen lassen.

Wenn Sie in dieser Entwicklungsphase ein wenig mit der Anwendung experimentieren, stellen Sie einen kleinen Unterschied zwischen den beiden Arten von Threads fest. Wenn alle Threads laufen und Sie die Maus ausgiebig bewegen, ist zu bemerken, dass die OnIdle- Fächer langsamer drehen (das kann schwierig festzustellen sein, da die Rechner heutzutage zu schnell sind, als dass diese Aktivität einen ausreichenden Effekt auf die OnIdle-Operationen ausüben würde). Die unabhängigen Threads beanspruchen einen guten Teil der Prozessorzeit vom Haupt-Thread der Anwendung und lassen kaum Prozessorzeit für den Leerlauf übrig. Im Ergebnis ist es einfacher, die Anwendung beschäftigt zu halten. Wenn Sie Menüs aktivieren oder das Info-Fenster öffnen, ist weiterhin festzustellen, dass zwar die OnIdle-Tasks komplett stoppen, die unabhängigen Threads aber weiterlaufen, wie es Abbildung 17.9 verdeutlicht. Die beiden Threads sind vollkommen unabhängige Prozesse innerhalb Ihrer Anwendung, sodass sie nicht vom übrigen Teil der Anwendung beeinflusst werden.

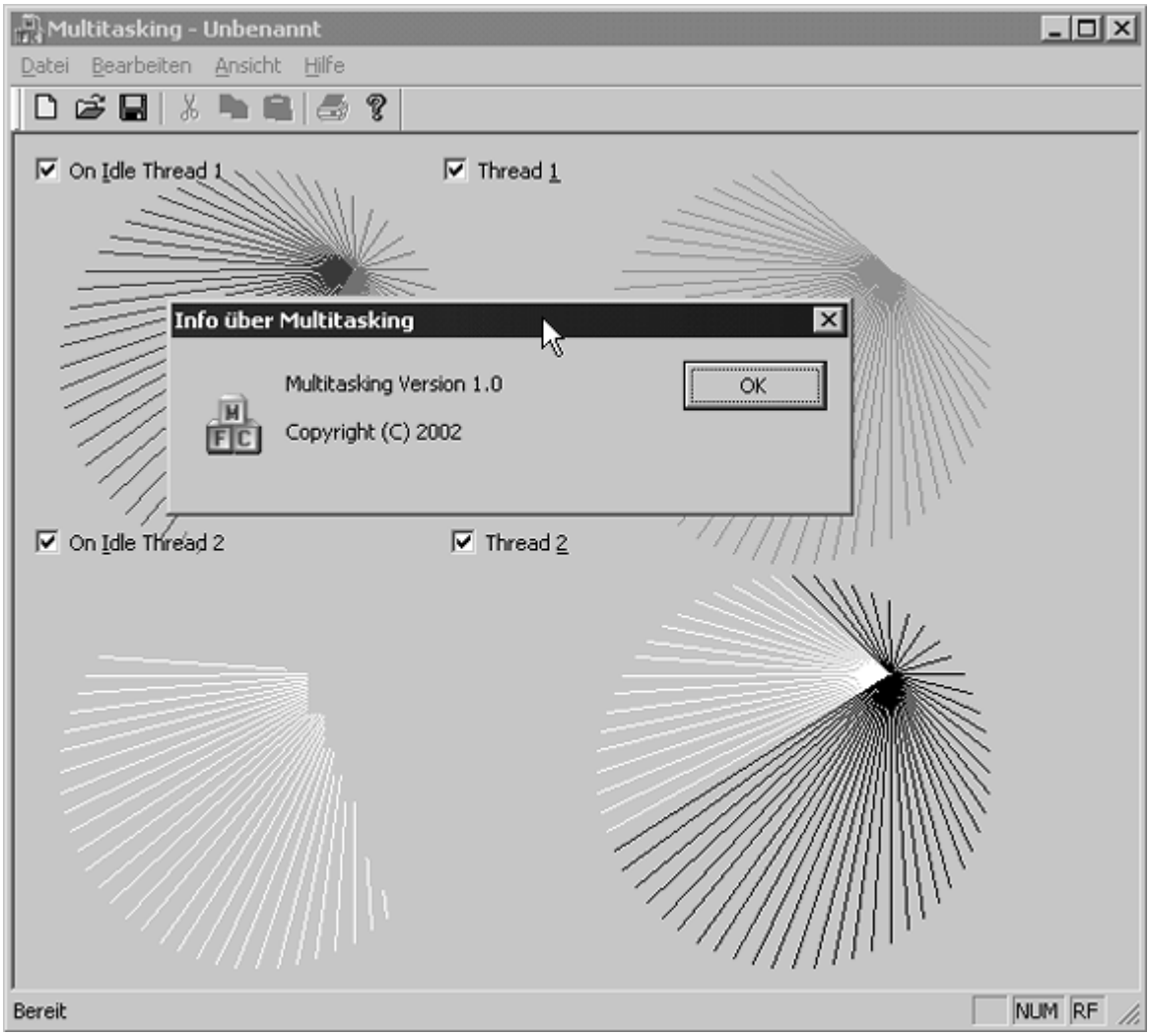


Abbildung 17.9: Die Threads werden nicht durch das Menü beeinflusst

Sauberes Herunterfahren

Vielleicht sind Sie der Meinung, dass die Anwendung fertig gestellt ist. Versuchen Sie aber nun, die Anwendung zu schließen, während ein oder beide unabhängigen Threads laufen. Leider erhalten Sie einen unwillkommenen Hinweis wie in Abbildung 17.10, weshalb Sie noch etwas unternehmen müssen. Es scheint, dass laufende Threads beim Schließen der Anwendung einen Absturz herbeiführen können.

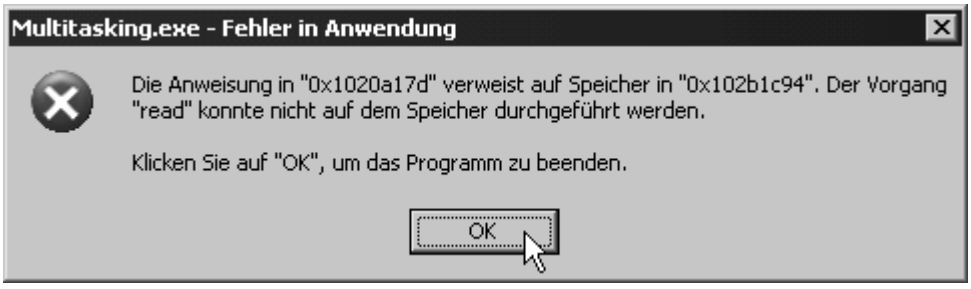


Abbildung 17.10: Fehlermeldung der Anwendung

Auch wenn Sie die Anwendung schließen, laufen die Threads unbekümmert weiter. Sobald diese den Wert der Variablen testen, die angibt, ob ein Thread weiterlaufen oder den zugehörigen Fächer drehen soll, greifen sie auf ein Speicherobjekt zu, das nicht mehr existiert. Dieses Problem verursacht einen der grundlegenden und am schwersten wiegenden Speicherfehler in einer Anwendung und Sie sollten diesen Fehler eliminieren, bevor Sie irgendjemandem die Anwendung zum Einsatz übergeben.

Um den Fehler zu unterbinden, stoppen Sie beide Threads, bevor Sie erlauben, die Anwendung zu schließen. Diese Aktion realisiert man am besten in der Behandlungsroutine für die Nachricht WM_DESTROY in der Ansichtsklasse. Mit dieser Nachricht wird der Ansichtsklasse mitgeteilt, dass vor dem Schließen der Anwendung noch alle erforderlichen Aufräumarbeiten zu erledigen sind. Mit dem Code in der Behandlungsfunktion setzen Sie die Variablen der beiden Kontrollkästchen auf FALSE, sodass sich die Threads selbst anhalten. Dann rufen Sie die Funktion SuspendSpinner für jeden Thread auf, um sich davon zu überzeugen, dass beide gestoppt haben, bevor Sie der Anwendung das Schließen erlauben. Es ist kein Aufruf von UpdateData erforderlich, um die Variablen mit den Steuerelementen zu synchronisieren, da der Benutzer nicht mehr sehen muss, wenn Sie den Wert der Kontrollkästchen ändern.

Um diese Funktionalität zu Ihrer Anwendung hinzuzufügen, folgen Sie diesen Schritten:

1. Nehmen Sie über den Modus Meldungen im Eigenschaftenfenster eine Behandlungsroutine für die Nachricht WM_DESTROY in die Ansichtsklasse Ihrer Anwendung auf. Der MFC-Anwendungs-Assistent erstellt diese Funktion normalerweise nicht für die Ansichtsklasse, weshalb Sie sie bei Bedarf in der abgeleiteten Ansichtsklasse hinzufügen müssen.

2. In die Funktion übernehmen Sie den Code von Listing 17.14.

Listing 17.14: Die Funktion CMultitaskingView.OnDestroy

```
 1: void CMultitaskingView::OnDestroy()
 2: {
 3:   CFormView::OnDestroy();
 4:
 5:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
 6:   // Läuft der erste Thread?
 7:   if (m_bThread1)
 8:   {
 9:     // Ersten Thread zum Anhalten festlegen
10:     m_bThread1 = FALSE;
11:     // Zeiger auf das Dokument holen
12:     CMultitaskingDoc* pDocWnd = GetDocument();
13:     // Ist der Zeiger gültig?
14:     ASSERT_VALID(pDocWnd);
15:
16:     // Fächerthread anhalten oder starten
17:     pDocWnd->SuspendSpinner(0, m_bThread1);
18:   }
19:   // Läuft der zweite Thread?
20:   if (m_bThread2)
21:   {
22:     // Zweiten Thread zum Anhalten festlegen
23:     m_bThread2 = FALSE;
24:     // Zeiger auf das Dokument holen
25:     CMultitaskingDoc* pDocWnd = GetDocument();
26:     // Ist der Zeiger gültig?
27:     ASSERT_VALID(pDocWnd);
28:
29:     // Fächerthread anhalten oder starten
30:     pDocWnd->SuspendSpinner(1, m_bThread2);
31:   }
32: }
```

Die Funktion führt genau das aus, was zu tun ist. Zuerst prüft sie die Variablen der Kontrollkästchen. Wenn eines der Kontrollkästchen den Wert TRUE aufweist, wird die entsprechende Variable auf FALSE gesetzt, ein Zeiger auf das Dokument ermittelt und die

Funktion SuspendSpinner für den betreffenden Thread aufgerufen. Wenn Sie nun die Anwendung schließen, stürzt sie nicht mehr ab, auch wenn gerade unabhängige Threads laufen.

17.3 Zusammenfassung

Der heutige Tag hat eine Menge Stoff gebracht. Es ging um die verschiedenen Möglichkeiten, wie man mehrere Aufgaben (so genannte Tasks) mit Anwendungen gleichzeitig ausführen kann. Sie haben wichtige Punkte kennen gelernt, die zu beachten sind, wenn man die Anwendungen mit Multitasking-Fähigkeiten ausstatten möchte. Es wurde gezeigt, wie sich Tasks im Leerlauf einer Anwendung ausführen lassen. In diesem Zusammenhang wurden auch die Einschränkungen und Nachteile dieser Lösung behandelt. Weiterhin haben Sie gelernt, wie man unabhängige Threads in einer Anwendung erstellt. Die Threads arbeiten vollkommen unabhängig von der übrigen Anwendung. In der heutigen Beispielanwendung haben Sie die beiden Lösungsansätze verwendet, sodass Sie damit Erfahrung sammeln können.

Wenn Sie in Ihre Anwendungen die Fähigkeiten zum Multitasking einbauen wollen, sollten Sie sich darüber im Klaren sein, dass diese Aufgaben zu den komplizierteren Aspekten der Windows-Programmierung gehören. Sie müssen eine ganze Reihe von Faktoren verstehen und weit mehr in Betracht ziehen, als wir an einem einzigen Tag behandeln konnten. Wenn Sie Multitasking-Anwendungen erstellen möchten, sollten Sie sich mit einschlägigen Büchern zur Programmierung von Windows-Anwendungen mit MFC oder Visual C++ beschäftigen. Diese Bücher sollen profundes Wissen zum Multithreading mit MFC vermitteln und alle Synchronisierungsklassen detaillierter behandeln, als es an dieser Stelle möglich war. Denken Sie daran, dass Sie ein Buch brauchen, das sich auf MFC konzentriert und nicht auf die Entwicklungsumgebung von Visual C++. (MFC wird von den meisten kommerziellen C++-Entwicklungswerkzeugen für Windows-Anwendungen unterstützt. Dazu gehören auch die Compiler von Borland und Symantec. Eine Behandlung dieser Themen geht also über die Umgebung von Visual C++ hinaus.)

17.4 Workshop

Fragen und Antworten

Frage:
Wie kann ich die andere Version von AfxBeginThread einsetzen, um einen Thread in einer benutzerdefinierten Klasse zu verkapseln?

Antwort:
Erstens dient die andere Version von AfxBeginThread hauptsächlich dazu, Threads für Benutzeroberflächen zu erzeugen. Die in der heutigen Beispielanwendung eingesetzte Version ist für so genannte Worker-Threads vorgesehen, die sofort mit einem bestimmten Task beginnen. Wenn Sie einen Thread für eine Benutzeroberfläche erzeugen wollen, müssen Sie Ihre benutzerdefinierte Klasse von der Klasse CWinThread ableiten. Als Nächstes sind mehrere Vorgängerfunktionen in der benutzerdefinierten Klasse zu überschreiben. Nachdem die Klasse für den Einsatz vorbereitet ist, holen Sie mit dem Makro RUNTIME_CLASS einen Zeiger auf die Laufzeitklasse Ihrer Klasse und übergeben diesen Zeiger folgendermaßen an die Funktion AfxBeginThread:

```
CWinThread* pMyThread = AfxBeginThread(RUNTIME_CLASS(CMyThreadClass));
```

Frage:
Kann ich mit SuspendThread und ResumeThread meine unabhängigen Threads in der Beispielanwendung starten und stoppen?

Antwort:
Ja, es sind aber ein paar wesentliche Änderungen an der Anwendung erforderlich. Erstens müssen Sie in der

Funktion OnNewDocument die beiden Thread-Zeiger mit NULL initialisieren, wie es Listing 17.15 zeigt.

Listing 17.15: Die modifizierte Funktion CMultitaskingDoc.OnNewDocument

```
 1: BOOL CMultitaskingDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Fächer initialisieren
 9:   InitSpinners();
10:
11:   // Thread-Zeiger initialisieren
12:   m_pSpinThread[0] = NULL;
13:   m_pSpinThread[1] = NULL;
14:
15:   return TRUE;
16: }
```

Antwort:
Als Nächstes modifizieren Sie die Thread-Funktion gemäß Listing 17.16, damit sich der Thread nicht selbst anhält, sondern weiter in der Schleife läuft, wenn die Variable des Kontrollkästchens den Wert FALSE aufweist.

Listing 17.16: Die modifizierte Funktion CMultitaskingDoc.TreadFunc

```
 1: UINT CMultitaskingDoc::ThreadFunc(LPVOID pParam)
 2: {
 3:   // Argument in Zeiger auf den Fächer für diesen Thread
 4:   // umwandeln
 5:   CSpinner* lpSpin = (CSpinner*)pParam;
 6:   // Zeiger auf das Fortfahren-Flag holen
 7:   BOOL* pbContinue = lpSpin->GetContinue();
 8:
 9:   // Schleife solange Flag wahr ist
10:   while (TRUE)
11:     // Fächer drehen
12:     lpSpin->Draw();
13:   return 0;
14: }
```

Antwort:
Schließlich ist die Funktion SuspendSpinner entsprechend Listing 17.17 zu modifizieren, sodass sie bei einem gültigen Thread-Zeiger die Funktion SuspendThread auf dem Thread-Zeiger aufruft, um den Thread zu stoppen, und die Funktion ResumeThread, um ihn erneut zu starten.

Listing 17.17: Die modifizierte Funktion CMultitaskingDoc.SuspendSpinner

```
 1: void CMultitaskingDoc::SuspendSpinner(int iIndex, BOOL bRun)
 2: {
 3:   // Wenn der Thread angehalten wird
 4:   if (!bRun)
 5:   {
 6:     // Ist der Zeiger auf den Thread gültig?
 7:     if (m_pSpinThread[iIndex])
 8:     {
 9:       // Thread anhalten
10:       m_pSpinThread[iIndex]->SuspendThread();
11:     }
12:   }
```

```
13:   else    // Thread läuft
14:   {
15:     // Ist der Zeiger auf den Thread gültig?
16:     if (m_pSpinThread[iIndex])
17:     {
18:       // Thread wieder aufnehmen
19:       m_pSpinThread[iIndex]->ResumeThread();
20:     }
21:     else
22:     {
23:       int iSpnr;
24:       // Welcher Fächer soll verwendet werden?
25:       switch (iIndex)
26:       {
27:         case 0:
28:           iSpnr = 1;
29:           break;
30:         case 1:
31:           iSpnr = 3;
32:           break;
33:       }
34:       // Thread starten, Zeiger auf Fächer übergeben
35:       m_pSpinThread[iIndex] = AfxBeginThread(ThreadFunc,
36:                                (LPVOID)&m_cSpin[iSpnr]);
37:     }
38:   }
39: }
```

Quiz

1. Wann wird die Funktion OnIdle aufgerufen?

2. Wie kann man den wiederholten Aufruf der Funktion OnIdle bewirken, während sich die Anwendung im Leerlauf befindet?
3. Worin besteht der Unterschied zwischen einer OnIdle-Task und einem Thread?
4. Wie heißen die vier Objekte zur Synchronisierung von Threads?
5. Warum sollten Sie keine höhere als die normale Priorität für die Threads in Ihrer Anwendung festlegen?

Übungen

1. Wenn Sie die heutige Beispielanwendung ausführen und die Auslastung Ihres Computers mit einem Systemmonitor überwachen, zeigt sich, dass die Prozessorauslastung sogar ohne laufende Threads bei 100 Prozent liegt (siehe Abbildung 17.11). Die Funktion OnIdle wird fortlaufend aufgerufen, selbst wenn nichts zu tun ist.

   Modifizieren Sie die Funktion OnIdle, damit keine der OnIdle-Tasks aktiv ist, wenn es nichts zu tun gibt. Somit wird die Funktion OnIdle erst dann wieder fortlaufend aufgerufen, wenn einer der Threads aktiv wird. Dieser Zustand dauert nur so lange an, bis beide Threads wieder ausgeschaltet sind. Der Prozessor kann dann zu einer minimalen Auslastung zurückkehren (siehe Abbildung 17.12).

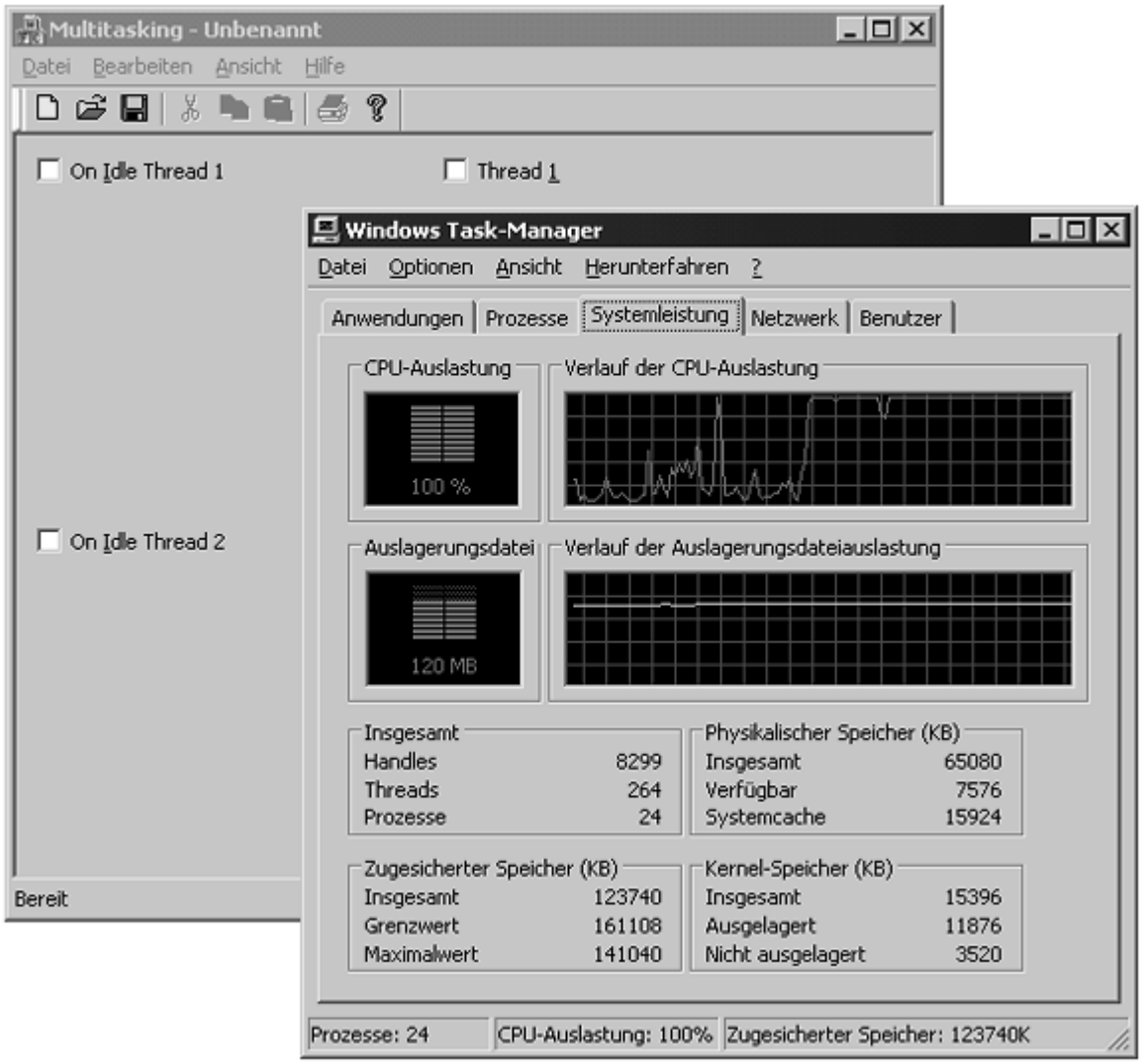


Abbildung 17.11: Prozessorauslastung von 100%

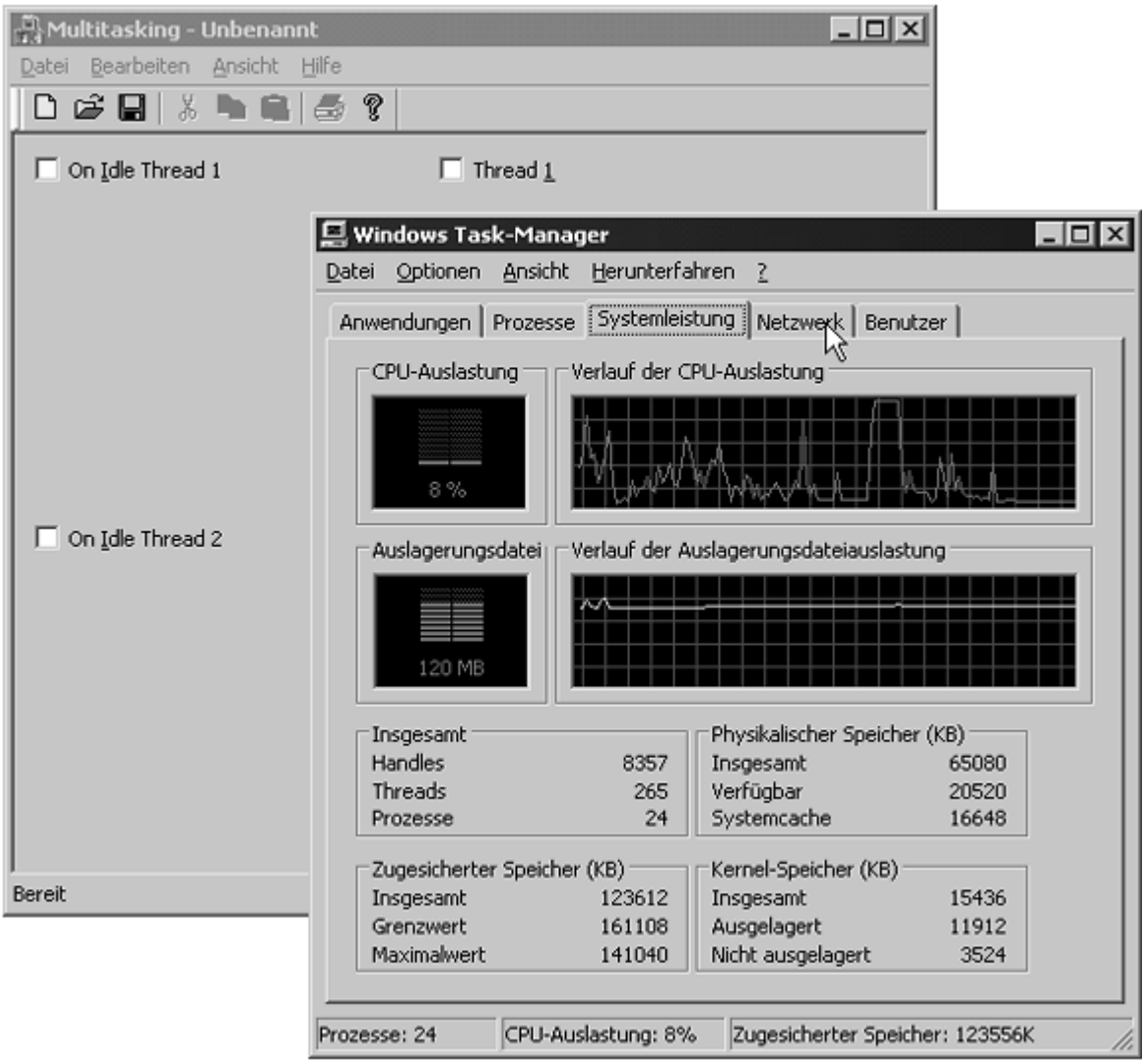


Abbildung 17.12: Prozessorauslastung auf normalem Level

2. Geben Sie beim Starten der unabhängigen Threads einem der Threads die Priorität THREAD_PRIORITY_NORMAL und dem anderen die Priorität THREAD_PRIORITY_LOWEST.

Tag 18

Internet und Netzwerke

Immer mehr Anwendungen besitzen die Fähigkeit, mit anderen Anwendungen über Netzwerke einschließlich des Internets zu kommunizieren. Das ist nicht zuletzt auf die explosionsartige Zunahme der Popularität des Internets zurückzuführen. Microsoft hat mit Windows NT und Windows 95 erstmals die Netzwerkfähigkeiten in seine Betriebssysteme integriert und damit zu einer allgemeinen Verbreitung in allen Arten von Anwendungen beigetragen.

Einige Anwendungen führen einfache Netzwerkaufgaben aus. Beispielsweise prüfen sie eine Website, ob irgendwelche Updates für das Programm vorhanden sind, und bieten dann dem Benutzer die Option, das Programm mit diesen Updates auf den neuesten Stand zu bringen. Verschiedene Textverarbeitungen formatieren Dokumente als Webseiten und geben damit dem Benutzer die Möglichkeit, die Seiten auf den Webserver zu laden. Es gibt auch Computerspiele, bei denen Benutzer auf der ganzen Welt gegeneinander spielen können und nicht nur für sich selbst am und gegen den Computer werkeln müssen.

Anwendungen können eine Vielzahl von Netzwerkfunktionen aufweisen und sie sind alle um die Winsock-Schnittstelle herum aufgebaut. Wenn Sie sich mit der Winsock- Schnittstelle auskennen und wissen, wie man Programme mit ihr und den MFC-Winsock- Klassen erstellt, eröffnet sich Ihnen dieses wunderbare Reich der Anwendungsprogrammierung, womit Sie Ihre Programmiermöglichkeiten beträchtlich erweitern können.

Heute lernen Sie, ...

- wie Anwendungen mit der Winsock-Schnittstelle arbeiten, um die Netzwerkverbindungen zwischen zwei oder mehr Computern zu realisieren,
- worin der Unterschied zwischen einer Client- und einer Server-Anwendung liegt und welche Rolle jede beim Einrichten einer Kommunikationsverbindung spielt,
- wie die MFC-Winsock-Klassen das Schreiben von Internet-Anwendungen vereinfachen,
- wie man eine eigene Winsock-Klasse von den MFC-Winsock-Klassen ableitet und damit eine ereignisgesteuerte Netzwerkanwendung erstellt.

18.1 Wie funktionieren Netzwerkübertragungen?

Die meisten Anwendungen, die über ein Netzwerk kommunizieren, ob es sich nun um das Internet oder ein kleines lokales Netzwerk handelt, arbeiten nach den gleichen Prinzipien und bauen auf der gleichen Funktionalität auf, um die Datenübertragung zu realisieren. Eine Anwendung wartet auf dem einen Computer darauf, dass eine andere Anwendung eine Verbindung öffnet. Die erste Anwendung »lauscht« (daher Listener) auf diese Verbindungsanforderung, genau wie man auf das Klingeln des Telefons hört, wenn man einen Anruf erwartet.

In der Zwischenzeit versucht eine andere Anwendung, höchstwahrscheinlich (aber nicht notwendigerweise) auf einem anderen Computer, sich mit der ersten Anwendung zu verbinden. Dieser Versuch, eine Verbindung zu öffnen, entspricht dem Anruf eines anderen Telefonteilnehmers. Man wählt die Nummer und hofft, dass der Empfänger das Telefonklingeln am anderen Ende der Leitung hört. Als Anrufender müssen Sie die Rufnummer des Angerufenen kennen. Wenn Sie die Nummer nicht wissen, können Sie sie im Telefonbuch anhand des Namens ermitteln. Analog dazu muss die Anwendung, die eine Verbindung zur ersten Anwendung herzustellen versucht, den Netzwerkstandort - oder die Adresse - der ersten Anwendung kennen.

Nachdem eine Verbindung zwischen den beiden Anwendungen hergestellt ist, lassen sich Nachrichten in beiden Richtungen zwischen ihnen austauschen - auch hier wieder die Analogie zu einem Telefongespräch. Diese Verbindung stellt einen bidirektionalen Kommunikationskanal dar, bei dem beide Seiten Informationen senden können (siehe Abbildung 18.1).

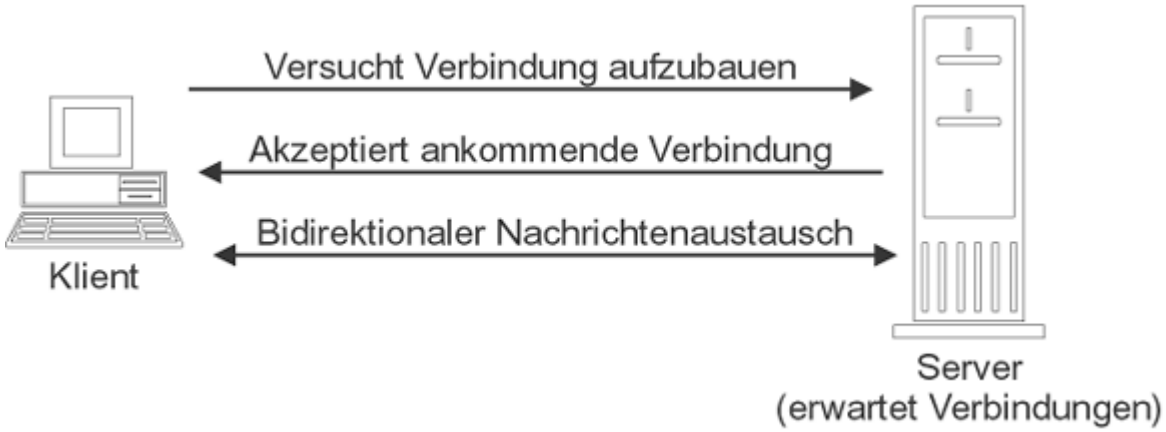


Abbildung 18.1: Der grundlegende Socket-Verbindungsprozess

Schließlich beendet eine Seite (oder beide Seiten) die Konversation. Die Verbindung wird geschlossen, so wie Sie den Telefonhörer auflegen, wenn Sie das Gespräch beenden. Nachdem die Verbindung von einer der beiden Seiten geschlossen wurde, erkennt die andere Seite diesen Zustand und schließt ihrerseits die Verbindung, so wie Sie feststellen, dass der andere Telefonteilnehmer aufgelegt hat oder wenn Sie aus anderen Gründen unterbrochen wurden. Grundsätzlich laufen Netzwerkverbindungen zwischen zwei oder mehr Anwendungen nach diesem stark vereinfacht dargestellten Schema ab.

Diese grundlegende Beschreibung bezieht sich auf Netzwerkverbindungen, die nach dem TCP/IP-Protokoll arbeiten, das hauptsächlich als Netzwerkprotokoll im Internet zum Einsatz kommt. Viele andere Netzwerkprotokolle weichen etwas von dieser Beschreibung ab. TCP/IP besteht aus einer ganzen Reihe verschiedener Protokolle. Die Verbindungen, die wir in diesem Kapitel aufbauen werden, laufen mit dem TCP-Protokoll. Andere Protokolle wie etwa das UDP-Protokoll verhalten sich eher wie Rundfunkübertragungen, wo es keine Verbindung zwischen den beiden Anwendungen gibt. Die eine Anwendung sendet Nachrichten und die andere ist dafür verantwortlich, den Empfang aller dieser Nachrichten sicherzustellen. Neben diesen beiden zur Datenübertragung verwendeten Protokollen gibt es noch Protokolle zur Kontrolle des Netzwerkes selber wie etwa ICMP oder ARP. Diesen Protokollen können wir uns im Rahmen der heutigen Lektion allerdings nicht widmen. Wenn Sie mehr über Netzwerkprotokolle und deren Arbeitsweise erfahren möchten, sollten Sie sich mit der einschlägigen Literatur beschäftigen. Gerade zum Thema Kommunikation gibt es zahlreiche Bücher, die sich auch mit den verschiedenartigen Internet-Anwendungen befassen und die Kommunikation über die eingerichteten Verbindungen beschreiben.

Sockets, Ports und Adressen

Das grundlegende Objekt, über das eine Anwendung den größten Teil der Netzwerkkommunikation realisiert, bezeichnet man als Socket. Sockets wurden zuerst unter Unix an der Berkeley-Universität entwickelt. Ziel war es, dass Anwendungen den Hauptteil der Netzwerkkommunikation zwischen Anwendungen in der gleichen Weise abwickeln können wie

das Lesen und Schreiben von Dateien. Seit dieser Zeit haben sich Sockets natürlich etwas verändert, wobei aber die grundsätzliche Arbeitsweise gleich geblieben ist.

Während der Tage von Windows 3.x, bevor die Netzwerke in das Betriebssystem Windows integriert wurden, konnte man Netzwerkprotokolle für die Datenübertragung von zahlreichen Anbietern erwerben. Jede dieser Firmen ging etwas anders an die Netzwerkübertragung zwischen Anwendungen heran. Im Ergebnis musste jede Anwendung, die Netzwerkverbindungen realisieren wollte, eine Liste von unterschiedlichen Netzwerkprogrammen beachten, mit denen die Anwendung arbeiten konnte. Viele Anwendungsentwickler waren nicht gerade glücklich mit dieser Situation. Das führte letztendlich dazu, dass die Netzwerkfirmen - einschließlich Microsoft - zusammenkamen und die Winsock-API (Windows Sockets) entwickelten, die Vorstellung der Schnittstelle erfolgte im Januar 1992. Damit steht allen Anwendungsentwicklern eine einheitliche Programmierschnittstelle (API - Anwendungsprogrammierschnittstelle) zur Verfügung, die unabhängig von der verwendeten Netzwerksoftware ist.

Wenn man eine Datei lesen oder schreiben will, verwendet man ein Dateiobjekt, das auf die Datei zeigt. Obwohl dies in den meisten Visual C++-Anwendungen, die Sie bisher erstellt haben, vor Ihnen verborgen geblieben ist, mussten Sie für das vor ein paar Tagen erzeugte ActiveX-Steuerelement diese Schritte beim Anlegen des Dateiobjekts zum Lesen und Schreiben durcharbeiten. Ein Socket ist damit vergleichbar. Es handelt sich um ein Objekt, das man zum Lesen und Schreiben von Nachrichten verwendet, die zwischen den Anwendungen reisen.

Um eine Socket-Verbindung zu einer anderen Anwendung herzustellen, sind andere Informationen erforderlich als beim Öffnen einer Datei. Wenn man eine Datei öffnen möchte, muß man den Dateinamen und den Standort der Datei kennen. Bei einer Socket- Verbindung muss man wissen, auf welchem Computer die andere Anwendung läuft und über welchen Port die Anwendung lauscht. Ein Port entspricht etwa einem Nebenstellenanschluss, während die Computeradresse mit der Telefonnummer der Telefonzentrale vergleichbar ist. In gleicher Weise werden Ports benutzt, um Netzwerkübertragungen weiterzuleiten (siehe Abbildung 18.2). Wie für eine Telefonnummer gibt es auch Mittel, um die Portnummer zu ermitteln, wenn man sie nicht bereits kennt. Das erfordert aber, dass der Computer mit den Angaben konfiguriert ist, über welchen Port die verbindende Anwendung lauscht. Wenn man die falsche Computeradresse - oder Portnummer - spezifiziert, erhält man gegebenenfalls eine Verbindung zu einer anderen Anwendung. Das entspricht dem vom Telefon bekannten »falsch verbunden«. Vielleicht erhalten Sie aber auch überhaupt keine Antwort, wenn es am anderen Ende der Leitung keine Anwendung gibt, die lauscht.

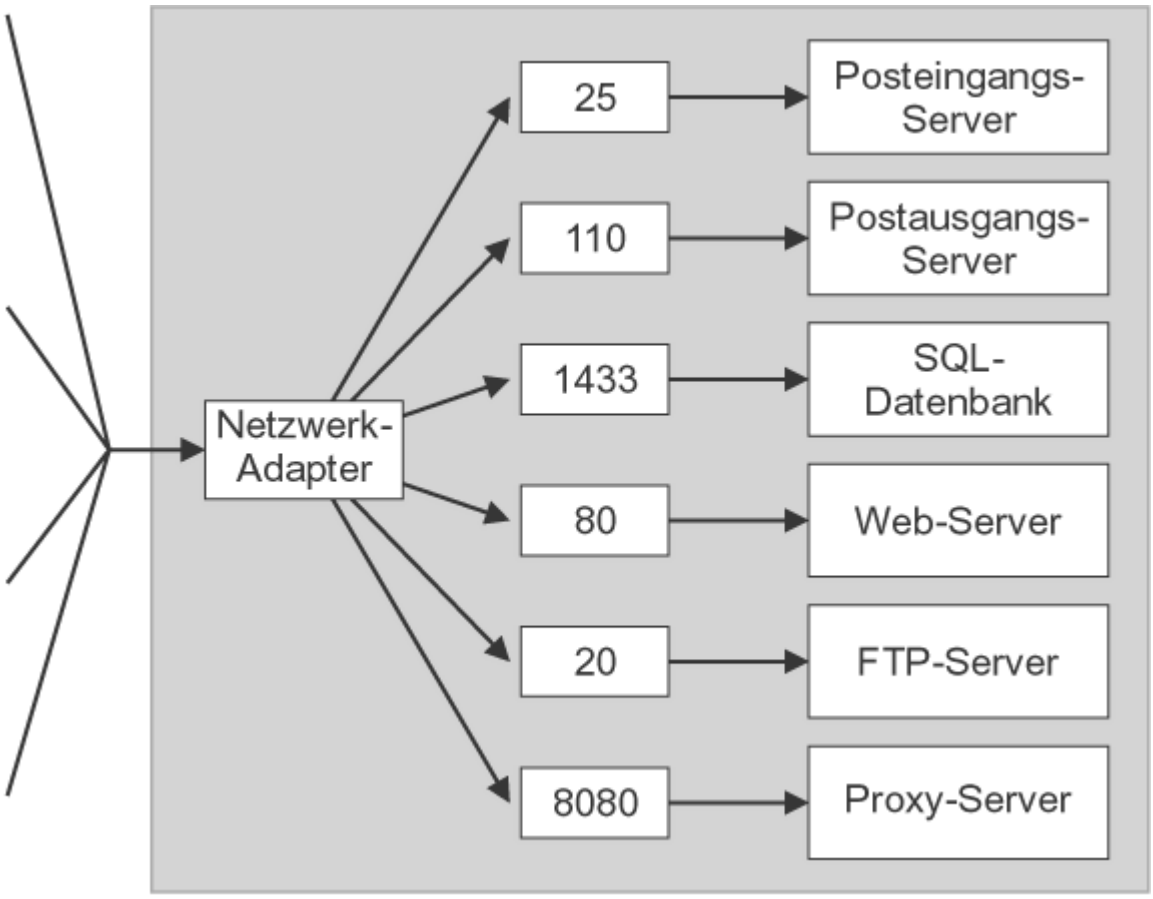


Abbildung 18.2: Portangaben dienen dazu, die Netzwerkverbindungen zur richtigen Anwendung weiterzuleiten.

Nur eine Anwendung kann auf einem einzelnen Computer an einem bestimmten Port lauschen. Obwohl auch mehrere Anwendungen auf ein und demselben Computer gleichzeitig auf Verbindungsgesuche warten können, müssen diese Anwendungen dazu verschiedene Ports verwenden. Bekannte Portnummern sind etwa 80 für Webserver oder 25 für Mailserver.

Die Winsock-Umgebung initialisieren

Bevor Sie Winsock-MFC-Klassen verwenden können, müssen Sie die Winsock-Umgebung für Ihre Anwendung initialisieren. Das geschieht durch einen einzigen Funktionsaufruf in der Initialisierung der Anwendungsinstanz und zwar von AfxSocketInit. Diese Funktion kann eine WSADATA-Struktur als optionalen Parameter übernehmen. Wenn Sie diese Struktur an die Funktion übergeben, wird sie mit Informationen über die derzeit auf dem Computer, auf dem Ihre Anwendung läuft, verwendete Winsock-Version ausgefüllt. Wenn Sie keine der in dieser Struktur enthaltenen Informationen benötigen, müssen Sie sie nicht als Parameter übergeben:

```
BOOL CSockApp::InitInstance()
{
  if (!AfxSocketInit())
  {
    AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
    return FALSE;
  }
...
}
```

Wenn Sie diese Funktion in die Initialisierungsfunktion der Instanz einbinden, wird die Winsock-Umgebung durch Ihre Anwendung korrekt initialisiert und heruntergefahren.

Wenn Sie Ihr Projektgerüst mit dem Visual C++-Assistenten erstellen und dabei die Unterstützung von Winsock mit aufnehmen, wird die Funktion AfxSocketInit Ihrem Anwendungsgerüst automatisch hinzugefügt.

Einen Socket erstellen

Wenn Sie Anwendungen mit Visual C++ erstellen, können Sie auf die MFC-Winsock- Klassen zurückgreifen, um die Fähigkeiten zur Netzwerkübertragung ziemlich einfach zu realisieren. Die Basisklasse CAsyncSocket stellt vollständige, ereignisgesteuerte Verbindungen bereit. Sie können Ihre eigene abgeleitete Socket-Klasse erstellen, die jedes dieser Ereignisse abfangen und darauf antworten kann. Die Klasse CSocket ist aus der Klasse CAsyncSocket abgeleitet und verkapselt und vereinfacht einen Teil der Funktionalität ihrer Basisklasse. Die Klasse CSocket wird in erster Linie verwendet, wenn Sie einen blockierenden Socket benötigen, bei dem die Socket-Funktionen keine Ereignisse auslösen, sondern vor der Rückgabe der Steuerung auf die Abarbeitung eines jeden Funktionsaufrufs warten.

Um einen Socket zu erstellen, den Sie in Ihrer Anwendung einsetzen können, müssen Sie zunächst eine Variable von CAsyncSocket (oder Ihrer davon abgeleiteten Klasse) als Klassenelement für eine der Hauptanwendungsklassen deklarieren:

```
class CMyDlg : public CDialog
{
...
private:
    CAsyncSocket m_sMySocket;
};
```

Bevor Sie das Socket-Objekt verwenden können, müssen Sie dessen Create-Methode aufrufen. Diese Methode erstellt den eigentlichen Socket und bereitet ihn zum Einsatz vor. Wie Sie die Methode Create aufrufen, hängt von der Verwendungsart des Sockets ab. Wollen Sie - als Client - einen Ruf an eine andere Anwendung absetzen, um eine Verbindung zu dieser Anwendung herzustellen, dann brauchen Sie der Create-Methode keinerlei Parameter zu übergeben:

```
if (m_sMySocket.Create())
{
  // Fortfahren
}
else
  // Hier Fehlerbehandlung durchführen
```

Wenn allerdings der Socket auf den Verbindungswunsch einer anderen Anwendung lauschen soll, um sich mit dieser Anwendung zu verbinden, und dabei - als Server - auf den Ruf wartet, dann müssen Sie zumindest die Nummer des Ports übergeben, auf dem der Socket lauschen soll:

```
if (m_sMySocket.Create(4000))
{
  // Fortfahren
}
else
  // Hier Fehlerbehandlung durchführen
```

In den Aufruf der Create-Methode können Sie weitere Parameter einbinden, wie etwa den Typ des zu erzeugenden Sockets, die Ereignisse, auf die der Socket reagieren soll, und die Adresse, die der Socket abhören soll (falls der Computer über mehrere Netzwerkkarten verfügt).

Sie können mit den MFC-Winsock-Klassen zwei Arten von Sockets erzeugen: streaming (oder TCP) und datagram (oder UDP). Streaming-Sockets sind verbindungsbasiert und besitzen eine eingebaute Garantie, dass die Pakete abgeliefert werden. Datagram-Sockets sind verbindungslos und Sie müssen den Code schreiben, der sicherstellt, dass die Pakete empfangen und in der empfangenden Anwendung in der richtigen Reihenfolge (in der sie gesendet wurden) angeordnet werden. Wird ein bestimmtes Paket nicht empfangen, müssen Sie auch den Code in der empfangenden Anwendung schreiben, der ein erneutes Senden dieses Pakets anfordert, und den Code in der sendenden Anwendung, der das fehlende Datenpaket noch einmal abschickt. Um einen Streaming-Socket zu erzeugen, übergeben Sie an die Create-Methode als zweites Argument SOCK_STREAM. Um einen Datagram-Socket zu erzeugen, übergeben Sie stattdessen SOCK_DGRAM als zweites Argument.

Wenn Sie eine Server-Anwendung erstellen, die möglicherweise auf einem Computer mit mehr als einer Netzwerkkarte läuft, müssen Sie eventuell die Netzwerkadresse angeben, auf der der Socket lauscht. Damit teilen Sie dem Socket mit, dass er nur über eine bestimmte Netzwerkkarte auf hereinkommende Verbindungsanfragen lauscht. Dazu übergeben Sie die

Netzwerkadresse, an die der Socket gebunden werden soll, als letztes Argument an die Create-Methode. In der Klasse CAsyncSocket ist dies das vierte Argument, in der Klasse CSocket das dritte. Die Netzwerkadresse sollte als String im Standard-TCP/IP-Format übergeben werden:

```
"127.0.0.1"
```

Eine Verbindung herstellen

Nachdem Sie einen Socket erstellt haben, können Sie damit eine Verbindung öffnen. Das Öffnen einer einzelnen Verbindung verläuft in drei Schritten. Zwei davon finden auf dem Server - der Anwendung, die auf die Verbindung prüft - statt und der dritte Schritt läuft auf dem Client ab, der den Anruf ausführt.

Auf der Client-Seite ist einfach die Methode Connect aufzurufen. Der Client muss zwei Parameter an die Methode übergeben: den Computernamen oder die Netzwerkadresse und den Port der Anwendung, mit der die Verbindung herzustellen ist. Die Methode Connect lässt sich auf folgende zwei Arten verwenden:

- Verbindung über Computernamen herstellen

```
if (m_sMySocket.Connect("thatcomputer.com", 4000))
{
  // Fortfahren
}
else
{
  // Hier Fehlerbehandlung durchführen
}
```

- Verbindung über IP-Adresse herstellen

```
if (m_sMySocket.Connect("127.0.0.1", 4000))
{
  // Fortfahren
}
else
{
  // Hier Fehlerbehandlung durchführen
}
```

Nachdem die Verbindung hergestellt ist, wird ein Ereignis ausgelöst, um Ihre Anwendung wissen zu lassen, dass sie verbunden ist oder dass es Probleme gegeben hat und sich die Verbindung (temporär oder dauerhaft) nicht einrichten lässt. (Auf diese Ereignisse geht der Abschnitt »Socket-Ereignisse« weiter hinten in dieser Lektion näher ein.) Wenn Sie die Klasse CSocket verwenden, kehrt die Funktion Connect nicht zurück, bevor die Verbindung hergestellt ist oder ein Fehler aufgetreten ist, der den Aufbau der Verbindung verhindert.

Auf der Server-Seite (der lauschenden Seite) der Verbindung muss die Anwendung zuerst den Socket anweisen, hereinkommende Verbindungen zu prüfen. Dazu ist die Methode Listen aufzurufen. Die Methode Listen übernimmt nur ein einziges Argument, das Sie nicht bereitstellen müssen. Dieser Parameter legt die Anzahl der anhängigen Verbindungen fest, die in eine Warteschlange eingereiht werden können und auf den Abschluss der Verbindung warten. Per Vorgabe ist dieser Wert 5, der das Maximum darstellt. Die Methode Listen lässt sich folgendermaßen aufrufen:

```
if (m_sMySocket.Listen())
{
  // Fortfahren
}
else
{
  // Hier Fehlerbehandlung durchführen
}
```

Immer dann, wenn eine andere Anwendung versucht, sich mit der lauschenden Anwendung in Verbindung zu setzen, wird ein Ereignis ausgelöst, um der Anwendung mitzuteilen, dass eine Verbindungsanforderung vorliegt. Die lauschende Anwendung muss die Verbindungsanforderung durch Aufruf der Methode Accept entgegennehmen. Diese Methode erfordert eine zweite CAsyncSocket-Variable, die mit der anderen Anwendung verbunden ist. Nachdem ein Socket in den Lauschmodus gebracht wurde, bleibt er in diesem Modus. Immer wenn ein Verbindungsgesuch empfangen wird, erzeugt der lauschende Socket einen weiteren Socket, der mit der anderen Anwendung verbunden wird. Für diesen zweiten Socket sollte man nicht die Methode Create aufrufen, da die Methode Accept den Socket erstellt. Die Methode Accept lässt sich folgendermaßen aufrufen:

```
if (m_sMySocket.Accept(m_sMySecondSocket))
{
  // Fortfahren
}
else
{
  // Hier Fehlerbehandlung durchführen
}
```

Zu diesem Zeitpunkt ist die verbindende Anwendung mit dem zweiten Socket der lauschenden Anwendung verbunden.

Bei der Klasse CSocket werden hereinkommende Verbindungen wie oben über die Funktion Accept erkannt und entgegengenommen. Allerdings kehrt die Funktion Accept hier nicht zurück, bevor ein Verbindungsgesuch empfangen und entgegengenommen wurde.

Nachrichten senden und empfangen

Das Senden und Empfangen von Nachrichten über eine Socket-Verbindung ist nicht ganz trivial. Da man Sockets einsetzen kann, um alle Arten von Daten zu senden, erwarten die Funktionen zum Senden und Empfangen von Daten einen Zeiger auf einen generischen Puffer. Zum Senden von Daten sollte dieser Puffer die zu sendenden Daten enthalten. Beim Empfangen von Daten werden die empfangenen Daten in diesen Puffer kopiert. Solange Sie Strings und Text senden und empfangen, können Sie ziemlich einfache Umwandlungen der Pufferinhalte über CStrings vornehmen.

Um eine Nachricht über eine Socket-Verbindung zu versenden, rufen Sie die Methode Send auf. Diese Methode erfordert zwei Parameter und besitzt einen dritten optionalen Parameter, mit denen man steuern kann, wie die Nachricht zu versenden ist. Der erste Parameter ist ein Zeiger auf den Puffer, der die zu sendenden Daten enthält. Wenn Ihre Nachricht in einer CString-Variablen steht, können Sie die CString-Variable mithilfe des Operators LPCTSTR als Puffer übergeben. Der zweite Parameter gibt die Länge des Puffers an. Die Methode liefert die Menge der Daten zurück, die an die andere Anwendung gesendet wurden. Wenn ein Fehler auftritt, liefert die Funktion Send den Wert SOCKET_ERROR zurück. Die Methode Send rufen Sie folgendermaßen auf:

```
CString strMyMessage;
int iLen;
int iAmtSent;
...
iLen = strMyMessage.GetLength();
iAmtSent = m_sMySocket.Send(LPCTSTR(strMyMessage), iLen);
if (iAmtSent == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  // Alles OK
}
```

Wenn Sie mit UNICODE-Strings arbeiten, gibt die Funktion GetLength die Länge des Strings selbst zurück, aber dabei handelt es sich nicht um die Länge der zu sendenden Daten. Da jedes Zeichen nicht durch ein Byte, sondern durch zwei dargestellt wird, muss die Stringlänge verdoppelt werden, um alle zu sendenden Daten aufnehmen zu können. Um die Daten maximal portierbar zu halten, sollten Sie sie vor dem Versenden in Standard-ASCII-Text umwandeln. Zu diesem Thema liefert Microsoft einen ausfühlichen Artikel in der Online-Hilfe: Multibyte Character Set (MBCS) Survival Guide von Chau Vu, Seiichi Satoh, and Matt Grove.

Wenn von einer anderen Anwendung empfangene Daten verfügbar sind, wird in der empfangenden Anwendung ein Ereignis für die Klasse AsyncSocket und ihre Abkömmlinge ausgelöst. Damit wird der Anwendung mitgeteilt, dass sie die Nachricht empfangen und verarbeiten kann. Um die Nachricht zu erhalten, ist die Methode Receive aufzurufen. Diese Methode übernimmt die gleichen Parameter wie die Methode Send, mit einem kleinen Unterschied. Der erste Parameter ist ein Zeiger auf einen Puffer, in den die Nachricht kopiert werden kann. Der zweite Parameter gibt die Größe des Puffers an. Aus diesem Wert weiß der Socket, wie viele Daten zu kopieren sind (falls mehr empfangen wurden, als in den Puffer passen). Wie bei der Methode Send liefert die Methode Receive die Menge der in den Puffer kopierten Daten zurück. Wenn ein Fehler auftritt, gibt die Methode ebenfalls den Wert SOCKET_ERROR zurück. Wenn die Nachricht, die Ihre Anwendung empfängt, eine Textnachricht ist, können Sie sie direkt in eine CString- Variable kopieren. Damit lässt sich die Methode Receive wie folgt aufrufen:

```
char *pBuf = new char[1025];
int iBufSize = 1024;
int iRcvd;
CString strRecvd;
iRcvd = m_sMySocket.Receive(pBuf, iBufSize);
if (iRcvd == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  // den gelesenen String mit einen Nullzeichen abschliessen
  pBuf[iRcvd] = '\0';
  strRecvd = pBuf;
  // Nachricht weiterverarbeiten
}
```

Beim Empfang einer Textnachricht empfiehlt es sich immer, eine 0 unmittelbar nach dem letzten empfangenen Zeichen in den Puffer zu schreiben, wie es das obige Beispiel zeigt. Im Puffer können noch willkürliche Zeichen stehen, die Ihre Anwendung vielleicht als Teil der Nachricht interpretiert, wenn Sie den String nicht mit 0 abtrennen.

Wie die meisten CSocket-Versionen dieser Funktionen kehrt auch Receive nicht zurück, bevor Daten von der verbundenen Anwendung empfangen wurden.

Wenn Sie mit Datagram-Sockets arbeiten, sollten Sie alternative Methoden verwenden: SendTo und ReceiveFrom. Diese Funktionen arbeiten genau wie ihre Streaming- Gegenstücke, mit dem Zusatz der Netzwerkadresse und des Ports, an den die Daten zu senden sind (bei SendTo), oder von Variablen, in denen die Adresse der sendenden Anwendung gespeichert wird (bei ReceiveFrom).

Die Verbindung schließen

Nachdem die Anwendung den Datenaustausch mit der anderen Anwendung beendet hat, kann sie die Verbindung durch Aufruf der Methode Close schließen. Die Methode Close übernimmt keinerlei Parameter und lässt sich folgendermaßen aufrufen:

```
m_sMySocket.Close();
```

Die Methode Close gehört zu den wenigen Methoden von CAsyncSocket, die keinen Statuscode zurückgibt. Bei allen bisher untersuchten Member-Funktionen können Sie anhand des Rückgabewerts prüfen, ob ein Fehler aufgetreten ist.

Manchmal möchte man einen Socket herunterfahren, bevor er geschlossen ist. Das können Sie mit der Methode ShutDown tun. Diese Methode übernimmt einen einzigen Integer-Parameter, der angibt, ob das Senden oder das Empfangen über den Socket heruntergefahren werden soll. Standardmäßig fährt ShutDown das Senden über einen Socket herunter. Sie können angeben, welcher Socket deaktiviert wird, indem Sie die in Tabelle 18.1 gezeigten Werte übergeben.

Wert	Beschreibung
0	Verhindert den Empfang hereinkommender Datenpakete über den Socket
1	Verhindert das Versenden von Datenpaketen durch den Socket
2	Verhindert Senden und Empfangen von Datenpaketen durch den Socket

Tabelle 18.1: Parameterwerte der Socketfunktion ShutDown

Der Aufruf der Methode ShutDown auf einem Socket schließt weder die Verbindung, noch werden vom Socket beanspruchte Ressourcen freigegeben. Sie müssen den Socket mit der Methode Close schließen.

Socket-Ereignisse

Der Hauptgrund, weshalb Sie Ihre eigene von CAsyncSocket abgeleitete Klasse erstellen, liegt darin, die Ereignisse abzufangen, die zum Beispiel beim Empfang einer Nachricht oder beim Beenden der Verbindung auftreten. Die Klasse CAsyncSocket verfügt über eine Reihe von Funktionen, die für die jeweiligen Ereignisse aufgerufen werden. Die Funktionen verwenden alle die gleiche Definition - nur der Funktionsname unterscheidet sich - und sind dafür vorgesehen, in den abgeleiteten Klassen überschrieben zu werden. Die Funktionen sind als geschützte (protected) Elemente der Klasse CAsyncSocket deklariert. Auch in Ihren abgeleiteten Klassen sollten Sie diese Funktionen als geschützt deklarieren. Die Funktionen haben alle einen einzelnen Integer-Parameter, der einen Fehlercode darstellt und den Sie auswerten sollten, um gegebenenfalls auf Fehler zu reagieren. Tabelle 18.2 listet die Ereignisfunktionen zusammen mit den auslösenden Ereignissen auf.

Funktion	Ereignisbeschreibung
OnAccept	Diese Funktion wird auf einem hörenden Socket aufgerufen, um zu signalisieren, dass eine Verbindungsanforderung von einer anderen Anwendung auf die Annahme wartet.
OnClose	Diese Funktion wird auf einem Socket aufgerufen, um zu signalisieren, dass die Anwendung am anderen Ende der Verbindung ihren Socket geschlossen hat oder die

	Verbindung unterbrochen wurde. Daraufhin sollte man den Socket schließen, der diese Nachricht erhalten hat.
OnOutOfBandData	Diese Funktion wird aufgerufen, wenn out-of-band-Daten empfangen werden. Out-of-band-Daten werden über einen logisch unabhängigen Kanal gesendet und zum Versenden dringender Daten verwendet, die kein Teil der normalen Kommunikation zwischen den beiden verbundenen Anwendungen sind. Die Methoden Send und Receive besitzen beide einen dritten Parameter, dem ein Flag, MSG_OOB, übergeben werden kann, um out-of-band-Daten zu senden und zu empfangen.
OnConnect	Diese Funktion wird auf einem Socket aufgerufen, um zu signalisieren, dass die Verbindung mit einer anderen Anwendung eingerichtet wurde und dass die Anwendung jetzt Nachrichten über den Socket senden und empfangen kann.
OnReceive	Diese Funktion wird aufgerufen, um zu signalisieren, dass Daten über die Socket-Verbindung empfangen wurden und über die Funktion Receive zum Abruf bereitliegen.
OnSend	Diese Funktion wird aufgerufen, um zu signalisieren, dass der Socket für das Senden von Daten bereit und verfügbar ist. Die Funktion wird unmittelbar im Anschluss an die erfolgreiche Einrichtung der Verbindung aufgerufen. Ansonsten ruft man die Funktion gewöhnlich auf, wenn die Anwendung der Funktion Send mehr Daten übergeben hat, als in ein einziges Paket passen. In diesem Fall ist das ein Signal, dass alle Daten gesendet wurden und die Anwendung den nächsten mit Daten gefüllten Puffer übertragen kann.

Tabelle 18.2: Überschreibbare Benachrichtigungsfunktionen der Klasse CAsyncSocket

Zusätzlich zu diesen überschreibbaren Ereignisfunktionen stellt die Klasse CSocket eine weitere überschreibbare Funktion bereit: OnMessagePending, die aufgerufen wird, wenn Nachrichten in der Ereignisnachrichten-Warteschlange der Anwendung anhängen. So können Sie in Ihrer CSocket-Klasse nach bestimmten Windows-Nachrichten suchen und sie beantworten.

Auslösen der Ereignisse kontrollieren

Standardmäßig ruft die Klasse CAsyncSocket alle überschreibbaren Funktionen in Tabelle 18.2 auf, die Klasse CSocket ruft dagegen keine von ihnen auf. Was ist nun, wenn Ihre abgeleitete Klasse einen Kompromiss schließen und manche der Funktionen aufrufen, andere dagegen ignorieren soll? Sie haben Glück. Es gibt zwei Möglichkeiten, zu steuern, welche dieser Ereignisfunktionen ausgelöst werden sollen.

Die erste Möglichkeit, festzulegen, welche Ereignisfunktionen auszulösen sind, ist nur in der Klasse CAsyncSocket und in direkt von ihr abgeleiteten benutzerdefinierten Klassen verfügbar. In der Create-Methode ist der dritte Parameter, den Sie übergeben können, ein Flag-Wert, der angibt, welche Ereignisse ausgelöst werden sollen. Die Klasse CSocket überlädt diese Methode und hindert Sie daran, diesen Flag-Wert zu verwenden. Standardmäßig kombiniert die Create-Methode von CAsyncSocket alle Ereignis-Flag-Werte und legt damit fest, dass alle Ereignisse ausgelöst werden.

Die zweite Möglichkeit, festzulegen, welche Ereignisfunktionen auszulösen sind, ASyncSelect, ist in den abgeleiteten Klassen von sowohl CAsyncSocket als auch CSocket verfügbar. Diese Methode übernimmt nur das Kombinations-Flag, das definiert, welche Ereignisse ausgelöst werden sollen. Sie können AsyncSelect folgendermaßen aufrufen:

```
iErr = m_sMySocket.AsyncSelect(FD_READ | FD_CONNECT | FD_CLOSE);
if (iErr == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  // Mit Verarbeitung fortfahren
}
```

Der Standardwert für den Parameter von AsyncSelect ist, dass alle Ereignisfunktionen ausgelöst werden.

Wenn Sie also für alle Ereignisse das Auslösen deaktivieren und dann wieder aktivieren möchten, deaktivieren Sie zuerst alle Ereignisse durch Übergabe von 0 als Flag-Wert:

```
iErr = m_sMySocket.AsyncSelect(0);
```

Um das Auslösen aller Ereignisse wieder zu aktivieren, übergeben Sie keinen Wert als Flag:

```
iErr = m_sMySocket.AsyncSelect();
```

In Tabelle 18.3 sind die Flag-Werte aufgelistet, die Sie an AsyncSelect und Create (nur CAsyncSocket) übergeben können.

Flag	Beschreibung
FD_READ	Löst aus und ruft die Funktion OnReceive auf, wenn zu lesende Daten eingetroffen sind
FD_WRITE	Löst aus und ruft die Funktion OnSend auf, wenn die ausgehenden Winsock-Puffer für das Senden von Daten verfügbar sind. Diese Ereignisfunktion teilt Ihrer Anwendung mit, wann sie Daten senden kann.
FD_OOB	Löst aus und ruft die Funktion OnOutOfBandData auf, wenn out-of-band-Daten eintreffen und gelesen werden müssen
FD_ACCEPT	Löst aus und ruft die Funktion OnAccept auf, um Ihre Anwendung zu informieren, dass auf Ihrem lauschenden Socket ein Verbindungsgesuch hereinkommt. Rufen Sie die Funktion Accept auf, um die Verbindung herzustellen.
FD_CONNECT	Löst aus und ruft die Funktion OnConnect aus, um Ihre Anwendung zu informieren, dass die von Ihrer Anwendung mit der Methode Connect initiierte Verbindung zustande gekommen ist. Unmittelbar auf dieses Ereignis folgt das Ereignis OnSend, das Ihre Anwendung informiert, dass sie jetzt Daten an die verbundene Anwendung senden kann.
FD_CLOSE	Löst aus und ruft die Funktion OnClose aus, um Ihre Anwendung zu informieren, dass die Socket-Verbindung durch die verbundene Anwendung geschlossen wurde

Tabelle 18.3: Benachrichtigungs-Flags für Socket-Ereignisse

Fehler erkennen

Wenn eine Member-Funktion von CAsyncSocket einen Fehler zurückgibt, entweder FALSE bei den meisten Funktionen oder SOCKET_ERROR bei den Funktionen Send und Receive, können Sie mit der Methode GetLastError den Fehlercode ermitteln. Die Funktion liefert nur Fehlercodes zurück und Sie müssen sich selbst um eine Übersetzung in Klartext kümmern. Alle Winsock-Fehlercodes sind mit Konstanten definiert, sodass Sie die Konstanten in Ihrem Code verwenden können, um die anzuzeigende Fehlermeldung (falls vorhanden) für den Benutzer zu bestimmen. Die Funktion GetLastError lässt sich wie folgt verwenden:

```
int iErrCode;

iErrCode = m_sMySocket.GetLastError();
switch (iErrCode)
{
  case WSANOTINITIALISED:
...
}
```

Socket-Informationen abfragen

Von Zeit zu Zeit benötigen Sie Informationen über den Status der Sockets in Ihrer Anwendung, wie zum Beispiel die Adresse und den Port der Anwendung am anderen Ende der Verbindung und ob der Socket auf die Fertigstellung einer blockierenden Funktion wartet. Sie können auch verschiedene Optionen für die Sockets Ihrer Anwendung setzen und abfragen.

Verbindungsadresse ermitteln

Wenn ein Socket mit einer anderen Anwendung verbunden ist, können Sie die Netzwerkadresse der anderen Anwendung durch einen Aufruf der Methode GetPeerName ermitteln, der Sie einen Zeiger auf einen CString und einen Integer ohne Vorzeichen übergeben. Die Adresse und der Port der anderen Anwendung werden in diesen beiden Variablen zurückgeliefert. Die Methode GetPeerName rufen Sie folgendermaßen auf:

```
CString sPeerAddress;
UINT iPeerPort;
iErr = m_sMySocket.GetPeerName(&sPeerAddress, &iPeerPort);
if (iErr == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  cout << "Peer Network Address: " << sPeerAddress << "\n";
  cout << "Peer Port: " << iPeerPort << "\n";
  // Fortfahren
}
```

Wenn Sie Ihren Socket nicht an einen bestimmten Port oder an eine bestimmte Netzwerkadresse gebunden haben (was Sie gewöhnlich nicht tun, solange Ihr Socket nicht auf hereinkommende Verbindungen lauscht), können Sie die gleichen Informationen über den Socket Ihrer Anwendung abfragen, indem Sie die Methode GetSockName aufrufen:

```
CString sMyAddress;
UINT iMyPort;
iErr = m_sMySocket.GetSockName(&sMyAddress, &iMyPort);
if (iErr == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  cout << "My Network Address: " << sMyAddress << "\n";
  cout << "My Port: " << iMyPort << "\n";
  // Fortfahren
}
```

Mit der Methode GetSockName erhalten Sie wahrscheinlich 0.0.0.0 als Netzwerkadresse Ihrer Anwendung. Das liegt daran, dass Ihr Socket nicht an eine bestimmte Netzwerkadresse gebunden wurde und daher die Standardadresse (0.0.0.0) für seine ausgehenden Verbindungsgesuche verwendet. Die Winsock-Schnittstelle übersetzt diese Adresse in die Netzwerkadresse Ihres Computers, sodass die Anwendung, mit der Sie verbunden sind, die tatsächliche Netzwerkadresse des Computers erhält, auch wenn Sie selbst durch Ihre Anwendung nur Nullen sehen.

Optionen abfragen und setzen

Sie können für einen Socket verschiedene Optionen setzen, um sein Verhalten zu beeinflussen. Sie können die meisten Anwendungen jedoch erstellen, ohne auch nur eine dieser Optionen anpassen zu müssen. Wenn Sie einige der Einstellungen abfragen oder anpassen wollen, verwenden Sie die Methoden GetSockOpt und SetSockOpt.

Die Methode GetSockOpt überprüft die aktuellen Einstellungen der verschiedenen Socket- Optionen. Diese

Methode übernimmt vier Parameter, von denen der letzte optional ist:

- die Option, deren Wert Sie abfragen möchten
- einen Zeiger auf einen Puffer, in den der aktuelle Wert der Option kopiert werden soll
- einen Integer-Zeiger auf eine Variable, in der die Größe des Puffers gespeichert ist, in den der Einstellungswert kopiert werden soll
- für welche Ebene die Option definiert ist: Socket oder Protokoll. Standard ist die Socket- Ebene, SOL_SOCKET, jedoch ist eine Option, IPPROTO_TCP, auf Protokoll-Ebene definiert. In Tabelle 18.4 sind die verfügbaren Socket-Optionen und ihre Datentypen aufgelistet.

Option	Datentyp	Beschreibung
SO_ACCEPTCONN	BOOL	Der Socket lauscht auf ein hereinkommendes Verbindungsgesuch.
SO_BROADCAST	BOOL	Der Socket ist für die Übertragung von Broadcast-Nachrichten konfiguriert.
SO_DEBUG	BOOL	Für den Socket ist Debugging aktiviert.
SO_DONTLINGER	BOOL	Wenn man diese Option auf TRUE setzt, wird die Option SO_LINGER deaktiviert.
SO_DONTROUTE	BOOL	Routing ist deaktiviert.
SO_ERROR	int	Fehlerstatus abfragen und löschen
SO_KEEPALIVE	BOOL	Es werden Pakete zum Aufrechterhalten der Verbindung gesendet.
SO_LINGER	struct LINGER	Gibt die aktuelle Verzögerungsoption zurück
SO_OOBINLINE	BOOL	Es werden out-of-band-Daten im normalen Datenstrom empfangen.
SO_RCVBUF	int	Die für den Datenempfang verwendete Puffergröße
SO_REUSEADDR	BOOL	Der Socket kann an eine bereits verwendete Adresse (und einen Port) gebunden werden.
SO_SNDBUF	int	Die für das Senden von Daten verwendete Puffergröße
SO_TYPE	int	Der Socket-Typ (SOCK_STREAM oder SOCK_DGRAM)
SO_NODELAY	BOOL	Deaktiviert den Nagle-Algorithmus für kumuliertes Senden

Tabelle 18.4: Socket-Optionen

Um eine dieser Optionen zu setzen oder zu ändern, übernimmt die Methode SetSockOpt die gleichen vier Parameter mit einer kleinen Ausnahme: der dritte Parameter, die Größe des Puffers, der den Wert enthält, auf den die Option zu setzen ist, wird als Integer übergeben anstatt als Zeiger auf einen Integer. Denken Sie auch daran, dass Sie mit der Methode SetSockOpt den Wert jeder Option in Tabelle 18.4 außer SO_ACCEPTONLY, SO_ERROR und SO_TYPE setzen können. Diese Optionen sind Read-only.

Um den Wert einer bestimmten Option abzufragen und zu setzen, verwenden Sie wie folgt diese beiden Methoden:

```
BOOL bStatus;
int iStatusSize;
iStatusSize = sizeof(BOOL);
iErr = m_sMySocket.GetSockOpt(SO_KEEPALIVE, &bStatus, &iStatusSize);
if (iErr == SOCKET_ERROR)
{
  // Hier Fehlerbehandlung durchführen
}
else
{
  // Senden wir Pakete zum Aufrechterhalten der Verbindung?
  if (!bStatus)
  {
```

```
    // Nein, mit dem Senden beginnen
    bStatus = TRUE;
    iErr = m_sMySocket.SetSockOpt(SO_KEEPALIVE, &bStatus,
                                  sizeof(BOOL));
    if (iErr == SOCKET_ERROR)
    {
      // Hier Fehlerbehandlung durchführen
    }
  }
  // Fortfahren
}
```

Blockiert ein Socket?

Wenn Sie die Klasse CSocket verwenden, blockieren alle Socket- Kommunikationsfunktionen standardmäßig die gesamte Thread-Verarbeitung, bis die Funktion abgearbeitet ist. Wenn Sie auf einem Socket die Funktion Connect aufgerufen haben, gibt die Funktion die Steuerung des Threads nicht zurück, bevor die Verbindung hergestellt ist oder eine Zeitüberschreitung eingetreten ist. Das gilt genauso für die Funktionen Accept, Receive und Send (sowie ReceiveFrom und SendTo). Was, wenn Sie eine dieser Funktionen vor ihrer Rückkehr abbrechen wollen? Dazu lassen sich zwei Methoden in der Klasse CSocket verwenden.

Zuerst müssen Sie überprüfen, ob der Socket einen Thread (und zusätzliche Operationen) blockiert. Sie können die Methode IsBlocking verwenden, um festzustellen, ob sich ein Socket in einer blockierenden Funktion befindet. Diese Methode übernimmt keine Parameter und gibt einen Booleschen Wert zurück, der Ihnen mitteilt, ob der Socket blockiert.

Nachdem Sie festgestellt haben, dass ein Socket einen Thread blockiert, können Sie die Methode durch einen Aufruf der Methode CancelBlockingCall abbrechen. Diese Methode veranlasst den Socket, die derzeit blockierende Funktion abzubrechen. Die blockierende Funktion kehrt mit der Fehlerbedingung WSAEINTR zurück.

> *Die Verwendung der Methode CancelBlockingCall, um eine blockierende Funktion abzubrechen, kann einen Socket destabilisieren. Die einzige Funktion, die Sie mit einigermaßen vorhersagbarem Ergebnis nach dem Abbruch einer blockierenden Funktion aufrufen können, ist Close.*

Um festzustellen, ob ein Socket blockiert und um ihn in diesem Fall zu beenden, können Sie Folgendes tun (natürlich in einem zweiten Thread):

```
if (m_sMySocket.IsBlocking())
  m_sMySocket.CancelBlockingCall();
```

Sockets und E/A-Serialisierung

Wenn die zwischen zwei durch eine Socket-Verbindung kommunizierenden Anwendungen ausgetauschten Daten ein bekanntes Format besitzen und problemlos serialisiert werden können, ermöglicht Ihnen eine spezialisierte MFC-Klasse die Serialisierung der Kommunikation. Bei dieser Klasse handelt es sich um CSocketFile, die Sie an eine offene CSocket-Klasse anhängen und dann genau wie ein CFile-Klassenobjekt behandeln können.

Wenn Sie einen CSocket verbinden, können Sie dem CSocket-Objekt ein CSocketFile- Klassenobjekt anhängen und dabei angeben, ob das CSocketFile archivkompatibel sein soll:

```
CSocketFile sMySocketFile(&m_sMySocket, TRUE);
```

Der erste Parameter, den Sie an den CSocketFile-Konstruktor übergeben müssen, ist ein Zeiger auf das zu serialisierende CSocket-Objekt. Der zweite Parameter ist ein boolescher Wert, der angibt, ob das CSocketFile-Objekt kompatibel zu einem CArchive-Objekt sein soll. Wenn Sie als zweiten Parameter TRUE übergeben, können Sie das CSocketFile-Objekt nehmen und mit einem CArchive-Objekt verknüpfen:

```
CArchive lArchive(&sMySocketFile, CArchive::load);
```

Ab hier können Sie das CArchive-Objekt an die Standard-MFC-Funktion Serialize übergeben, um Daten aus der Socket-Verbindung zu lesen und in sie zu schreiben.

Die Serialisierung der Socket-Kommunikation erfordert, dass beide verbundenen Anwendungen das gleiche Datenformat in den Socket schreiben und aus ihm lesen. Wenn eine der beiden verbundenen Anwendungen nicht das gleiche serialisierte Datenformat wie die andere verwendet, empfangen und senden Sie am Ende möglicherweise Datenmüll.

18.2 Eine Netzwerkanwendung erstellen

Mit der heutigen Beispielanwendung erstellen Sie eine einfache dialogbasierte Anwendung, die entweder als Client oder als Server in einer Winsock-Verbindung agieren kann. Damit können Sie für jedes Ende der Verbindung zwei Kopien derselben Anwendung starten, entweder auf demselben Computer, oder Sie kopieren die Anwendung auf einen anderen Computer, sodass Sie die beiden Kopien auf getrennten Computern ausführen und die Nachrichtenübertragung über das Netzwerk verfolgen können. Nachdem die Anwendung eine Verbindung mit einer anderen Anwendung eingerichtet hat, können Sie Textnachrichten eingeben und sie an die andere Anwendung verschicken. Wenn die Nachricht gesendet wurde, kommt sie in eine Liste der gesendeten Nachrichten. Jede empfangene Nachricht wird in eine andere Liste aller empfangenen Nachrichten kopiert. Damit können Sie sich die vollständige Liste aller gesendeten und empfangenen Nachrichten ansehen. Außerdem können Sie daraus ersehen, was eine der Anwendungskopien gesendet und was die andere empfangen hat. (Die beiden Listen sollten gleich sein.)

Das Anwendungsgerüst erstellen

Um die heutige Beispielanwendung einfach zu halten, erstellen Sie eine dialogbasierte Anwendung. Alles, was in der heutigen Anwendung zu tun ist, lässt sich genauso einfach auch in einer SDI- oder MDI-Anwendung realisieren. Durch den Einsatz einer dialogbasierten Anwendung halten wir alles, was sonst von der grundlegenden Socket- Funktionalität ablenkt (beispielsweise Fragen, ob die Socket-Variable zur Dokument- oder zur Ansichtsklasse gehört, welche Funktionen der Anwendung in welche der beiden Klassen gehören usw.) von der Beispielanwendung fern.

Um mit der heutigen Beispielanwendung zu beginnen, folgen Sie diesen Schritten:

1. Erstellen Sie ein neues MFC-Anwendungs-Visual C++-Projekt und nennen Sie es Sock.

2. Wählen Sie im Bereich Anwendungstyp die Option Dialogbasierend.
3. Im Bereich Erweiterte Features des Anwendungs-Assistenten legen Sie fest, dass die Anwendung die Unterstützung von Windows-Sockets einbinden soll (siehe Abbildung 18.3).
4. Klicken Sie auf Fertig stellen, um das Anwendungsgerüst zu erzeugen.

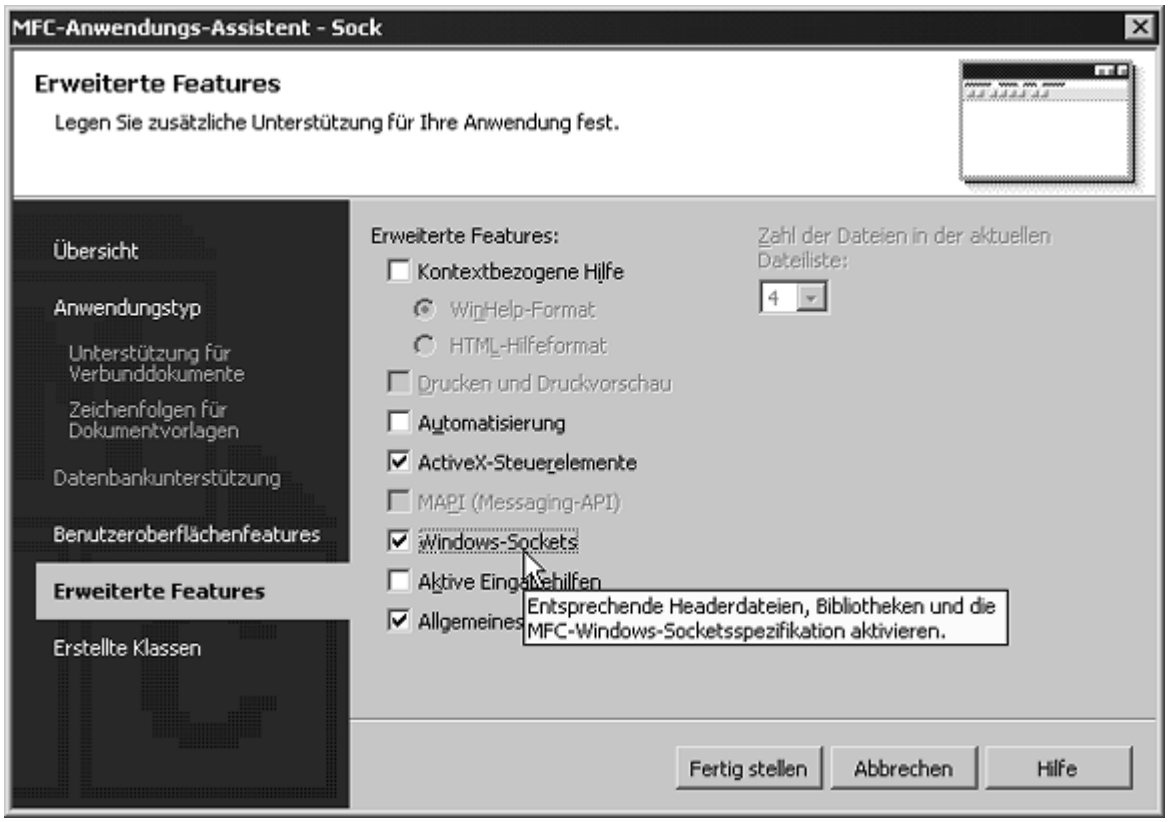


Abbildung 18.3: Unterstützung für Sockets einbinden

Fenstergestaltung und Funktionalität beim Start

Nachdem Sie über das Anwendungsgerüst verfügen, können Sie das Hauptdialogfeld für Ihre Anwendung entwerfen. In diesem Dialogfeld brauchen Sie eine Gruppe von Optionsfeldern, um festzulegen, ob die Anwendung als Client oder als Server laufen soll. Außerdem ist eine Reihe von Eingabefeldern für den Computernamen und den Port, auf dem der Server lauschen soll, erforderlich. Als Nächstes brauchen Sie eine Schaltfläche, um das Lauschen der Anwendung auf dem Socket zu starten oder die Verbindung zum Server zu öffnen, sowie eine Schaltfläche, um die Verbindung zu schließen. Weiterhin sind ein Eingabefeld für die zu verschickenden Nachrichten an die andere Anwendung und eine Schaltfläche zum Senden der Nachrichten erforderlich. Schließlich brauchen wir eine Reihe von Listenfeldern, in die sich die gesendeten und empfangenen Nachrichten eintragen lassen.

Um die Dialogbox zu gestalten, folgen Sie diesen Schritten:

1. Platzieren Sie die Steuerelemente im Dialogfeld, wie es Abbildung 18.4 zeigt. Die Eigenschaften der Steuerelemente legen Sie gemäß Tabelle 18.5 fest.

Objekt	Eigenschaft	Einstellung
Gruppenfeld	ID	IDC_STATICTYPE
	Beschriftung	Socket-Typ
Optionsfeld	ID	IDC_RCLIENT
	Beschriftung	&Client
	Gruppe	True
Optionsfeld	ID	IDC_RSERVER
	Beschriftung	&Server

Text	ID Beschriftung	IDC_STATICNAME Server-&Name:
Eingabefeld	ID	IDC_ESERVNAME
Text	ID Beschriftung	IDC_STATICPORT Server-&Port:
Eingabefeld	ID	IDC_ESERVPORT
Schaltfläche	ID Beschriftung	IDC_BCONNECT &Verbinden
Schaltfläche	ID Beschriftung Deaktiviert	IDC_BCLOSE Schlie&ssen True
Text	ID Beschriftung Deaktiviert	IDC_STATICMSG Nachrich&t: True
Eingabefeld	ID Deaktiviert	IDC_EMSG True
Schaltfläche	ID Beschriftung Deaktiviert	IDC_BSEND Sen&den True
Text	Titel	Gesendet:
Listenfeld	ID Tabstopp Sortieren Auswahl	IDC_LSENT False False Keine
Text	Titel	Empfangen:
Listenfeld	ID Tabstopp Sortieren Auswahl	IDC_LRECVD False False Keine

Tabelle 18.5: Eigenschaftseinstellungen der Steuerelemente

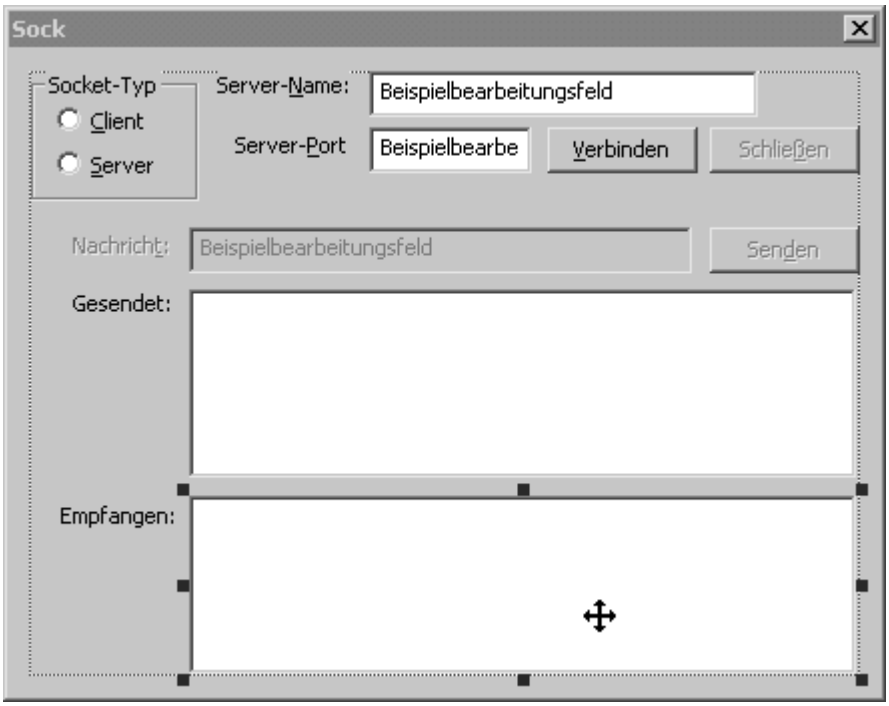


Abbildung 18.4: Die Gestaltung des Hauptdialogfelds

2. Nachdem Sie das Dialogfeld fertig entworfen haben, verbinden Sie die Variablen gemäß Tabelle 18.6 mit den Steuerelementen im Dialogfeld.

Objekt	Name	Kategorie	Typ
IDC_RCLIENT	m_iType	Value	int
IDC_ESERVNAME	m_strName	Value	CString
IDC_ESERVPORT	m_iPort	Value	int
IDC_BCONNECT	m_ctlConnect	Control	CButton
IDC_EMSG	m_strMessage	Value	CString
IDC_LSENT	m_ctlSent	Control	CListBox
IDC_LRECVD	m_ctlRecvd	Control	CListBox

Tabelle 18.6: Variablen der Steuerelemente

Damit man die Schaltfläche Verbinden zweifach nutzen kann, um auch den Server in den Lauschmodus zu versetzen, fügen Sie für das Klickereignis der beiden Optionsfelder eine Funktion hinzu, die den Text auf der Schaltfläche in Abhängigkeit von der momentan ausgewählten Option ändert. Um diese Funktionalität in die Anwendung einzubauen:

1. fügen Sie für die Nachricht BN_CLICKED der Steuerelement-ID IDC_RCLIENT eine Funktion namens OnRType hinzu,

2. öffnen Sie die Quellcode-Datei SockDlg.cpp und suchen Sie den Abschnitt mit der Nachrichtenzuordnungstabelle. Kopieren Sie das Makro ON_BN_CLICKED für die Steuerelement-ID IDC_RCLIENT und fügen Sie einen neuen Eintrag für die Steuerelement-ID IDC_RSERVER ein, der die gleiche Fehler-Behandlungsroutine aufruft:

```
BEGIN_MESSAGE_MAP(CSockDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
```

```
    ON_WM_QUERYDRAGICON()
    //} } AFX_MSG_MAP
    ON_BN_CLICKED(IDC_RCLIENT, OnBnClickedRType)
    ON_BN_CLICKED(IDC_RSERVER, OnBnClickedRType)
END_MESSAGE_MAP()
```

3. In die Funktion übernehmen Sie den Code aus Listing 18.1.

Listing 18.1: Die Funktion CSockDlg.OnBnClickedRType

```
 1: void CSockDlg::OnBnClickedRType()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Steuerelemente mit Variablen synchronisieren
 6:   UpdateData(TRUE);
 7:   // In welchem Modus sind wir?
 8:   if (m_iType == 0)    // Entsprechenden Text auf die
 9:                        // Schaltfläche setzen
10:     m_ctlConnect.SetWindowText("&Verbinden");
11:   else
12:     m_ctlConnect.SetWindowText("Sen&den");
13: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollte sich der Text auf der Schaltfläche ändern und die jeweilige Rolle der Anwendung wiedergeben, je nachdem, welches der beiden Optionsfelder ausgewählt ist (siehe Abbildung 18.5).

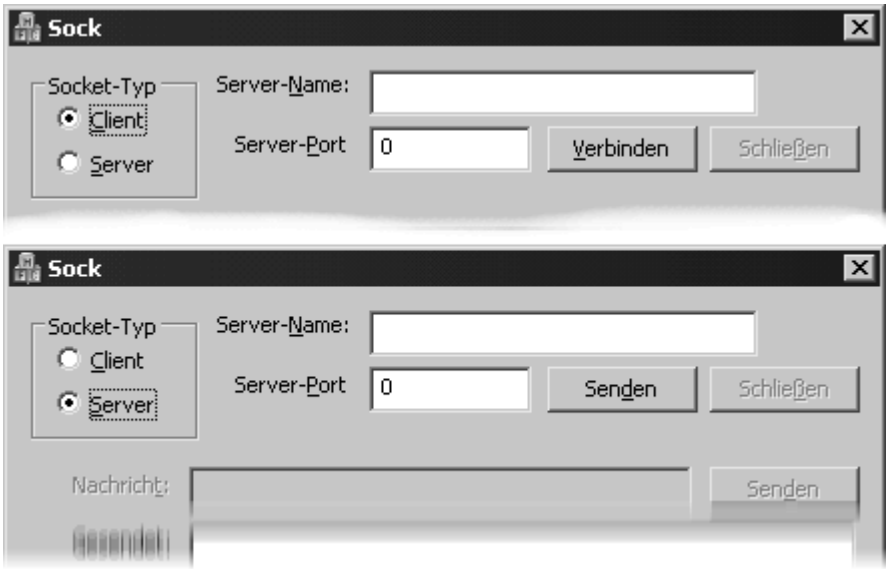


Abbildung 18.5: Der Schaltflächentext ändert sich je nach gewählter Option.

Von der Klasse CAsyncSocket erben

Damit Sie die Socket-Ereignisse abfangen und darauf reagieren können, erstellen Sie Ihre eigene Klasse, die von CAsyncSocket abgeleitet ist. Diese Klasse benötigt sowohl ihre eigenen Versionen der Ereignisfunktionen als auch die Mittel, das Ereignis an das Dialogfeld weiterzuleiten, zu dem das Objekt gehört. Um jedes dieser Ereignisse auf der Ebene der Dialogfeldklasse übergeben zu können, fügen Sie einen Zeiger auf die übergeordnete Dialogfeldklasse als Elementvariable Ihrer Socket-Klasse hinzu. Über diesen Zeiger rufen Sie Ereignisfunktionen, die Elementfunktionen des Dialogfelds sind, für die Socket-Ereignisse auf, nachdem Sie - selbstverständlich - geprüft haben, dass keine Fehler aufgetreten sind.

Um diese Klasse in Ihrer Anwendung zu erzeugen, folgen Sie diesen Schritten:

1. Klicken Sie in der Klassenansicht mit der rechten Maustaste auf den obersten Knoten (den Projektknoten).

2. Wählen Sie Hinzufügen / Klasse hinzufügen aus dem Kontextmenü.
3. Im Dialogfeld Klasse hinzufügen wählen Sie in der Baumansicht auf der linken Seite MFC und auf der rechten Seite MFC-Klasse (siehe Abbildung 18.6). Klicken Sie auf die Schaltfläche Öffnen.

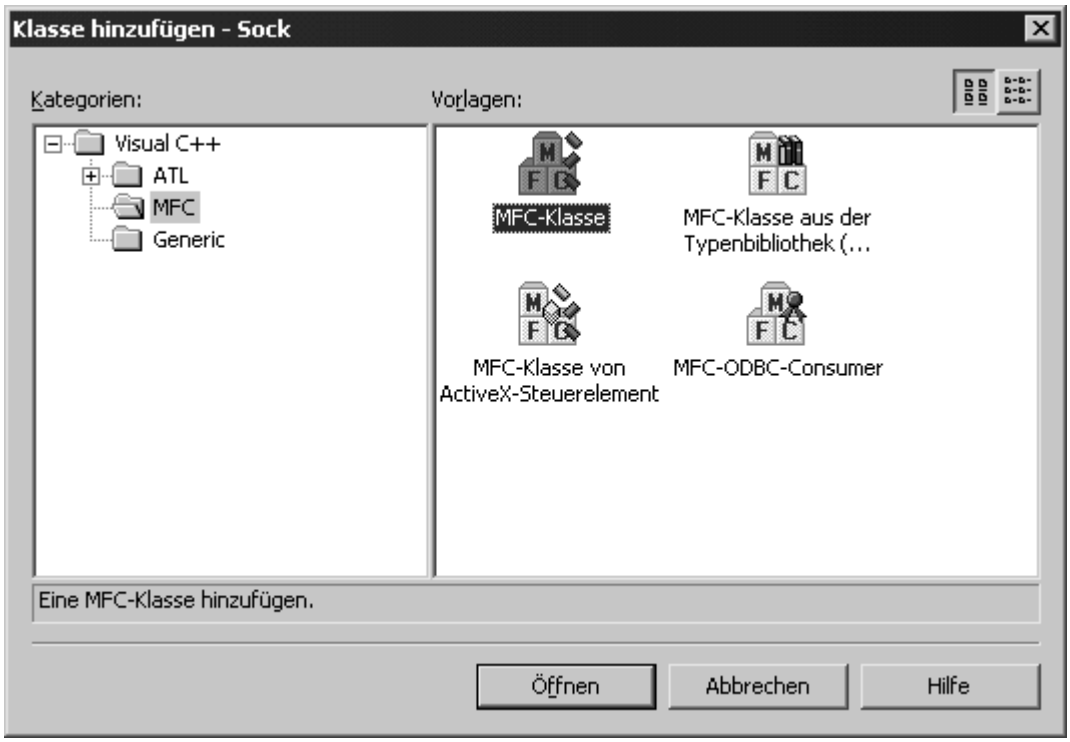


Abbildung 18.6: Den zu erstellenden Klassentyp festlegen

4. Nennen Sie Ihre neue Klasse CMySocket und wählen Sie CAsyncSocket aus der Liste der verfügbaren Basisklassen (siehe Abbildung 18.7). Klicken Sie auf Fertig stellen, um diese neue Klasse in Ihre Anwendung einzubinden.

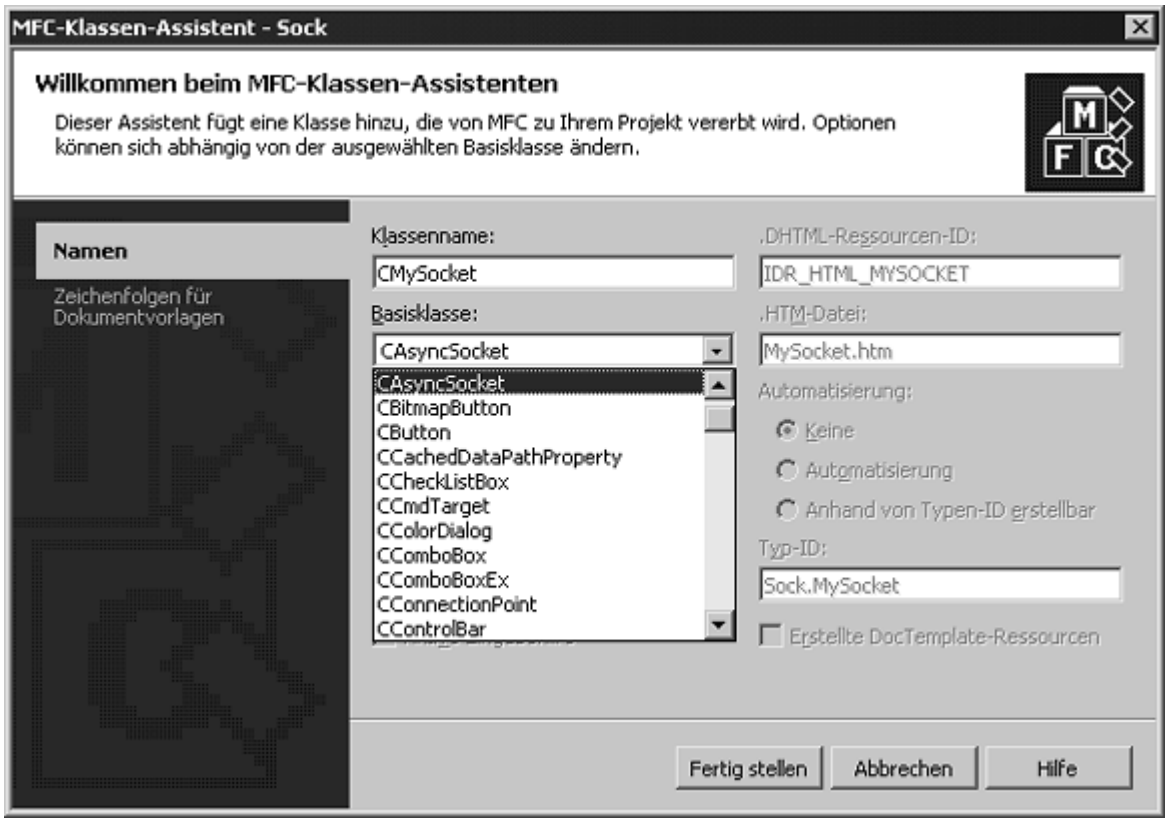


Abbildung 18.7: Die zu erbende MFC-Klasse festlegen

5. Nehmen Sie in die Klasse eine Member-Variable auf, die als Zeiger auf das übergeordnete Dialogfeld dient. Legen Sie den Variablentyp als CDialog*, den Variablennamen mit m_pWnd und den Zugriff als private fest.
6. Nehmen Sie in die neue Socket-Klasse eine Member-Funktion auf, um den Zeiger zu setzen. Legen Sie den Funktionstyp als void und den Namen als SetParent fest und fügen Sie einen Parameter vom Typ CDialog* namens pWnd ein. Legen Sie den Zugriff als Public fest.
7. In dieser Funktion setzen Sie den als Parameter übergebenen Zeiger auf den Member-Variablenzeiger, wie folgt:

```
void CMySocket::SetParent(CDialog *pWnd)
{
  // Member-Zeiger setzen
  m_pWnd = pWnd;
}
```

Das Einzige, was Sie noch in Ihre Socket-Klasse aufnehmen müssen, sind die Ereignisfunktionen, über die Sie ähnlich benannte Funktionen in der Dialogfeldklasse aufrufen. Um eine Funktion für das Ereignis OnAccept aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie im Modus Überschreibungen des Eigenschaftenfensters der Klasse CMySocket eine Überschreibungsfunktion für die Funktion OnAccept hinzu.

2. In die Funktion schreiben Sie den Code aus Listing 18.2.

Listing 18.2: Die Funktion CMySocket.OnAccept

```
1: void CMySocket::OnAccept(int nErrorCode)
2: {
3:   // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
4:   // und/oder rufen Sie die Basisklasse auf.
5:   // Sind Fehler aufgetreten?
6:   if (nErrorCode == 0)
7:     // Nein, die OnAccept-Funktion des Dialogfelds aufrufen
```

```
 8:     ((CSockDlg*)m_pWnd)->OnAccept();
 9:
10:   CAsyncSocket::OnAccept(nErrorCode);
11: }
```

3. Analoge Funktionen fügen Sie der Socket-Klasse für die Funktionen OnConnect, OnClose, OnReceive und OnSend hinzu, die gleichnamige Funktionen in der Dialogfeldklasse aufrufen. (Die Funktionen der Dialogfeldklasse realisieren wir später.)
4. Binden Sie die Header-Datei für das Anwendungsdialogfeld in Ihre Socket-Klasse ein, wie es die letzte Zeile im Folgenden zeigt:

```
// MySocket.cpp : Implementierungsdatei
//
#include "stdafx.h"
#include "Sock.h"
#include "MySocket.h"
#include "SockDlg.h"
```

Nachdem Sie die Ereignisfunktionen in Ihre Socket-Klasse eingebunden haben, nehmen Sie eine Variable Ihrer Socket-Klasse in die Dialogfeldklasse auf. Für die Server- Funktionalität sind zwei Variablen in der Dialogfeldklasse erforderlich - eine zum Lauschen auf Verbindungsgesuche und die andere zum Herstellen einer Verbindung zur anderen Anwendung.

Da Sie zwei Socket-Objekte brauchen, nehmen Sie zwei Member-Variablen in die Dialogfeldklasse (CSockDlg) auf. Legen Sie den Typ der beiden Variablen mit Ihrer Socket- Klasse (CMySocket) und den Zugriff als private fest. Die Variable, die für das Lauschen auf Verbindungswünsche verwendet wird, nennen Sie m_sListenSocket, die andere Variable, die für die Übertragung der Nachrichten in beide Richtungen verantwortlich ist, m_sConnectSocket.

Nachdem Sie die Socket-Variablen hinzugefügt haben, bauen Sie noch den Initialisierungscode für alle Variablen ein. Als Vorgabe setzen Sie den Anwendungstyp auf Client, den Servernamen auf loopback und den Port auf 4000. Zusammen mit diesen Variablen setzen Sie die Zeiger des übergeordneten Dialogfelds in Ihren zwei Socket- Objekten, sodass sie auf die Dialogfeldklasse zeigen. Dazu nehmen Sie den Code aus Listing 18.3 in die Funktion OnInitDialog der Dialogfeldklasse auf.

Der Computername loopback ist ein spezieller Name, der im Netzwerkprotokoll TCP/IP den Computer kennzeichnet, auf dem Sie arbeiten. Es handelt sich um einen internen Computernamen, der in die Netzwerkadresse 127.0.0.1 aufgelöst wird. Diesen Computernamen und diese Adresse verwenden gewöhnlich Anwendungen, die eine Verbindung zu anderen Anwendungen, die auf demselben Computer laufen, herstellen müssen. Der Name entspricht dem speziellen Namen localhost. Der Name localhost findet sich als Standardname in der hosts-Datei des Systems (eine lokale Datei, die zur Auflösung von Hostnamen verwendet wird, bevor eine Anfrage an den DNS-Server geht), während der Name loopback auf tieferer Ebene von der Socket- und Netzwerk-Infrastruktur verwendet wird.

Listing 18.3: Die Funktion CSockDlg.OnInitDialog

```
 1: BOOL CSockDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4:
    ...
30:
31:   // TODO: Hier zusätzliche Initialisierung einfügen
32:   // Steuerelementvariablen initialisieren
33:   m_iType = 0;
34:   m_strName = "loopback";
```

```
35:   m_iPort = 4000;
36:   // Steuerelemente aktualisieren
37:   UpdateData(FALSE);
38:   // Zeiger auf Socket-Dialogfeld setzen
39:   m_sConnectSocket.SetParent(this);
40:   m_sListenSocket.SetParent(this);
41:
42:   return TRUE;  // Geben Sie TRUE zurück, außer ein
43:                 // Steuerelement soll den Fokus erhalten
44: }
```

Die Anwendung verbinden

Wenn der Benutzer auf die Schaltfläche Verbinden klickt, deaktivieren Sie alle Steuerelemente im oberen Teil des Dialogfelds. Zu diesem Zeitpunkt soll der Benutzer nicht annehmen, dass er die Einstellungen des Computers, der gerade eine Verbindung aufbaut, oder die Art und Weise, wie die Anwendung lauscht, ändern kann. Sie rufen die Funktion Create auf der jeweiligen Socket-Variablen auf, je nachdem, ob die Anwendung als Client oder als Server läuft. Schließlich rufen Sie entweder die Funktion Connect oder die Funktion Listen auf, um die Anwendungsseite der Verbindung zu initialisieren.

Um diese Funktionalität in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie eine Behandlungsroutine für das Ereignis BN_CLICKED der Schaltfläche Verbinden (IDC_BCONNECT) hinzu.

2. In die Funktion übernehmen Sie den Code aus Listing 18.4.

Listing 18.4: Die Funktion CSockDlg.OnBnClickedBconnect

```
 1: void CSockDlg::OnBnClickedBconnect()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4: Benachrichtigung ein.
 5:   // Variablen mit Steuerelementen synchronisieren
 6:   UpdateData(TRUE);
 7:   // Steuerelemente Verbindung und Typ deaktivieren
 8:   GetDlgItem(IDC_BCONNECT)->EnableWindow(FALSE);
 9:   GetDlgItem(IDC_ESERVNAME)->EnableWindow(FALSE);
10:   GetDlgItem(IDC_ESERVPORT)->EnableWindow(FALSE);
11:   GetDlgItem(IDC_STATICNAME)->EnableWindow(FALSE);
12:   GetDlgItem(IDC_STATICPORT)->EnableWindow(FALSE);
13:   GetDlgItem(IDC_RCLIENT)->EnableWindow(FALSE);
14:   GetDlgItem(IDC_RSERVER)->EnableWindow(FALSE);
15:   GetDlgItem(IDC_STATICTYPE)->EnableWindow(FALSE);
16:   // Sind wir Client oder Server?
17:   if (m_iType == 0)
18:   {
19:     // Client, Standard-Socket erzeugen
20:     m_sConnectSocket.Create();
21:     // Verbindung zum Server öffnen
22:     m_sConnectSocket.Connect(m_strName, m_iPort);
23:   }
24:   else
25:   {
26:     // Server, an den angegebenen Port gebundenen Socket
27:     // erzeugen
28:     m_sListenSocket.Create(m_iPort);
29:     // Auf Verbindungsgesuche lauschen
30:     m_sListenSocket.Listen();
31:   }
32: }
```

Um die Verbindung zu vervollständigen, fügen Sie in die Dialogfeldklasse die Socket- Ereignisfunktionen für

die Ereignisfunktionen OnAccept und OnConnect hinzu. Diese Funktionen ruft Ihre Socket-Klasse auf. Es sind keine Parameter erforderlich und die Funktionen liefern auch keine Rückgabewerte. In der Funktion OnAccept, die für den lauschenden Socket aufgerufen wird, wenn eine andere Anwendung eine Verbindung herstellen will, rufen Sie die Funktion Accept des Socket-Objekts auf und übergeben dabei die Socket-Variable für die Verbindung. Nachdem Sie die Verbindung akzeptiert haben, aktivieren Sie die Aufforderung und das Eingabefeld, damit der Benutzer Nachrichten an die andere Anwendung eingeben und abschicken kann.

Folgen Sie diesen Schritten, um die Funktion in Ihre Anwendung aufzunehmen:

1. Fügen Sie der Dialogfeldklasse (CSockDlg) eine Member-Funktion hinzu.

2. Legen Sie den Funktionstyp als void, den Namen mit OnAccept und den Zugriff als public fest.
3. In die Funktion übernehmen Sie folgenden Code:

```
void CSockDlg::OnAccept(void)
{
  // Verbindungsgesuch entgegennehmen
  m_sListenSocket.Accept(m_sConnectSocket);
  // Text- und Nachrichtensteuerelemente aktivieren
  GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
}
```

Nachdem die Verbindung vollständig eingerichtet ist, bleibt auf der Client-Seite nichts weiter zu tun, als die Steuerelemente für die Eingabe und das Senden der Nachrichten zu aktivieren. Weiterhin aktivieren Sie die Schaltfläche Schliessen, damit sich die Verbindung von der Client-Seite (nicht jedoch von der Server-Seite) schließen lässt. Um die Funktionalität in Ihre Anwendung aufzunehmen, folgen Sie diesen Schritten:

1. Fügen Sie der Dialogfeldklasse (CSockDlg) eine weitere Member-Funktion hinzu.

2. Legen Sie den Funktionstyp als void, den Funktionsnamen als OnConnect und den Zugriff als public fest.
3. In die Funktion schreiben Sie folgenden Code:

```
void CSockDlg::OnConnect(void)
{
  // Text- und Nachrichtensteuerelemente aktivieren
  GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
  GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
  GetDlgItem(IDC_BCLOSE)->EnableWindow(TRUE);
}
```

Falls sich die Anwendung jetzt schon kompilieren und ausführen ließe, könnten Sie zwei Kopien der Anwendung starten, eine davon in den Lauschmodus setzen und zu dieser mit der anderen Anwendung eine Verbindung herstellen. Leider können Sie die Anwendung wahrscheinlich noch nicht einmal kompilieren, da die Socket-Klasse nach verschiedenen Funktionen in Ihrer Dialogfeldklasse sucht, die Sie noch nicht hinzugefügt haben. Nehmen Sie die noch fehlenden drei Funktionen mit folgenden Schritten auf:

1. Fügen Sie in die Dialogfeldklasse (CSockDlg) drei Member-Funktionen ein.

2. Legen Sie alle drei Funktionen als void mit Zugriffsstatus public fest. Den Namen der ersten Funktion geben Sie mit OnSend, den der zweiten Funktion mit OnReceive und den der dritten mit OnClose an.

Nun sollte sich die Anwendung kompilieren lassen. Nachdem Sie die Anwendung erfolgreich kompiliert haben, starten Sie zwei Kopien und platzieren Sie nebeneinander auf dem Bildschirm. Die eine Anwendung legen Sie als Server fest und klicken auf die Schaltfläche Lauschen, um sie in den Lauschmodus zu versetzen. Die andere Anwendung belassen Sie als Client und klicken auf die Schaltfläche Verbinden. Die Verbindungssteuerelemente sollten nun deaktiviert und die Steuerelemente zum Senden der Nachrichten aktiviert sein, nachdem die Verbindung hergestellt ist, wie es Abbildung 18.8 zeigt.

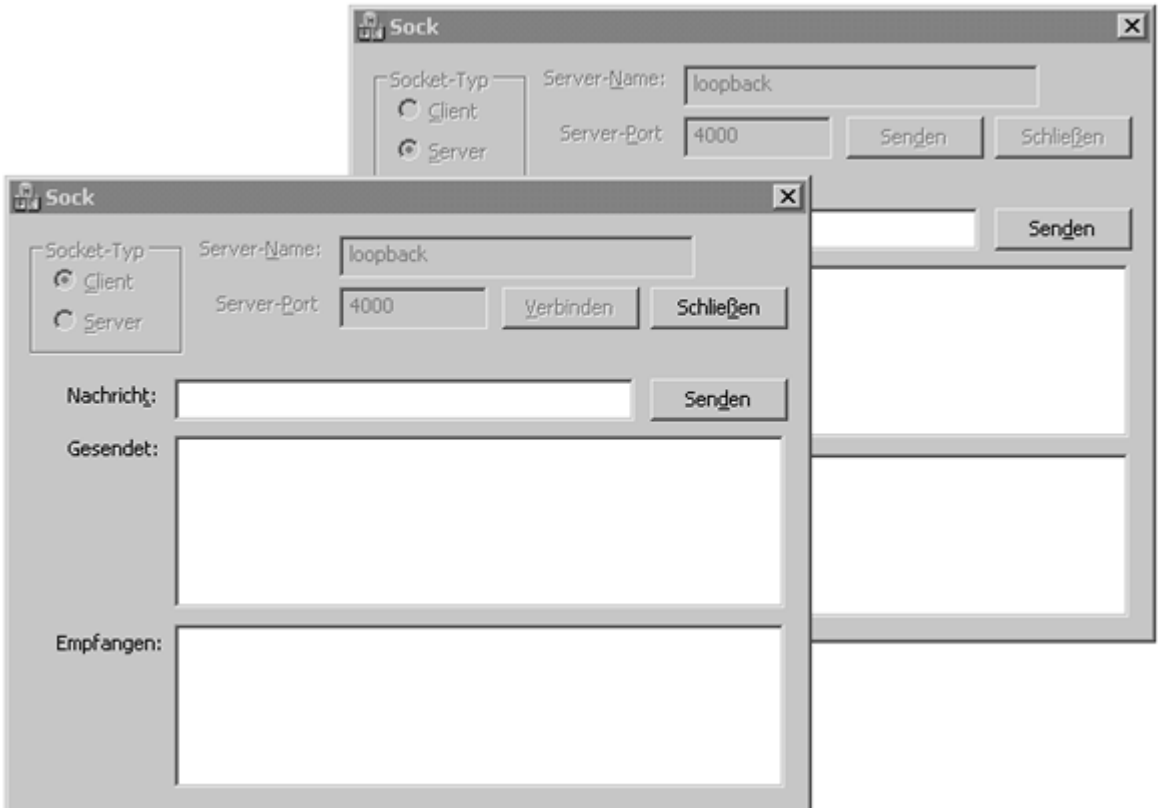


Abbildung 18.8: Die beiden Anwendungen verbinden

Stellen Sie sicher, dass die Server-Anwendung bereits lauscht, bevor Sie versuchen, die Verbindung mit der Client-Anwendung herzustellen. Wenn Sie mit dem Client eine Verbindung aufbauen wollen und der Server noch nicht lauscht, wird die Verbindung zurückgewiesen. Ihre Anwendung erkennt aber nicht, dass die Verbindung abgelehnt wurde, da Sie bisher keinerlei Fehlerbehandlung für dieses Ereignis vorgesehen haben.

Sie können immer nur eine Applikation mit Debuggen / Starten ausführen. Entweder starten Sie die zweite Applikation über den Windows Explorer oder Sie starten die erste Applikation über den Befhel Debuggen / Starten ohne Debuggen.

Um die Anwendungen ausführen und verbinden zu können, muss TCP/IP auf Ihrem Computer laufen. Diese Bedingung ist wahrscheinlich schon erfüllt, wenn Ihr Computer mit einer Netzwerkkarte ausgestattet ist. Haben Sie keine Netzwerkkarte und bauen Verbindungen zum Internet über ein Modem auf, müssen Sie sich wahrscheinlich mit dem Internet verbinden, wenn Sie die Anwendungen ausführen und testen. Falls Sie die Verbindung zum Internet per Modem abwickeln, startet Ihr Computer gewöhnlich TCP/IP, nachdem die Internetverbindung hergestellt ist. Wenn Sie weder über eine Netzwerkkarte verfügen, noch eine Möglichkeit haben, sich mit dem Internet oder einem anderen Netzwerk zu verbinden, das Ihnen die Ausführung von Netzwerkanwendungen gestattet, können Sie die heutigen Beispielanwendungen nicht auf Ihrem Computer ausführen und testen.

Eine weitere Möglichkeit für den Fall, dass Sie über keinerlei Netzwerkverbindung verfügen, ist

die Installation des MS-Loopback-Adapters, einer in Software nachgebildeten Netzwerkkartze ohne Hardware. Konsultieren Sie die Dokumentation in der Microsoft Knowledge Base im MSDN oder auf der Microsoft-Webseite, um zu erfahren, wie Sie das auf Ihrem Betriebssystem tun können.

Senden und Empfangen

Wenn Sie die beiden laufenden Anwendungen verbinden können, müssen Sie nun noch die Funktionalität hinzufügen, um Nachrichten zu senden und zu empfangen. Nachdem die Verbindung zwischen den Anwendungen eingerichtet ist, kann der Benutzer Textnachrichten in das Eingabefeld in der Mitte des Dialogfelds eingeben und auf die Schaltfläche Senden klicken, um die Nachricht an die andere Anwendung abzuschicken. Nachdem die Nachricht gesendet wurde, erscheint sie im Listenfeld der gesendeten Nachrichten. Die Anwendung hat also zu prüfen, ob beim Klicken auf die Schaltfläche Senden eine zu sendende Nachricht vorhanden ist. In diesem Fall muss die Anwendung die Länge der Nachricht ermitteln, die Nachricht senden und sie dann in das Listenfeld der gesendeten Nachrichten eintragen. Folgen Sie diesen Schritten:

1. Fügen Sie eine Behandlungsroutine für das Klickereignis der Schaltfläche Senden (IDC_BSEND) hinzu.

2. In die Funktion schreiben Sie den Code gemäß Listing 18.5.

Listing 18.5: Die Funktion CSockDlg.OnBnClickedBsend

```
 1: void CSockDlg::OnBnClickedBsend()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   int iLen;
 6:   int iSent;
 7:
 8:   // Steuerelemente mit Variablen synchronisieren
 9:   UpdateData(TRUE);
10:   // Ist eine Nachricht zu senden?
11:   if (m_strMessage != "")
12:   {
13:     // Länge der Nachricht ermitteln
14:     iLen = m_strMessage.GetLength();
15:     // Nachricht senden
16:     iSent = m_sConnectSocket.Send(LPCTSTR(m_strMessage), iLen);
17:     // Konnten wir senden?
18:     if (iSent == SOCKET_ERROR)
19:     {
20:     }
21:     else
22:     {
23:       // Nachricht in das Listenfeld eintragen
24:       m_ctlSent.AddString(m_strMessage);
25:       // Variablen mit den Steuerelementen synchronisieren
26:       UpdateData(FALSE);
27:     }
28:   }
29: }
```

Wenn die Ereignisfunktion OnReceive ausgelöst wird, weil eine Nachricht angekommen ist, rufen Sie die Nachricht aus dem Socket über die Funktion Receive ab. Dann konvertieren Sie die Nachricht in einen CString und fügen sie dem Listenfeld der empfangenen Nachrichten hinzu. Um diese Funktionalität zu realisieren, übernehmen Sie den Code aus Listing 18.6 in die Funktion OnReceive.

Listing 18.6: Die Funktion CSockDlg.OnReceive

```
 1: void CSockDlg::OnReceive(void)
 2: {
```

```
 3:    char *pBuf = new char[1025];
 4:    int iBufSize = 1024;
 5:    int iRcvd;
 6:    CString strRecvd;
 7:
 8:    // Nachricht empfangen
 9:    iRcvd = m_sConnectSocket.Receive(pBuf, iBufSize);
10:    // Haben wir etwas empfangen?
11:    if (iRcvd == SOCKET_ERROR)
12:    {
13:    }
14:    else
15:    {
16:      // Ende der Nachricht abschneiden
17:      pBuf[iRcvd] = '\0';
18:      // Nachricht in einen CString kopieren
19:      strRecvd = pBuf;
20:      // Nachricht in Listenfeld Empfangen eintragen
21:      m_ctlRecvd.AddString(strRecvd);
22:      // Variablen mit Steuerelementen synchronisieren
23:      UpdateData(FALSE);
24:    }
25: }
```

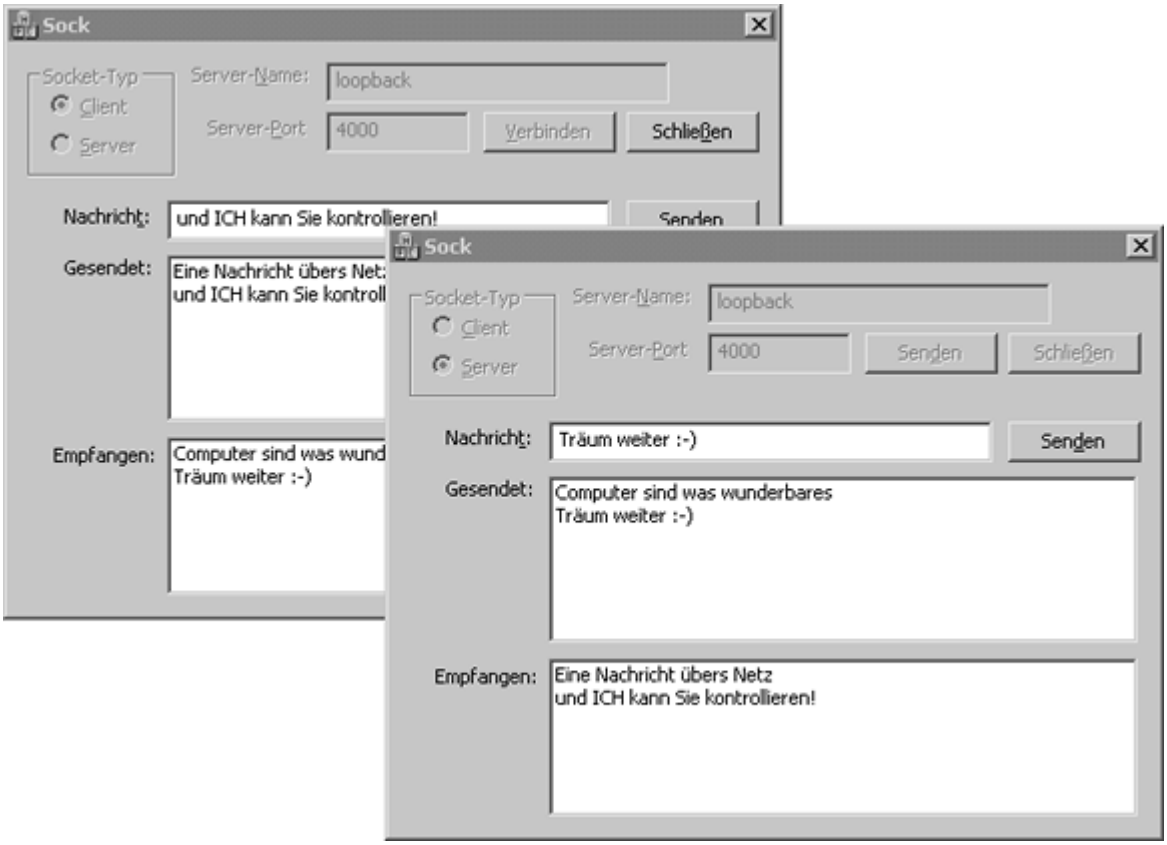


Abbildung 18.9: Nachrichten zwischen den Anwendungen senden

Jetzt sollten Sie die Anwendung kompilieren und zwei Kopien ausführen können, wobei Sie die beiden Anwendungen - wie bereits weiter oben geschehen - miteinander verbinden. Nachdem Sie die Verbindung eingerichtet haben, können Sie eine Nachricht in die eine Anwendung eingeben und sie an die andere Anwendung schicken, wie es Abbildung 18.9 verdeutlicht.

Die Verbindung beenden

Um die Verbindung zwischen den beiden Anwendungen zu schließen, kann der Benutzer der Client-Anwendung auf die Schaltfläche Schliessen klicken, um die Verbindung zu beenden. Die Server-Anwendung empfängt daraufhin das Socket-Ereignis OnClose. Das Gleiche muss in beiden Fällen passieren. Der verbundene Socket ist zu schließen und die Steuerelemente für das Senden von Nachrichten sind zu

deaktivieren. Auf der Client-Seite können die Verbindungssteuerelemente aktiviert bleiben, da der Client beispielsweise die Informationen ändern und eine Verbindung zu einer anderen Server-Anwendung öffnen kann. In der Zwischenzeit lauscht die Server-Anwendung weiter auf dem Port, für den sie zu diesem Zweck konfiguriert wurde.

Um die beschriebene Funktionalität in Ihrer Anwendung umzusetzen, bearbeiten Sie die Funktion OnClose und übernehmen den Code aus Listing 18.7.

Listing 18.7: Die Funktion CSockDlg.OnClose

```
 1: void CSockDlg::OnClose(void)
 2: {
 3:   // Verbundenen Socket schließen
 4:   m_sConnectSocket.Close();
 5:   // Steuerelemente zum Senden deaktivieren
 6:   GetDlgItem(IDC_EMSG)->EnableWindow(FALSE);
 7:   GetDlgItem(IDC_BSEND)->EnableWindow(FALSE);
 8:   GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
 9:   GetDlgItem(IDC_BCLOSE)->EnableWindow(FALSE);
10:   // Sind wir im Client-Modus?
11:   if (m_iType == 0)
12:   {
13:     // Ja, Verbindungssteuerelemente aktivieren
14:     GetDlgItem(IDC_BCONNECT)->EnableWindow(TRUE);
15:     GetDlgItem(IDC_ESERVNAME)->EnableWindow(TRUE);
16:     GetDlgItem(IDC_ESERVPORT)->EnableWindow(TRUE);
17:     GetDlgItem(IDC_STATICNAME)->EnableWindow(TRUE);
18:     GetDlgItem(IDC_STATICPORT)->EnableWindow(TRUE);
19:     GetDlgItem(IDC_RCLIENT)->EnableWindow(TRUE);
20:     GetDlgItem(IDC_RSERVER)->EnableWindow(TRUE);
21:     GetDlgItem(IDC_STATICTYPE)->EnableWindow(TRUE);
22:   }
23: }
```

Schließlich rufen Sie für die Schaltfläche Schliessen die Funktion OnClose auf. Folgen Sie diesen Schritten:

1. Fügen Sie eine Behandlungsroutine für das Klickereignis der Schaltfläche Schliessen (IDC_BCLOSE) hinzu. In die Funktion schreiben Sie folgenden Code, um die Funktion OnClose aufzurufen:

```
void CSockDlg::OnBnClickedBclose()
{
  // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
  //Benachrichtigung ein.
  // OnClose aufrufen
  OnClose();
}
```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie die Client-Anwendung mit dem Server verbinden, Nachrichten in beiden Richtungen austauschen und die Verbindung des Clients trennen, indem Sie auf die Schaltfläche Schliessen klicken. In beiden Anwendungen werden die Steuerelemente zum Senden von Nachrichten deaktiviert, wie es Abbildung 18.10 zeigt. Eine erneute Verbindung des Clients mit dem Server ist möglich, indem Sie wieder auf die Schaltfläche Verbinden klicken und dann weitere Nachrichten zwischen den beiden austauschen, als ob sie noch gar nicht verbunden gewesen wären. Wenn Sie eine dritte Kopie der Anwendung starten, deren Portnummer ändern, sie als Server bekannt machen und in den Lauschmodus setzen, können Sie Ihren Client abwechselnd mit beiden Servern verbinden. Dazu schließen Sie die eine Verbindung, ändern die Portnummer und stellen dann die Verbindung zum anderen Server her.

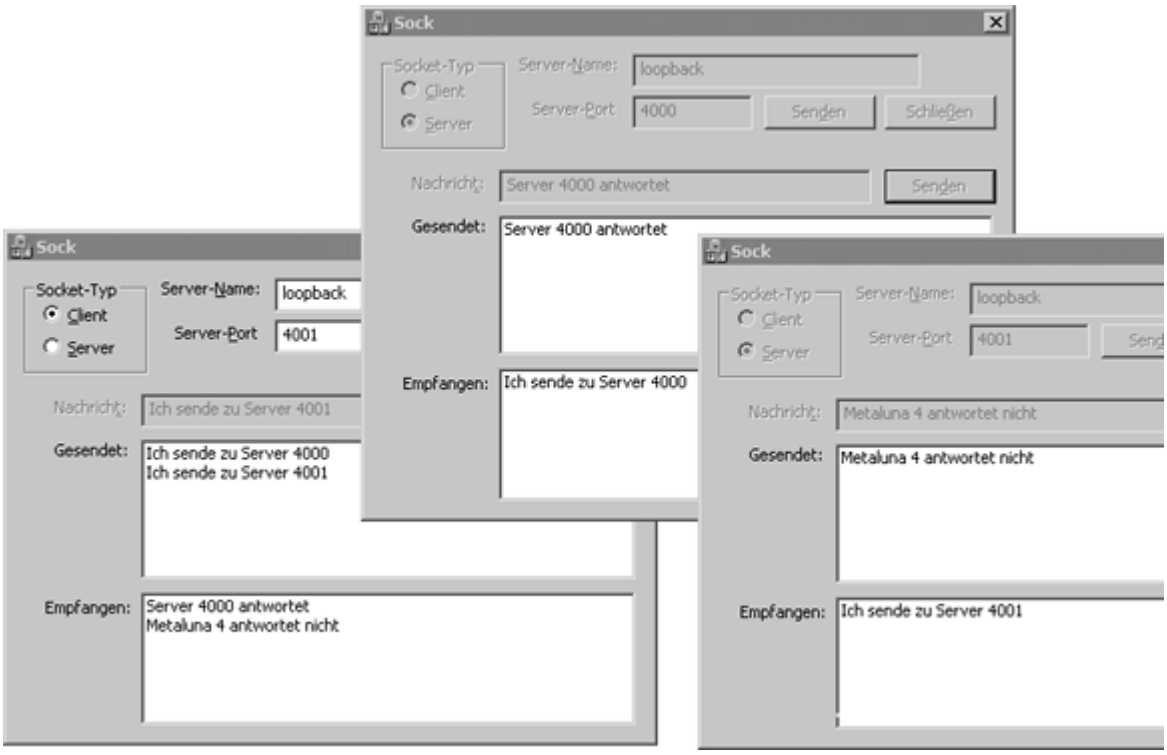


Abbildung 18.10: Die Verbindung zwischen den Anwendungen schließen

18.3 Zusammenfassung

Heute haben Sie gelernt, wie man Anwendungen mit MFC-Winsock-Klassen erstellt, um mit anderen Anwendungen über ein Netzwerk oder das Internet kommunizieren zu können. Die Klasse CAsyncSocket wurde näher beleuchtet und Sie haben gesehen, wie man davon eigene Klassen ableitet, um in einer Anwendung die ereignisgesteuerte Kommunikation zu realisieren. Es wurde gezeigt, wie man eine Server-Anwendung erstellt, die Verbindungen mit anderen Anwendungen prüft und die Verbindung herstellt. Weiterhin haben Sie gelernt, wie man eine Client-Anwendung erstellt, die eine Verbindung mit einem Server herstellen kann. Die Lektion ist ebenfalls darauf eingegangen, wie sich Nachrichten über eine Socket-Verbindung zwischen zwei Anwendungen senden und empfangen lassen. Schließlich haben Sie erfahren, wie man die Verbindung schließt und wie man erkennt, dass die Verbindung geschlossen wurde.

18.4 Workshop

Fragen und Antworten

Frage:
Wie arbeiten Internet-Anwendungen?

Antwort:
Die meisten Internet-Anwendungen bauen auf der gleichen Funktionalität auf, die Sie mit der heutigen Beispielanwendung realisiert haben. Der Hauptunterschied besteht darin, dass die Anwendungen mit Nachrichtenscripten arbeiten, die in beiden Richtungen übertragen werden. Die Nachrichten bestehen aus einem Befehl und den Daten, die für diesen Befehl erforderlich sind. Der Server liest den Befehl, verarbeitet die Daten entsprechend und schickt einen Statuscode zurück, aus dem der Client den Erfolg oder das Scheitern des Befehls ablesen kann. Wenn Sie mehr über diese Abläufe in Internet-Anwendungen erfahren möchten, sollten Sie sich mit der einschlägigen Literatur befassen - einige Bücher behandeln dieses Thema in allen Einzelheiten. Sie finden außerdem auf der Webseite für die Internet Engineering Task Force auf www.ietf.org die RFC-Dokumente, die das Kommunikationsprotokoll für verschiedene standardisierte Anwendungen spezifizieren. RFC-Dokumente sind die Spezifikationsdokumente für verschiedene Anwendungen, Dateiformate und andere Standards, die im Internet verwendet werden.

Listing 18.8: Ausschnitt aus dem HTTP-Protokoll

```
GET /javascript/ HTTP/1.0
User-Agent: Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+.NET)
HOST: 192.168.49.10:80
Header: HTTP/1.1 200 OK
Header: Server: Microsoft-IIS/4.0
Header: MicrosoftOfficeWebServer: 5.0_Pub
Header: Content-Location: http://192.168.49.10/javascript/index.htm
Header: Date: Sun, 19 May 2002 12:43:00 GMT
Header: Content-Type: text/html
Header: Accept-Ranges: bytes
Header: Last-Modified: Tue, 26 Mar 2002 09:02:28 GMT
Header: ETag: "504c67f4a4d4c11:8a29"
Header: Content-Length: 1172
<html>
...
```

Frage:
Wie behandelt eine Server-Anwendung eine größere Anzahl gleichzeitiger Client- Verbindungen?

Antwort:
Bei einem Vollserver sind die Verbindungs-Sockets nicht als Klassenvariablen deklariert. Der Server verwendet stattdessen eine Art dynamischer Zuweisung von Sockets in einem Array oder einer verknüpften Liste, um Sockets für die Clients zu erzeugen, sobald Verbindungsanforderungen hereinkommen. Eine andere von Servern oftmals verwendete Lösung besteht darin, einen eigenen Thread für jede Verbindungsanforderung einzurichten. Damit kann die Anwendung eine einzige Socket-Verbindung pro Thread behandeln, was es wesentlich erleichtert, die Sockets zu verfolgen. In allen Fällen arbeiten Server-Anwendungen normalerweise nicht nur mit einer einzigen Variablen für die Verbindungs- Sockets. Denken Sie jedoch daran, dass bei zu vielen laufenden Threads die Anzahl der Kontextwechsel die Leistung des Servers deutlich beeinträchtigt, sodass der Ansatz »ein Thread pro Verbindung« für eine Server-Anwendung mit hoher Last möglicherweise nicht angebracht ist.

Quiz

1. Welche Dinge muss eine Client-Anwendung kennen, damit sie sich mit einer Server- Anwendung verbinden kann?

2. Welche CAsyncSocket-Funktion kommt zum Einsatz, um eine Server-Anwendung in die Lage zu versetzen, Verbindungsgesuche von Client-Anwendungen zu erkennen?
3. Welche CAsyncSocket-Elementfunktion wird aufgerufen, um zu signalisieren, dass Daten über eine Socket-Verbindung angekommen sind?
4. Welche Funktion wird aufgerufen, um zu signalisieren, dass eine Verbindung eingerichtet wurde?
5. Welche Funktion verwenden Sie, um eine Nachricht über eine Socket-Verbindung zur Anwendung am anderen Ende der Verbindung zu senden?

Übung

Die Server-Anwendung, die Sie heute geschrieben haben, kann nicht mehr als eine einzelne Verbindung zu einem Zeitpunkt behandeln. Wenn eine zweite Anwendung versucht, eine Verbindung zum Server zu öffnen, während der Server mit einer anderen Anwendung in Verbindung steht, stürzt die Server-Anwendung ab. Der Server versucht, die zweite Verbindung in den Socket, der bereits mit der ersten Client-Anwendung verbunden ist, zu übernehmen. Fügen Sie Ihrer Anwendung ein drittes Socket-Objekt hinzu, das die zusätzlichen Client-Verbindungen zurückweist, bis der erste Client die Verbindung schließt.

Tag 19

Verwalteter Code

Vor kurzem stellte Microsoft eine neue Plattform-Initiative namens »dot-net« (.NET) vor. Bei der Vorstellung zeigte Microsoft mehrere neue Geräte, die Teil der neuen Plattform sind, wie beispielsweise Mobiltelefone mit eingebauten Farbbrowsern und Stimmerkennung, Tablettcomputer mit Handschrifterkennung und Autos, die sich mit allen Kommunikationsgeräten verbinden (Büro- und Mobiltelefon, E-Mail usw.). Diese Geräte wirkten wie einem Science-Fiction-Film entsprungen, doch sie sind echt und stellen funktionierende Technologie dar, die bald in den Läden auftauchen wird.

Was Microsoft da zeigte, war die neue, sich derzeit in Entwicklung befindende .NET- Plattform. Mit der .NET-Plattform können Sie mehrere Computergeräte verbinden und Daten und Anwendungen zwischen ihnen austauschen. Das heißt wenn Sie an einem Ort mit einer Anwendung arbeiten und dann an anderer Stelle die gleiche Anwendung weiterverwenden müssen (und Ihren Rechner nicht mitnehmen können), können Sie mit jedem anderen .NET-Gerät auf diese Anwendung zugreifen und mit ihr arbeiten. Wenn Sie Ihren Bürorechner verwendet haben, können Sie beispielsweise mit Ihrem Mobiltelefon auf die gleiche Anwendung und die gleichen Daten zugreifen.

Heute lernen Sie die neue .NET-Plattform kennen, wie sie funktioniert und was Sie benötigen, um Anwendungen dafür zu schreiben.

Unter anderem lernen Sie heute, ...

- wie die .NET-Plattform von Microsoft die Prozessverantwortung aufteilt und das Web mit Client-/Server-Prozessmodellen verschmilzt, um das so genannte »Active Web« oder »Active Internet« zu kreieren,
- was es mit der Common Language Runtime (CLR) auf sich hat und wie Sie mit ihrer Hilfe Anwendungen auf jedem beliebigen .NET-Gerät ausführen können,
- den Unterschied zwischen Standard-C++ und verwaltetem C++, einer neuen Variante von C++, die die Common Language Runtime nutzt,
- wie Sie Ihre vorhandenen MFC-Anwendungen nach CLR portieren und damit auf .NET-Geräten verwendbar machen,
- wie Sie mit verwaltetem C++ Anwendungen für die .NET-Plattform erstellen.

19.1 Die .NET-Plattform von Microsoft und die Common Language Runtime

Wie also funktionieren .NET und die CLR (Common Language Runtime) und welchen Unterschied macht das für Sie als Software-Entwickler? Es gibt ein paar wichtige Aspekte in der Bedeutung von CLR und .NET für Sie. Erstens werden Sie in der Lage sein, eine Anwendung einmal zu schreiben und dann auf jedem .NET-Gerät auszuführen. Wenn Sie eine Desktop-Anwendung schreiben, können Sie sie auf Ihrem Palmtop ausführen oder sogar auf Ihrem digitalen Fernsehdecoder. Das bedeutet aber auch, dass Sie anders an den Entwurf und die Planung Ihrer Anwendungen herangehen müssen, als Sie bisher gewohnt sind. Ihre Anwendungen müssen aufgeteilt werden, in Teile, die auf dem Server laufen und Teile, die auf dem Gerät des Benutzers laufen.

Die .NET-Architektur

Die .NET-Architektur ist die neueste Entwicklung in der Funktionsweise des Webs. Ursprünglich befand sich die gesamte Funktionalität auf dem Webserver und der Browser konnte über das Anzeigen von Seiten und das Senden von Rohdaten an den Server hinaus nicht viel tun. Verglichen mit dem Client-/Server-Modell für Anwendungen, das dem Web vorausging, schien das vielen Benutzern ein großer Rückschritt zu sein. Da sich die gesamte Funktionalität auf dem Server befand, gab es keine Interaktion mit dem Benutzer, wenn

dieser nicht eine Anfrage an den Server stellte.

Im Client-/Server-Modell war die Anwendungsfunktionalität über mehrere Computer verteilt. Der Client (der Rechner des Benutzers) führte die Funktionalität der Benutzerschnittstelle aus, zeigte Daten an und interagierte mit dem Benutzer. Der Datenbankserver beherbergte die Daten und führte gelegentlich Business Logic oder Prozesslogik mit diesen Daten aus. In manchen Client-/Server-Modellen gab es auch noch einen Middleware-Server, der den größten Teil der Rechenlast durchführte. Im Middleware-Server fanden sich die Business Rules, sodass sich die Funktionalität nur an zentraler Stelle änderte. Dieses Modell wird auch als 3-Tier-Modell bezeichnet.

Beim Übergang vom funktionalitätsreichen Client-/Server-Modell zur nackten Funktionalität des Webs war sich die Computerindustrie einig, dass das Web einige der Probleme des Client-/Server-Modells löste, wie zum Beispiel die Verteilung der Client- Software und die Unbrauchbarkeit für die Verwendung über ein WAN (wide-area network, zum Beispiel das Internet). Gleichzeitig aber mochten die Benutzer den Mangel an Interaktivität im Frontend gar nicht. Wegen dieser Mängel erhielt das Web eine Reihe von Zusätzen und Verbesserungen, um es dem funktionalitätsreichen Client-/Server-Modell anzunähern. Das begann mit Scriptsprachen, die im Browser liefen, und führte weiter zu Java-Applets und ActiveX-Komponenten. Gleichzeitig wurden Anwendungsserver erstellt, um die Funktionalität auf Server-Seite zu verbessern. Man wollte näher an das Client-/ Server-Modell heranrücken, ohne dabei die Netzwerkfreundlichkeit des Webs zu verlieren, die seine Stärke über Client-/Server darstellt.

Nach einigen Jahren erleben wir nun die neueste Entwicklung in dieser Reihe von Änderungen am Web. Diese neue Entwicklung ist die .NET-Plattform von Microsoft. Die .NET-Plattform besteht aus der nächsten Generation mehrerer Microsoft-Technologien (COM+, ASP+, IIS, Passport usw.), kombiniert mit verschiedenen offenen Standards (XML, SOAP, HTTP usw.) und einigen neuen Technologien und Programmiersprachen (CLR, C# usw.), um das so genannte Active Web zu kreieren. Die aus dieser Kombination entstehende Architektur sieht wie in Abbildung 19.1 aus. Die Server-Komponenten übernehmen einen großen Teil der Verarbeitung und der Datenverwaltung, mit direkter Anbindung zur Datenbank oder einer anderen Datenquelle. Die Client-Anwendungen laufen innerhalb des Webbrowsers Internet Explorer oder auf jedem anderen .NET-Client-Gerät, wobei der Benutzer eine reiche Funktionalität erlebt. XML wird als primäres Datenformat für die gesamte Kommunikation über das Webprotokoll HTTP zwischen Client und Server und zwischen Komponenten verwendet.

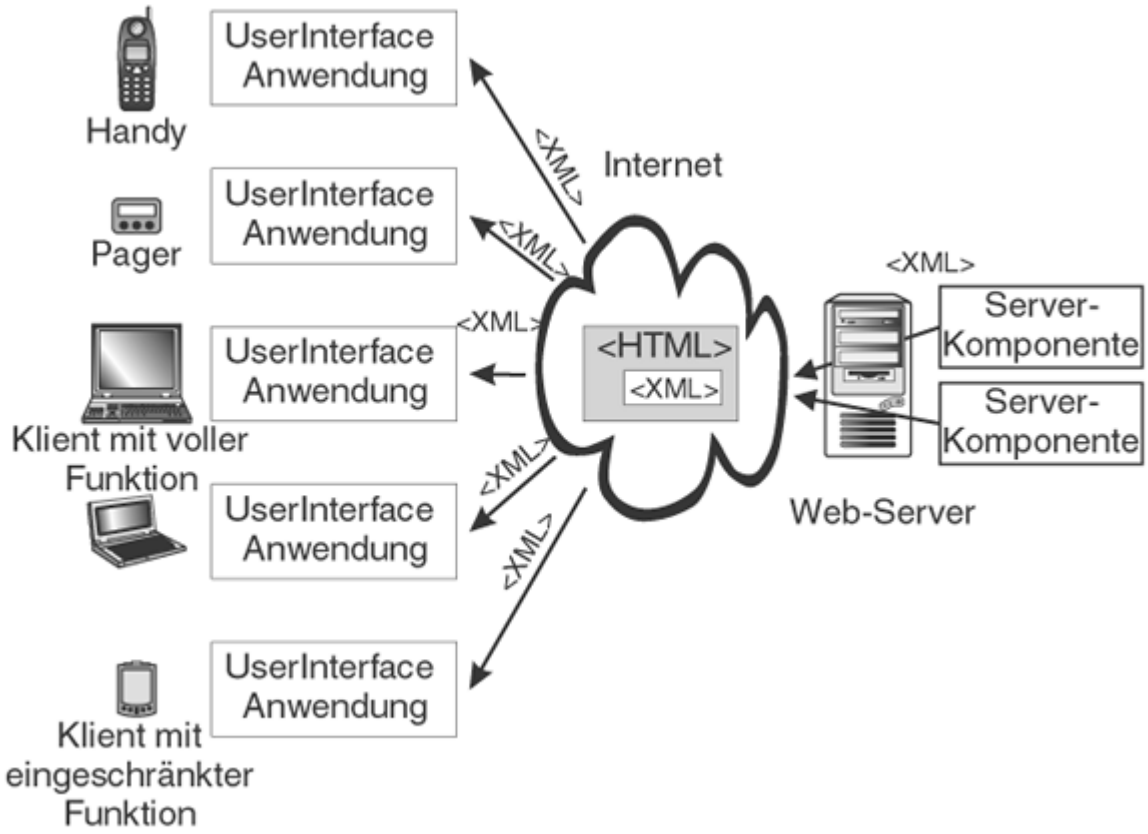


Abbildung 19.1: Die grundlegende .NET-Architektur

Server-Komponenten

Auf Server-Seite ist der größte Teil der Funktionalität in Komponenten enthalten. Komponenten sind DLLs, sie sind kein Teil einer Anwendung und können von vielen Anwendungen aufgerufen und verwendet werden. Diese Komponenten besitzen keine Benutzerschnittstelle für Interaktionen. Sie besitzen nur exponierte Funktionen, die von verschiedenen Prozesstasks aus aufgerufen werden können.

Die Server-Komponenten haben gewöhnlich mindestens zwei Verantwortungsbereiche:

- alle Business Rules für die Anwendung ausführen
- die gesamte Interaktion mit der Datenbank durchführen

Ein Grund dafür, diese Funktionalität dem Server zu überlassen, ist seine Nähe zum Datenbankserver, sodass große Mengen von Daten zu ihrer Verarbeitung nur kurze Wege über eine Netzwerkverbindung mit hoher Geschwindigkeit anstatt über das Internet zurücklegen müssen. Ein weiterer Grund ist, dass sich Business Rules auf zentralisierten Servern viel leichter warten und aktualisieren lassen, als wenn alle Benutzer ihre Anwendungen auf den neuesten Stand bringen müssten. Ein dritter Grund ist, dass es viel einfacher ist, große, schnelle, von allen Benutzern verwendete Server zu kaufen, als die Ausgaben und die Mühe auf sich zu nehmen, alle Benutzer einer Anwendung mit minimaler Verarbeitungskapazität auszustatten, besonders wenn es sich um eine große Firma handelt.

Eine Business Rule bezieht sich auf ein Stück Logik, das ausgeführt werden muss, oder auf eine Regel, die erzwungen werden muss. Ein Beispiel dafür sind die Regeln, nach denen Ihre Bank Schecks einlöst. Zuerst gibt es wahrscheinlich eine Regel, dass auf Ihrem Konto genug Geld sein muss, um die Summe des Schecks zu decken. Dann gibt es eine Regel, nach der von Ihrem Konto die Summe des Schecks abgezogen und dem Einlöser des Schecks ausgezahlt werden muss. Ein weiteres Beispiel für eine Business Rule ist der Berechnung der Steuer für einen Einkauf. Es müssen bestimmte Berechnungen mit der Menge der gekauften Waren und dem entsprechenden Steuersatz ausgeführt werden. Das sind sehr einfache Beispiele für Business Rules.

Client-Anwendungen

Der Client-Teil von .NET-Anwendungen wird auf vielen verschiedenen Geräten laufen. Kurz gesagt, die Client-Anwendung läuft in allem, das grundsätzlich eine Version des Internet Explorers von Microsoft ist. Da einige Geräte, für die Microsoft .NET-Clients plant, keine gemeinsame Hardware-Plattform besitzen, sind nativ kompilierte Anwendungen nicht zwischen den Geräten portierbar, wie sich zum Beispiel für Windows geschriebene Anwendungen nicht auf Windows-CE-Geräten ausführen ließen. Stattdessen musste man einen Cross-Compiler verwenden, um auf Windows-CE-Geräten lauffähige Versionen der Anwendung zu erstellen.

Ein Cross-Compiler kompiliert Anwendungen für eine andere Hardware-Plattform als die, auf der er läuft. Wenn Sie beispielsweise auf einem Windows-Rechner arbeiten und eine Anwendung für ein Windows-CE-Gerät erstellen, verwenden Sie für die Erstellung der ausführbaren Datei einen Cross-Compiler. Die erzeugte Anwendung läuft nicht auf dem System, auf dem Sie sie erstellt haben, aber auf dem System, für das sie gedacht ist. Cross-Compiler findet man häufig in Firmen, die Anwendungen für spezialisierte Hardware erstellen, wie beispielsweise die Anwendung, die Ihren Mikrowellenherd steuert. Die für dieses Steuerprogramm verantwortlichen Programmierer verwenden wahrscheinlich entweder ein Windows- oder ein UNIX-System, um die Anwendung zu schreiben, und erstellen dann die Anwendung mit einem Cross-Compiler, um sie in einen ROM-Chip zu brennen und in die Mikrowelle zu stecken.

Durch die Verwendung eines Webbrowsers als primäre Schnittstelle kann Microsoft Webtechnologien geschickt nutzen und erweitern, um die Client-Schnittstelle für .NET- Anwendungen über Geräte und Plattformen portierbar zu machen. Um diese Portabilität bereitzustellen, hat Microsoft seine ASP-Technologie erweitert und ihr eine viel größere Fähigkeit zur Interaktion verliehen, um sie besser mit den Server-Komponenten interagieren zu lassen. Für die durch ActiveX-Steuerelemente bereitgestellte Funktionalität hat Microsoft die Common Language Runtime (CLR) geschaffen, um Anwendungen und Steuerelementen die Ausführung auf beliebiger Hardware zu ermöglichen, was vor .NET mit ActiveX nicht immer möglich war.

SOAP und XML

Die Kommunikation zwischen den Client- und Server-Anteilen von .NET-Anwendungen wird mit XML (eXtensible Markup Language) und SOAP (Simple Object Access Protocol) durchgeführt, zwei Internet-Standards für Daten- und Objekt-Kommunikation. Es handelt sich bei beiden um Beschreibungssprachen, ähnlich HTML (HyperText Markup Language). Wenn Sie sich XML- oder SOAP-Dateien ansehen, wirken diese wie außergewöhnliche HTML-Dateien. Der Grund dafür ist, dass beide textbasierte Formate sind, die die gleichen Tags verwenden, um die kommunizierten Informationen zu beschreiben und zu formatieren. Sie ähneln sich, weil sie alle aus der Standard Generalized Markup Language (SGML) hervorgehen, bei der es sich um einen Standard für die Spezifikation von Dokumentlayout und -beschreibung handelt, der allen Internet- Standards wie HTML, XML etc. vorausgeht.

Das Format XML wird verwendet, um Daten und Datenobjekte zu beschreiben. Das Format SOAP wird verwendet, um verfügbare Objekte und Dienste zu beschreiben (SOAP ist eine spezielle Anwendung von XML). Wenn man diese Informationsformate für die Kommunikation verwendet, kann die gesamte Kommunikation zwischen Client- Anwendung und Server-Komponenten über das HTTP-Protokoll stattfinden. Standardisierte Webtechnologien werden geschickt genutzt, um Rich-Client- Funktionalität über das Internet

bereitzustellen, die bisher nur in lokalen, geschlossenen Netzwerken praktikabel war. Wenn diese Anwendungen im Internet laufen können, können sie in jedem Netzwerk laufen.

Die Themen XML und SOAP sind umfangreich; es wurden ihnen diverse Bücher gewidmet. Wenn Sie ihre Funktionsweise und ihr Format verstehen wollen, sollten Sie ein diesem Thema gewidmetes Buch erwerben.[1]

Die Common Language Runtime (CLR)

Die Common Language Runtime entspricht grundsätzlich der Virtual Machine von Java. Es handelt sich um eine binäre Maschinensprache für einen Computerprozessor, der nicht wirklich existiert. CLR wurde tatsächlichen Prozessoren nachmodelliert, ist jedoch eher für die Übersetzung binärer Maschinensprache in Prozessoranweisungen gedacht. Microsoft nahm, was man aus der Erstellung einer der besten Java Virtual Machines gelernt hatte und verwendete dieses Wissen, um seine eigene Common Language Runtime zu erstellen.

Der Zweck der CLR ist, Anwendungen für die CLR anstatt für einen bestimmten Computerprozessor zu kompilieren. Der Vorteil dabei ist, dass nun jede Anwendung auf jeder Hardware laufen kann, für die es eine Version der CLR gibt. Bei der explosionsartig wachsenden Zahl von Handgeräten und deren verschiedenen Hardware-Konfigurationen ist das wichtig, um die Erstellung von Anwendungen zu erleichtern, die auf so vielen Geräten wie möglich laufen können.

Eigentlich ist das für Microsoft eine alte Idee. Obwohl wahrscheinlich damals keiner der Entwickler dabei war, die jetzt die CLR geschaffen haben, verwendete Microsoft den gleichen Ansatz bei der Entwicklung erster Kundenanwendungen wie Tabellenkalkulationen oder Textverarbeitungen. Diese Anwendungen wurden zu einem Halbbinärformat kompiliert und dann auf verschiedenen Plattformen durch eine Laufzeit- Engine ausgeführt. Wenn man Haare spalten will, könnte man sogar behaupten, dass die Idee bis zum ersten Microsoft-Produkt zurückreicht, Microsoft BASIC, da man damit die gleiche Anwendung auf allen Computern ausführen konnte, auf denen Microsoft BASIC geladen war. Reste dieses Ansatzes waren noch bis Visual Basic 5 deutlich, als Microsoft den Programmierern schließlich die Option gab, ihre VB-Anwendungen als nativ ausführbare Dateien zu kompilieren.

Die Idee mag alt sein, doch die CRL hat ein paar neue Wendungen. Eine davon nennt sich Just In Time (JIT) Compiler. Der JIT rekompiliert die Anwendung von CLR in eine nativ ausführbare Datei, wenn die Anwendung gestartet wird. Damit fällt die den meisten Virtual Machines gemeine Interpretationsebene weg. So können in die CLR kompilierte Anwendungen ebenso schnell ausgeführt werden wie in nativer Maschinensprache kompilierte Anwendungen.

Eine weiterer Wendung der CLR ist, dass sie als Ersatz für COM dienen wird. Sie wird die Arbeit mit COM deutlich erleichtern und die Interaktion zwischen in unterschiedlichen Programmiersprachen erstellten Komponenten nahtloser denn je machen. Die CLR enthält einen gemeinsamen Satz von Objekttypen, die von allen Sprachen verwendet werden, mit denen man Anwendungen für sie erstellt.

19.2 Mit verwaltetem C++ arbeiten

Ein Teil der Anziehungskraft der Programmiersprachen C und C++ liegt darin, dass sie sich direkt zu Maschinensprache kompilieren lassen. Um schnelle und effiziente Anwendungen zu erstellen, gibt es nur eine schnellere, effizientere Programmiersprache - Assembler. Mit Assembler programmieren Sie im Grunde Maschinensprache, mit einfachen Übersetzungen in Maschinensprache.

Ein weiterer Aspekt von C/C++, der sie zu sehr wünschenswerten Sprachen macht, ist, dass sie Ihnen keine Grenzen bei dem setzen, was Sie mit ihnen tun können. Sie können auf alle Aspekte des Computers zugreifen, auf denen Ihre Anwendung läuft, wenn das Betriebssystem diese Zugriffe erlaubt. So ist C/C++ ideal für die Erstellung von Gerätetreibern, um die Interaktion zwischen Betriebssystem und dem Computer hinzugefügten Geräten wie Sound- und Grafikkarten zu steuern.

Leider widerspricht der Zugriffslevel, den C/C++ seinen Programmierern gibt, der ganzen Idee hinter CLR. Die Idee hinter CLR ist, dass Anwendungen zu virtueller Maschinensprache kompiliert werden, die dann während der Ausführung der Anwendung in tatsächliche Maschinensprache umgewandelt wird. Außerdem führt die CLR die gesamte Speicherverwaltung für die Anwendungen durch, die auf ihr laufen. In C/C++ sind Sie für die Speicherverwaltung selbst zuständig.

> *Die Speicherverwaltung steuert, wie und wann Objekte und Variablen vom im Computer verfügbaren Speicher alloziert werden und wann und wie der Speicher in die Gesamtmenge verfügbaren Speichers zurückgegeben wird. In C++ erledigt man diese Aufgaben mit den Schlüsselworten new und delete. Wenn Sie mit dem Schlüsselwort new Speicher für eine Variable oder ein Objekt allozieren, müssen Sie darauf achten, diesen Speicher mit dem Schlüsselwort delete wieder freizugeben, bevor die Anwendung beendet wird. In einigen Programmiersprachen wie beispielsweise Visual Basic bleibt diese Aufgabe vor dem Programmierer verborgen.*

Die Lösung, die Microsoft für dieses Problem entwickelte, hat zwei Seiten. Zuerst wurde eine neue Programmiersprache, C# (C-sharp ausgesprochen, ein Thema für ein anderes Buch), entworfen und entwickelt, bei der es sich um eine Adaption von C++ mit Einwürfen von verschiedenen aus Java und objektorientierten Sprachen gelernten Lektionen handelt. Die zweite Lösung war verwaltetes C++.

Mit verwaltetem C++ wird Ihre Anwendung für die CLR kompiliert anstatt für native Maschinensprache. Die Speicherverwaltung findet immer noch statt, erfordert nun aber andere Schlüsselworte für die Allozierung und für die Interaktion mit CLR-Objekten.

Verwaltete C++-Anwendungen erstellen

Sie können verwaltete C++-Anwendungen auf drei Arten erstellen:

- Sie können eine verwaltete C++-Anwendung von Grund auf erstellen. Leider ist das wohl die schwierigste Herangehensweise für die Erstellung verwalteter C++- Anwendungen, da Visual Studio Ihnen keinen der hilfreichen Assistenten bietet, um das Anwendungsgerüst und die Funktionalität zusammenzubauen. Dieser Ansatz ist der herkömmliche Stil der Erstellung von C++-Anwendungen, bevor es Assistenten und andere Code erzeugende Werkzeuge gab. Sie werden heute mit dieser Methode eine einfache Anwendung erstellen.
- Sie können eine bestehende C++-Anwendung in eine verwaltete C++-Anwendung umwandeln. Das ist wahrscheinlich die einfachste Möglichkeit. Sie werden heute auch mit dieser Methode eine verwaltete C++-Anwendung erstellen.
- Um eine verwaltete C++-Anwendung oder -Komponente zu erstellen, können Sie die bestehende Komponente in einer verwalteten C++-Schale verpacken. Dieser Ansatz ermöglicht Ihnen, die bestehende C++-Komponente unberührt zu lassen und ihr einen CLR-Wrapper zu verleihen, der seine Funktionen für die Verwendung durch andere CLR-Steuerelemente und -Anwendungen exponiert. Dieser Ansatz ist für ganze Anwendungen unbrauchbar, jedoch sehr praktikabel für Komponenten. Sie werden morgen eine nicht verwaltete Komponente erstellen und mit verwaltetem C++ verpacken.

Verwaltete Datentypen

Es gibt in verwaltetem C++ zwei grundlegende verwaltete Datentypen: gc und value. Die gc-Typen sind speicherbereinigte Typen, die auf dem CLR-Heap alloziert und von der CLR-Engine verwaltet werden. Die value-Typen sind kurzlebige Typen, die auf dem Stapel oder innerhalb anderer Objekt alloziert werden und den Overhead der Speicherbereinigung nicht rechtfertigen. (Speicherbereinigung bezieht sich auf die automatische Freigabe von Objekten und Variablen zugewiesenem Speicher.)

gc-Klassen und -Strukturen

Wenn Sie in Ihrem Projekt eine gc-Klasse definieren wollen, fügen Sie der Klassendeklaration einfach das Schlüsselwort __gc hinzu:

```
__gc class MyGCClass{
public:
   int i;
   int k;
...
}
```

Wenn Sie diese Klasse in einer verwalteten C++-Anwendung verwenden, allozieren Sie mit dem Schlüsselwort new eine Instanz der Klasse:

```
MyGCClass* pMgc = new MyGCClass;
```

Sie müssen zwar nicht das Schlüsselwort delete aufrufen, um den dieser Variable zugewiesenen Speicher freizugeben, doch Sie können die sofortige Deallozierung des Speichers erzwingen, wenn Sie möchten. Allerdings können Sie das Schlüsselwort delete nicht auf einer Klasse aufrufen, die keinen benutzerdefinierten Destruktor besitzt.

Wenn Sie eigene gc-Klassen und -Strukturen erstellen, müssen Sie sich an ein paar Regeln halten:

- Sie können keine gc-Klasse aus einer nicht verwalteten Klasse ableiten und umgekehrt.
- Eine gc-Klasse kann sich nicht aus mehreren Klassen ableiten.
- gc-Klassen können keine Friend-Klassen oder -Funktionen haben.
- Sie können keinen zweiten Konstruktor für eine gc-Klasse deklarieren, genauso wenig wie Kopien der Operatoren new und delete.
- Sie können auf einer gc-Klasse nicht die Funktionen sizeof und offsetof verwenden.
- Sie können die Modifizierer const und volatile nicht in Member-Funktionen der gc- Klasse verwenden.
- Da alle gc-Typen den in CLR integrierten Operator new besitzen müssen, können Sie keine Instanz eines gc-Typs deklarieren, nur einen Zeiger darauf.

Die hier dargestellten Regeln erwähnen einige Dinge, die Sie wahrscheinlich nicht kennen, falls Sie nicht schon vor der Lektüre dieses Buchs mit der Programmiersprache C++ vertraut waren. Es handelt sich dabei um die Möglichkeit, Friend-Klassen und -Funktionen zu erstellen, eigene Klassenoperatoren, Mehrfachvererbung und Kopien von Konstruktoren. Das sind alles wichtige Aspekte und Fähigkeiten von C++, doch sie alle würden den Rahmen dieses Buchs sprengen. Dieses Buch versucht, Ihnen die Grundlagen von C++ beizubringen, doch man kann nicht alle Aspekte der Sprache als Exkurs aus dem Hauptthema abdecken. Wenn Sie mit C++ noch nicht vertraut waren, sind Ihnen bis jetzt wahrscheinlich die Grundlagen geläufig. Sie wären besser damit bedient, ein Buch über die Sprache zu erwerben, das auf alle hier erwähnten Aspekte der Sprache eingeht.[2]

Das sind keinesfalls alle Regeln, denen Sie bei der Erstellung von und der Arbeit mit gc-Klassen folgen müssen. Eine vollständige Liste der Regeln finden Sie in der Spezifikation »Managed Extensions for C++« in der MSDN-Dokumentation, die Sie zusammen mit Ihrer Ausgabe von Visual Studio oder Visual C++ erhalten haben.

value-Klassen und -Strukturen

Wenn Sie eine value-Klasse in Ihrem Projekt definieren wollen, fügen Sie der Klassendeklaration einfach das Schlüsselwort __value hinzu:

```
__value class MyValueClass{
public:
   int i;
   int k;
...
}
```

Sie können value-Typen verwenden, indem Sie entweder eine Instanz von ihnen deklarieren:

```
MyValueClass c;
```

oder indem Sie sie als nicht verwaltetes Objekt behandeln, wobei Sie sicherstellen müssen, den von dem Objekt verwendeten Speicher nach der Verwendung zu deallozieren:

```
MyValueClass* p = new MyValueClass;
...
delete p;
```

Wie bei den gc-Klassen und -Strukturen müssen Sie auch hier bestimmten Regeln für die Verwendung von value-Klassen und -Strukturen folgen:

- Sie können keine value-Klasse aus einer nicht verwalteten Klasse ableiten und umgekehrt.
- Eine value-Klasse kann sich nicht aus mehreren Klassen ableiten.
- value-Klassen können keine Friend-Klassen oder -Funktionen besitzen.
- Sie können für eine value-Klasse keinen kopierten Konstruktor deklarieren, ebenso wenig kopierte new- und delete-Operatoren.
- Sie können in einer value-Klasse nicht die Funktionen sizeof und offsetof verwenden.
- Sie können die Modifizierer const und volatile nicht in einer Member-Funktion der value-Klasse verwenden.
- Eine value-Klasse kann keine virtual-Methoden enthalten.
- Eine value-Klasse kann sich nur aus einer gc-Schnittstelle ableiten. Eine value-Klasse kann sich nicht aus einer gc-Klasse oder aus anderen value-Klassen ableiten.

Wie bei den gc-Klassen sind dies keinesfalls alle Regeln, denen Sie bei der Erstellung und Verwendung von value-Klassen und -Strukturen folgen müssen. Eine vollständige Liste der Regeln finden Sie in der MSDN-Spezifikation.

Verwaltete Zeiger

Wenn eine Anwendung unter der CLR läuft, die eine integrierte Speicherbereinigung besitzt, könnte man denken, dass die Zeiger wegfallen oder zumindest wie in Visual Basic verborgen sind. Aber in Wirklichkeit sind sie komplizierter geworden. Sie haben jetzt zwei Arten von Zeigern: nicht verwaltete und speicherbereinigte.

Die nicht verwalteten Zeiger sind Ihre Standard-C/C++-Zeiger, die Sie selbst verwalten müssen. Die verwalteten Zeiger besitzen zwei Schlüsselworte, die anzeigen, ob der Zeiger etwas auf dem speicherbereinigten CLR-Heap zeigen kann. Es handelt sich dabei um die Schlüsselworte __gc (speicherbereinigt, bereits besprochen) und __nogc (no garbage collection - keine Speicherbereinigung). Eins dieser beiden Schlüsselworte muss unmittelbar vor dem Zeigeroperator (*) platziert werden:

```
G __gc* pG;
V __nogc* pV;
```

Sie können das Schlüsselwort __gc für Zeiger verwenden, die auf gc-Klassen und Strukturen zeigen, die auf dem CLR-Heap alloziert wurden. Außerdem kann es verwendet werden, um auf value-Klassen oder -Strukturen zu zeigen, bei denen es sich um Elemente von gc-Klassen oder -Strukturen handelt:

```
__value struct Val{
    int x;
    int y;
}
__gc class Cgc
{
public:
   // Die value-Struktur Val wird als public-Element deklariert
   Val m_vMem;
}
// Später im Code der Anwendung ...
Cgc __gc* pCgc = new Cgc;
Val __gc* pVal = &(pCgc->m_vMem);
```

Beim Erzeugen von Zeigern auf value-Klassen oder -Strukturen, die keine Elemente von gc-Klassen oder -Strukturen sind, müssen Sie das Schlüsselwort __nogc verwenden:

```
Val __nogc* pVal = new Val;
```

gc-Zeiger haben den Zweck, der CLR die Verfolgung aller Verweise im CLR-Heap zu ermöglichen. Das ist notwendig, damit die CLR eine korrekte Speicherbereinigung ausführen kann.

Verwaltete Stringliterale

Wenn Sie in einer verwalteten C++-Anwendung Stringliterale verwenden, ist eine neue, verwaltete Stringliterale effizienter als eine Standard-C++-Stringliterale. Der Grund dafür ist, dass Standard-C++-Stringliterale aus ASCII-Zeichen bestehen, mit denen verwaltete Anwendungen nicht umgehen können. Wegen der Inkompatibilität mit verwalteten Objekten und Anwendungen müssten Standard-C++-Stringliterale in eine Form umgewandelt werden, die verwaltete Objekte handhaben können. Sie können diesen Overhead vermeiden, indem Sie stattdessen verwaltete Stringliterale verwenden.

Um verwaltete Stringliterale zu verwenden, müssen Sie angeben, dass Sie mscorlib.dll und das Objekt System::String verwenden:

```
#using <mscorlib.dll>
using System::String;
```

> *Die Präprozessor-Direktive #using und das Schlüsselwort using sind zwei neue Elemente in C++, die im folgenden Abschnitt, ».NET-Objekte in verwalteten C++-Anwendungen«, erklärt werden.*

Um eine verwaltete Stringliterale zu erzeugen, deklarieren Sie einen Zeiger auf den String-Typ und weisen ihm die Stringliterale zu:

```
String *s = "This is a managed string literal.";
```

Sie können auch eine verwaltete UNICODE-Stringliterale erzeugen, indem Sie das Präfix L einfügen:

```
String *s2 = L"This is a managed, wide-character string literal.";
```

Für die beste Performance erzeugen Sie mit dem neuen Präfix S einen neuen Stringliteral- Typ:

```
String *s3 = S"This is a new, more efficient managed string literal.";
```

Schlüsselworte in verwaltetem C++

In die verwalteten C++-Erweiterungen wurden einige neue Schlüsselworte eingeführt (siehe Tabelle 19.1).

Schlüsselwort	Beschreibung
__abstract	Wird vor die Klassendeklaration gestellt, um anzugeben, dass es sich bei der Klasse um eine Basisklasse handelt, die nicht direkt instanziiert werden kann. Nur Abkömmlinge von als __abstract gekennzeichneten Klassen können in Anwendungen verwendet werden.
__box	Wird verwendet, um eine Kopie eines __value-Objekts zu erstellen und auf dem CLR-Heap zu platzieren
__delegate	Wird verwendet, um einen Funktionszeiger auf eine Funktion in einer __gc-Klasse zu deklarieren
__event	Wird in einer Klassendeklaration verwendet, um eine Ereignismethode in der Klasse zu deklarieren
__identifier	Wird verwendet, um auf nicht verwaltete C++-Objekte zuzugreifen, die Schlüsselworte (Identifier) als Namen verwenden
__pin	Wirt verwendet, um einen Zeiger auf ein verwaltetes, speicherbereinigtes Objekt zu deklarieren. Diese Art von Zeiger gibt an, dass das Objekt, auf das er zeigt, nicht während der Speicherbereinigung durch die CLR verschoben werden darf.
__property	Wird in der Klassendeklaration verwendet, um Eigenschaften der Klasse zu deklarieren
__sealed	Wird in der Klassendeklaration verwendet. Das Schlüsselwort kennzeichnet eine Methode in der Klasse, die in abgeleiteten Klassen nicht überschrieben werden kann oder um die ganze Klasse so zu kennzeichnen, dass sie nicht von anderen Klassen ererbt werden kann.
__try_cast	Wird verwendet, um festzulegen, dass eine Variable in einen anderen Objekttyp umgewandelt werden soll. Kann die Variable nicht in den angegebenen Typ umgewandelt werden, wird eine Ausnahme ausgelöst.
__typeof	Wird verwendet, um System::Typ eines ihm übergebenen Objekts zurückzuliefern.

Tabelle 19.1: Verwaltete Erweiterungen für C++-Schlüsselworte

.NET-Objekte in verwalteten C++-Anwendungen

Ein Vorteil bei der Erstellung verwalteter C++-Anwendungen ist die eingebaute Objekt- Bibliothek in der .NET-Plattform. Ihre Anwendung kann auf eine umfangreiche Liste von Objekten zugreifen. Diese eingebauten Objekte stellen eine bedeutende Menge von Funktionalität bereit, die in jeder .NET-Anwendung verwendet werden kann, nicht nur in verwalteten C++-Anwendungen.

Allerdings können Sie nicht einfach anfangen, diese Objekte in Ihrem Code zu verwenden. Sie müssen ein paar Dinge beachten:

Die Präprozessor-Direktive #using

Am Tag 13 haben Sie die Präprozessor-Direktive #import kennen gelernt, die Microsoft in Visual C++ aufgenommen hat, um COM-Objekte ins Projekt zu importieren und Ihrer Anwendung zur Verfügung zu stellen. Die Präprozessor-Direktive #using ist im Grunde das Gleiche, doch sie kann nur DLLs und eigens für die CLR kompilierte Objekte importieren. In Zukunft wird man mit der Direktive #using auch ausführbare Dateien importieren können, die für die CLR kompiliert wurden, aber diese Fähigkeit ist noch nicht verfügbar.

Die wichtigste DLL, die Sie in die meisten Ihrer verwalteten C++-Anwendungen importieren sollten, ist mscorlib.dll, die viele der Core-Objekte enthält, die Teil der .NET-Plattform sind. Sie verwenden diese Direktive genau wie die #import-Direktive am Anfang Ihrer Quellcode-Dateien:

```
#using <mscorlib.dll>
```

Die Direktive using namespace

Während die Präprozessor-Direktive #using neu in verwaltetem C++ ist, stellt die Direktive using einen Teil der Sprachspezifikation von C++ dar. Die Direktive using ermöglicht Ihnen, einen Namespace und verschachtelte Namespace-Bezeichner festzulegen, sodass Sie Objekte aus dem Namespace verwenden können, ohne den vollständig qualifizierten Namen der Objekte zu benötigen. Betrachten Sie beispielsweise folgenden Code:

```
#using <mscorlib.dll>
using namespace System
main()
{
  Console::Write("Hello World.");
}
```

Wenn Sie die Direktive für die Verwendung des Namespace System weggelassen hätten, müsste der Code folgendermaßen aussehen:

```
#using <mscorlib.dll>
main()
{
  System::Console::Write("Hello World.");
}
```

Die Direktive using muss nicht am Anfang des Quellcodes verwendet werden, sondern kann überall eingefügt werden, wo Sie einen Namespace verwenden möchten, um die Objektnamen in diesem Code-Abschnitt nicht vollständig qualifizieren zu müssen.

Die Deklaration using

Sie finden in verwalteten C++-Anwendungen auch ab und an die Deklarationsform von using. Diese Verwendung des Schlüsselworts using ist ebenfalls nicht neu von den verwalteten C++-Erweiterungen eingeführt worden, sondern Teil der Sprache C++. Die Deklaration using ermöglicht Ihnen, einen Objekttyp oder eine Funktion direkt innerhalb des Kontexts zu verwenden, in dem Sie die Verwendung festgelegt haben. Ein Beispiel dafür ist die Verwendung von Stringliteralen:

```
#using <mscorlib.dll>
using System::String;
main()
{
  String *s = "This is a managed string literal.";
}
```

Ohne die Deklaration using müsste der Code so aussehen:

```
#using <mscorlib.dll>
main()
{
  System::String *s = "This is a managed string literal.";
}
```

Eine weitere Verwendungsmöglichkeit der Deklaration using ist der Zugriff auf bestimmte Funktionen in einer Basisklasse von einer abgeleiteten Klasse aus:

```
class B
```

```
{
public:
   void f(char c);
};
class D : B
{
public:
   using B::f;
   void f(int i)
   {
      f('a');
   }
};
main ()
{
   D d;
   // Abgeleitete Methode aufrufen
   d.f(1);
   // Basismethode aufrufen
   d.f('b');
}
```

oder für den direkten Zugriff auf eine Klassenmethode, ohne die Klasse angeben zu müssen:

```
class C
{
public:
   void f(char c);
};
void main()
{
   using C::f;
   f('a');
}
```

In all diesen Situationen verwenden Sie die Deklaration using, um eine Abkürzung zu einem bestimmten Objekt oder einer bestimmten Methode zu erzeugen.

Einige Standard-.NET-Objekte

Wenn Sie die MSDN-Dokumentation öffnen und sich die .NET-Framework- Klassenbibliothek ansehen, finden Sie eine Liste von fast 100 Namespaces, die Sie in Ihrer Anwendung verwenden können. In jedem Namespace finden Sie mehrere Klassen, die Funktionalität für bestimmte Dinge enthalten, die in das Hoheitsgebiet des Namespace fallen. Die meisten der Namespaces sind recht selbsterklärend, was die in den eingeschlossenen Klassen enthaltene Art der Funktionalität angeht. Die Namespace- Gruppe System.Data enthält beispielsweise Klassen für Datenbank-Funktionalität. Genauer gesagt, enthält der Namespace System.Data.OleDb Klassen, die die OLEDB- Schnittstelle für den Zugriff auf Datenquellen verkapseln. In Tabelle 19.2 sind einige der Namespaces auf höherer Ebene aufgelistet.

Namespace	Inhalt
System	Basisklassen für häufig genutzte Ressourcen, Datentypen, Ereignisse, Schnittstellen, Attribute und Ausnahmebehandlung
System.Data	Klassen, die die ADO.NET-Architektur ausmachen
System.Drawing	Klassen, die grundlegende Zeichenfunktionalität bereitstellen. Dieser Namespace verkapselt einen Großteil der GDI-Funktionalität von Windows.
System.Net	Klassen, die Funktionalität für Netzwerkkommunikation bereitstellen
System.Security	Klassen, die einen großen Teil der zu Grunde liegenden CLR-Sicherheit verkapseln, wie beispielsweise Berechtigungen und Zugriffsfunktionalität
System.Threading	Klassen, die einen großen Teil der Threading-Funktionalität verkapseln, unter anderem

	die Synchronisationsobjekte
System.Web	Klassen, die Browser-/Server-Kommunikation ermöglichen
System.Xml	Klassen, die für die Arbeit mit XML-Daten verwendet werden

Tabelle 19.2: .NET-Framework-Namespaces

Wenn Sie in verwaltetem C++ einen Namespace verwenden, müssen Sie die Namespace- Bereiche (sowie die Klassen innerhalb des Namespace) mit dem Scope-Resolution- Operator (zwei Doppelpunkte, ::) unterteilen, obwohl die Dokumentation eine Unterteilung durch Punkte (.) angibt:

```
using namespace System::Net;
```

.NET-Objekte in MFC-Anwendungen

Die einfachste Möglichkeit, GUI-verwaltete C++-Anwendungen zu erstellen, ist, eine MFC-Anwendung zu produzieren und dann in die CLR zu portieren. Das geschieht durch eine einfache Änderung einiger Kompilierungsoptionen des Projekts. Um die Änderungen durchzuführen, folgen Sie diesen Schritten:

1. Öffnen Sie ein bestehendes MFC-Projekt.

2. Klicken Sie in der Projektmappen-Explorer-Ansicht mit der rechten Maustaste auf den Projektknoten am oberen Ende des Dateibaums und wählen Sie Eigenschaften aus dem Kontextmenü.
3. Wählen Sie Alle Konfigurationen aus dem Kombinationsfeld Konfiguration im oberen Teil des Dialogfelds Projekteigenschaften.
4. Wählen Sie C/C++, allgemein im linken Teil des Dialogfelds und legen Sie als Als verwaltet kompilieren-Option Assemblyuntertützung (/clr) fest. Für Debuginformationsformat legen Sie Programmdatenbank (/Zi) fest, wie es Abbildung 19.2 zeigt.

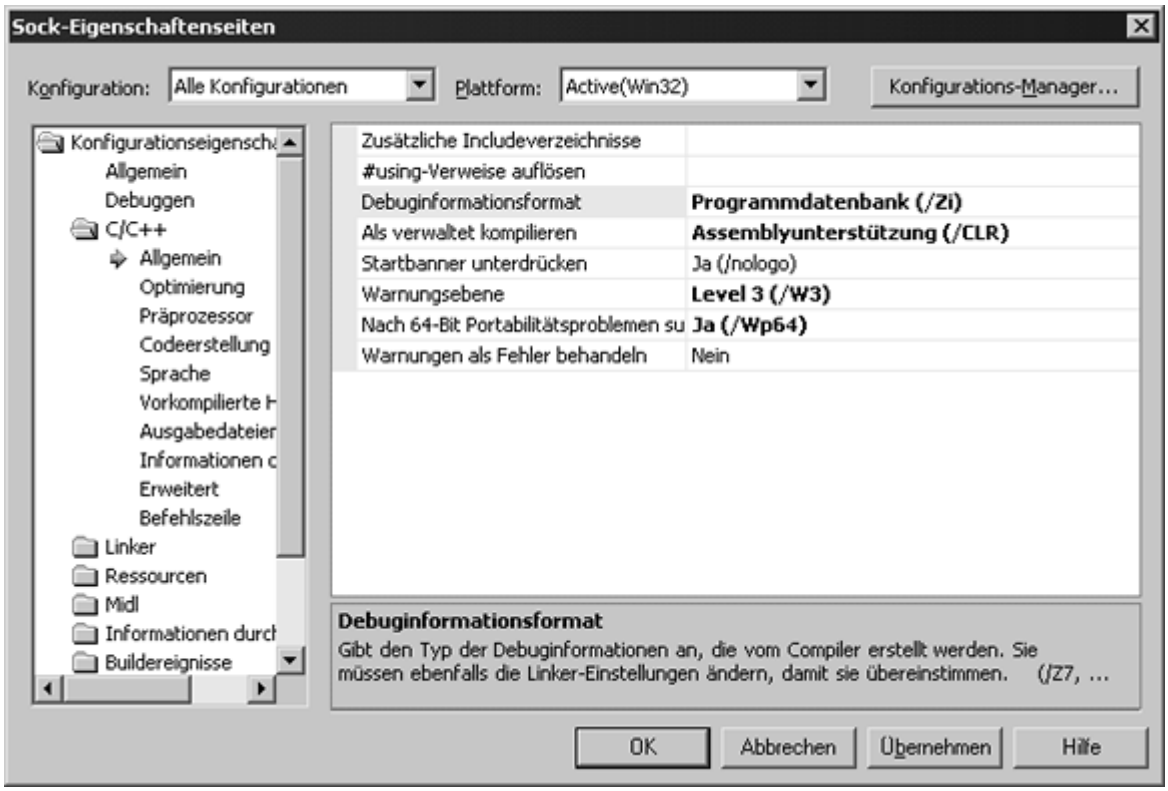


Abbildung 19.2: Festlegen, dass ein Projekt für CLR kompiliert werden soll

5. Wählen Sie C/C++, Codeerstellung im linken Teil des Dialogfelds und legen Sie für Vollständige Laufzeitüberprüfungen die Option Standard fest. Legen Sie Nein für Minimale Neuerstellung aktivieren

fest.

6. Klicken Sie auf OK, um diese Änderungen in Ihr Projekt zu übernehmen. Damit wandeln Sie Ihr Projekt so um, dass es für die CLR kompiliert wird.

Wenn es Zeit wird, in Ihre MFC-Anwendung einige .NET-Foundation-Klassen aufzunehmen, müssen Sie eine kleine Einzelheit beachten. Die Debug-Version von MFC definiert den Operator new neu, sodass er allozierten Speicher im Auge behalten und Speicherlecks in Ihrer Anwendung finden kann. Das ist mit den .NET-Objekten inkompatibel. Wenn Sie Instanzen von .NET-Objekten oder anderen verwalteten Objekten (zum Beispiel speicherbereinigte Klassen oder Strukturen, die Sie definiert haben) erzeugen, müssen Sie daher die Direktiven #pragma push_macro und pop_macro zusammen mit der Direktive #undef verwenden, um die Neudefinition des Operators new für die Verwendung mit verwalteten Objekten rückgängig zu machen. Dieser Code sieht ungefähr so aus:

```
#ifdef _DEBUG
#pragma push_macro("new")
#undef new
#endif
String* s;
s = new String("This is a managed string.");
#ifdef _DEBUG
#pragma pop_macro("new")
#endif
```

19.3 Eine verwaltete C++-Anwendung schreiben

Für das heutige Beispiel erstellen Sie eine einfache verwaltete C++-Anwendung. Diese Anwendung im Zeichenmodus extrahiert verschiedene Informationen über das System, auf dem die Anwendung läuft, und zeigt sie an. Die Anwendung bittet außerdem den Benutzer, zwei Zahlen einzugeben und addiert dann diese beiden Zahlen.

Um das Anwendungsgerüst für die heutige Anwendung zu erstellen, erzeugen Sie ein neues Verwaltete C++-Anwendungs-Projekt und nennen es ManagedEnv.

Beim heutigen Projekt wird Ihnen nicht der Anwendungs-Assistent angezeigt, an den Sie gewöhnt sind. Stattdessen gelangen Sie direkt ins Projekt, in dem schon ein paar Dateien erzeugt sind. Wenn Sie sich die Datei ManagedEnv.cpp ansehen, stellen Sie fest, dass die Funktion main mit der einfachen »Hello World«-Anwendung erstellt wurde.

Die Umgebungsinformationen-Klasse erzeugen

Um mit dieser Anwendung zu beginnen, erstellen Sie eine neue speicherbereinigte Klasse, die Umgebungsinformationen über den Computer und das Betriebssystem anzeigt, auf dem die Anwendung läuft. Um die Klasse anzulegen, folgen Sie diesen Schritten:

1. Fügen Sie auf die gleiche Methode wie an den vergangenen Tagen eine neue allgemeine Klasse in das Projekt ein. Nennen Sie die Klasse CEnvironmentInfo.

2. Fügen Sie wie in Listing 19.1 gezeigt das Schlüsselwort __gc in die Klassendeklaration ein.

Listing 19.1: Die Klassendeklaration von CEnvironmentInfo

```
 1: #using <mscorlib.dll>
 2:
 3: using namespace System;
 4:
 5: __gc class CEnvironmentInfo
 6: {
 7: public:
 8:     CEnvironmentInfo(void);
 9:     ~CEnvironmentInfo(void);
10: };
```

3. Nehmen Sie in die Klasse CEnvironmentInfo eine neue Funktion auf, um die Version des Betriebssystems anzuzeigen. Legen Sie den Rückgabetyp als void, den Funktionsnamen als ShowOSVersion und den Zugriff als public fest.
4. Fügen Sie in die Funktion ShowOSVersion den Code aus Listing 19.2 ein.

Listing 19.2: Die Funktion ShowOSVersion

```
 1: void CEnvironmentInfo::ShowOSVersion(void)
 2: {
 3:   String* strLine;
 4:
 5:   // Aktuelle Betriebssystem-Version ermitteln
 6:   strLine = String::Concat(S"OS-Version: ",
 7:                            Environment::get_OSVersion());
 8:   // Betriebssystem-Version anzeigen
 9:   Console::WriteLine(strLine);
10: }
```

Listing 19.2 nutzt die get_OSVersion-Methode der .NET-Klasse Environment, um die Betriebssystem-Version auf dem Computer zu ermitteln, auf dem die Anwendung läuft. OSVersion ist eigentlich als Eigenschaft von Environment spezifiziert, also würde man diese Methode in anderen Programmiersprachen nicht direkt aufrufen. Da es sich hier allerdings um C++ handelt, sind Ihnen die Interna von Klasseneigenschaften nicht verborgen, also müssen Sie die tatsächliche Methode aufrufen, die die Funktionalität dieser Eigenschaft bereitstellt.

Der Anzeigestring wird mit der Klasse System.String erstellt. Diese Klasse besitzt nicht die Verkettungsoperatoren, mit denen Sie in der Klasse CString gearbeitet haben. Damit ist sie näher an der Stringfunktionalität in Standard-C, in dem Sie die Methode Concat aufrufen, um zwei Strings zu einem zu kombinieren.

Nachdem der String erstellt wurde, zeigt die WriteLine-Methode der Klasse Console ihn an. Die Klasse Console verkapselt einen großen Teil der Anzeigefunktionalität im Zeichenmodus.

Um die Funktionalität hinzuzufügen, die weitere Informationen über die Umgebung anzeigt, fahren Sie mit diesen Schritten fort:

1. Nehmen Sie vier weitere Funktionen in die Klasse CEnvironmentInfo auf, um zusätzliche Umgebungsinformationen anzuzeigen. Verwenden Sie die Funktionsnamen und rufen Sie die Methoden der Klasse Environment auf (siehe Tabelle 19.3). Legen Sie den Zugriff für alle Funktionen als public und ihren Rückgabetyp als void fest.

Funktionsname	Zu verwendende Umgebungseigenschaftsmethode
ShowCurDirectory	Environment::get_CurrentDirectory
ShowMachineName	Environment::get_MachineName
ShowUserName	Environment::get_UserName
ShowDomainName	Environment::get_UserDomainName

Tabelle 19.3: Funktionen und aufzurufende Methoden für die Beispielanwendung

2. Nehmen Sie eine letzte Funktion in die Klasse CEnvironmentalInfo auf. Legen Sie den Rückgabetyp als void, den Namen als ShowClrVersion und den Zugriff als public fest. Fügen Sie in diese Funktion den Code aus Listing 19.3 auf.

Listing 19.3: Die Funktion ShowClrVersion

```
 1: void CEnvironmentInfo::ShowClrVersion(void)
 2: {
 3:   String* strLine;
 4:
 5:   // Version der CLR-Umgebung ermitteln
 6:   Version* verClr = Environment::get_Version();
 7:   // Version der CLR in einem String formatieren
 8:   strLine = String::Concat(S"CLR-Version: ",
 9:                            verClr->ToString());
10:   // CLR-Version anzeigen
11:   Console::WriteLine(strLine);
12: }
```

Diese Funktion erzeugt eine Instanz der Klasse Version, indem Sie die Eigenschaftsmethode get_Version der Klasse Environment aufruft. Danach verwendet sie die Methode ToString der Klasse Version, um diese CLR-Information in einen anzeigbaren String umzuwandeln.

3. Modifizieren Sie die main-Funktion der Anwendung gemäß Listing 19.4, indem Sie die Klasse CEnvironmentInfo instanziieren und dann ihre Methoden aufrufen, um die Systeminformationen anzuzeigen.

Listing 19.4: Die modifizierte main-Funktion

```
 1: #include "stdafx.h"
 2: #include "EnvironmentInfo.h"
 3:
 4: #using <mscorlib.dll>
 5: #include <tchar.h>
 6: using namespace System;
 7:
 8: // Dies ist der Einstiegspunkt für die Anwendung
 9: int _tmain(void)
10: {
11:   // TODO: Ersetzen Sie den Beispielcode durch Ihren eigenen
12:   // Code.
13:   CEnvironmentInfo* eiInfo = new CEnvironmentInfo;
14:
15:   // Aktuelles Verzeichnis anzeigen
16:   eiInfo->ShowCurDirectory();
17:   // Rechnernamen anzeigen
18:   eiInfo->ShowMachineName();
19:   // Betriebssystem-Version anzeigen
20:   eiInfo->ShowOSVersion();
21:   // Benutzernamen anzeigen
22:   eiInfo->ShowUserName();
23:   // Domainnamen anzeigen
24:   eiInfo->ShowDomainName();
25:   // CLR-Version anzeigen
26:   eiInfo->ShowClrVersion();
27:
28:   return 0;
29: }
```

Sie können Ihre Anwendung nun kompilieren und ausführen. Wie in Abbildung 19.3 zu sehen, werden die Systeminformationen angezeigt. Sie sollten die Anwendung ohne Debugging laufen lassen, damit sie nach

dem Beenden anhält und darauf wartet, dass Sie eine Taste drücken, um das Fenster zu schließen.

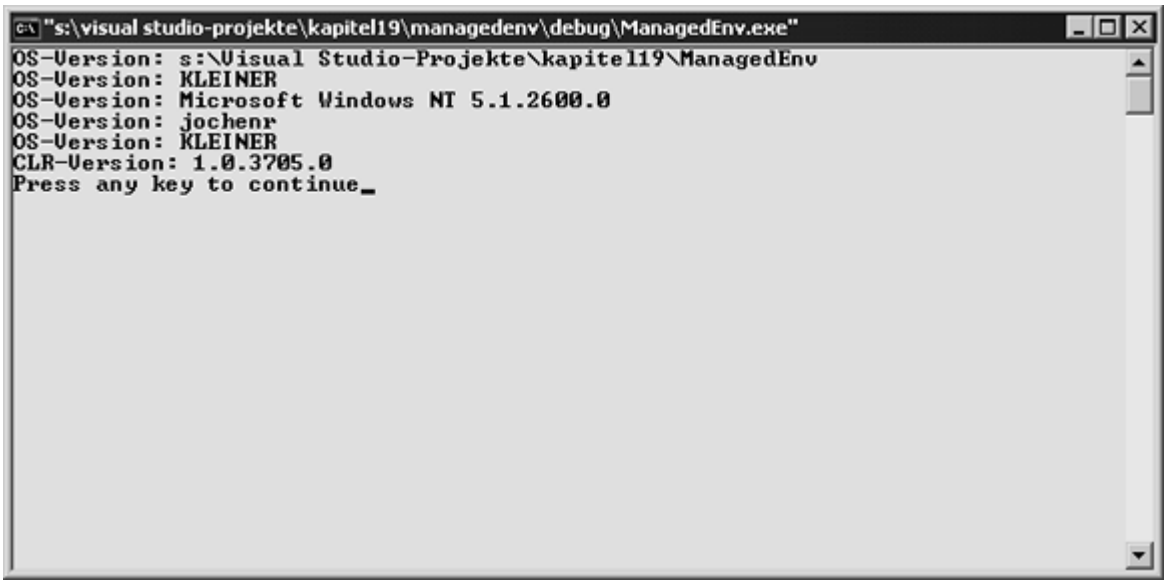


Abbildung 19.3: Systeminformationen anzeigen

Die Additionsklasse erstellen

Um die Eingabe- und Additionsanteile dieser Anwendung zu erstellen, erzeugen Sie eine neue speicherbereinigte Klasse, die den Benutzer um die Eingabe zweier Zahlen bittet, diese addiert und dann das Ergebnis anzeigt. Um die Klasse zu erstellen, folgen Sie diesen Schritten:

1. Nehmen Sie mit den gleichen Schritten wie an den vergangenen Tagen eine neue Klasse in das Projekt auf. Nennen Sie die Klasse CAddNumbers.

2. Modifizieren Sie die Klassendeklaration gemäß Listing 19.5 und fügen Sie das Schlüsselwort __gc ein.

Listing 19.5: Die Klassendeklaration von CAddNumbers

```
 1: #using <mscorlib.dll>
 2:
 3: using namespace System;
 4:
 5: __gc class CAddNumbers
 6: {
 7: public:
 8:     CAddNumbers(void);
 9:     ~ CAddNumbers(void);
10: };
```

Um der Klasse Member-Variablen hinzuzufügen, in denen die beiden vom Benutzer eingegebenen Werte gespeichert werden, fahren Sie mit diesen Schritten fort:

1. Nehmen Sie zwei Member-Variablen in die Klasse CAddNumbers auf. Legen Sie die Variablentypen als int und den Zugriff als private fest. Nennen Sie eine Variable m_iNumber1 und die andere m_iNumber2.

2. Nehmen Sie dann eine neue Funktion in die Klasse CAddNumbers auf, um die erste Zahl vom Benutzer abzufragen. Legen Sie den Rückgabetyp als void, den Namen als GetFirstNumber und den Zugriff als public fest. Fügen Sie in die Funktion den Code aus Listing 19.6 ein.

Listing 19.6: Die Funktion GetFirstNumber

```
 1: void CAddNumbers::GetFirstNumber(void)
```

```
 2: {
 3:   String* strIn;
 4:
 5:   // Den Benutzer um eine Zahl bitten
 6:   Console::Write(S"Geben Sie eine Zahl ein: ");
 7:   // Benutzereingabe lesen
 8:   strIn = Console::ReadLine();
 9:   // Eingabe in Zahl umwandeln
10:   m_iNumber1 = Convert::ToInt32(strIn);
11: }
```

Diese Funktion verwendet die ReadLine-Methode der Klasse Console, um zu lesen, was der Benutzer eingetippt hat. Wenn es sich bei der Benutzereingabe um einen String handelt, wandelt die ToInt32-Methode der Klasse Convert den eingegebenen Wert in eine Zahl um und speichert ihn in der ersten der beiden Member-Variablen.

3. Nehmen Sie eine neue Funktion in die Klasse CAddNumbers auf, um die zweite Zahl vom Benutzer abzufragen. Legen Sie den Rückgabetyp als void, den Namen als GetSecondNumber und den Zugriff als public fest. Fügen Sie in die Funktion den Code aus Listing 19.7 ein.

Listing 19.7: Die Funktion GetSecondNumber

```
 1: void CAddNumbers::GetSecondNumber(void)
 2: {
 3:   String* strIn;
 4:
 5:   // Den Benutzer um eine Zahl bitten
 6:   Console::Write(S"Geben Sie eine Zahl ein: ");
 7:   // Benutzereingabe lesen
 8:   strIn = Console::ReadLine();
 9:   // Eingabe in Zahl umwandeln
10:   m_iNumber2 = Int32::Parse(strIn);
11: }
```

Diese Funktion verwendet die Parse-Methode der Klasse Int32, um die vom Benutzer eingegebene Ziffernfolge in eine Zahl umzuwandeln. Ansonsten entspricht diese Funktion der in Listing 19.6.

4. Fügen Sie eine neue Funktion in die Klasse CAddNumbers ein, um die beiden Zahlen zu addieren und das Ergebnis anzuzeigen. Legen Sie den Rückgabetyp als void, den Namen als AddNumbers und den Zugriff als public fest. Nehmen Sie in die Funktion den Code aus Listing 19.8 auf.

Listing 19.8: Die Funktion AddNumbers

```
 1: void CAddNumbers::AddNumbers(void)
 2: {
 3:   int iResult;
 4:   String* strMsg;
 5:
 6:   // Die beiden Zahlen addieren
```

```
 7:   iResult = m_iNumber1 + m_iNumber2;
 8:   // Die erste eingegebene Zahl auf das Anzeigen vorbereiten
 9:   strMsg = String::Concat(S"Erste eingegebene Zahl: ",
10:                           Convert::ToString(m_iNumber1));
11:   // Erste eingegebene Zahl anzeigen
12:   Console::WriteLine(strMsg);
13:   // Die zweite eingegebene Zahl auf das Anzeigen vorbereiten
14:   strMsg = String::Concat(S"Zweite eingegebene Zahl: ",
15:                           Convert::ToString(m_iNumber2));
16:   // Zweite angegebene Zahl anzeigen
17:   Console::WriteLine(strMsg);
18:   // Ergebnis auf das Anzeigen vorbereiten
19:   strMsg = String::Concat(S"Ergebnis der Addition: ",
20:                           iResult.ToString());
21:   // Ergebnis der Addition anzeigen
22:   Console::WriteLine(strMsg);
23: }
```

Listing 19.8 addiert zuerst die beiden Variablen. Danach zeigt es die beiden vom Benutzer eingegebenen Zahlen an. Um diese beiden Zahlen in Strings umzuwandeln, übergibt sie der ToString-Methode der Klasse Convert die umzuwandelnde Variable. Dann erstellt sie die Strings, um das Ergebnis anzuzeigen; hier jedoch haben wir die ToString-Methode der Klasse Int32 verwendet, um zwei verschiedene Methoden der Umwandlung eines Integer in einen String zu zeigen.

5. Um die Anwendung fertig zu stellen, modifizieren Sie die main-Funktion, indem Sie die fett gedruckten Zeilen aus Listing 19.9 einfügen.

Listing 19.9: Die modifizierte main-Funktion

```
 1: #include "stdafx.h"
 2: #include "EnvironmentInfo.h"
 3: #include "AddNumbers.h"
 4:
 5: #using <mscorlib.dll>
 6: #include <tchar.h>
 7: using namespace System;
 8:
 9: // Dies ist der Einstiegspunkt für die Anwendung
10: int _tmain(void)
11: {
12:   // TODO: Ersetzen Sie den Beispielcode durch Ihren
13:   // eigenen Code.
14:   CEnvironmentInfo* eiInfo = new CEnvironmentInfo;
15:   CAddNumbers* anNbrs = new CAddNumbers;
16:   // Aktuelles Verzeichnis anzeigen
17:   eiInfo->ShowCurDirectory();
18:   // Rechnernamen anzeigen
19:   eiInfo->ShowMachineName();
20:   // Betriebssystem-Version anzeigen
21:   eiInfo->ShowOSVersion();
22:   // Benutzernamen anzeigen
23:   eiInfo->ShowUserName();
24:   // Domainnamen anzeigen
25:   eiInfo->ShowDomainName();
26:   // CLR-Version anzeigen
27:   eiInfo->ShowClrVersion();
```

```
28:
29:    Console::WriteLine("");
30:
31:    // Erste zu addierende Zahl abfragen
32:    anNbrs->GetFirstNumber();
33:    // Zweite zu addierende Zahl abfragen
34:    anNbrs->GetSecondNumber();
35:    // Die beiden Zahlen addieren
36:    anNbrs->AddNumbers();
37:    return 0;
38: }
```

Wenn Sie Ihre Anwendung nun kompilieren und ausführen, zeigt sie nicht nur Informationen über Ihr System an, sondern bittet Sie auch noch um zwei Zahlen, die sie dann addiert (siehe Abbildung 19.4).

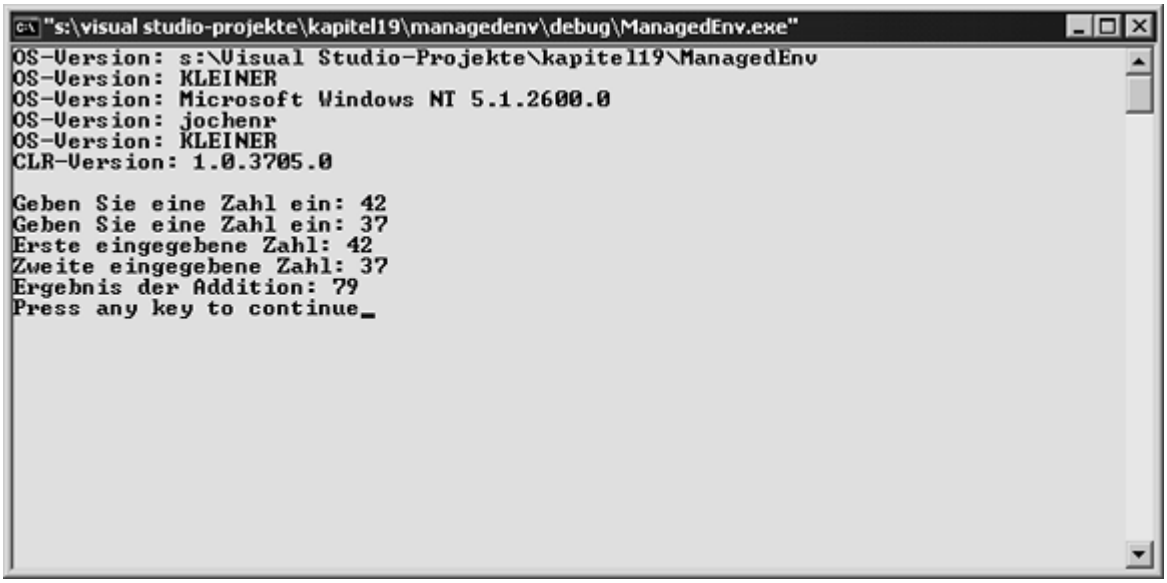


Abbildung 19.4: Systeminformationen anzeigen und zwei eingegebene Zahlen addieren

19.4 Zusammenfassung

Heute haben Sie gelernt, wie man verwaltete Erweiterungen für C++ verwendet, um Anwendungen für die neue Common Language Runtime (CLR) zu erstellen, die den Aspekt der Plattformportabilität für die .NET-Plattform von Microsoft bereitstellt. Sie haben einige der neuen Schlüsselworte kennen gelernt und gesehen, wie man einige der in CLR integrierten Klassen verwendet. Sie haben auch gelernt, wie einfach Sie Ihre MFC-Anwendungen portieren können, sodass sie auf der CLR laufen, ohne großartige Änderungen am Code vornehmen zu müssen.

An den nächsten beiden Tagen werden Sie lernen, wie man die Active Template Library (ATL) verwendet, um Komponenten zu konstruieren, die auf der Server-Seite von .NET- Anwendungen laufen, und wie man einen verwalteten C++-Wrapper um diese Komponenten herum erstellt. Sie werden außerdem lernen, wie man mit in anderen CLR- Sprachen wie C# und Visual Basic.NET erstellten Komponenten interagiert.

19.5 Workshop

Fragen und Antworten

Frage:
Warum fühlt sich die Erstellung verwalteter C++-Anwendungen wie ein Rückschritt in die Zeit an, bevor es all die Werkzeuge und Assistenten gab, die mir dabei helfen, meine Anwendung zu erstellen?

Antwort:

Die Werkzeuge und Assistenten sind aus mehreren Gründen nicht verfügbar, um Ihnen bei der Erstellung verwalteter C++-Anwendungen zu helfen:

- Verwaltete Erweiterungen für C++ waren als ein Mittel dafür gedacht, bestehende Anwendungen und Funktionalität auf die .NET-Plattform zu bringen, ohne die Anwendung in einer anderen Sprache neu zu schreiben.
- Es ist beabsichtigt, ASP+ als .NET-Werkzeug für die Erstellung von Benutzerschnittstellen einzusetzen und VB.NET, C# und verwaltetes C++ nur noch für die Erstellung serverseitiger Komponenten einzusetzen. Das heißt nicht, dass Sie mit diesen Sprachen keine GUI-Anwendungen erstellen können, doch sie sind eher für den Gebrauch auf Server-Seite gedacht. So gesehen ist der Hauptgrund für die Verwendung von C++ die Geschwindigkeit der nativen Ausführung. Obwohl die CLR diese Geschwindigkeit fast erreicht, werden die meisten Entwickler vorziehen, ihre Komponenten in nicht verwaltetem C++ zu erstellen und dann einen verwalteten Wrapper um die Komponenten zu erzeugen, der von der CLR verwendet wird.

Frage:
Wie kann ich verwaltetes und nicht verwaltetes C++ zusammen in meiner Anwendung einsetzen?

Antwort:
Auf die gleiche Art, auf die Sie die Projekteigenschaften verwenden, um das gesamte MFC-Projekt als für die CLR zu kompilieren zu kennzeichnen, können Sie auch einzelne Dateien im Projekt kennzeichnen. So können Sie Teile Ihrer Anwendung vermischen und manche Komponenten als verwaltete kennzeichnen, während andere nicht verwaltet bleiben. Für gewöhnlich sollten Sie Wrapper- Komponenten in eigenen verwalteten Dateien hinzufügen, um den Zugriff auf nicht verwaltete Komponenten von anderen CLR-Objekten und -Anwendungen aus zu gewährleisten. Wie Sie das tun, lernen Sie an den nächsten zwei Tagen.

Quiz

1. Wie kennzeichnen Sie eine Klasse für die Speicherbereinigung?

2. Mit welchen beiden Methoden können Sie einen verwalteten String in einen Integer umwandeln?
3. Wie kennzeichnen Sie eine verwaltete Klasse oder Struktur so, dass für sie keine Speicherbereinigung durchgeführt wird?
4. Mit welcher Methode geben Sie Text in einem DOS-Fenster aus?
5. Wie importieren Sie eine CLR-DLL oder Objektdatei, um sie in einer verwalteten C++-Anwendung zu verwenden?

Übung

Modifizieren Sie die heutige Anwendung so, dass der Benutzer festlegen kann, ob die beiden eingegebenen Zahlen addiert oder subtrahiert werden sollen. Hinweis: Sie sollten die Equals-Methode der Klasse System.String verwenden, um den Vergleich durchzuführen

---

1

Zum Beispiel »XML in 21 Tagen« von Devan Shepherd, ISBN: 3-8272-6265-8

2

Zum Beispiel »C++ in 21 Tagen« von Jesse Liberty, ISBN: 3-8272-5624-0

Tag 20

ATL-Komponenten

In manchen Situationen lassen sich Anwendungen oder Komponenten einfach nicht richtig mit MFC erstellen. MFC ist dafür gedacht, die Erstellung von GUI-Anwendungen zu erleichtern, doch was ist, wenn Ihre Anwendung als Service ohne Benutzerschnittstelle laufen soll? Oder was ist mit Business Logic-Komponenten, die aus mehreren Anwendungen heraus aufgerufen werden sollen? Was, wenn Sie eine logische Komponente erstellen müssen, die von einer Webseite aus aufgerufen werden soll? In all diesen Situationen bietet sich eher die Verwendung der Active Template Library (ATL) anstelle von MFC an.

Heute lernen Sie, wie man mit ATL Komponenten erstellt.

Sie lernen heute unter anderem, ...

- wie man herausfindet, wann ATL die passende Bibliothek ist,
- den Unterschied zwischen ATL und MFC,
- wie Ihnen ATL und Visual C++ einen großen Teil der Plackerei beim Erstellen von COM-Komponenten abnehmen,
- wie Sie eine Komponente mit ATL erstellen, die einfach in verwaltetem C++ verpackt und so von verwalteten und nicht verwalteten Anwendungen aus aufgerufen werden kann.

20.1 Was ist ATL?

Die Active Template Library (ATL) ist ein Satz von Vorlagen und Klassen, die für die Erstellung von Komponenten und Diensten gedacht sind. ATL bietet keine Unterstützung für das Zeichnen von Fenstern oder Bildern und erfordert, dass Sie in den meisten Fällen unveränderte API-Aufrufe für von Windows bereitgestellte Dienste verwenden. Daher ist ATL für die Erstellung von GUI-Anwendungen weniger attraktiv, doch es hat seine Vorteile bei der Erstellung von Business Logic-Komponenten, da es nicht den Overhead von MFC-Anwendungen mitschleppt.

ATL und COM

Vor einigen Jahren, als Microsoft erstmals COM vorstellte, gab es nur eine Möglichkeit, COM-Objekte zu erstellen. Sie bestand aus einer Menge C/C++-Code, viel Aufwand und einer sehr genauen Kenntnis der Interface Definition Language (IDL - Schnittstellen- Definitionssprache). Zumindest war sie kompliziert und benötigte einiges an Aufwand. Es gab keine Werkzeuge und Assistenten zur Unterstützung.

Seitdem wurden bedeutende Fortschritte gemacht. Sie können MFC zur Erstellung von ActiveX-Steuerelementen verwenden und dabei auf die Hilfe der Assistenten zurückgreifen, die die zur Implementierung der COM-Schnittstellen notwendigen IDL- Dateien erstellen. MFC kommt jedoch mit der Erstellung von COM-Komponenten ohne Benutzerschnittstelle oder mit solchen, die für die Verwendung im Internet abgespeckt sein müssen, nicht sehr gut zurecht. Hier kommt ATL ins Spiel.

Microsoft entwickelte ATL vor einigen Jahren, um Programmierern die Erstellung von abgespeckten und damit internetfreundlichen (kleinen) ActiveX-Steuerelementen zu ermöglichen. Microsoft gestaltete ATL als gutes Werkzeug für die Erstellung von COM- Komponenten, die Business Logic enthalten und keine Benutzerschnittstelle besitzen. Diese Komponenten waren ideal, um sie in den Transaktionsserver von Microsoft, COM+ oder als Erweiterungen in den Microsoft-Webserver, Internet Information Server (IIS), zu laden.

Jetzt ist ATL durch die Möglichkeit der Erstellung von ATL-Komponenten, in denen die Business Logic von der COM-Schnittstelle getrennt ist, das perfekte Medium für die Erstellung von Komponenten mit Dual-Schnittstellen, auf die man von .NET- Anwendungen aus über COM oder über CLR (Common Language

Runtime) zugreifen kann. Sie können die ATL-Assistenten verwenden, um die COM-Schnittstelle zu definieren und die IDL-Dateien zu erstellen und dann einfach Wrapper aus verwaltetem C++ erzeugen, die die CLR-Schnittstelle für die nicht verwaltete Komponente bereitstellen.

ATL oder MFC?

Also warum sollte man ATL an Stelle von MFC verwenden? Könnte man Komponenten nicht auch mit MFC erstellen? Sie können das machen, jedoch nicht ohne bedeutenden, wahrscheinlich unerwünschten, Overhead. MFC ist für die Erstellung grafischer Objekte und Anwendungen gedacht, nicht für Komponenten ohne Benutzerschnittstelle (obwohl Sie diese mit MFC durchaus erstellen können). Jedes mit MFC erstellte Objekt und jede Anwendung benötigt Funktionen der mfc70.dll (oder eine ihrer Vorgängerversionen bei älteren MFC-Anwendungen, sowie noch weitere MFC-Dlls), eine große DLL. Sie können mfc70.dll in Ihre MFC-Anwendung oder -Komponente statisch linken und damit die Notwendigkeit ausschließen, die DLL mitzuverteilen, doch Ihre Komponente wird dadurch bedeutend größer. Als Beispiel die Anwendung Sock.exe von vorgestern:

Sock.exe mit dynamischer MFC-DLL	Sock.exe mit statischer MFC-DLL
143.360 Byte	1.961.984 Byte

ATL wurde mit Blick auf geringe Größe gestaltet. Natürlich muss auch hier eine Laufzeit- DLL eingebunden werden, atl70.dll, doch diese besitzt nur einen Bruchteil der Größe von mfc70.dll. Der Größenunterschied zwischen mit MFC oder ATL erstellten Komponenten macht ATL-basierende Komponenten viel internetfreundlicher. Der Nachteil ist allerdings, dass ATL keine der API-Funktionalitäten von Win32 abstrahiert. Das heißt wenn Sie die Benutzerschnittstelle für die Komponente zeichnen wollen, müssen Sie alle API-Aufrufe selbst ausführen, anstatt sich darauf zu verlassen, dass MFC diese Arbeit für Sie erledigt.

Ein weiterer Punkt, was die Größe angeht, ist, dass Sie bei der Verwendung von MFC die gesamte MFC-Bibliothek einbinden müssen, selbst wenn Sie diese in ihre Anwendung einlinken (und daher mfc70.dll nicht mit verteilen müssen). Da ATL über C++-Vorlagen erzeugt wird, wird nur das, was Sie verwenden, in die Anwendung eingelinkt. So bleibt die Größe Ihrer Anwendung oder Komponente auf das beschränkt, was Sie tatsächlich verwenden. ATL-Komponenten benötigen immer noch eine unterstützende DLL, atl70.dll, doch diese ist viel kleiner als die MFC-DLL.

In gewissem Sinn verhalten sich Vorlagen wie Makros, nur auf Klassenebene. Sie sind ein als Teil der Sprache C++ eingeführter Mechanismus, während Makros ein Überbleibsel aus C sind. Vorlagen ermöglichen Ihnen, eine Prototyp-Klasse mit vollständiger Funktionalität zu erstellen, die als Vorlage für mehrere Klassen verschiedener Datentypen verwendet werden kann.

Ein Beispiel dafür, wie das funktioniert, können die Arrayklassen von MFC sein. Sie alle besitzen die gleiche Funktionalität, arbeiten jedoch mit unterschiedlichen Datentypen. Sie können eine generische Arrayklasse als Vorlage implementieren und dann aus der Vorlage spezielle Arrayklassen für die Behandlung der verschiedenen Datentypen generieren.

Um zu sehen, wie das funktioniert, implementieren Sie Ihre Vorlagenklasse vor der Klassendeklaration und zwar mit dem Schlüsselwort template gefolgt von den als Parameter in spitzen Klammern (<>) aufgelisteten Schlüssel-Datenelementen, die für die Implementierung der speziellen Instanzen der Klasse notwendig sind. Beginnend mit der nächsten Zeile haben Sie eine gewöhnliche Klassendeklaration, die die Platzhalter-Namen für die Parameter verwendet und sie durch die tatsächlichen Variablen oder Werte ersetzt, wenn eine tatsächliche Klasse aus der Vorlage abgeleitet wird. Das könnte folgendermaßen aussehen:

```
template<class Tdatatype, int iMaxEntries>
class SimpleArray
{
private:
    int m_iCurrentNbr;
public:
    SimpleArray() {  m_iCurrentNbr = 0;}
    ~SimpleArray() { }
    BOOL Add(Tdatatype value);
    Tdatatype Get(int iPos);
private:
```

```
    Tdatatype m_array[iMaxEntries];
};
```

Diese Vorlagenklassendeklaration wird von den Implementierungen der Member- Funktionen gefolgt, die das gleiche Format verwenden:

```
template<class Tdatatype, int iMaxEntries>
BOOL SimpleArray<class Tdatatype, int
                 iMaxEntries>::Add(Tdatatype value)
{
  if (m_iCurrentNbr < iMaxEntries)
  {
    m_array[m_iCurrentNbr++] = value;
    return TRUE;
  }
  return FALSE;
}
template<class Tdatatype, int iMaxEntries>
Tdatatype SimpleArray<class Tdatatype,
                      int iMaxEntries>::Get(int iPos)
{
  if (iPos < m_iCurrentNbr)
    return m_array[iPos];
  return NULL;
}
```

Wenn Sie aus dieser Vorlage eine tatsächliche Klasse erstellen wollen, deklarieren Sie in Ihrem Code einfach eine Instanz dieser Klasse und übergeben Sie ihr alle notwendigen Parameter:

```
SimpleArray<long, 25> saMyLongArray;
```

Sie können sie auch mit Ihren eigenen Klassen verwenden, abhängig davon, wie Sie die Vorlage implementiert haben. Die vorstehende Beispielvorlage würde beispielsweise nicht mit einer Ihrer benutzerdefinierten Klassen zusammenarbeiten, mit einem Zeiger auf Ihre Klasse dagegen schon:

```
SimpleArray<&CMyClass, 25> saMyClassArray;
```

Sie können auch Funktionsvorlagen erzeugen, wenn Sie nur eine einfache Funktion und nicht eine ganze Klasse benötigen. Die Funktionsvorlagen funktionieren genauso wie Klassenvorlagen, nur eben für eine einzelne Funktion. Sie können Ihre Funktion wie folgt erzeugen:

```
template<class Tdatatype>
void DoSomething(Tdatatype value, int iOtherParam)
{
  for (int i = 0; i < iOtherParam; i++)
  {
    // Den Wert verarbeiten
  }
}
```

Die Funktionsvorlage erzeugt die eigentliche Funktion, wenn sie in Ihrem Code aufgerufen wird und ersetzt dabei den Datentyp-Platzhalter durch den Datentyp des Werts, den Sie ihr übergeben:

```
int iMyInt;
double dbMyDouble;
DoSomething(iMyInt, 4);
DoSomething(dbMyDouble, 8);
```

So umfangreich die Klassenbibliothek für ATL auch sein mag, manchmal möchten Sie etwas verwenden, das sie nicht enthält. Rein zufällig gibt es auch noch eine andere gehaltvolle Vorlagenbibliothek - die Standard Template Library (STL).

STL ist eine umfassende Bibliothek von Vorlagen, die ursprünglich von Hewlett Packard erstellt wurden und seitdem ein Teil des ISO-C++-Standars geworden sind. Sie ist viel zu groß, um sie hier auch nur annähernd abzudecken; es gibt jede Menge Bücher, die sich ihrer Verwendung widmen.[1] In Tabelle 20.1 sind einige der wichtigsten enthaltenen Vorlagen aufgelistet.

Klasse	Beschreibung
allocator	Alloziert Speicher für Objekte. Dies ist der Standard-Allozierer für die meisten STL-Vorlagen.
iterator	Durchquert Arrays, verknüpfte Listen, Warteschlangen und andere Speicherstrukturen. Es gibt eigentlich mehrere iterator-Vorlagen mit unterschiedlichen Navigationscharakteristika, also sollten Sie die verschiedenen iterator-Klassen studieren, um die für Ihre Bedürfnisse passende zu finden.
vector	Führt Funktionalität aus, die den Arrayklassen in MFC sehr ähnlich ist. Die vector-Klassen erstellen eine dynamisch allozierte und gewachsene Sammlung von Objekten, die am Ende der Sammlung hinzugefügt werden, auf die jedoch leicht nach Belieben zugegriffen werden kann.
deque	Führt der vector-Klassenvorlage ähnliche Funktionalität aus. Sie können jedoch sowohl am Ende als auch am Anfang der Sammlung Objekte einfügen.
queue	Implementiert eine Objekt-Warteschlange, in der neue Objekte am Ende hinzugefügt und dann Objekte am Anfang entfernt werden können
priority_queue	Implementiert eine Warteschlange, in der es eine Rangordnung und verschiedene Prioritäten für die Elemente geben kann. Elemente in der Warteschlange werden nach ihrer Priorität entfernt.
list	Implementiert eine Liste von Objekten, in der Elemente hinzugefügt oder entfernt werden können und auf die auf jedem Punkt der Liste zugegriffen werden kann, nicht nur am Anfang oder Ende
map	Implementiert eine Sammlung von Elementen, in der jedem Element ein Schlüssel zugewiesen ist. Jeder Schlüsselwert muss eindeutig sein und darf nur einmal in der Map auftauchen.
multimap	Funktioniert ähnlich wie die map-Klassenvorlage, es können jedoch mehrere Schlüssel den gleichen Wert besitzen
set	Ähnlich der map-Klassenvorlage, jedoch gibt es keinen zugeordneten Wert, nur den Schlüssel
multiset	Ähnlich der set-Klassenvorlage, jedoch sind doppelte Schlüssel im Set erlaubt
stack	Implementiert einen Stapel, in dem Elemente am gleichen Ende in die Sammlung eingefügt und aus ihr entfernt werden. Das heißt, dass nur das zuletzt zum Stapel hinzugefügte Element entfernt werden kann (die Elemente werden in der entgegengesetzten Reihenfolge ihres Hinzufügens entfernt).
basic_string	Implementiert Stringfunktionalität. Diese Klassenvorlage dient als Vorlage für die string- (ASCII) und wstring- (UNICODE) Typen, die ohne Berührung der Vorlagenklasse verwendet werden können.

Tabelle 20.1: Einige wichtige Klassen-Vorlagen in der STL

ATL-Exkurs: Debug-Makros

So wie die MFC-Klassenbibliothek nicht in ATL-Projekte verfügbar ist, sind es auch die Debug-Makros nicht, die Sie an Tag 2 kennen gelernt haben. Stattdessen hat die ATL- Klassenbibliothek eine eigene Reihe von Makros, die auf die gleiche Art wie ihre MFC- Äquivalente funktionieren:

- Das Makro ATLASSERT überprüft Annahmen in Ihrem Code. Sie übergeben ihm einen Booleschen Ausdruck, der immer als TRUE ausgewertet werden sollte und der Sie im Debug-Modus alarmiert, wenn der Ausdruck nicht als TRUE ausgewertet wird. Es entspricht dem Makro ASSERT in MFC-Anwendungen.
- Das Makro ATLTRACE gibt Nachrichten an ein offenes Debug-Ausgabefenster aus. Sie können in den

Parametern dieses Makros Standard-C-Formatausdrücke verwenden. Es entspricht dem Makro TRACE in MFC-Anwendungen.

- Das Makro ATLTRACENOTIMPL verfolgt Aufrufe an nicht implementierte Funktionen. Sie übergeben diesem Makro den Funktionsnamen und es schickt den String "functionname is not implemented" an das Dump-Device.

20.2 Eine einfache ATL-Komponente erstellen

Die heutige Beispielanwendung ist ein Business-Objekt, das die Steuer für gekaufte Waren berechnet. Diese Komponente erhält eine Kategorie für die gekauften Waren und ihren Preis und berechnet die zu zahlende Steuer. Sie verpacken diese Komponente zuerst in einer COM-Schnittstelle und erstellen dann einen einfachen C++-Client, um die Komponente aufzurufen. Zuletzt fügen Sie einen verwalteten Wrapper um die Komponente ein, um verwalteten und nicht verwalteten Code zu vereinigen und erstellen einen verwalteten C++-Client, um die Komponente aufzurufen.

Der Schlüssel zur Erstellung einer Komponente mit Dual-Schnittstelle wie dieser ist, die eigentliche Business Logic von den Schnittstellen getrennt zu halten. Dazu platzieren Sie den Code zur Steuerberechnung in einer generischen Klasse innerhalb der ATL- Komponente. Sie fügen dann eine einfache ATL-Klasse ein, die als externe Schnittstelle für die Business Logic-Klasse dient. Zuletzt fügen Sie eine verwaltete Klasse ein, die als zweite externe Schnittstelle für die Business Logic-Klasse dient.

Die ATL-Komponente erstellen

Um das heutige Beispiel zu erstellen, müssen Sie zuerst das ATL-Projekt erzeugen. Es handelt sich um ein Standard-ATL-Projekt, also folgen Sie diesen Schritten, um das Projektgerüst zu erstellen:

1. Starten Sie ein neues Projekt. Wählen Sie ATL-Projekt aus den C++-Vorlagen und nennen Sie das Projekt TaxCalculations. Klicken Sie auf Ok, um den ATL-Projekt- Assistenten zu starten.

2. Im ATL-Projekt-Assistenten klicken Sie auf Fertig stellen, um das Projekt mit den Standardeinstellungen zu erzeugen.

Die Business Logic-Klasse erstellen

Nachdem Sie nun das Projektgerüst erstellt haben, müssen Sie die Business Logic-Klasse erzeugen. Sie werden diese Klasse in einem eigenen Namespace platzieren und ihr eine einzige Methode für die Berechnung der zu zahlenden Steuer geben. Die in diesem Beispiel verwendeten Steuersätze sind rein fiktiv, etwaige Ähnlichkeiten mit lebenden oder anderweitigen Steuersätzen sind rein zufällig und nicht beabsichtigt. Um diese Klasse zu erstellen, folgen Sie diesen Schritten:

1. Fügen Sie nach den an den vergangenen Tagen verwendeten Schritten eine allgemeine C++-Klasse zum Projekt TaxCalculations hinzu. Nennen Sie die Klasse CTaxCalculator.

2. Öffnen Sie die Header-Datei der CTaxCalculator-Klasse und ändern Sie sie wie in Listing 20.1 gezeigt - binden Sie ein paar Standard-C-Header ein und verkapseln Sie die Klasse in einem Namespace namens TaxCalcs.

Listing 20.1: Die Klassendeklaration von CTaxCalculator

```
 1: #pragma once
 2: #include <windows.h>
 3: #include <stdio.h>
 4:
 5: namespace TaxCalcs
 6: {
 7:   class CTaxCalculator
 8:   {
 9:   public:
10:       CTaxCalculator(void);
11:       ~CTaxCalculator(void);
12:   };
```

```
13: }
```

Namespaces sind ein konzeptionelles Mittel, um Klassen und Objekte in Gruppierungen ähnlicher Funktionalität zu organisieren. Für dieses Beispiel ist es zwar eigentlich nicht notwendig, doch es ist ein extensiv in der .NET-Objekt-Bibliothek wie auch in der C++- Standard Template Library (STL) verwendetes Konzept, also schadet es nicht, die Arbeit mit Namespaces ein wenig zu üben.

3. Öffnen Sie die Quellcode-Datei für die Klasse CTaxCalculator und schließen Sie sie wie in Listing 20.2 im Namespace TaxCalcs ein.

Listing 20.2: Die Implementierung der Klasse CTaxCalculator

```
 1: #include "StdAfx.h"
 2: #include "taxcalculator.h"
 3:
 4: namespace TaxCalcs
 5: {
 6:   CTaxCalculator::CTaxCalculator(void)
 7:   {
 8:   }
 9:
10:   CTaxCalculator::~CTaxCalculator(void)
11:   {
12:   }
13: }
```

4. Erweitern Sie in der Klassenansicht den Baum und den TaxCalcs-Namespace, um die Klasse CTaxCalculator anzuzeigen. Fügen Sie der Klasse mit den an den vergangenen Tagen verwendeten Schritten eine neue Methode hinzu.
5. Legen Sie den Rückgabetyp für die neue Funktion als BOOL, den Namen als CalculateTaxes und den Zugriff als public fest. Fügen Sie drei Parameter ein. Legen Sie den Typ des ersten Parameters als float und den Namen als fPurchaseAmt fest. Legen Sie den Typ des zweiten Parameters als wchar_t* und den Namen als wszCategory fest. Legen Sie den Typ des dritten Parameters als float* und den Namen als pfTaxAmt fest. Klicken Sie auf Fertig stellen, um die Methode Ihrer Klasse hinzuzufügen.
6. Nehmen Sie in die neue Methode den Code aus Listing 20.3 auf.

Listing 20.3: Die Funktion CalculateTaxes

```
 1: BOOL TaxCalcs::CTaxCalculator::CalculateTaxes(float
 2:      fPurchaseAmt, wchar_t* wszCategory,
 3:                    float* pfTaxAmt)
 4: {
 5:   // Das Array mit den Steuerkategorien deklarieren
 6:   static const wchar_t *wszCategories[] =
 7:               { L"Nahrungsmittel", L"Kleidung",
 8:                 L"Musik"} ;
 9:   // Dass Array mit den Steuersätzen deklarieren
10:   static const float fRates[] = { 0.0725, 0.0835, 0.081} ;
11:
12:   // Schleife durch die Kategorien
13:   for (int i = 0; i < 3; i++)
14:   {
15:     // Wurde diese Kategorie gekauft?
16:     if (wcscmp(wszCategory, wszCategories[i]) == 0)
17:     {
18:       // Ja, zu zahlende Steuer berechnen
19:       *pfTaxAmt = fPurchaseAmt * fRates[i];
20:       // TRUE zurückgeben
21:       return TRUE;
22:     }
23:   }
24:   // Gekaufte Kategorie nicht gefunden, FALSE
25:   // zurückgeben
```

```
26:   return FALSE;
27: }
```

Da COM-Methoden einen Erfolg-/Misserfolg-Status zurückgeben, müssen Sie die Ergebnisse der Berechnung durch als Parameter übergebene Zeiger zurückgeben. Beachten Sie, dass es sich bei allen in diesem Beispiel verwendeten Strings um UNICODE-Strings handelt, da COM intern vollständig UNICODE ist. Außerdem verwenden Sie eine C-Standardfunktion, wcscmp, um den Stringvergleich durchzuführen. Die Stringvergleichsfunktionen von C geben 0 zurück, wenn die beiden Strings gleich sind, und ansonsten einen negativen oder positiven Wert, der das Ergebnis des für den Vergleich der beiden Strings verwendeten Vergleichsalgorithmus ist.

Die COM-Schnittstelle hinzufügen

Der nächste Schritt ist das Hinzufügen der COM-Schnittstelle zu Ihrer ATL-Komponente. Sie fügen dazu eine einfache ATL-Klasse mit einer einzelnen Methode ein, die die Methode der bereits erstellten Klasse widerspiegelt. Um diese Klasse einzufügen, folgen Sie diesen Schritten:

1. Fügen Sie dem Projekt TaxCalculations mit den gleichen Schritten wie an den vorangegangenen Tagen eine neue Klasse hinzu. Markieren Sie den ATL-Knoten auf der linken Seite des Dialogfelds Klasse hinzufügen und wählen Sie Einfaches ATL-Objekt in den Vorlagen auf der rechten Seite. Klicken Sie auf Öffnen, um die neue Klasse hinzuzufügen.

2. Legen Sie im Feld Kurzer Name den Klassennamen als GetTaxForPurchase fest. Beachten Sie, dass der ATL-Assistent die Namen für die COM-Schnittstellen und die Dateinamen selbsttätig einfügt. Klicken Sie auf Fertig stellen, um die Klasse zu erzeugen.
3. Öffnen Sie die Header-Datei von GetTaxForPurchase. Fügen Sie wie in Listing 20.4 eine Include-Anweisung für die Header-Datei TaxCalculation.h Ihrer Business Logic- Klasse ein.

Listing 20.4: Die Header-Datei für GetTaxForPurchase

```
1: // GetTaxForPurchase.h : Deklaration der Klasse
2: // CGetTaxForPurchase
3:
4: #pragma once
5: #include "resource.h"       // Hauptsymbole
6: #include "TaxCalculator.h"
7:
8: // IGetTaxForPurchase
```

4. Scrollen Sie in der Header-Datei nach unten, um die Klassendeklaration zu finden. Beachten Sie, dass der Klassen-Konstruktor innerhalb der Header-Datei als Inline- Funktion implementiert ist. Fügen Sie unmittelbar vor dem Klassen-Konstruktor einen privaten Member-Zeiger auf Ihre CTaxCalculation-Klasse ein und initialisieren Sie ihn so, dass er auf eine neue Instanz der CTaxCalculation-Klasse im Konstruktor zeigt, wie es in Listing 20.5 dargestellt ist. Löschen Sie den allozierten Zeiger in der Funktion FinalRelease weiter unten in der Klassendeklaration, um Speicherlecks zu verhindern.

Listing 20.5: Die Klassendeklaration für CGetTaxForPurchase

```
 1: class ATL_NO_VTABLE CGetTaxForPurchase :
 2:     public IGetTaxForPurchase
 3: {
 4: private:
 5:     TaxCalcs::CTaxCalculator* m_pTaxCalc;
 6: public:
 7:     CGetTaxForPurchase()
 8:     {
 9:       // Zeiger auf das Berechnungsmodul initialisieren
10:       m_pTaxCalc = new TaxCalcs::CTaxCalculator();
11:     }
12:     DECLARE_PROTECT_FINAL_CONSTRUCT()
13:
14:     HRESULT FinalConstruct()
```

```
15:     {
16:       return S_OK;
17:     }
18:
19:     void FinalRelease()
20:     {
21:       delete m_pTaxCalc;
22:     }
```

Das Makro ATL_NO_VTABLE, das in dieser Klassendeklaration verwendet wird, unterdrückt die Initialisierung der virtuellen Funktionstabelle im Konstruktor und im Destruktor der Klasse. Die virtuelle Funktionstabelle ist eine Tabelle mit Zeigern auf alle virtuellen Funktionen in einer Klasse. Wenn Sie die Initialisierung dieser Tabelle unterdrücken, wird die Klasse noch kleiner als gewöhnlich.

5. Sie müssen die exponierte Methode hinzufügen, über die Clients auf die Business Logic-Klasse in Ihrer Komponente zugreifen werden. Fügen Sie eine neue Methode zur IGetTaxForPurchase-Schnittstelle hinzu. Achten Sie darauf, die Methode der Schnittstelle und nicht der Klasse hinzuzufügen.
6. Legen Sie den Namen der Methode als GetPurchaseTaxes fest. Fügen Sie drei Parameter ein, um den drei an die Business Logic-Klasse übergebenen Parametern zu entsprechen.

- Legen Sie für den ersten Parameter FLOAT als Parametertyp fest, geben Sie fPurchaseAmt als Parametername ein, aktivieren Sie das Kontrollkästchen in und klicken Sie auf Hinzufügen.
- Legen Sie für den zweiten Parameter BSTR* als Parametertyp fest, geben Sie bstrCategory als Parametername ein, aktivieren Sie das Kontrollkästchen in und klicken Sie auf Hinzufügen.
- Legen Sie für den dritten Parameter FLOAT* als Parametertyp fest, geben Sie pfTax als Parametername ein, aktivieren Sie das Kontrollkästchen out und klicken Sie auf Hinzufügen.

Der Assistent zum Hinzufügen von Methoden sollte nun wie in Abbildung 20.1 aussehen. Klicken Sie auf Fertig stellen, um die Methode zur Schnittstelle hinzuzufügen.

Abbildung 20.1: Der Assistent zum Hinzufügen von Methoden

7. Nehmen Sie in die Methode den Code aus Listing 20.6 auf.

Listing 20.6: Die Methode GetPurchaseTaxes

```
 1: STDMETHODIMP CGetTaxForPurchase::GetPurchaseTaxes(FLOAT
 2:     fPurchaseAmt, BSTR* bstrCategory, FLOAT* pfTax)
 3: {
 4:   // TODO: Fügen Sie hier Ihren Implementierungscode
 5:   // ein.
 6:   // Haben wir einen gültigen Zeiger auf die Variable,
 7:   // in der die zu zahlende Steuer platziert wird?
 8:   if (!pfTax)
 9:     return S_FALSE;
10:   // Zu zahlende Steuer berechnen
11:   if (m_pTaxCalc->CalculateTaxes(fPurchaseAmt,
12:       (wchar_t*)*bstrCategory, pfTax))
13:     return S_OK;
14:   else
15:     return S_FALSE;
16: }
```

Den C++-Client erstellen

An diesem Punkt sollten Sie Ihre Anwendung kompilieren können, doch Sie können sie noch nicht verwenden. Dazu müssen Sie eine Client-Anwendung erstellen. Die Client- Anwendung ist eine einfache Konsolenanwendung, die einen Kategoriestring und die Menge der gekauften Waren übergibt und dann die zurückgegebene Steuer anzeigt, die berechnet werden muss. Um die Client-Anwendung zu erstellen, folgen Sie diesen Schritten:

1. Markieren Sie in der Projektmappen-Explorer-Ansicht den Knoten Projektmappe am oberen Ende des Baums und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Neues Projekt aus dem Kontextmenü.

2. Wählen Sie Win32-Projekt aus den Visual C++-Projektvorlagen. Nennen Sie das Projekt TaxCalcClient. Klicken Sie auf Ok.
3. Wählen Sie im Win32-Anwendungs-Assistenten den Bereich Anwendungseinstellungen aus und geben Sie als Applikationstyp Konsolenanwendung an. Klicken Sie auf Fertig stellen, um das Projekt zu erzeugen.
4. Öffnen Sie die Quellcode-Datei TaxCalcClient.cpp. Fügen Sie die Include-Anweisung für die Header-Datei atlbase.h ein und importieren Sie die von Ihrem ATL-Projekt erzeugte DLL. Geben Sie dabei wie in Listing 20.7 den korrekten Pfad für Ihr Projekt an.

Listing 20.7: Die Include- und Import-Anweisungen für den Client

```
1: // TaxCalcClient.cpp : Definiert den Einstiegspunkt für
2: // die Konsolenanwendung.
3: //
4:
5: #include "stdafx.h"
6: #include <atlbase.h>
7: // TODO: Ersetzen Sie den Pfad in der Import-Direktive
8: // durch den korrekten Pfad für Ihr Projekt
9: #import "..\TaxCalculations\Debug\TaxCalculations.dll"
```

5. Nehmen Sie in die Hauptfunktion den Code aus Listing 20.8 auf.

Listing 20.8: Die Hauptfunktion des Clients

```
 1: int _tmain(int argc, _TCHAR* argv[])
 2: {
 3:   // COM-Umgebung starten
 4:   CoInitialize(NULL);
 5:   {
 6:     // Namespace des Steuerberechnungsmoduls verwenden
 7:     using namespace TaxCalculations;
 8:     // Den Schnittstellenzeiger für das
```

```
 9:     // Steuerberechnungsmodul holen
10:     IGetTaxForPurchasePtr pGetTax(
11:                 L"TaxCalculations.GetTaxForPurchase");
12:
13:     // String für die gekaufte Kategorie erzeugen
14:     CComBSTR bstrCat1(OLESTR("Kleidung"));
15:     float fPurchaseAmt, fTaxAmt;
16:
17:     // Kaufpreis angeben
18:     fPurchaseAmt = 45.0;
19:     // Zu berechnende Steuern abfragen
20:     if (pGetTax->GetPurchaseTaxes(fPurchaseAmt,
21:                                   &bstrCat1,
22:                                   &fTaxAmt) == S_OK)
23:     {
24:       // String-Umwandlungsmakros einfügen
25:       USES_CONVERSION;
26:       // Gekaufte Kategorie anzeigen
27:       printf("%s\n", W2A(bstrCat1));
28:       // Gekaufte Menge anzeigen
29:       printf("Kaufpreis = %9.2f\n", fPurchaseAmt);
30:       // Zu zahlende Steuer anzeigen
31:       printf("   Steuer = %9.2f\n", fTaxAmt);
32:       // Summe anzeigen
33:       printf("    Summe = %9.2f\n", fPurchaseAmt +
34:              fTaxAmt);
35:     }
36:   }
37:   // COM-Umgebung stoppen
38:   CoUninitialize();
39:   return 0;
40: }
```

Der Code in Listing 20.8 initialisiert zuerst mit dem Funktionsaufruf von CoInitialize die COM-Umgebung. Am anderen Ende der Hauptfunktion stoppt er die COM-Umgebung mit der Funktion CoUninitialize.

Indem Sie den Namespace für die vorher erstellte Komponente verwenden, erzeugen Sie einen Schnittstellenzeiger für die implementierte COM-Schnittstelle. Alles zwischen den Funktionen CoInitialize und CoUninitialize ist in Klammern eingeschlossen, sodass der Schnittstellenzeiger durch Verlust seines Geltungsbereichs freigegeben wird, bevor die COM-Umgebung beendet wird. Als Nächstes gibt ein CComBSTR-String die Kategorie für den zu berechnenden Steuersatz an. CComBSTR ist eine ATL-Klasse, die den Stringtyp BSTR verkapselt. Nachdem die Funktion zurückgekehrt ist, binden Sie das Makro USES_CONVERSION ein, um die BSTR-Umwandlungsmakros einzuschließen und rufen dann das Makro W2A auf, um den Kategoriestring in einen ASCII-String für die Anzeige auf der Konsole umzuwandeln. Da es sich um eine Anwendung für den DOS-Modus handelt, verwenden Sie die C-Funktion printf, um Textstrings für den Benutzer auf das Konsolenfenster auszugeben.

Sie können nun die Client-Anwendung kompilieren aber noch nicht starten, beim Versuch zu starten fragt Sie Visual C++ nach der zu startenden Applikation. Geben Sie deshalb den Klient als Startprojekt an. Damit weisen Sie Visual Studio an, die Client- Anwendung zu starten, wenn Sie Debuggen / Starten wählen. Folgen Sie diesen Schritten:

1. Markieren Sie das TaxCalcClient-Projekt in der Projektmappen-Explorer-Ansicht und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.

2. Kompilieren und starten Sie Ihr Projekt. Sie werden ein DOS-Fenster sehen, das den Kaufpreis und die zu berechnende Steuer anzeigt (siehe Abbildung 20.2).

Abbildung 20.2: Die laufende Client-Anwendung

Ein weiterer Schönheitsfehler ist, daß in Listing 20.7 der direkte Pfad zur verwendeten DLL enthalten ist, dies ist dann unpraktisch, wenn Sie den fertigen Klienten im Release- Modus erstellen, die DLL aber weiterhin aus dem Debug-Modus verwendet wird. Ersetzen Sie die #import-Anweisung in Listing 20.7 durch folgende Anweisung:

```
#import "TaxCalculations.dll"
```

Bevor Sie versuchen, die Client-Anwendung zu kompilieren und zu starten, müssen Sie ein paar Änderungen an der Konfiguration vornehmen. Um die Änderungen durchzuführen, folgen Sie diesen Schritten:

1. Markieren Sie das Projekt TaxCalcClient in der Projektmappen-Explorer-Ansicht und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Eigenschaften aus dem Kontextmenü.

2. Wählen Sie Alle Konfigurationen aus dem Kombinationsfeld Konfiguration.
3. Wählen Sie C/C++, allgemein auf der linken Seite des Dialogfelds Eigenschaftsseiten aus. Wählen Sie auf der rechten Seite Zusätzliche Include- Verzeichnisse aus und geben Sie dann $(SolutionDir)\$(ConfigurationName) ein. Klicken Sie auf Ok, um den Eigenschaftendialog zu schließen.

Sie haben in die Liste der Verzeichnisse, in denen nach Header-Dateien, Bibliotheks-Dateien und DLLs gesucht wird, das Verzeichnis eingebunden, in dem die aktuelle ATL- Komponente kompiliert wird. Die erste Hälfte des Strings, $(SolutionDir), gibt das Verzeichnis an, in dem sich die Projektmappen-Datei (TaxCalculations.sln) befindet. Da dieses Projekt mit dem ATL-Komponenten-Projekt begonnen wurde, befindet sich in diesem Verzeichnis auch die Projektmappen-Datei. Die zweite Hälfte des Strings, $(ConfigurationName), gibt das Verzeichnis an, das der aktuellen Build-Konfiguration entspricht. Also legt der gesamte String fest, dass die ATL-DLL im Debug-Modus des Debug- Verzeichnisses und im Release-Modus des Release-Verzeichnisses unter dem ATL- Projektverzeichnis zu suchen ist.

Den verwalteten C++-Wrapper hinzufügen

Jetzt müssen Sie den verwalteten Wrapper für Ihre Business Logic-Klasse einfügen. Sie fügen dem ATL-Projekt eine neue Klasse hinzu, die Sie als verwaltet konfigurieren, und duplizieren die Funktionalität der COM-Schnittstelle in der neuen, verwalteten Klasse.

Folgen Sie diesen Schritten:

1. Wählen Sie in der Klassenansicht das TaxCalculations-Projekt aus und fügen Sie mit der an den vorangegangenen Tagen verwendeten Methode eine neue allgemeine C++-Klasse ein.

2. Legen Sie den Klassennamen als MgdGetTaxForPurchase fest und klicken Sie auf Fertig stellen, um

die Klasse hinzuzufügen.

3. Öffnen Sie die Header-Datei MgdGetTaxForPurchase.h und nehmen Sie den Code aus Listing 20.9 auf. Binden Sie den Header für die Business Logic-Klasse ein. Legen Sie fest, dass die CLR-DLL mscorlib.dll verwendet werden soll und konfigurieren Sie die Klasse als speicherbereinigte Klasse. Fügen Sie den gleichen Zeiger auf die Business Logic-Klasse ein, den Sie auch in der COM-Schnittstellenklasse verwendet haben. Zuletzt entfernen Sie die Deklaration des Klassendestruktors.

Listing 20.9: Die Header-Datei für MgdGetTaxForPurchase

```
 1: #pragma once
 2: #include "TaxCalculator.h"
 3: #using <mscorlib.dll>
 4: using namespace System;
 5:
 6: public __gc class MgdGetTaxForPurchase
 7: {
 8: private:
 9:     TaxCalcs::CTaxCalculator* m_pTaxCalc;
10: public:
11:     MgdGetTaxForPurchase(void);
12: };
```

4. Öffnen Sie die Quellcode-Datei MgdGetTaxForPurchase.cpp und ändern Sie den Klassenkonstruktor wie in Listing 20.10, um den Klassenzeiger zu initialisieren.

Listing 20.10: Der Klassenkonstruktor von MgdGetTaxForPurchase

```
1: MgdGetTaxForPurchase::MgdGetTaxForPurchase(void)
2: {
3:   // Zeiger auf Berechnungsmodul initialisieren
4:   m_pTaxCalc = new TaxCalcs::CTaxCalculator();
5: }
```

5. Fügen Sie dieser neuen Klasse eine neue Funktion hinzu. Die Klasse ist in der Klassenansicht nicht mehr sichtbar, also können Sie die Funktion nicht mit der bisher verwendeten Methode hinzufügen. Stattdessen öffnen Sie die Header-Datei noch einmal und fügen Sie wie in Listing 20.11 die Funktionsdeklaration ein.

Listing 20.11: Die modifizierte Header-Datei für MgdGetTaxForPurchase

```
 1: #pragma once
 2: #include "TaxCalculator.h"
 3: #using <mscorlib.dll>
 4: using namespace System;
 5:
 6: public __gc class MgdGetTaxForPurchase
 7: {
 8: private:
 9:     TaxCalcs::CTaxCalculator* m_pTaxCalc;
10: public:
11:     MgdGetTaxForPurchase(void);
12:     BOOL GetPurchaseTaxes(float fPurchaseAmt,
13:          System::String* pstrCategory, float* pfTax);
14: };
```

6. Öffnen Sie die Quellcode-Datei für die Klasse MgdGetTaxForPurchase noch einmal und fügen Sie wie in Listing 20.12 die Definition für die neue Funktion ein.

Listing 20.12: Die Funktionsdefinition für GetPurchaseTaxes

```
 1: BOOL MgdGetTaxForPurchase::GetPurchaseTaxes(
 2:     float fPurchaseAmt, System::String* pstrCategory,
```

```
 3:     float* pfTax)
 4: {
 5:   wchar_t *pCat;
 6:   int iLen;
 7:
 8:   // Haben wir einen gültigen Zeiger auf die Variable,
 9:   // in der die zu zahlende Steuer platziert wird?
10:
11:   if (!pfTax)
12:     return FALSE;
13:   // Kategorie aus verwaltetem String in wchar_t
14:   // umwandeln
15:   __wchar_t pCategory __gc[] =
16:                       pstrCategory->ToCharArray();
17:   iLen = pstrCategory->get_Length();
18:   pCat = new wchar_t[iLen + 1];
19:   for (int i = 0; i < iLen; i++)
20:   {
21:     pCat[i] = pCategory[i];
22:   }
23:   pCat[i] = NULL;
24:   // Zu zahlende Steuer berechnen
25:   if (m_pTaxCalc->CalculateTaxes(fPurchaseAmt, pCat,
26:                                  pfTax))
27:     return TRUE;
28:   else
29:     return FALSE;
30: }
```

Als Nächstes müssen Sie in Ihrem ATL-Projekt eine Klasse so konfigurieren, dass sie als verwaltetes C++ kompiliert wird. Die erste Änderung an der Konfiguration ist die Kennzeichnung der verwalteten Klasse als für CLR kompiliert, mit Standard-Debug- Unterstützung. Als Zweites müssen Sie die Verwendung der vorkompilierten Header für das gesamte Projekt unterbinden und das Debug-Informationsformat auf »nur Programmdatenbank« ändern. Um diese Änderungen an der Konfiguration vorzunehmen, folgen Sie diesen Schritten:

1. Markieren Sie im Projektmappen-Explorer die Datei MgdGetTaxForPurchase.cpp und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Eigenschaften aus dem Kontextmenü.

2. Im Kombinationsfeld Konfiguration wählen Sie Alle Konfigurationen, sodass sich die von Ihnen vorgenommenen Änderungen auf alle Build-Konfigurationen auwirken und nicht nur auf die aktuelle.
3. Auf der linken Seite des Eigenschaftendialogs wählen Sie C/C++, Allgemein und auf der rechten Seite setzen Sie den Eintrag Als verwaltet kompilieren auf die Assemblyunterstützung (/clr).
4. Auf der linken Seite des Eigenschaftendialogs wählen Sie Codeerstellung. Setzen Sie auf der rechten Seite den Eintrag Minimale Neuerstellung aktivieren auf Nein. Setzen Sie Vollständige Laufzeitüberprüfungen auf Standard.
5. Klicken Sie auf Übernehmen, um die Änderungen an der Konfiguration zu übernehmen und den Eigenschaftendialog geöffnet zu halten.
6. Klicken Sie, während der Eigenschaftendialog noch geöffnet ist, auf den Projektknoten TaxCalculations in der Projektmappen-Explorer-Ansicht.
7. Auf der linken Seite des Eigenschaftendialogs wählen Sie Vorkompilierte Header. Setzen Sie auf der rechten Seiten des Dialogfelds den Eintrag Erstellen/ Verwenden eines vorkompilierten Headers auf Vorkompilierte Header nicht verwenden.
8. Auf der linken Seite des Eigenschaftendialogs wählen Sie Allgemein. Setzen Sie auf der rechten Seite des Dialogfelds den Eintrag Debuginformationsformat auf Programmdatenbank (/Zi).
9. Klicken Sie auf Übernehmen, um die Konfigurationsänderungen zu übernehmen und den Eigenschaftendialog offen zu halten.
10. Klicken Sie, während der Eigenschaftendialog noch geöffnet ist, in der Projektmappen-Explorer-Ansicht auf den Datei-Knoten Stdafx.cpp unter dem TaxCalculations-Projekt.
11. Auf der linken Seite des Eigenschaftendialogs wählen Sie Vorkompilierte Header. Setzen Sie auf der rechten Seite des Dialogfelds den Eintrag Erstellen/Verwenden eines vorkompilierten Headers auf Vorkompilierte Header nicht verwenden.
12. Klicken Sie auf Ok, um die Konfigurationsänderungen zu übernehmen und den Eigenschaftendialog zu schließen.

Sie sollten Ihr Projekt nun kompilieren können. Wenn Sie das Projekt starten, läuft wieder der COM-Client, sodass Sie sehen können, dass die COM-Schnittstelle noch funktioniert.

Den verwalteten C++-Client erstellen

Der letzte Schritt bei der Erstellung der heutigen Beispielanwendung ist die Erstellung des verwalteten C++-Clients. Dieser Client dupliziert die Funktionalität des COM-Clients, nur eben als verwaltete C++-Anwendung, die die verwaltete Wrapper-Schnittstelle zu Ihrer ATL-Komponente verwendet. Um diesen Client zu erstellen, folgen Sie diesen Schritten:

1. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektmappen-Knoten am oberen Ende des Baums und klicken Sie mit der rechten Maustaste darauf. Wählen Sie dann Hinzufügen / Neues Projekt aus dem Kontextmenü.

2. Wählen Sie aus den Visual C++-Projektvorlagen Verwaltete C++-Anwendung. Nennen Sie das Projekt MgdTaxCalcClient.
3. Öffnen Sie die Datei MgdTaxCalcClient.cpp und nehmen Sie die beiden Änderungen aus Listing 20.13 vor.

Listing 20.13: Die Funktiondefinition für GetPurchaseTaxes

```
 1: // Die Hauptprojektdatei für ein VC++-Anwendungsprojekt,
 2: // das mit dem Anwendungs-Assistenten generiert wurde.
 3:
 4: #include "stdafx.h"
 5:
 6: #using <mscorlib.dll>
 7: #using "TaxCalculations.dll"
 8: #include <tchar.h>
 9: using namespace System;
10:
11: // Dies ist der Einstiegspunkt für die Anwendung
12: int _tmain(void)
13: {
14:   // TODO: Ersetzen Sie den Beispielcode durch Ihren
15:   // eigenen Code.
16:   // Schnittstellenzeiger für Steuerberechnungsmodul
17:   // holen
18:   MgdGetTaxForPurchase* pGetTax =
19:                         new MgdGetTaxForPurchase();
20:
21:   float fPurchaseAmt, fTaxAmt, fTotalDue;
22:
23:   // String für die gekaufte Kategorie erzeugen
24:   String* pstrCategory = L"Kleidung";
25:
26:   // Kaufpreis angeben
27:   fPurchaseAmt = 45.0;
28:   // Zu berechnende Steuer abfragen
29:   if (pGetTax->GetPurchaseTaxes(fPurchaseAmt,
30:                                 pstrCategory,
31:                                &fTaxAmt))
32:   {
33:     // Summe berechnen
34:     fTotalDue = fPurchaseAmt + fTaxAmt;
35:     // Gekaufte Kategorie anzeigen
36:     Console::WriteLine(pstrCategory);
37:     // Kaufpreis anzeigen
38:     Console::Write("Kaufpreis = ");
39:     Console::WriteLine(fPurchaseAmt.ToString("c"));
40:     // Zu zahlende Steuer anzeigen
41:     Console::Write("   Steuer = ");
42:     Console::WriteLine(fTaxAmt.ToString("c"));
```

```
43:     // Summe anzeigen
44:     Console::Write("    Summe = ");
45:     Console::WriteLine(fTotalDue.ToString("c"));
46:   }
47:   return 0;
48: }
```

Mit der ToString-Methode der verwalteten numerischen Variablen haben Sie den Buchstaben c als einziges Argument übergeben. Es gibt die zu verwendende Formatierung bei der Umwandlung des Werts in einen String an. Die möglichen Formatwerte für diese Methode sind in Tabelle 20.2 aufgelistet.

Formatzeichen	Beschreibung
c, C	Währungsformat (Currency)
d, D	Dezimalformat
e, E	Exponential- (wissenschaftliches) Format
f, F	Festkommaformat
g, G	Allgemeines Format (General)
n, N	Zahlenformat (Zahl)
r, R	Roundtrip-Format (garantiert, dass bei der Rückumwandlung der Strings in Zahlen diese die gleichen Werte wie vor der Umwandlung haben)
x, X	Hexadezimalformat

Tabelle 20.2: Numerische Formate von .NET-ToString

Zuletzt müssen Sie das verwaltete C++-Client-Projekt zur Verwendung in der ATL- Projektmappe konfigurieren. Sie müssen die Verwendung vorkompilierter Header unterbinden und ein vorerstelltes Ereignis einfügen, um die ATL-DLL in das Build- Verzeichnis für den verwalteten Client zu kopieren. Schließlich müssen Sie noch angeben, dass der Visual Studio-Compiler im Ausgabeverzeichnis suchen soll, um #using-Verweise aufzulösen. So kann der Compiler die in der #using-Anweisung angegebene ATL-DLL finden, die Sie in den verwalteten Client-Code eingefügt haben.

Um die Konfigurationsänderungen vorzunehmen, folgen Sie diesen Schritten:

1. Markieren Sie in der Projektmappen-Explorer-Ansicht das Projekt MgdTaxCalcClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Eigenschaften aus dem Kontextmenü.

2. Wählen Sie Alle Konfigurationen im Kombinationsfeld Konfiguration, sodass sich die vorgenommenen Änderungen auf alle Build-Konfigurationen auswirken und nicht nur auf die aktuelle.
3. Auf der linken Seite des Eigenschaftendialogs wählen Sie C/C++, Vorkompilierte Header. Setzen Sie auf der rechten Seite des Dialogfelds den Eintrag Erstellen/ Verwenden vorkompilierter Header auf Vorkompilierte Header nicht verwenden.
4. Auf der linken Seite des Eigenschaftendialogs wählen Sie Buildereignisse, Ereignis vor dem Erstellen. Geben Sie auf der rechten Seite des Dialogfelds beim Eintrag Befehlszeile Folgendes ein:

   ```
   copy "$(SolutionDir)\$(ConfigurationName)\
                       TaxCalculations.dll" $(ConfigurationName)
   ```

   Dadurch wird die DLL vor der Erstellung des verwalteten Clients ins Build- Verzeichnis für den verwalteten Client kopiert.

5. Auf der linken Seite des Eigenschaftendialogs wählen Sie C/C++, Allgemein. Geben Sie auf der rechten Seite des Dialogfelds beim Eintrag #using-Verweise auflösen ein: $(outdir). Das weist den Compiler an, im Build-Ausgabeverzeichnis zu suchen, um #using-Verweise für das Projekt aufzulösen.
6. Klicken Sie auf Ok, um die Änderungen zu übernehmen und den Eigenschaftendialog zu schließen.
7. Markieren Sie in der Projektmappen-Explorer-Ansicht das Projekt MgdTaxCalcClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.

Nun sollten Sie den verwalteten C++-Client für Ihre ATL-Komponente kompilieren und starten können. Wenn Sie den verwalteten Client starten, werden Sie ein Konsolenfenster mit den Ergebnissen des Aufrufs der ATL-Komponente sehen (siehe Abbildung 20.3).

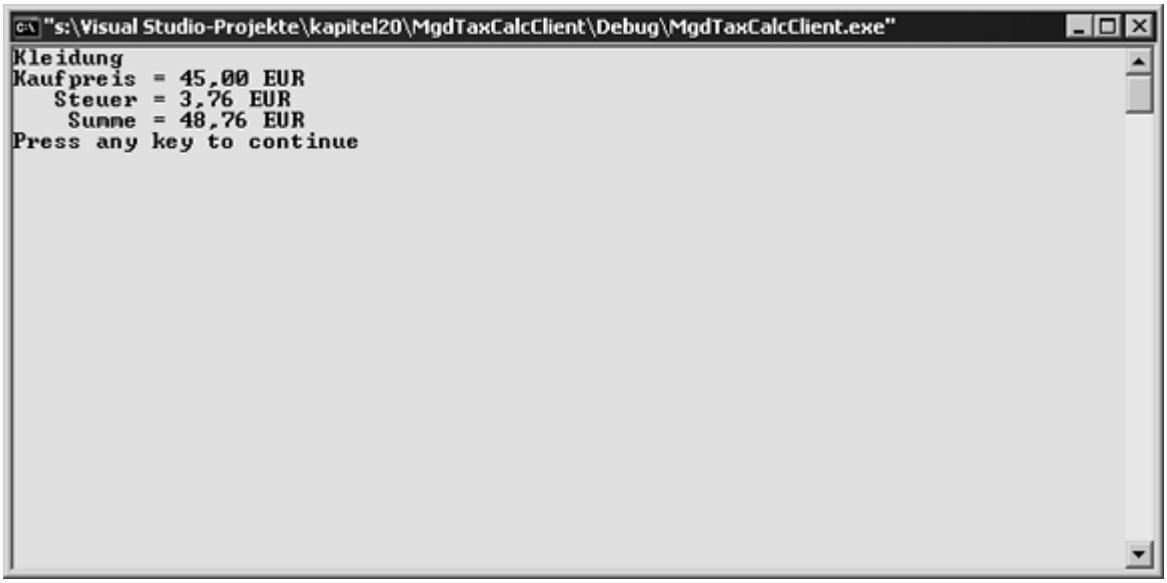


Abbildung 20.3: Die laufende verwaltete Client-Anwendung

Das bei der Ausgabe verwendete Zahlenformat richtet sich nach den in der Systemsteuerung als Währungsformat hinterlegten Angaben. Es kann möglich sein, daß hier ein Fragezeichen ausgegeben wird, das liegt daran, daß ihr Währungsformat mit dem Eurozeichen angegeben ist (_), dieses Zeichen aber vom Konsolenfenster nicht dargestellt werden kann.

20.3 Zusammenfassung

Heute haben Sie die Active Template Library kennen gelernt und erfahren, wie man sie zur Erstellung kleiner COM-Komponenten verwendet, die mit in vielen Programmiersprachen erstellten Anwendungen benutzt werden. Sie haben außerdem gelernt, wie Sie einen verwalteten C++-Wrapper um die Komponente erzeugen und sie damit unabhängig von der Programmiersprache für jede CLR-Anwendung verfügbar machen, indem Sie die Business Logic von den COM-Schnittstellen Ihrer Anwendung getrennt halten. So erhalten Sie eine einzigartige Kombination, die eingesetzt werden kann, um die Geschwindigkeit und die Vorteile von nicht verwaltetem C++ zusammen mit der .NET-Architektur zu verwenden, sodass Sie diese nicht verwalteten Komponenten jeder Anwendung auf allen Plattformen zur Verfügung stellen können. Die Flexibilität dieses Vorgehens ermöglicht Ihnen, bei jeder Komponente zu entscheiden, was wünschenswert ist - verwaltete oder nicht verwaltete Ausführung!

Um ehrlich zu sein, war der heutige Tag nicht mehr als eine Einführung in die Welt der ATL-Programmierung. Es ist ein umfangreiches Thema, über das ganze Bücher geschrieben wurden. Wenn Sie diesen Bereich der Programmierung weiter erforschen möchten, sei Ihnen ein Buch wie Sams Teach Yourself ATL Programming in 21 Days von Kenn Scribner (ISBN 0-672-31867-9) ans Herz gelegt.

20.4 Workshop

Fragen und Antworten

Frage:
Beim Einfügen der Parameter für die COM-Schnittstellenmethode - wozu waren die Kontrollkästchen (in, out, retval) über dem Bereich, in dem die Parameter eingefügt wurden, gut?

Antwort:
Diese Kontrollkästchen fügen die Richtung für jeden Parameter in der IDL- Definition der COM-Schnittstelle hinzu. Wenn Sie die IDL-Sprache in der Header-Datei für die COM-Schnittstelle untersuchen, sehen Sie, dass die Schnittstellendefinition der Methode [in] vor den Parametern enthält, für die Sie das Kontrollkästchen in

aktiviert haben, und [out] vor den Parametern, für die Sie das Kontrollkästchen out aktiviert haben. Das gibt die Richtung und den Zweck eines jeden dieser Parameter in der Methode an. [in] legt fest, dass ein Parameter in die Methode übergeben wird. [out] legt fest, dass der Parameter durch einen Ergebniswert in der Methode ersetzt wird, der an die aufrufende Funktion zurückgegeben werden soll. Sie können für einen Parameter sowohl in als auch out angeben und so festlegen, dass der Parameterwert sowohl beim Aufruf der Methode als auch bei der Rückkehr aus der Methode importiert wird. Wenn Sie das Kontrollkästchen retval aktiviert haben, gibt dieser Parameter an, dass er anstelle des Erfolgs- oder Misserfolgsanzeigers von der Methode zurückgegeben wird.

Frage:
Warum ist die Klasse aus der Klassenansicht verschwunden, als ich den verwalteten Schnittstellen-Wrapper modifizierte?

Antwort:
Das liegt daran, dass das Projekt als nicht verwaltetes Projekt erstellt wurde. Als solches versteht es die in der Deklaration für die verwaltete Wrapper-Klasse verwendeten Schlüsselworte aus verwaltetem C++ nicht. Jede Klasse, Struktur oder Funktion, die Visual Studio im Kontext des Projekts nicht versteht, zeigt es nicht in der Klassenansicht an.

Quiz

1. Was ist eine C++-Klassenvorlage?

2. Welche Standardbibliothek mit Vorlagen ist in jeder C++-Anwendung verfügbar?
3. Was ist der Schlüssel zur Erstellung einer ATL-Komponente, die Sie mit einem verwalteten C++-Wrapper bereitstellen können, sodass sie sowohl eine COM- als auch eine CLR-Schnittstelle haben kann?
4. Warum müssen Sie die Verwendung vorkompilierter Header unterbinden, nachdem Sie dem Projekt den verwalteten Wrapper hinzugefügt haben?

Übung

Fügen Sie Ihrer ATL-Klasse eine zweite Methode hinzu, um den Steuersatz für die angegebene Kategorie zurückzugeben. Modifizieren Sie beide Schnittstellen für diese neue Methode und beide Clients so, dass sie den Steuersatz abfragen und anzeigen.

1

Zum Beispiel »Komponenten entwerfen mit der C++ STL« von Ulrich Breymann, ISBN: 3-8273-1474-7

Tag 21

Interaktion mit Visual Basic .NET und C# .NET-Komponenten

Einer der seit der ersten Ankündigung von Microsoft proklamierten Vorteile der .NET- Plattform und von CLR ist die Leichtigkeit und Mächtigkeit der Interaktion mit in anderen Sprachen erstellten Objekten. Natürlich war das auch schon mit COM möglich, doch dafür mussten Sie COM-Objekte in den verschiedenen Sprachen erstellen und durch die von COM in den Weg gehaltenen Ringe hüpfen. Es bleibt also die Frage, wie einfach und mühelos ist es wirklich, mit in anderen Sprachen erstellten Objekten zu interagieren? Das werde Sie heute untersuchen und lernen.

Heute werden Sie lernen, ...

- verwaltete C++-Komponenten zu erstellen, die von C#- und VB.NET-Anwendungen aus aufgerufen werden können,
- VB.NET- und C#-Komponenten von verwalteten C++-Anwendungen aus aufzurufen,
- andere CLR-Komponenten von nicht verwalteten Anwendungen aus aufzurufen.

21.1 Sprachen mischen: das Versprechen der CLR erfüllen

Gestern haben Sie gelernt, wie man eine COM-Komponente über ATL erstellt und diese Komponente in einem verwalteten C++-Wrapper verpackt, um sie anderen CLR- Anwendungen oder -Objekten zur Verfügung zu stellen. Das ist eine Möglichkeit, nicht verwaltete C++-Objekte für die Verwendung in mit anderen CLR-Anwendungen erstellten Anwendungen bereitzustellen, aber wie funktioniert das eigentlich? Und können Sie auch den entgegengesetzten Weg gehen und in anderen Sprachen erstellte .NET-Objekte in nicht verwalteten C++-Anwendungen verwenden?

Die schnelle Antwort ist: Ja, Sie können andere .NET-Objekte in nicht verwalteten C++- Anwendungen verwenden und Sie können verwaltete Objekte erstellen, die in anderen .NET-Sprachen verwendet werden können. Es ist zwar viel einfacher, mit verwaltetem C++ erstellte (oder in verwaltetem C++ verpackte) Objekte in anderen .NET-Sprachen zu verwenden, doch es ist möglich, den entgegengesetzten Weg zu nehmen und in anderen .NET-Sprachen erstellte Objekte sowohl in verwalteten als auch in nicht verwalteten C++-Anwendungen zu verwenden.

Verwaltete C++-Objekte in C# .NET und VB.NET

Von C#- und VB.NET-Projekten aus auf in anderen Sprachen erstellte Objekte zuzugreifen ist wirklich einfach. Wenn Sie ältere Versionen von Visual Basic verwendet haben, wissen Sie wahrscheinlich, wie man Verweise auf COM-Objekte einfügt, um diese Objekte in Ihrem Projekt zu verwenden. Bei VB.NET und C# ist es grundsätzlich der gleiche Prozess, der Ihnen die Verwendung anderer CLR-Objekte ermöglicht, unabhängig von der Sprache, in der die Objekte erstellt wurden - einschließlich verwaltetem C++.

Wenn Sie ein C#- (C-sharp ausgesprochen, englisch für die Musiknote »Cis«) oder VB.NET-Projekt unter dem Projektknoten in der Projektmappen-Explorer-Ansicht erstellen, finden Sie einen mit Verweise beschrifteten Knoten. Wenn Sie diesen Knoten markieren und mit der rechten Maustaste anklicken, finden Sie im Kontextmenü die Option Verweis hinzufügen, die das Dialogfeld Verweis hinzufügen öffnet (siehe Abbildung 21.1).

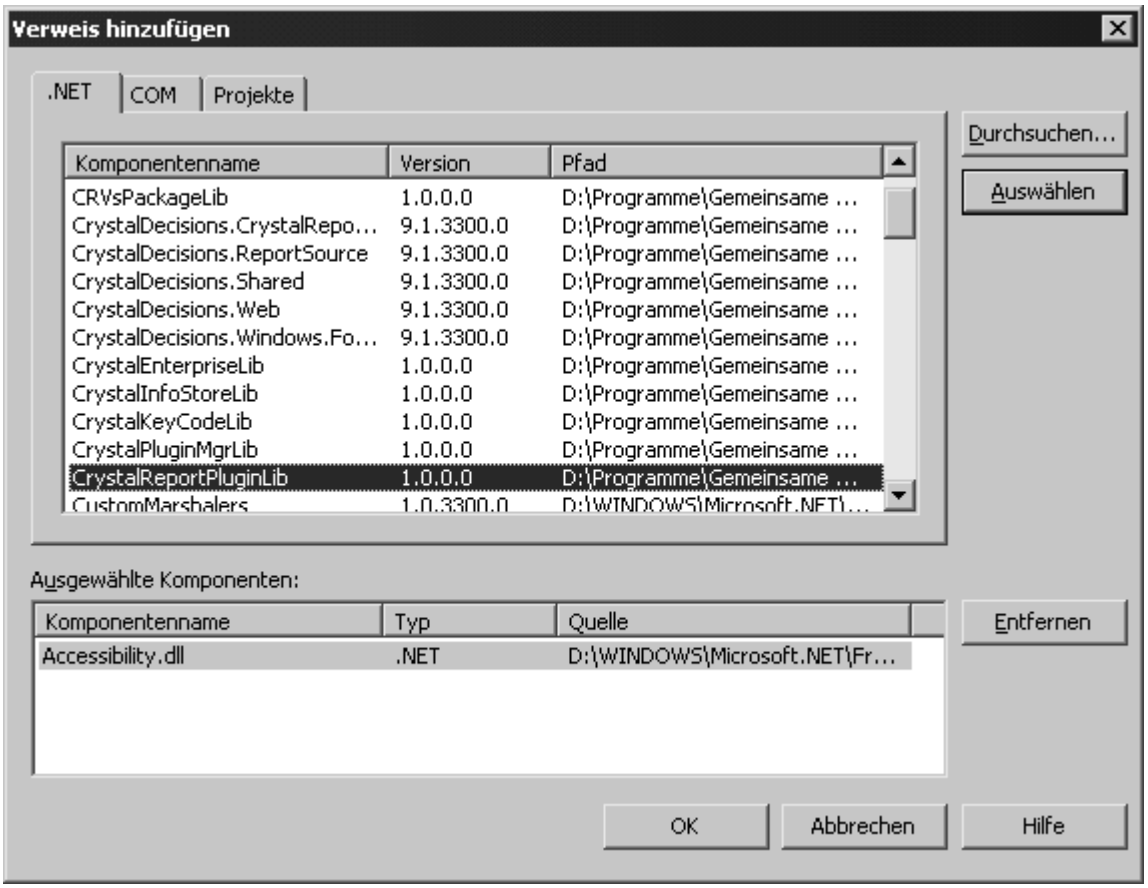


Abbildung 21.1: Das Dialogfeld Verweis hinzufügen

Auf der .NET-Seite finden Sie die verschiedenen für Ihre Anwendung verfügbaren registrierten .NET-Objekte. Auf der COM-Seite finden Sie alle auf Ihrem Computer registrierten COM-Objekte einschließlich der gestern erstellten Komponente (wenn Sie gestern die Beispielanwendung erstellt haben). Auf der Projektseite finden Sie eine Liste von Projekten in der aktuellen Projektmappe. Zusätzlich können Sie nach anderen Objekten in anderen Projekten suchen.

Nachdem Sie ein Objekt, auf das Sie verweisen möchten, markiert haben, klicken Sie auf die Schaltfläche Auswählen, um es zu der Liste der referenzierten Objekte im unteren Teil des Dialogfelds hinzuzufügen. Wenn Sie eine Objektreferenz löschen möchten, markieren Sie sie im unteren Teil und klicken Sie auf die Schaltfläche Entfernen. Nachdem Sie Verweise auf alle benötigten Objekte aufgenommen haben, klicken Sie auf Ok, um all diese Objekte als Knoten unter dem Ordner Verweise der Projektmappen- Explorer-Ansicht hinzuzufügen.

Nachdem Sie Verweise auf die Objekte aufgenommen haben, können Sie diese beliebig in Ihrem VB.NET- oder C#-Code einbinden. Wenn Sie beispielweise einen Verweis auf die gestern erstellte Komponente in ein C#-Projekt aufgenommen haben, verwenden Sie es folgendermaßen:

```
MgdGetTaxForPurchase pGetTax = new MgdGetTaxForPurchase();

if (pGetTax.GetPurchaseTaxes(fPurchaseAmt, pstrCategory,
    ref fTaxAmt))
...
```

Wenn Sie die Komponente in einer VB.NET-Anwendung verwenden, tun Sie das folgendermaßen:

```
Dim pGetTax as MgdGetTaxForPurchase

pGetTax = New MgdGetTaxForPurchase()
if (pGetTax.GetPurchaseTaxes(fPurchaseAmt, pstrCategory,
    byref fTaxAmt))
...
```

Wenn Sie nun noch einmal zum gestrigen Beispiel zurückkehren und versuchen, C#- und VB.NET-Clients für die Komponente zu erzeugen, stellen Sie fest, dass diese nicht korrekt funktionieren. Es müssen ein paar Änderungen im verwalteten C++-Wrapper von gestern vorgenommen werden, bevor Sie ihn in C#- oder VB.NET-Anwendungen einsetzen können.

- Die zu berechnenden Steuern sollten als Ergebnis zurückgeliefert und nicht als Parameter übergeben werden.
- Für die float-Variablen müssen Sie in VB.NET Single-Variablentypen verwenden. Stattdessen können Sie auch double an Stelle von float verwenden.

Nachdem Sie diese Änderungen vorgenommen haben, sieht Ihr C#-Code so aus:

```
MgdGetTaxForPurchase pGetTax = new MgdGetTaxForPurchase();

fTaxAmt = pGetTax.GetPurchaseTaxes(fPurchaseAmt,
                                   pstrCategory);
...
```

und der VB.NET-Code sieht so aus:

```
Dim pGetTax as MgdGetTaxForPurchase

pGetTax = New MgdGetTaxForPurchase()
sgTaxAmt = pGetTax.GetPurchaseTaxes(sgPurchaseAmt,
                                    pstrCategory)
...
```

An Ihrem verwalteten C++-Wrapper müssen Sie folgende Änderungen vornehmen:

```
float MgdGetTaxForPurchase::GetPurchaseTaxes(
      float fPurchaseAmt, System::String* pstrCategory)
{
  wchar_t *pCat;
  int iLen;
  float fTax;
  // Kategorie von BSTR in wchar_t umwandeln
  __wchar_t pCategory __gc[] = pstrCategory->ToCharArray();
  iLen = pstrCategory->get_Length();
  pCat = new wchar_t[iLen + 1];
  for (int i = 0; i < iLen; i++)
  {
    pCat[i] = pCategory[i];
  }
  pCat[i] = '\0';
  // Zu zahlende Steuer berechnen
  if (m_pTaxCalc->CalculateTaxes(fPurchaseAmt, pCat, &fTax))
    return fTax;
  else
    return 0.0;
}
```

C#- und VB.NET-Objekte in verwaltetem C++

Die Formel umzudrehen und zu versuchen, auf in C# und VB.NET erstellte Objekte von verwalteten C++-Anwendungen aus zuzugreifen, ist grundsätzlich das Gleiche wie die gestern verwendete Methode, um auf den verwalteten C++-Wrapper des COM-Objekts zuzugreifen. Sie platzieren eine #using-Anweisung am Anfang des Quellcodes, kopieren die DLL (oder eine andere Objektdatei, in der sich die Objekte befinden) in das Ausgabeverzeichnis und weisen dann den Compiler an, das Ausgabeverzeichnis zum Auflösen von #using-Anweisungen zu verwenden.

Der größte Unterschied in der Konfiguration ist, dass Sie bei der Arbeit mit C#- und VB.NET-Komponenten, die Teil des gleichen Projekts sind, ein bin-Verzeichnis zwischen dem Projektverzeichnis und dem

Konfigurationsnamen der Komponente einfügen müssen. Hätten Sie die gestrige Anwendung beispielsweise mit C# anstatt mit ATL und verwaltetem C++ erstellt, würden Sie den Befehl im Pre-Build-Ereignisbefehl folgendermaßen ersetzen:

```
copy "$(SolutionDir)\bin\$(ConfigurationName)\
        ProjectName.DLL" $(ConfigurationName)
```

Denselben String hätten Sie eingegeben, wenn die Komponente mit VB.NET erstellt worden wäre. Dieses Pre-Build-Ereignis erstellt eine Kopie der DLL, in der die Komponente verpackt ist, im Ausgabeverzeichnis des Client-Projekts. Das geschieht vor der Erstellung des Client-Projekts, sodass dieses seine #using-Anweisung mit dem Ausgabeverzeichnis auflösen kann.

C#- und VB.NET-Objekte in nicht verwaltetem C++

Kompliziert wird es, wenn Sie von einer nicht verwalteten Anwendung aus auf verwaltete Komponenten zugreifen möchten. Es ist so, als würden Sie Ihre ATL-COM-Komponente in einem verwalteten Wrapper verpacken, nur dass Sie diesmal einen verwalteten C++- Wrapper um die CLR-Komponenten herum einfügen.

Sie müssen eine verwaltete C++-Klasse als Teil der nicht verwalteten Anwendung erstellen. Diese verwaltete C++-Klasse dient als Schnittstelle zwischen den CLR- Komponenten, unabhängig von der Sprache, in der sie erstellt wurden, und dem Rest der nicht verwalteten Anwendung. Um das Ganze so einfach wie möglich zu halten, sollte Ihre verwaltete C++-Klasse nicht speicherbereinigt sein und Sie sollten alle #using-Direktiven nicht in der Header-Datei, sondern in der Quellcode-Datei (.cpp) unterbringen. Das ist eine Vereinfachung, da der restliche nicht verwaltete C++-Code, der auf die verwaltete Klasse zugreifen soll, in der Anwendung die Header-Datei der verwalteten Klasse einbinden muss. Wenn die Header-Datei die #using-Direktiven oder verwalteten Erweiterungsschlüsselworte (wie __gc, das die Klasse speicherbereinigt) enthält, müssen diese Klassen ebenfalls als verwaltetes C++ kompiliert werden, um die Direktiven und Schlüsselworte zu verstehen. Alternativ können Sie das alles mithilfe der Präprozessor- Direktive #define definieren und die Schlüsselworte durch passende Alternativen (beispielsweise Leerzeichen) ersetzen.

Wie bei der ATL-Komponente, die Sie gestern verpackt haben, müssen Sie die Verwendung vorkompilierter Header in Ihrer Anwendung unterbinden, ebenso wie die Option, im Debug-Modus zu bearbeiten und fortzufahren. Abhängig von Ihrem nicht verwalteten Projekt müssen Sie möglicherweise ein paar andere Konfigurationsoptionen anpassen. Wenn Sie versuchen, das Projekt zu kompilieren und wenn eine andere Option eine andere Einstellung benötigt, erhalten Sie eine Fehlermeldung vom Compiler, die die umzustellende Compileroption angibt. Die Compileroptionen werden in Befehlszeilenform angezeigt (die an den Compiler übergebenen Flags, wenn Sie ihn vom DOS-Prompt aus aufrufen würden), also müssen Sie möglicherweise den Projekt- Eigenschaftendialog durchsuchen, um die umzustellende Option zu finden.

Ein weiterer Aspekt, der beachtet werden sollte, ist die veränderliche Kompatiblität. Ein wichtiger Variablentyp, der Probleme verursachen kann, ist der string-Typ. Wenn die CLR-Komponenten, die Sie aufrufen, eine Stringvariable als Parameter für eine Methode benötigt, können Sie meist wie folgt ein wchar_t-Array übergeben:

```
wchar_t wszString[20];

pClrObject->SomeMethod(wszString);
```

21.2 Eine verwaltete C++-Komponente mit C#-Client erstellen

Heute erstellen Sie nicht nur eine, sondern drei Kombinationen aus Komponente und Anwendung, um zu zeigen, wie die verschiedenen Interaktionsszenarien funktionieren. Die erste Kombination erstellt noch einmal das gestrige Projekt als reine verwaltete C++- Komponente und erzeugt für den Zugriff darauf einen C#- und/oder VB.NET-Client. Als Nächstes erstellen Sie die Komponente entweder mit C# oder mit VB.NET und greifen über einen verwalteten C++-Client darauf zu. Zuletzt greifen Sie über einen nicht verwalteten C++-Client auf die C#-/VB.NET-Komponente zu.

Die verwaltete C++-Komponente erstellen

Um mit dem ersten dieser Projekte zu beginnen, folgen Sie diesen Schritten:

1. Erzeugen Sie ein neues Verwaltete C++-Klassenbibliotheks-Projekt und nennen Sie es MgdTaxCalc

2. Öffnen Sie die Header-Datei MgdTaxCalc.h.
3. Ersetzen Sie dann den Klassennamen Class1 durch den neuen Klassennamen CTaxCalculator.
4. Fügen Sie der Klasse CTaxCalculator eine neue Member-Funktion hinzu (Sie müssen den Namespace-Knoten ({}) von MgdTaxCalc in der Klassenansicht erweitern). Legen Sie den Rückgabetyp der Funktion als double und den Namen als CalculateTaxes fest und fügen Sie zwei Parameter ein. Legen Sie für den ersten Parameter den Typ als double und den Namen als dbPurchaseAmt fest. Legen Sie für den zweiten Parameter den Typ als String* und den Namen als strCategory fest. Legen Sie als Zugriff für die Funktion public fest.
5. Bearbeiten Sie die Funktion in der Header-Datei und nehmen Sie den Code aus Listing 21.1 auf.

Listing 21.1: Die Funktion CalculateTaxes

```
 1: double CalculateTaxes(double dbPurchaseAmt,
 2:                       String* strCategory)
 3: {
 4:   // Array mit den Steuerkategorien deklarieren
 5:   String* strCategories[] = {S"Nahrungsmittel", S"Kleidung",
 6:                              S"Musik"};
 7:   // Array mit den Steuersätzen deklarieren
 8:   double dbRates[] = {0.0725, 0.0835, 0.081};
 9:   // Schleife durch die Kategorien
10:   for (int i=0; i < 3; i++)
11:   {
12:     // Wurde die Kategorie gekauft?
13:     if (strCategory == strCategories[i])
14:     {
15:       // Ja, zu zahlende Steuer berechnen
16:       return dbPurchaseAmt * dbRates[i];
17:     }
18:   }
19:   // Gekaufte Kategorie nicht gefunden, 0 zurückgeben
20:   return 0;
21: }
```

Mehr braucht es nicht für die verwaltete C++-Komponente. Der Code entspricht weitgehend dem der nicht verwalteten Komponente von gestern, er verwendet nur verwaltete Strings und gibt die zu zahlende Steuer als Ergebniswert zurück.

Den C#- oder VB.NET-Client erstellen

Der nächste Schritt in diesem Beispiel fügt dieser Projektmappe den C#- bzw. VB.NET- Client hinzu. Die Client-Anwendung dupliziert die Funktionalität aus der verwalteten C++-Client-Anwendung, die Sie gestern erstellt haben. Um den Client zu erstellen, folgen Sie diesen Schritten:

1. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektmappen-Knoten am oberen Ende des Baums und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Neues Projekt aus dem Kontextmenü.

2. Klicken Sie auf Visual Basic-Projekte oder Visual C#-Projekte auf der linken Seite des Dialogfelds Neues Projekt hinzufügen. Wählen Sie auf der rechten Seite des Dialogfelds Konsolenanwendung (dieser Anwendungstyp ist sowohl für VB.NET- als auch für C#-Projekte verfügbar). Wenn Sie ein Visual Basic-Projekt erstellen, nennen Sie es VBClient. Ein C#-Projekt nennen Sie CSClient. Klicken Sie auf Ok, um das Projekt zu erzeugen.
3. Markieren Sie in der Projektmappen-Explorer-Ansicht den Knoten Verweise im gerade erzeugten Projekt und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Verweis hinzufügen aus dem Kontextmenü.
4. Wählen Sie die Registerkarte Projekte im Dialogfeld Verweis hinzufügen. Sie sollten Ihr verwaltetes C++-Komponenten-Projekt (MgdTaxCalc) in der Projektliste sehen. Klicken Sie auf das Projekt und dann auf Auswählen. Klicken Sie auf Ok, um den Verweis auf das Projekt einzufügen.
5. Wenn Sie ein C#-Projekt erstellt haben, öffnen Sie die Datei Class1.cs. Wenn Sie ein VB-Projekt

erstellt haben, öffnen Sie die Datei Module1.vb.

6. Wenn Sie ein C#-Projekt erstellt haben, nehmen Sie den Code aus Listing 21.2 in die Funktion Main auf. Wenn Sie ein VB-Projekt erstellt haben, nehmen Sie den Code aus Listing 21.3 in die Funktion Main auf.

Listing 21.2: Die Main-Funktion des C#-Clients

```
 1: static void Main(string[] args)
 2: {
 3:   //
 4:   // TODO: Fügen Sie hier Code hinzu, um die Anwendung zu
 5:   // starten
 6:   //
 7:   MgdTaxCalc.CTaxCalculator pGetTax =
 8:                             new MgdTaxCalc.CTaxCalculator();
 9:
10:   double dbPurchaseAmt, dbTaxAmt, dbTotalDue;
11:
12:   // String für die gekaufte Kategorie erzeugen
13:    String strCategory = "Kleidung";
14:
15:   // Kaufpreis
16:   dbPurchaseAmt = 45.0;
17:   // Zu berechnende Steuer abfragen
18:   dbTaxAmt = pGetTax.CalculateTaxes(dbPurchaseAmt,
19:                                     strCategory);
20:   if (dbTaxAmt > 0.0)
21:   {
22:     // Summe berechnen
23:     dbTotalDue = dbPurchaseAmt + dbTaxAmt;
24:     // Gekaufte Kategorie anzeigen
25:     Console.WriteLine(strCategory);
26:     // Preis der Ware anzeigen
27:     Console.Write("Kaufpreis = ");
28:     Console.WriteLine(dbPurchaseAmt.ToString("c"));
29:     // Zu zahlende Steuer anzeigen
30:     Console.Write("   Steuer = ");
31:     Console.WriteLine(dbTaxAmt.ToString("c"));
32:     // Summe anzeigen
33:     Console.Write("    Summe = ");
34:     Console.WriteLine(dbTotalDue.ToString("c"));
35:   }
36: }
```

Listing 21.3: Die Main-Funktion des VB.NET-Clients

```
 1: Sub Main()
 2:   Dim pGetTax As MgdTaxCalc.CTaxCalculator
 3:   pGetTax = New MgdTaxCalc.CTaxCalculator()
 4:
 5:   Dim dbPurchaseAmt As Double
 6:   Dim dbTaxAmt As Double
 7:   Dim dbTotalDue As Double
 8:
 9:   ' String für die gekaufte Kategorie erzeugen
10:   Dim strCategory As String
11:   strCategory = "Kleidung"
12:
13:   ' Kaufpreis angeben
14:   dbPurchaseAmt = 45
15:   ' Zu berechnende Steuer abfragen
16:   dbTaxAmt = pGetTax.CalculateTaxes(dbPurchaseAmt,
17:                                     strCategory)
```

```
18:    If (dbTaxAmt > 0) Then
19:      ' Summe berechnen
20:      dbTotalDue = dbPurchaseAmt + dbTaxAmt
21:      ' Gekaufte Kategorie anzeigen
22:      Console.WriteLine(strCategory)
23:      ' Kaufpreis anzeigen
24:      Console.Write("Kaufpreis = ")
25:      Console.WriteLine(dbPurchaseAmt.ToString("c"))
26:      ' Zu zahlende Steuer anzeigen
27:      Console.Write("   Steuer = ")
28:      Console.WriteLine(dbTaxAmt.ToString("c"))
29:      ' Summe anzeigen
30:      Console.Write("    Summe = ")
31:      Console.WriteLine(dbTotalDue.ToString("c"))
32:    End If
33:
34: End Sub
```

7. Markieren Sie in der Projektmappen-Explorer-Ansicht das eben erstellte Projekt und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.

Sie sollten die Beispielanwendung nun kompilieren und ausführen können und die Ausgabe in Abbildung 21.2 sehen. Sie können diese Schritte auch noch einmal wiederholen, um die Client-Anwendung in der zweiten Sprache zu erstellen, wenn Sie Visual Studio und nicht nur Visual C++ benutzen (und damit beide zusätzliche Sprachen zur Verfügung haben).

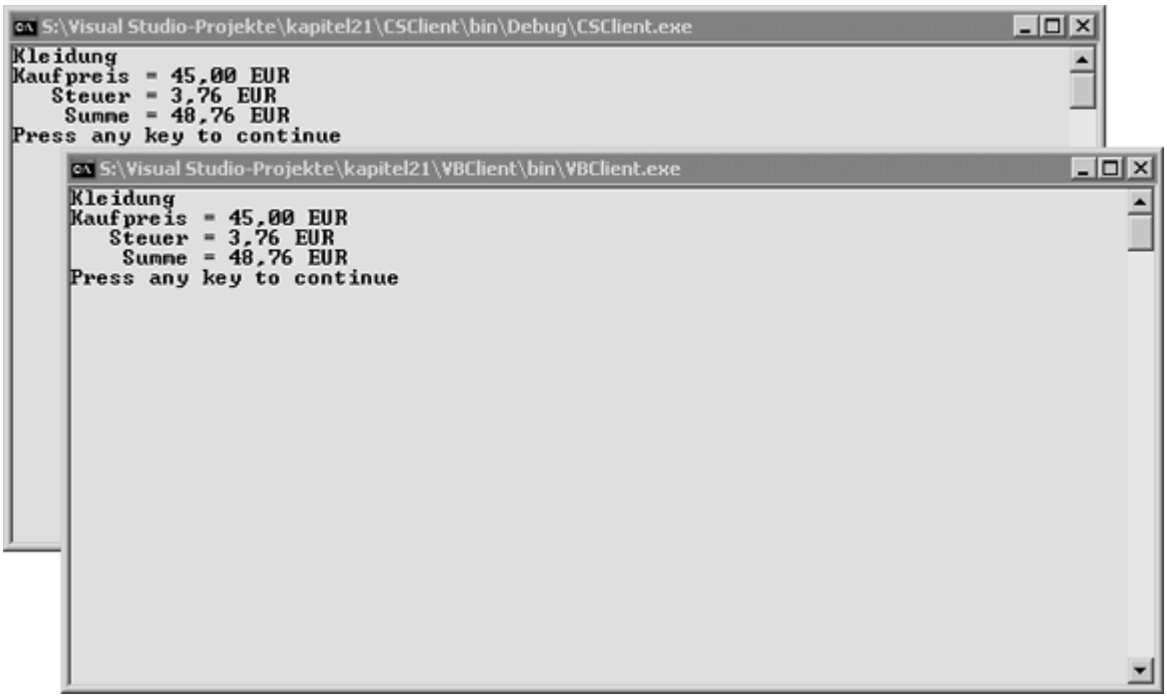


Abbildung 21.2: Die laufende Anwendung

21.3 Eine C#-Komponente mit verwaltetem C++-Client erstellen

Das nächste Beispiel kehrt die Situation des vorhergegangenen um. In diesem Beispiel erstellen Sie die Komponente entweder mit VB.NET oder mit C# und erzeugen dann einen verwalteten C++-Client, der auf die Komponente zugreift. Wegen der im verwalteten C++- Client verwendeten Makros müssen Sie eine vollständig neue Solution beginnen, wenn Sie die Komponente in einer anderen Sprache als der bisher gewählten erstellen wollen.

Die C#- oder VB.NET-Komponente erstellen

Um dieses Beispielprojekt zu erstellen, folgen Sie diesen Schritten:

1. Wählen Sie Neu / Projekt aus dem Menü Datei.

2. Um die Komponente in C# zu erstellen, wählen Sie Visual C#-Projekte auf der linken Seite des Dialogfelds Neues Projekt. Um die Komponente in VB zu erstellen, wählen Sie Visual Basic-Projekte auf der linken Seite des Dialogfelds. Bei beiden Komponentenarten wählen Sie Klassenbibliothek auf der rechten Seite des Dialogfelds. Wenn Sie eine C#-Komponente erstellen, nennen Sie das Projekt CSTaxCalc. Wenn Sie eine VB-Komponente erstellen, nennen Sie das Projekt VBTaxCalc. Wenn das vorherige Projekt noch geöffnet ist, achten Sie darauf, dass das Optionsfeld Projektmappe schliessen aktiviert ist. Klicken Sie auf Ok, um das neue Projekt zu erzeugen.
3. Wenn Sie ein C#-Projekt erstellt haben, öffnen Sie die Datei Class1.cs. Wenn Sie ein VB-Projekt erstellt haben, öffnen Sie die Datei Class1.vb. Ersetzen Sie Class1 an allen Stellen durch CTaxCalculations.
4. Wenn Sie die Datei umbenennen möchten, in der sich die Komponente befindet, markieren Sie die Datei Class1.cs bzw. Class1.vb in der Projektmappenexplorer- Ansicht. Drücken Sie (F2) und nennen Sie die Datei TaxCalculations. Achten Sie darauf, die Dateinamenserweiterung nicht zu ändern (wenn es sich beispielsweise um ein C#-Projekt handelt, sollte der neue Dateiname TaxCalculations.cs sein, bei einem VB-Projekt sollte er TaxCalculations.vb sein).
5. Wenn Sie eine Visual Basic-Komponente erstellen, nehmen Sie in die Quellcode- Datei die Funktion aus Listing 21.4 auf (Sie können bei einer Visual Basic- Komponente nicht das Dialogfeld Memberfunktion hinzufügen verwenden). Die restlichen Schritte in diesem Abschnitt können Sie dann überspringen, die VB- Komponente ist damit fertig gestellt.

Listing 21.4: Die VB.NET-Funktion CalculateTaxes

```
 1: Public Class CTaxCalculations
 2:   Public Function CalculateTaxes(ByVal fPurchaseAmt
 3:          As Double, _ ByVal strCategory As String) As Double
 4:     ' Array mit den Steuerkategorien deklarieren
 5:     Dim strCategories() As String = {"Nahrungsmittel",
 6:                                      "Kleidung", "Musik"}
 7:     ' Array mit den Steuersätzen deklarieren
 8:     Dim fRates() As Double = {0.0725, 0.0835, 0.081}
 9:     Dim i As Int32
10:
11:     ' Standardrückgabewert voreinstellen
12:     CalculateTaxes = 0
13:     ' Schleife durch die Kategorien
14:     For i = 0 To 2
15:       ' Wurde diese Kategorie gekauft?
16:       If (strCategory = strCategories(i)) Then
17:         ' Ja, zu zahlende Steuer berechnen
18:         CalculateTaxes = fPurchaseAmt * fRates(i)
19:       End If
20:     Next
21:   End Function
22:
23: End Class
```

> Wenn Sie eine C#-Komponente erstellen, markieren Sie die Klasse CTaxCalculations in der Klassenansicht und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Methode hinzufügen aus dem Kontextmenü.

6. Legen Sie im C#-Methoden-Assistenten den Zugriff für die Methode als public, den Rückgabetyp als double und den Namen als CalculateTaxes fest und fügen Sie zwei Parameter ein. Legen Sie für den ersten Parameter den Modifizierer als Keine, den Parametertyp als double und den Parameternamen als dbPurchaseAmt fest. Für den zweiten Parameter legen Sie den Modifizierer als Keine, den Parametertyp als string und den Parameternamen als strCategory fest. Klicken Sie auf Fertig stellen, um die Methode hinzuzufügen.
7. Nehmen Sie in die Methode den Code aus Listing 21.5 auf.

Listing 21.5: Die C#-Funktion CalculateTaxes

```
 1: public double CalculateTaxes(double dbPurchaseAmt,
 2:                               string strCategory)
 3: {
 4:   // Array mit den Steuerkategorien deklarieren
 5:   String[] strCategories = {"Nahrungsmittel", "Kleidung",
 6:                             "Musik"};
 7:   // Array mit den Steuersätzen deklarieren
 8:   double[] dbRates = {0.0725, 0.0835, 0.081};
 9:   // Schleife durch die Kategorien
10:   for (int i=0; i < 3; i++)
11:   {
12:     // Wurde diese Kategorie gekauft?
13:     if (strCategory == strCategories[i])
14:     {
15:       // Ja, zu zahlende Steuer berechnen
16:       return dbPurchaseAmt * dbRates[i];
17:     }
18:   }
19:   // Gekaufte Kategorie nicht gefunden, 0 zurückgeben
20:   return 0;
21: }
```

Den verwalteten C++-Client erstellen

Der nächste Schritt in diesem Beispiel erzeugt die verwaltete C++-Client-Anwendung, die auf die mit VB oder C# erstellte Komponente zugreift. Um den Client zu erstellen, folgen Sie diesen Schritten:

1. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektmappen-Knoten am oberen Ende des Baums und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Hinzufügen / Neues Projekt, wenn Sie dieses Beispiel zum ersten Mal erstellen. Wählen Sie Hinzufügen / Vorhandenes Projekt, wenn Sie dieses Beispiel zum zweiten Mal mit der zweiten Sprache erstellen und navigieren Sie dann zum Projektverzeichnis des verwalteten C++-Clients und wählen die Projektdatei aus.

2. Wenn Sie ein neues Projekt hinzufügen, wählen Sie Visual C++-Projekte auf der linken Seite des Dialogfelds und Verwaltete C++-Anwendung auf der rechten Seite. Nennen Sie das Projekt MgdClient. Klicken Sie auf Ok, um das Projekt zu erzeugen.
3. Wenn Sie die Komponente mit C# erstellt haben, nehmen Sie den fett gedruckten Code in Listing 21.6 auf. Wenn Sie die Komponente mit VB erstellt haben, nehmen Sie ebenfalls den fett gedruckten Code aus Listing 21.6 auf, ersetzen jedoch CSTaxCalc an allen Stellen durch VBTaxCalc.

Listing 21.6: Die Main-Funktion des verwalteten C++-Clients

```
 1: // Die Hauptprojektdatei für ein VC++-Anwendungsprojekt,
 2: // das mit dem Anwendungs-Assistenten generiert wurde.
 3:
 4: #using <mscorlib.dll>
 5: #include <tchar.h>
 6:
 7: #using "CSTaxCalc.dll"
 8:
 9: using namespace System;
10:
11: // Dies ist der Einstiegspunkt für die Anwendung
12: int tmain(void)
13: {
14:   // TODO: Ersetzen Sie den Beispielcode durch Ihren eigenen
15:   // Code.
16:   CSTaxCalc::CTaxCalculations* pGetTax =
17:                           new CSTaxCalc::CTaxCalculations();
18:
19:   double dbPurchaseAmt, dbTaxAmt, dbTotalDue;
20:
21:   // String für die gekaufte Kategorie erzeugen
```

```
22:   String* pstrCategory = S"Kleidung";
23:
24:   // Kaufpreis angeben
25:   dbPurchaseAmt = 45.0;
26:   // Zu berechnende Steuer abfragen
27:   dbTaxAmt = pGetTax->CalculateTaxes(dbPurchaseAmt,
28:                                      pstrCategory);
29:   if (dbTaxAmt > 0.0)
30:   {
31:     // Summe berechnen
32:     dbTotalDue = dbPurchaseAmt + dbTaxAmt;
33:     // Gekaufte Kategorie anzeigen
34:     Console::WriteLine(pstrCategory);
35:     // Gekaufte Menge anzeigen
36:     Console::Write("Kaufpreis = ");
37:     Console::WriteLine(dbPurchaseAmt.ToString("c"));
38:     // Zu zahlende Steuer anzeigen
39:     Console::Write("   Steuer = ");
40:     Console::WriteLine(dbTaxAmt.ToString("c"));
41:     // Summe anzeigen
42:     Console::Write("    Summe = ");
43:     Console::WriteLine(dbTotalDue.ToString("c"));
44:   }
45:
46:   return 0;
47: }
```

4. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektknoten des Projekts MgdClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Eigenschaften aus dem Kontextmenü.
5. Wählen Sie Alle Konfigurationen aus dem Kombinationsfeld Konfiguration.
6. Wählen Sie Buildereignisse, Ereignis vor dem Erstellen auf der linken Seite des Dialogfelds Eigenschaften. Auf der rechten Seite des Dialogfelds geben Sie beim Eintrag Befehlszeile Folgendes ein, wenn Sie die Komponente mit C# erstellt haben:

   ```
   copy "$(SolutionDir)\bin\$(ConfigurationName)\CSTaxCalc.dll"
        $(ConfigurationName)
   ```

   Wenn Sie die Komponente mit VB erstellt haben, geben Sie dies ein:

   ```
   copy "$(SolutionDir)\bin\VBTaxCalc.dll" $(ConfigurationName)
   ```

7. Wählen Sie auf der linken Seite des Dialogfelds C/C++, Allgemein. Geben Sie $(outdir) auf der rechten Seite des Dialogfelds beim Eintrag #using-Verweise auflösen ein.
8. Klicken Sie auf Ok, um die Änderungen an der Konfiguration zu übernehmen und den Eigenschaftendialog zu schließen.
9. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektknoten des Projekts MgdClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.

Sie können diese Beispielanwendung nun kompilieren und ausführen. Sie sollten das gleiche Ausgabefenster wie beim letzten Beispiel mit den vertauschten Sprachrollen sehen (siehe Abbildung 21.2).

21.4 Einen nicht verwalteten C++-Client erstellen

Beim letzten Beispiel wird es richtig zäh. Den Code zu erstellen, der einer nicht verwalteten C++-Komponente den Zugriff auf mit C# und VB.NET erstellte Komponenten ermöglicht, ist ziemlich einfach. Bei der Konfiguration des Projekts wird es allerdings sehr unübersichtlich.

Um dieses Projekt zu erstellen, folgen Sie diesen Schritten:

1. Wählen Sie Neu / Projekt aus dem Menü Datei.

2. Wählen Sie Visual C++-Projekte auf der linken Seite des Dialogfelds Neues Projekt und Win32-Projekt auf der rechten Seite aus. Nennen Sie das Projekt UnmgdClient. Wenn das vorherige Beispiel noch geöffnet ist, achten Sie darauf, dass das Optionsfeld Projektmappe schliessen aktiviert ist. Klicken Sie auf Ok, um das Projekt zu erzeugen.

Die verwaltete C++-Schnittstelle erstellen

Die verwaltete Schnittstelle für die CLR-Komponenten wird genauso wie der verwaltete Wrapper für Ihre ATL-Komponente erstellt. Um die Schnittstelle zu erstellen, folgen Sie diesen Schritten:

1. Fügen Sie mit der Methode der vergangenen Tage eine neue generische Klasse zum Projekt hinzu. Nennen Sie die neue Klasse MgdClient.

2. Fügen Sie der Klasse MgdClient eine neue Funktion hinzu. Legen Sie den Rückgabewert für die neue Funktion als double, den Namen als GetTaxes und den Zugriff als public fest und fügen Sie zwei Parameter ein. Legen Sie für den ersten Parameter den Parametertyp als double und den Namen als dbPurchaseAmt fest. Für den zweiten Parameter legen Sie den Typ als wchar_t* und den Namen als strCategory fest.
3. Nehmen Sie in die neue Funktion den Code aus Listing 21.7 auf.

Listing 21.7: Die Funktion GetTaxes

```
 1: double MgdClient::GetTaxes(double dbPurchaseAmt,
 2:                            wchar_t* strCategory)
 3: {
 4:   double dbTaxAmt;
 5:
 6:   // Eine Instanz des CLR-Objekts deklarieren und erzeugen
 7:   CSTaxCalc::CTaxCalculations* pGetTax =
 8:                           new CSTaxCalc::CTaxCalculations();
 9:   // Zu berechnende Steuer abfragen
10:   dbTaxAmt = pGetTax->CalculateTaxes(dbPurchaseAmt,
11:                                      strCategory);
12:   // Ergebnis zurückgeben
13:   return dbTaxAmt;
14: }
```

4. Wenn Sie die C#-Komponente verwenden, nehmen Sie am oberen Ende der Datei MgdClient.cpp die Direktiven aus Listing 21.8 auf. Wenn Sie die VB-Komponente verwenden, ersetzen Sie das Wort CSTaxCalc in Listing 21.7 und 21.8 an allen Stellen durch VBTaxCalc.

Listing 21.8: Die Direktiven für MgdClient

```
1: #include "StdAfx.h"
2: #include "mgdclient.h"
3:
4: #using <mscorlib.dll>
5: #using "CSTaxCalc.dll"
6:
7: using namespace System;
```

Programmierung und Konfiguration des nicht verwalteten C++-Clients

Die Main-Funktion des nicht verwalteten Clients wird ähnlich programmiert wie die Main- Funktion im gestern erstellten COM-Client. Um den Code für die Main-Funktion einzufügen, folgen Sie diesen Schritten:

1. Öffnen Sie die Datei UnmgdClient.cpp.

2. Fügen Sie die #include-Direktiven aus Listing 21.9 ein.

Listing 21.9: Die Direktiven für UnmgdClient

```
1: // UnmgdClient.cpp : Defines the entry point for the
2: // console application.
3: //
4:
5: #include "stdafx.h"
6: #include <windows.h>
7: #include <stdio.h>
8: #include "atlbase.h"
9: #include "mgdclient.h"
```

3. Nehmen Sie in die Hauptfunktion den Code aus Listing 21.10 auf.

Listing 21.10: Die Hauptfunktion

```
 1: int _tmain(int argc, _TCHAR* argv[])
 2: {
 3:   MgdClient* pGetTax = new MgdClient();
 4:
 5:   double dbPurchaseAmt, dbTaxAmt, dbTotalDue;
 6:
 7:   // Den String für die gekaufte Kategorie erzeugen
 8:   wchar_t pstrCategory[20];
 9:
10:   wcscpy(pstrCategory, L"Kleidung");
11:
12:   // Kaufpreis angeben
13:   dbPurchaseAmt = 45.0;
14:   // Zu berechnende Steuer abfragen
15:   dbTaxAmt = pGetTax->GetTaxes(dbPurchaseAmt, pstrCategory);
16:   if (dbTaxAmt > 0.0)
17:   {
18:     // Summe berechnen
19:     dbTotalDue = dbPurchaseAmt + dbTaxAmt;
20:     // Gekaufte Kategorie anzeigen
21:     USES_CONVERSION;
22:     printf("%s\n", W2A(pstrCategory));
23:     // Kaufpreis anzeigen
24:     printf("Kaufpreis = %9.2f\n", dbPurchaseAmt);
25:     // Zu zahlende Steuer anzeigen
26:     printf("   Steuer = %9.2f\n", dbTaxAmt);
27:     // Summe anzeigen
28:     printf("    Summe = %9.2f\n", dbTotalDue);
29:   }
30:   return 0;
31: }
```

4. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektknoten der Datei MgdClient.cpp und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Eigenschaften aus dem Kontextmenü.
5. Wählen Sie Alle Konfigurationen aus dem Kombinationsfeld Konfiguration.
6. Wählen Sie C/C++, Allgemein auf der linken Seite des Dialogfelds Eigenschaften. Setzen Sie den Eintrag Als Verwaltet kompilieren auf der rechten Seite auf Assemblyunterstützung (/clr). Geben Sie $(outdir) beim Eintrag #using-Verweise auflösen ein.
7. Klicken Sie auf die Schaltfläche Übernehmen, um die Konfigurationsänderungen zu übernehmen, ohne den Eigenschaftendialog zu schließen.
8. Markieren Sie in der Projektmappen-Explorer-Ansicht die Datei stdafx.cpp.
9. Wählen Sie C/C++, Vorkompilierte Header auf der linken Seite des Dialogfelds Eigenschaftsseiten. Setzen Sie auf der rechten Seite den Eintrag Erstellen/ Verwenden eines vorkompilierten Headers auf Vorkompilierte Header nicht verwenden.
10. Klicken Sie auf die Schaltfläche Übernehmen, um die Konfigurationsänderungen zu übernehmen, ohne den Eigenschaftendialog zu schließen.
11. Markieren Sie in der Projektmappen-Explorer-Ansicht den Projektknoten UnmgdClient.

12. Wählen Sie Buildereignisse, Ereignis vor dem Erstellen auf der linken Seite des Dialogfelds Eigenschaftsseiten. Geben Sie auf der rechten Seite beim Eintrag Befehlszeile Folgendes ein, wenn Sie die C#-Komponente verwenden:

```
copy "..\CSTaxCalc\bin\$(ConfigurationName)\CSTaxCalc.dll"
     $(ConfigurationName)
```

Wenn Sie die VB-Komponente verwenden, geben Sie dies ein:

```
copy "..\VBTaxCalc\bin\VBTaxCalc.dll" $(ConfigurationName)
```

Ersetzen Sie wenn nötig auf Ihrem System den angegebenen relativen Pfad durch den tatsächlichen Pfad zu den Projektverzeichnissen. Da sich die Projektmappen-Datei für die beiden Komponenten in diesem Beispiel nicht im Projektverzeichnis befindet, können Sie nicht den Platzhalter $(SolutionDir) verwenden.

13. Wählen Sie C/C++, Vorkompilierte Header auf der linken Seite des Dialogfelds Eigenschaftsseiten. Auf der rechten Seite wählen Sie Vorkompilierte Header nicht verwenden im Abschnitt Erstellen/Verwenden eines vorkompilierten Headers aus.
14. Wählen Sie Active(Debug) aus dem Kombinationsfeld Konfiguration. Die Einstellungen, die Sie hier ändern, wirken sich nur auf die Debug-Builds aus.
15. Wählen Sie C/C++, Codeerstellung auf der linken Seite des Dialogfelds Eigenschaftsseiten aus. Setzen Sie auf der rechten Seite den Eintrag Vollständige Laufzeitüberprüfungen auf Standard. Setzen Sie den Eintrag Laufzeitbibliothek auf Multithreaded-Debug (/MTd). Setzen Sie den Eintrag Minimale Neuerstellung auf Nein.
16. Wählen Sie C/C++, Allgemein auf der linken Seite des Dialogfelds. Setzen Sie auf der rechten Seite den Eintrag Debuginformationsformat auf Programmdatenbank (/Zi).
17. Klicken Sie auf Ok, um die Konfigurationsänderungen zu übernehmen und den Eigenschaftendialog zu schließen.

Sie sollten die nicht verwaltete Client-Anwendung nun kompilieren und ausführen und auf die im vorherigen Beisipiel erzeugte C#- oder VB.NET-Komponente zugreifen können. Der nicht verwaltete Client erzeugt die in Abbildung 21.3 gezeigte Ausgabe.

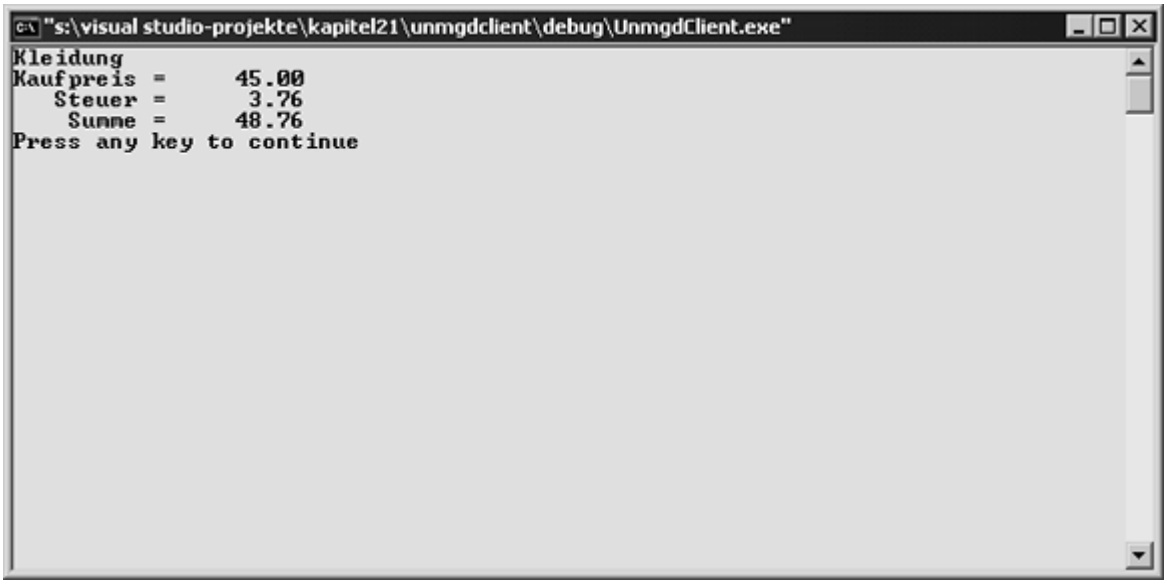


Abbildung 21.3: Der nicht verwaltete Client greift auf mit C# oder VB.NET erstellte CLR-Komponenten zu.

21.5 Zusammenfassung

Heute haben Sie gelernt, wie leicht man mit verwaltetem C++ auf in anderen Sprachen erstellte Komponenten zugreifen kann, sowohl in verwalteten als auch in nicht verwalteten C++-Anwendungen. Sie haben auch gelernt, wie man problemlos auf mit verwaltetem C++ erstellte Komponenten von anderen CLR-Sprachen wie

VB.NET oder C# aus zugreifen kann. Bei anderen Sprachen, die auf der CLR-Plattform verfügbar werden, sollte der Zugriff auf mit diesen Sprachen erzeugte Komponenten genauso einfach sein.

Sie haben in den letzten 21 Tagen eine Menge erreicht. Sie haben die erste Woche damit verbracht, die Grundlagen der Erstellung von Windows-Anwendungen mit Visual C++ zu erlernen. Sie haben während der zweiten Woche auf diesem Wissen aufgebaut und mehr über Themen der Windows-Programmierung gelernt, die sich auf das in der ersten Woche erarbeitete Wissen stützen. Zuletzt haben Sie in der dritten Woche auf allem aufgebaut, was Sie in den ersten beiden Wochen gelernt haben und sind in einige fortgeschrittene Themen der Windows-Programmierung eingetaucht. Sie haben die dritte Woche damit beendet, etwas über brandneue Themen zu lernen, die mit der neuen .NET-Plattform von Microsoft eingeführt wurden. An diesem Punkt besitzen Sie ein gutes Allgemeinwissen über die Erstellung von Anwendungen mit Visual C++. Ab hier sollten Sie sich speziellere Themen aussuchen, die Sie weiter erforschen wollen, und ein spezielles Buch wählen, das sich auf diese Themen konzentriert. Viel Glück![1]

21.6 Workshop

Fragen und Antworten

Frage:
Auf welche anderen für CLR verfügbaren Sprachen kann man mit verwaltetem C++ zugreifen?

Antwort:
Bisher erzeugt und vertreibt Microsoft nur Visual Basic, C# und verwaltetes C++. Microsoft hat aber die CLR-Spezifikationen veröffentlich, sodass andere Firmen Compiler für andere Sprachen erstellen können. Einige Firmen haben bereits ihre Unterstützung der Plattform zugesagt und Pläne angekündigt, Compiler für verschiedene andere Programmiersprachen herauszubringen. Eine der ersten Ankündigungen kam von einer Firma, die einen COBOL-Compiler für CLR herausbringt. Jedwede für CLR kompilierte Sprache sollte in der Lage sein, Objekte zu erzeugen, auf die Sie mit verwaltetem C++ zugreifen können, und Sie sollten Objekte mit verwaltetem C++ erzeugen können, auf die diese Sprachen zugreifen können.

Frage:
Warum konnte ich im nicht verwalteten C++-Client nicht den __gc-Modifizierer für die verwaltete Klasse verwenden?

Antwort:
Auf die verwaltete Klasse im nicht verwalteten Client muss vom nicht verwalteten Code in der Anwendung aus zugegriffen werden können. Wenn eine verwaltete Klasse für die Speicherbereinigung gekennzeichnet ist, kann nicht verwalteter Code nicht auf sie zugreifen. Die umgekehrte Situation haben Sie gestern gesehen, als eine verwaltete, speicherbereinigte Klasse auf eine nicht verwaltete Klasse zugriff. Scheinbar können verwaltete, speicherbereinigte Klassen aus der verwalteten Sandkiste herausgreifen, um auf nicht verwaltete Objekte zuzugreifen. Es ist nur so, dass speicherbereinigte Objekte sich auf dem CLR-verwalteten Heap befinden, auf den von außen nicht zugegriffen werden kann. Objekte auf dem verwalteten CLR-Heap können aber nicht speicherbereinigte Zeiger verwenden, um auf Objekte außerhalb zuzugreifen.

Quiz

1. Was mussten Sie tun, um auf eine verwaltete Komponente in einer VB.NET- oder C#- Anwendung zugreifen zu können?

2. Welche Konfigurationsänderungen mussten Sie vornehmen, um einer verwalteten C++-Anwendung den Zugriff auf in anderen CLR-Sprachen erstellte Objekte zu gewähren?
3. Welchen Datentyp können Sie von einer nicht verwalteten Anwendung an einen verwalteten String-Datentyp in einer verwalteten Komponente übergeben?
4. Können Sie vorkompilierte Header in einem gemischten verwalteten/unverwalteten C++-Projekt verwenden?

Übung

Fügen Sie Ihrer VB- oder C#-Komponente eine zweite Methode hinzu, um den Steuersatz für die angegebene Kategorie zurückzuliefern. Modifizieren Sie den verwalteten C++- Client so, dass er den Steuersatz abfragt

und anzeigt.

1

Sehen Sie sich doch einmal an, was der Markt+Technik-Verlag dazu zu bieten hat: www.mut.de

Woche 3 im Rückblick

Sie haben es geschafft! Nunmehr verfügen Sie über die Fertigkeiten, sich den meisten Programmieraufgaben mit Visual C++ unter Windows zu stellen, ausgenommen diejenigen, die spezielle Kenntnisse erfordern. Obwohl es noch viel zu lernen gibt, haben Sie den größten Teil der Themen in einer sehr kurzen Zeit abgehandelt. Von hier aus werden Sie sich wahrscheinlich einem oder zwei spezielleren Bereichen der Windows- Programmierung - zu denen selbst ganze Bücher erschienen sind - widmen, da Sie momentan über ein umfangreiches Allgemeinwissen verfügen.

Für den Fall, dass Sie noch nicht alle Themen beherrschen, empfiehlt es sich, dass Sie noch einmal an Ihren eigenen Anwendungen arbeiten und das Gelernte hier ausprobieren. Das hilft Ihnen, die Bereiche herauszufinden, die Sie vielleicht wiederholen sollten, bevor Sie sich komplizierteren Themen zuwenden. Sehen wir uns zur Sicherheit noch einmal kurz an, was Sie in der letzten Woche gelernt haben.

Sie haben gelernt, die Funktionalität Ihrer Module gemeinsam zu nutzen, ohne den eigentlichen Code freizulegen, indem Sie sie als ActiveX-Steuerelement verpackten. Sie haben mit dem Steuerelement-Assistenten und dem Klassen-Assistenten die Eigenschaften, Methoden und Ereignisse hinzugefügt, die für das Steuerelement erforderlich waren. Außerdem haben Sie gelernt, wie man die Eigenschaften des Steuerelements liest und schreibt. Die Eigenschaften haben Sie nach zwei unterschiedlichen Verfahren implementiert und Sie haben gelernt, wie man den jeweils passenden Typ für die Eigenschaften des Steuerelements wählt. Es wurde gezeigt, wie man Ereignisse in der Containeranwendung vom Steuerelement kommend auslöst, indem man das Ereignis im Code initiiert. Das Steuerelement haben Sie dann mit dem Testcontainer für ActiveX- Steuerelemente ausprobiert und dabei alle Methoden aufgerufen und die im Steuerelement abgefangenen Ereignisse desselben ausgelöst. Sie haben gesehen, wie man die Ereignisse überwacht, die das Steuerelement in der Containeranwendung auslöst, um zu gewährleisten, dass die Ereignisse zum richtigen Zeitpunkt ausgelöst werden. Wiederholen Sie Tag 15, falls Ihnen zu diesem Themengebiet noch etwas unklar ist.

Als Nächstes haben Sie den Webbrowser Internet Explorer von Microsoft ohne großen Aufwand in Ihre Anwendung integriert. Sie haben gelernt, wie man den Browser steuert, indem man den URL angibt, der zu laden und anzuzeigen ist. Es wurde auch dargestellt, wie man Informationen für den Benutzer sichtbar macht, was der Browser gerade ausführt und wann er beschäftigt ist. Sie haben auch gelernt, wie Visual C++ um ActiveX- Steuerelemente und COM-Objekte herum C++-Klassen erstellt, sodass Sie mit Steuerelementen und COM-Objekten wie mit jedem anderen C++-Objekt interagieren können. Sie sollten jetzt wissen, wie man auf eine beliebige COM-Objektschnittstelle in Ihrer Anwendung zugreift (bewaffnet mit der Dokumentation für das Steuerelement oder Objekt) und nahtlos mit ihm interagiert. Sie sollten in der Lage sein, eine Variable für das Steuerelement oder Objekt zu deklarieren, seine Methoden aufzurufen und auf seine Ereignisse zu reagieren, als sei es nichts weiter als ein Standardteil der Visual C++- Entwicklungsumgebung. Falls Sie Ihre Kenntnisse zu diesen Themen auffrischen möchten, gehen Sie noch einmal zurück zu Tag 16.

Die gleichzeitige Ausführung von Aufgaben bildete einen weiteren Schwerpunkt der dritten Woche. Mehr und mehr moderne Anwendungen erfordern eine derartige Funktionalität. Dabei haben Sie nicht nur gelernt, wie man mehrere Aufgaben gleichzeitig ausführt, sondern es wurden auch zwei unterschiedliche Lösungsmöglichkeiten dargestellt. Als Erstes wurde die Funktion OnIdle behandelt und gezeigt, wie man sich in diese Funktion einklinkt, um die eigene Funktionalität auszulösen, wenn die Anwendung im Leerlauf arbeitet. Dabei wurde auch auf die Mängel dieses Verfahrens eingegangen, die sich zeigen, wenn man einen zweiten Task in die Anwendung aufnimmt. Diese kann verhindern, dass die Anwendung auf den Benutzer reagieren kann. Die Hintergrundverarbeitung musste in kleinere Teile zerlegt werden, die sich schnell ausführen lassen. Dazu war eine Möglichkeit herauszuarbeiten, mit der man verfolgen kann, in welcher Phase die Task gerade arbeitet und wo sie die Arbeit aufnehmen muss, wenn die Anwendung wieder in den Leerlaufzustand gelangt.

Als zweites Verfahren, um einer Anwendung die Ausführung einer zweiten oder dritten Task zu ermöglichen, wurden separate Threads vorgestellt, die vollkommen unabhängig vom Thread der Benutzeroberfläche laufen. In diesem Zusammenhang haben Sie Callback-Funktionen kennen gelernt, die die höchste Ausführungsebene für die Threads steuern, und wie man den Thread bei Bedarf starten und anhalten kann. Sie haben auch gesehen, wie diese unabhängigen Threads vollkommen losgelöst von der übrigen

Anwendung arbeiten und wie sie weiterlaufen, selbst wenn die übrige Anwendung beschäftigt ist. Wenn Sie sich mit diesem Thema nochmals befassen möchten, gehen Sie zurück zu Tag 17.

Ein Bereich, der zunehmend Bedeutung gewinnt, ist das Erstellen von Internet- Anwendungen mit den Klassen der Winsock-Schnittstelle. Sie haben eine Anwendung erstellt, die eine Verbindung mit einer anderen Anwendung über ein Netzwerk eingeht und Nachrichten in beide Richtungen austauscht. Wie beim Telefon muss die zweite Anwendung auf ein Verbindungsgesuch der ersten Anwendung lauschen. Es wurde gezeigt, wie einfach es ist, Nachrichten zu senden und benachrichtigt zu werden, wenn die Nachricht angekommen ist, nachdem die Verbindung zwischen beiden Anwendungen hergestellt wurde. Sollten sich in diesem Zusammenhang bei Ihnen Fragen ergeben, wiederholen Sie den Stoff von Tag 18.

An den letzten drei Tagen sind Sie in die neue Welt von .NET und verwaltetem C++ eingetaucht. Am ersten dieser drei Tage haben Sie die verwalteten Erweiterungen für C++ studiert und wie sie Ihre Entwicklungen beeinflussen. Sie haben gelernt, wie einfach Sie Ihre MFC-Anwendungen zu verwalteten C++-Anwendungen migrieren können, sodass sie auf allen .NET-Clientgeräten laufen. Sie haben außerdem die Grundlagen der Erstellung einer eigenen verwalteten C++-Anwendung von Grund auf kennen gelernt. Wenn Sie eine Auffrischung benötigen, was die Lösung einer dieser Aufgaben betrifft, werfen Sie noch einen Blick auf Tag 19.

Am zweiten Tag der Themen rund um verwaltetes C++ haben Sie Erfahrung mit der Verwendung der ATL-Klassenbibliothek gesammelt, um abgespeckte COM-Objekte zu erstellen. Sie haben gesehen, wie Sie ein COM-Objekt auf eine bestimmte Art erstellen können, sodass es ein Leichtes ist, ihm eine verwaltete C++-Schnittstelle hinzuzufügen, um von anderen CLR-Sprachen wie VB.NET oder C# darauf zugreifen zu können. Wenn Sie dazu noch Fragen haben, schlagen Sie dieses Thema noch einmal an Tag 20 nach.

Zuletzt haben Sie, um die Woche zu beenden, die Erstellung verwalteter C++-Objekte erlernt, die in mit anderen Sprachen erstellten Anwendungen verwendet werden können. Sie haben gesehen, wie Sie umgekehrt in anderen CLR-Sprachen erstellte Objekte in Ihren verwalteten C++-Projekten verwenden können. Außerdem haben Sie gelernt, wie Sie von nicht verwalteten C++-Anwendungen aus mithilfe eines verwalteten Wrappers auf CLR-Objekte zugreifen und sie verwenden können. Wenn hier noch Fragen bleiben, sollten Sie Tag 21 noch einmal lesen.

Das war's. Sie haben es geschafft. Sie haben viel erreicht und Fähigkeiten und Wissen zu einigen komplizierteren Themen - insbesondere in der letzten Woche - gewonnen. Jetzt ist es an der Zeit, das Buch beiseite zu legen und aktiv in die Programmierung einzusteigen. Erstellen Sie auf der Basis des Gelernten eigene Anwendungen. Viel Glück! Wenn Sie Hilfe oder einen Rat brauchen, sollten Sie sich in den Newsgroups von Microsoft im Internet umsehen. Hier gibt es viele Gleichgesinnte, die sowohl kenntnisreich als auch hilfreich sind.

Anhang A

Workshop-Auflösung

Tag 1

Quiz-Antworten

1. Markieren Sie im Fenster-Layout-Editor die zu ändernde Schaltfläche. Ändern Sie den Wert im Feld Titel im Eigenschaftenfenster.

2. Mit dem Anwendungs-Assistenten erstellen Sie das Gerüst Ihrer Anwendung basierend auf dem Typ der Anwendung und der geplanten Funktionalität. Das Gerüst unterstützt dann die in den Dialogfeldern des Assistenten spezifizierte Funktionalität.
3. Mit dem Modus Steuerelementereignisse im Eigenschaftenfenster erzeugen Sie eine Funktion und verbinden sie mit einem Objekt, das eine bestimmte Windows- Nachricht behandelt. Visual C++ erzeugt die Funktion, danach können Sie mithilfe der Klassenansicht des Arbeitsbereichs an die richtige Stelle im Quellcode zu gehen, um den eigenen Code einzugeben.

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte Ressourcen.

2. Erweitern Sie den Zweig Dialog und doppelklicken Sie auf das Dialogfeld IDD_ABOUTBOX. Dieses Dialogfeld wird daraufhin im Editor von Visual C++ zur Bearbeitung angezeigt.
3. Klicken Sie in der Toolbox auf das Symbol für Schaltfläche.
4. Klicken Sie an der Stelle im Dialogfeld, wo Sie die Schaltfläche platzieren möchten.
5. Ändern Sie im Eigenschaftenfenster die ID und den Titel, um den Text auf der Schaltfläche festzulegen.
6. Wechseln Sie in den Modus Steuerelementereignisse des Eigenschaftenfensters und fügen Sie eine neue Funktion für die Klicknachricht (BN_CLICKED) für Ihre neue Schaltfläche hinzu.
7. Fügen Sie die Funktion MessageBox hinzu, um die Nachricht für den Benutzer auszugeben.
8. Kompilieren und starten Sie die Anwendung, um die neue Schaltfläche zu testen.

Tag 2

Quiz-Antworten

1. Die drei wichtigsten Navigationsbefehle bei der Fehlerbeseitigung sind Einzelschritt, Prozedurschritt und Ausführen bis Rücksprung.

2. Das Makro ASSERT entfernt bei der Erstellung einer Release-Build den ihm als Parameter übergebenen Bedingungscode, das Makro VERIFY belässt den Bedingungscode in der Anwendung.
3. Das Werkzeug Spy++ verwenden Sie, um die an ein bestimmtes Fenster oder eine Anwendung übergebenen Ereignisnachrichten zu sehen. So können Sie analysieren, welche Ereignisse ausgelöst werden und welche nicht. Diese Informationen sind unabdingbar, um herauszufinden, warum manche Ereignisse nicht erwartungsgemäß ausgelöst werden.

Lösung zur Übung

Listing 2.A: DEBUGGINGDLG.CPP - die Funktion CalulateSum

```
 1: int CDebuggingDlg::CalculateSum(int iLeftValue, int iRightValue)
 2: {
 3:   ASSERT(iLeftValue >= 0);
 4:   ASSERT(iRightValue >= 0);
 5:   int iSum;
 6:
 7:   // Parameter addieren
 8:   iSum = iLeftValue - iRightValue;
 9:   ASSERT(iSum >= 0);
10:   // Ergebnis zurückgeben
11:   return iSum;
12: }
```

Tag 3

Quiz-Antworten

1. Indem Sie die Tabulator-Reihenfolge der Steuerelemente in Ihrem Anwendungsfenster festlegen, können Sie bestimmen, auf welchem Weg der Benutzer durch das Anwendungsfenster navigiert. Wenn der Benutzer mit der Tastatur arbeitet, kann er mit der (ÿ)-Taste von einem Steuerelement zum nächsten wechseln und über die Zugriffstasten ein bestimmtes Steuerelement direkt anspringen. Die Tabulator-Reihenfolge bietet dem Benutzer ein einheitliches und vorhersagbares Verhalten, wenn er im Anwendungsfenster navigiert.

2. Wenn man eine Zugriffstaste für ein Textfeld vorsieht und dann sicherstellt, dass sich das Textfeld unmittelbar vor dem zugeordneten Eingabefeld befindet, kann der Benutzer die Zugriffstaste des Textfelds wählen, um direkt zum Eingabefeld zu gelangen.
3. Die eindeutigen Objekt-IDs für die beiden statischen Textfelder sind erforderlich, weil diese beiden Steuerelemente in Abhängigkeit vom Zustand der Kontrollkästchen, die bestimmte Gruppen von Steuerelementen aktivieren oder deaktivieren, manipuliert werden sollen.
4. Wenn der Benutzer den Wert des Steuerelements auf dem Bildschirm ändert, ist die Funktion UpdateData mit dem Argument TRUE aufzurufen, um die Werte aus den Steuerelementen im Fenster in die Variablen, die mit diesen Steuerelementen verbunden sind, zu übertragen. Ruft man UpdateData nicht auf, spiegeln die Variablen nicht die aktuellen Änderungen wider, die der Benutzer in den Steuerelementen auf dem Bildschirm vorgenommen hat.

Lösungen zu den Übungen

1. Fügen Sie im Eigenschaftenfenster eine Funktion für die Klicknachricht der Schaltfläche Standardnachricht hinzu. In die Funktion schreiben Sie den Code gemäß Listing 3.A.

Listing 3.A: Eine Standardnachricht in das Eingabefeld setzen

```
 1: void CControlsDlg::OnBnClickedDfltmsg()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Nachricht auf eine Standardnachricht setzen
 6:   m_strMessage = "Hier eine Nachricht eingeben";
 7:
 8:   // Bildschirm aktualisieren
 9:   UpdateData(FALSE);
10: }
```

2. Fügen Sie Funktionen für die Kontrollkästchen Programmaktion aktivieren und Programmaktion zeigen hinzu. In diese Funktionen übernehmen Sie den Code aus Listing 3.B.

Listing 3.B: Die Steuerelemente für die Programmausführung aktiveren oder deaktivieren bzw. zeigen oder verbergen

```
 1: void CControlsDlg::OnBnClickedCkenblpgm()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Aktuelle Werte vom Bildschirm holen
 6:   UpdateData(TRUE);
 7:
 8:   // Ist "Programmaktion aktivieren" aktiviert?
 9:   if (m_bEnablePgm == TRUE)
10:   {
11:     // Ja, alle Steuerelemente aktivieren, die für die
12:     // Programmausführung relevant sind
13:     GetDlgItem(IDC_PROGTORUN)->EnableWindow(TRUE);
14:     GetDlgItem(IDC_RUNPGM)->EnableWindow(TRUE);
15:     GetDlgItem(IDC_STATICPGM)->EnableWindow(TRUE);
16:   }
17:   else
18:   {
19:     // Nein, alle Steuerelemente deaktivieren, die für die
20:     // Programmausführung relevant sind
21:     GetDlgItem(IDC_PROGTORUN)->EnableWindow(FALSE);
22:     GetDlgItem(IDC_RUNPGM)->EnableWindow(FALSE);
23:     GetDlgItem(IDC_STATICPGM)->EnableWindow(FALSE);
24:   }
25: }
26:
27: void CControlsDlg::OnBnClickedCkshwpgm()
28: {
29:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
30:   // Benachrichtigung ein.
31:   // Aktuelle Werte vom Bildschirm holen
32:   UpdateData(TRUE);
33:
34:   // Ist "Programmaktion zeigen" aktiviert?
35:   if (m_bShowPgm == TRUE)
36:   {
37:     // Ja, alle Steuerelemente zeigen, die für die
38:     // Programmausführung relevant sind
39:     GetDlgItem(IDC_PROGTORUN)->ShowWindow(TRUE);
40:     GetDlgItem(IDC_RUNPGM)->ShowWindow(TRUE);
41:     GetDlgItem(IDC_STATICPGM)->ShowWindow(TRUE);
42:   }
43:   else
44:   {
45:     // Nein, alle Steuerelemente ausblenden, die für die
46:     // Programmausführung relevant sind
47:     GetDlgItem(IDC_PROGTORUN)->ShowWindow(FALSE);
48:     GetDlgItem(IDC_RUNPGM)->ShowWindow(FALSE);
49:     GetDlgItem(IDC_STATICPGM)->ShowWindow(FALSE);
50:   }
51: }
```

3. Modifizieren Sie die Funktion OnBnClickedRunpgm gemäß Listing 3.C.

Listing 3.C: Der Code, mit dem jedes in das Kombinationsfeld Programm starten eingegebene Programm ausgeführt werden kann

```
 1: void CControlsDlg::OnBnClickedRunpgm()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Aktuelle Werte vom Bildschirm holen
 6:   UpdateData(TRUE);
 7:
```

```
 8:    // Lokale Variable zur Aufnahme des Programmnamens deklarieren
 9:    CString strPgmName;
10:
11:    // Programmname in die lokale Variable kopieren
12:    strPgmName=m_strProgToRun;
13:
14:    // Programmname in Großbuchstaben umwandeln
15:    strPgmName.MakeUpper();
16:
17:    // Programm Paint gewählt?
18:    if(strPgmName=="PAINT")
19:      // Ja, Paint starten
20:      WinExec("mspaint.exe",SW_SHOW);
21:
22:    // Programm Editor (Notepad) gewählt?
23:    if(strPgmName=="EDITOR")
24:      // Ja, Editor starten
25:      WinExec("notepad.exe",SW_SHOW);
26:
27:    // Programm Solitär gewählt?
28:    if(strPgmName=="SOLITÄR")
29:      // Ja, Solitär starten
30:      WinExec("sol.exe",SW_SHOW);
31:
32:    // Anderer Programmname ins Kombinationsfeld eingegeben
33:    if ((strPgmName != "PAINT") && (strPgmName != "NOTEPAD") &&
34:        (strPgmName != "SOLITAIRE"))
35:      // Ja, eingegebenes Programm ausführen
36:      WinExec(strPgmName, SW_SHOW);
37: }
```

Tag 4

Quiz-Antworten

1. WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP, WM_RBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MBUTTONDBLCLK, WM_XBUTTONDOWN, WM_XBUTTONUP, WM_XBUTTONDBLCLK, WM_MOUSEMOVE und WM_MOUSEWHEEL.

2. Man kann die Flags, die an die Funktion OnMouseMove übergeben werden, mit dem Flag MK_LBUTTON wie folgt maskieren:

((nFlags & MK_LBUTTON) == MK_LBUTTON)

3. Geben Sie in der Behandlungsroutine OnSetCursor den Wert TRUE zurück. Damit verhindern Sie, dass die Funktion OnSetCursor der Basisklasse aufgerufen wird.

Lösungen zu den Übungen

1. Fügen Sie eine Funktion für die Nachricht WM_RBUTTONDOWN hinzu und schreiben Sie in die Funktion folgenden Code:

```
1: void CMouseDlg::OnRButtonDown(UINT nFlags, CPoint point)
2: {
3:    // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
4:    // und/oder benutzen Sie den Standard.
5:
6:    // Aktuellen Punkt als Anfangspunkt setzen
7:    m_iPrevX = point.x;
8:    m_iPrevY = point.y;
```

```
 9:
10:   CDialog::OnRButtonDown(nFlags, point);
11: }
```

Erweitern Sie die Funktion OnMouseMove wie folgt:

```
 1: void CMouseDlg::OnMouseMove(UINT nFlags, CPoint point)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 4:   // und/oder benutzen Sie den Standard.
 5:
 6:   // Linke oder rechte Maustaste gedrückt?
 7:   if (((nFlags & MK_LBUTTON) == MK_LBUTTON) ||
 8:       ((nFlags & MK_RBUTTON) == MK_RBUTTON))
 9:   {
10:     // Gerätekontext holen
11:     CClientDC dc(this);
12:     CPen* pPrevPen = NULL; // Zeiger auf den vorherigen Stift
13:
14:     if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
15:       // Neuen Stift erzeugen
16:       CPen lpen(PS_SOLID, 16, RGB(255, 0, 0));
17:
18:     if ((nFlags & MK_RBUTTON) == MK_RBUTTON)
19:       // Neuen Stift erzeugen
20:       CPen lpen(PS_SOLID, 16, RGB(0, 0, 255));
21:
22:     // Den neuen Stift verwenden
23:     pPrevPen = dc.SelectObject(&lpen);
24:
25:     // Linie vom letzten zum aktuellen Punkt zeichnen
26:     dc.MoveTo(m_iPrevX, m_iPrevY);
27:     dc.LineTo(point.x, point.y);
28:
29:     // Aktuellen Punkt als letzten Punkt speichern
30:     m_iPrevX = point.x;
31:     m_iPrevY = point.y;
32:
33:     // Vorherigen Stift wieder herstellen
34:     dc.SelectObject(pPrevPen);
35:   }
36:
37:   CDialog::OnMouseMove(nFlags, point);
38: }
```

2. Die modifizierte Funktion OnKeyDown kann etwa folgendermaßen aussehen:

```
 1: void CMouseDlg::OnKeyDown(UINT nChar, UINT nRepCnt,
 2:                           UINT nFlags)
 3: {
 4:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein,
 5:   // und/oder benutzen Sie den Standard.
 6:
 7:   char cChar;               // Zeichen der gedrückten Taste
 8:   HCURSOR hCursor = 0;      // Handle zum anzuzeigenden Cursor
 9:   HCURSOR hPrevCursor = 0;  // Handle zum letzten Cursor
10:
11:   // Code der gedrückten Taste in Zeichen umwandeln
12:   cChar = char(nChar);
13:
14:   // Ist Zeichen ein "A"?
15:   if (cChar == 'A')
16:     // Pfeilcursor laden
```

```
17:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
18:
19:   // Ist Zeichen ein "B"?
20:   if (cChar == 'B')
21:     // Balkencursor laden
22:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
23:
24:   // Ist Zeichen ein "C"?
25:   if (cChar == 'C')
26:     // Sanduhrcursor laden
27:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
28:
29:   // Ist Zeichen ein "D"?
30:   if (cChar == 'D')
31:     // Fadenkreuz-Cursor laden
32:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_CROSS);
33:
34:   // Ist Zeichen ein "E"?
35:   if (cChar == 'E')
36:     // Cursor "Pfeil nach oben" laden
37:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_UPARROW);
38:
39:   // Ist Zeichen ein "F"?
40:   if (cChar == 'F')
41:     // Cursor "Größe ändern" laden
42:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZEALL);
43:
44:   // Ist Zeichen ein "G"?
45:   if (cChar == 'G')
46:     // Cursor "Größe ändern Nordwest/Südost" laden
47:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENWSE);
48:
49:   // Ist Zeichen ein "H"?
50:   if (cChar == 'H')
51:     // Cursor "Größe ändern Nordost/Südwest" laden
52:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENESW);
53:
54:   // Ist Zeichen ein "I"?
55:   if (cChar == 'I')
56:     // Cursor "Größe ändern West/Ost" laden
57:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZEWE);
58:
59:   // Ist Zeichen ein "J"?
60:   if (cChar == 'J')
61:     // Cursor "Größe ändern Nord/Süd" laden
62:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENS);
63:
64:   // Ist Zeichen ein "K"?
65:   if (cChar == 'K')
66:     // Cursor "Kein" laden
67:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_NO);
68:
69:   // Ist Zeichen ein "L"?
70:   if (cChar == 'L')
71:     // Cursor "Anwendung wird gestartet" laden
72:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_APPSTARTING);
73:
74:   // Ist Zeichen ein "M"?
75:   if (cChar == 'M')
76:     // Cursor "Direkthilfe" laden
77:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_HELP);
78:
79:   // Ist Zeichen ein "X"?
80:   if (cChar == 'X')
```

```
 81:     // Pfeilcursor laden
 82:     hCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
 83:   // Cursorflag setzen
 84:   m_bCursor = TRUE;
 85:   // Bildschirmcursor setzen
 86:   if (hCursor)
 87:     hPrevCursor = SetCursor(hCursor);
 88:
 89:   // Letzten Cursor zerstören, um Ressourcen freizugeben
 90:   if (hPrevCursor)
 91:     DestroyCursor(hPrevCursor);
 92:
 93:   // Ist Zeichen ein "X"?
 94:   if (cChar == 'X')
 95:   {
 96:     // Exit the application
 97:     OnOK();
 98:   }
 99:
100:   CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
101: }
```

Tag 5

Quiz-Antworten

1. Durch die Definition der beiden IDs sind diese als Konstanten durch die gesamte Anwendung hindurch verfügbar.

2. Fügen Sie sie als #define-Konstanten in die Header-Datei der Klasse (TimerDlg.h) oder in die Quellcode-Datei (TimerDlg.cpp) wie folgt ein:

```
/////////////////////////////////////////////////////////////////////
// CTimersDlg dialog
#define ID_CLOCK_TIMER 1
#define ID_COUNT_TIMER 2
class CTimersDlg : public CDialog
{
...
```

3. Anhand der Timer-ID lässt sich bestimmen, welcher Timer das Ereignis ausgelöst hat.
4. Genau ein Timer-Ereignis

Lösung zur Übung

Um das Intervall zu ändern, mit dem ein Timer läuft, müssen Sie zuerst den Timer stoppen und dann erneut starten, wie es aus Listing 5.A hervorgeht.

Listing 5.A: Die überarbeiteten Funktionen OnBnClickedBstarttime und OnBnClickedBstoptimer

```
 1: void CTimersDlg::OnBnClickedBstarttime()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   UpdateData(TRUE);
 6:
 7:   // Zähler initialisieren
 8:   m_iCount = 0;
 9:   // Zähler für Anzeige formatieren
10:   m_sCount.Format("%d", m_iCount);
11:
```

```
12:    // Dialogfeld aktualisieren
13:    UpdateData(FALSE);
14:    // Timer starten
15:    SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
16:
17:    // Uhren-Timer anhalten
18:    KillTimer(ID_CLOCK_TIMER);
19:    // Uhren-Timer mit dem Zähler-Intervall neu starten
20:    SetTimer(ID_CLOCK_TIMER, m_iInterval, NULL);
21:
22:    // Schaltfläche Timer anhalten aktivieren
23:    m_cStopTime.EnableWindow(TRUE);
24:    // Schaltfläche Timer starten deaktivieren
25:    m_cStartTime.EnableWindow(FALSE);
26: }
27:
28: void CTimersDlg::OnBnClickedBstoptimer()
29: {
30:    // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die 31: Benachric
32:    // Timer anhalten
33:    KillTimer(ID_COUNT_TIMER);
34:
35:    // Uhren-Timer anhalten
36:    KillTimer(ID_CLOCK_TIMER);
37:    // Uhren-Timer mit dem Intervall 1 Sekunde neu starten
38:    SetTimer(ID_CLOCK_TIMER, 1000, NULL);
39:
40:    // Schaltfläche Timer anhalten deaktivieren
41:    m_cStopTime.EnableWindow(FALSE);
42:    // Schaltfläche Timer starten aktivieren
43:    m_cStartTime.EnableWindow(TRUE);
44: }
```

Tag 6

Quiz-Antworten

1. IDRETRY und IDCANCEL

2. Als MFC-Klassen sind die folgenden allgemeinen Windows-Dialogfelder definiert:

   - Dateiauswahl
   - Schriftauswahl
   - Farbauswahl
   - Seite zum Drucken einrichten
   - Drucken
   - Suchen und Ersetzen

3. Ein modales Dialogfeld stoppt alle Programmabläufe in der Anwendung, bis der Benutzer auf das Dialogfeld antwortet. Ein nicht modales Dialogfeld gestattet dem Benutzer, mit der übrigen Anwendung weiterzuarbeiten, während das Dialogfeld geöffnet bleibt und Eingaben entgegennehmen kann.
4. In der Variablendeklaration der Klasseninstanz übergeben Sie FALSE statt TRUE. Damit sieht die Variablendeklaration folgendermaßen aus:

   ```
   CFileDialog m_ldFile(FALSE);
   ```

5. Im benutzerdefinierten Dialogfeld ist lediglich die Funktion UpdateData aufzurufen, bevor das Dialogfeld geschlossen wird. Da Sie die Schaltflächen OK und Abbrechen nicht aus dem Dialogfeld gelöscht haben, führt die Schaltfläche OK automatisch diese Funktion aus.

Lösungen zu den Übungen

1. Modifizieren Sie die Funktion OnBnClickedBfileopen wie folgt:

```
 1: void CDialogsDlg::OnBnClickedBfileopen()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   CFileDialog ldFile(TRUE);
 6:
 7:   // Dialogfeld Öffnen zeigen und Ergebnis auffangen
 8:   if (ldFile.DoModal() == IDOK)
 9:   {
10:     // Gewählten Dateinamen ermitteln
11:     m_sResults = ldFile.GetPathName();
12:     // Dialogfeld aktualisieren
13:     UpdateData(FALSE);
14:   }
15: }
```

Die Funktion GetPathName gibt den Pfad und Dateinamen zurück, sodass bei Änderung des Funktionsaufrufs von GetFileName in GetPathName der Pfad zusammen mit dem Dateinamen in der Anzeige erscheint.

2. Führen Sie die folgenden Schritte aus:
1. Fügen Sie über die Klassenansicht eine Member-Variable in die Klasse CMsgDlg ein. Legen Sie den Variablentyp als int, den Namen mit m_iYesNo und den Zugriff als public fest.
2. Verwenden Sie die Ressourcenansicht, um das benutzerdefinierte Dialogfeld im Editorbereich zu öffnen. Fügen Sie eine Schaltfläche zum Fenster hinzu und tragen Sie IDC_BYESNO als ID und J&a oder Nein als Beschriftung ein.
3. Fügen Sie über das Eigenschaftenfenster eine Funktion für die neue Schaltfläche hinzu und schreiben Sie in diese Funktion den folgenden Code:

```
1: void CMsgDlg::OnBnClickedByesno()
2: {
3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die 4: Benach
5:   // Benutzer fragen
6:   m_iYesNo = MessageBox("Wählen Sie Ja oder Nein",
7:                         "Ja oder Nein", MB_YESNO);
8: }
```

4. Nehmen Sie in das Hauptdialogfeld eine Schaltfläche mit der ID IDC_BYESNO und der Beschriftung Ja oder &Nein auf.
5. Fügen Sie über das Eigenschaftenfenster eine Funktion für die neue Schaltfläche hinzu und schreiben Sie den folgenden Code in diese Funktion:

```
 1: void CDialogsDlg::OnBnClickedByesno()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Wie hat der Benutzer geantwortet?
 6:   switch (m_dMsgDlg.m_iYesNo)
 7:   {
 8:     case IDYES:    // Antwort JA?
 9:       m_strResults = "Ja!";
10:       break;
11:     case IDNO:     // Antwort NEIN?
12:       m_strResults = "Nein!";
13:       break;
14:   }
15:
16:   // Dialogfeld aktualisieren
17:   UpdateData(FALSE);
18: }
```

Tag 7

Quiz-Antworten

1. COMMAND (eigentlich WM_COMMAND, im Ereignis-Handler-Assistenten jedoch als COMMAND bezeichnet)

2. Öffnen Sie im Dialog-Designer den Eigenschaftsdialog für das Fenster und wählen Sie das Menü aus der Dropdown-Liste der Menüs aus. Alternativ können Sie die Funktion SetMenu verwenden, um das Menü in das Dialogfeld einzufügen.
3. Die Dialogklasse für das Fenster, in dem das Menü erscheint
4. Durch das Ereignis WM_CONTEXTMENU

Lösungen zu den Übungen

1. Führen Sie die folgenden Schritte aus:

1. Nehmen Sie eine Schaltfläche in das Dialogfeld auf. Tragen Sie für diese Schaltfläche die ID IDC_HELLO und den Titel &Hello ein.
2. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Schaltfläche hinzu. Verwenden Sie den vorgegebenen Namen für die Funktion, OnBnClickedHello.
2. Führen Sie die folgenden Schritte aus:
1. Bearbeiten Sie die Funktion OnContextMenu, indem Sie den hervorgehobenen Code ändern:

```
 1: void CMenusDlg::OnContextMenu(CWnd* pWnd, CPoint point)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
 4:   CMenu *pMenu;
 5:
 6:   // Zeiger auf Menü abfragen
 7:   pMenu = GetMenu();
 8:   // Zeiger auf Untermenü abfragen
 9:   pMenu = pMenu->GetSubMenu(1);
10:   // Als Kontextmenü öffnen
11:   pMenu->TrackPopupMenu(TPM_CENTERALIGN | TPM_LEFTBUTTON,
12:                         point.x, point.y, pWnd, NULL);
13: }
```

Tag 8

Quiz-Antworten

1. Übergeben Sie der Funktion CreateFont als Argument bUnderline den Wert 1.

2. Übergeben Sie der Funktion CreateFont als Argument nEscapement den Wert 1800.
3. Diese Funktion wird für jede im System verfügbare Schrift je einmal aufgerufen, außer wenn die Callback-Funktion den Wert 0 zurückgibt und die Auflistung der Schriften stoppt.

Lösungen zu den Übungen

1. Nehmen Sie das Kontrollkästchen in das Dialogfeld auf. Legen Sie seine Eigenschaften wie folgt fest:

Eigenschaft	Einstellung
ID	IDC_CBUSETEXT
Beschriftung	Text &verwenden

Weisen Sie dem Steuerelement eine Variable zu. Legen Sie den Variablentyp als BOOL und den Namen mit m_bUseText fest.

Fügen Sie eine Funktion für die Nachricht BN_CLICKED des Kontrollkästchens hinzu. In die Funktion schreiben Sie den folgenden Code:

```
 1: void CTextFontsDlg::OnBnClickedCbusetext(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen mit den Steuerelementen des Dialogfelds
 6:   // aktualisieren
 7:   UpdateData(TRUE);
 8:   // Schriftname für Schriftprobe verwenden?
 9:   if (!m_bUseText)
10:     // Schriftname verwenden
11:     m_strDisplayText = m_strFontName;
12:   else
13:     // Eingegebenen Text verwenden
14:     m_strDisplayText = m_strSampText;
15:
16:   // Dialogfeld aktualisieren
17:   UpdateData(FALSE);
18: }
```

Um das Kontrollkästchen zu initialisieren, modifizieren Sie OnInitDialog wie folgt:

Listing 8.A: Die Funktion OnInitDialog

```
 1: BOOL CTextFontsDlg::OnInitDialog()
 2: {
 3:   CDialog::OnInitDialog();
 4: ...
 5:   // TODO: Hier zusätzliche Initialisierung einfügen
 6:   // Listenfeld für Schriften füllen
 7:   FillFontList();
 8:
 9:   // Einzugebenden Text initialisieren
10:   m_strSampText = "Testen";
11:   // Text in Feld für Schriftbeispiel kopieren
12:   m_strDisplayText = m_strSampText;
13:   // Kontrollkästchen initialisieren
14:   m_bUseText = TRUE;
15:   // Dialogfeld aktualisieren
16:   UpdateData(FALSE);
17:
18:   return TRUE;  // Geben Sie TRUE zurück, außer ein
19:                 // Steuerelement soll den Fokus erhalten
20: }
```

Modifizieren Sie die Funktion OnSelchangeLfonts wie folgt:

Listing 8.B: Die Funktion OnSelchangeLfonts

```
 1: void CTextFontsDlg::OnLbnSelchangeLfonts(void)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Variablen mit den Steuerelementen des Dialogfelds
 6:   // aktualisieren
 7:   UpdateData(TRUE);
 8:   // Schriftname für Schriftprobe verwenden?
```

```
 9:   if (!m_bUseText)
10:   {
11:     // Schriftname in Feld für Schriftbeispiel kopieren
12:     m_strDisplayText = m_strFontName;
13:     // Dialogfeld mit Variablen aktualisieren
14:     UpdateData(FALSE);
15:   }
16:   // Set the font for the sample
17:   SetMyFont();
18: }
```

Schließlich ändern Sie noch die Funktion OnChangeEsamptext folgendermaßen ab:

Listing 8.C: Die Funktion OnChangeEsamptext

```
 1: void CTextFontsDlg::OnEnChangeEsamptext(void)
 2: {
 3:   // TODO:  If this is a RICHEDIT control, the control will not
 4:   // send this notification unless you override the
 5:   // CDialog::OnInitDialog()
 6:   // function and call CRichEditCtrl().SetEventMask()
 7:   // with the ENM_CHANGE flag ORed into the mask.
 8:
 9:   // TODO:  Add your control notification handler code here
10:   // Update the variables with the dialog controls
11:   UpdateData(TRUE);
12:   // Text für Schriftprobe verwenden?
13:   if (m_bUseText)
14:   {
15:     // Aktuellen Text in Schriftbeispiel kopieren
16:     m_strDisplayText = m_strSampText;
17:     // Dialogfeld mit Variablen aktualisieren
18:     UpdateData(FALSE);
19:   }
20: }
```

2. Nehmen Sie ein Kontrollkästchen in das Dialogfeld auf. Legen Sie seine Eigenschaften wie folgt fest:

Eigenschaft	Einstellung
ID	IOC_CBITALIC
Beschriftung	&Kursiv

Weisen Sie dem Steuerelement eine Variable zu. Legen Sie den Variablentyp als BOOL und den Namen mit m_bItalic fest.

Fügen Sie eine Funktion für die Nachricht BN_CLICKED des Kontrollkästchens hinzu. In die Funktion schreiben Sie den folgenden Code:

Listing 8.D: Die Funktion OnBnClickedCbitalic

```
 1: void CTextFontsDlg::OnBnClickedCbitalic()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4: Benachrichtigung ein.
 5:   // Variablen mit den Steuerelementen des Dialogfelds
 6:   // aktualisieren
 7:   UpdateData(TRUE);
 8:
 9:   // Schrift für Beispiel festlegen
```

```
10:   SetMyFont();
11: }
```

Ändern Sie die Funktion SetMyFont entsprechend dem nachstehenden Listing ab:

Listing 8.E: Die Funktion SetMyFont

```
 1: void CTextFontsDlg::SetMyFont()
 2: {
 3:   CRect rRect;        // Rechteck des Anzeigebereichs
 4:   int iItalic = 0;    // Schrift kursiv ausgeben?
 5:
 6:   // Wurde eine Schrift ausgewählt?
 7:   if (m_strFontName != "")
 8:   {
 9:     // Abmessungen des Feldes für Schriftbeispiel ermitteln
10:     m_ctlDisplayText.GetWindowRect(&rRect);
11:     if (m_bItalic)
12:       iItalic = 1;
13:     // Aktuelle Schrift freigeben
14:     m_fSampFont.Detach();
15:     // Zu verwendende Schrift erzeugen
16:     m_fSampFont.CreateFont((rRect.Height() - 5), 0, 0, 0,
17:             FW_NORMAL, iItalic, 0, 0, DEFAULT_CHARSET,
18:             OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS,
19:             DEFAULT_QUALITY, DEFAULT_PITCH |
20:             FF_DONTCARE, m_strFontName);
21:
22:     // Schrift für Anzeigefeld der Schriftprobe festlegen
23:     m_ctlDisplayText.SetFont(&m_fSampFont);
24:   }
25: }
```

Tag 9

Quiz-Antworten

1. Rot, Grün und Blau

2. Mit dem Gerätekontext
3. 8 mal 8 Pixel oder größer
4. Die Nachricht WM_PAINT
5. Rufen Sie die Funktion Invalidate auf diesem Fenster auf.

Lösungen zu den Übungen

1. Öffnen Sie das zweite Dialogfeld im Dialog-Designer. Setzen Sie die Eigenschaft Rahmen auf Grösse ändern. Fügen Sie eine Behandlungsroutine für die Nachricht WM_SIZE hinzu. In der eben erzeugten Funktion rufen Sie die Funktion Invalidate auf, wie es Listing 9.A zeigt.

Listing 9.A: Die Funktion OnSize

```
1: void CPaintDlg::OnSize(UINT nType, int cx, int cy)
2: {
3:   CDialog::OnSize(nType, cx, cy);
4:
5:   // TODO: Fügen Sie hier Ihren Meldungsbehandlungscode ein.
6:   // Fenster neu zeichnen
7:   Invalidate();
8: }
```

2. Öffnen Sie die Ressourcenansicht. Klicken Sie mit der rechten Maustaste auf den obersten Ordner des Ressourcenbaumes. Aus dem Kontextmenü wählen Sie den Befehl Hinzufügen / Ressource hinzufügen. Im Dialogfeld Ressource hinzufügen markieren Sie in der Liste Ressourcentyp den Eintrag Bitmap. Zeichnen Sie ein Muster in die gerade erstellte Bitmap. Klicken Sie auf die Bitmap-ID im Arbeitsbereich. Ändern Sie im Eigenschaftenfenster die Objekt-ID in IDB_BITMAPBRUSH. Öffnen Sie den Quellcode für die Funktion DrawRegion. Fügen Sie in diese Funktion den fett gedruckten Code von Listing 9.B ein. Vergrößern Sie die Anzahl der Schleifen in der for-Anweisung.

Listing 9.B: Die Funktion DrawRegion

```
 1: void CPaintDlg::DrawRegion(CPaintDC *pdc, int iColor,
 2:                                           int iTool, int iShape)
 3: {
 4:   // Stifte deklarieren und erzeugen
 5: ...
 6:   CBrush lVertBrush(HS_VERTICAL, m_crColors[iColor]);
 7:   CBrush lNullBrush(RGB(192, 192, 192));
 8:
 9:   CBitmap bmpBitmap;
10:   bmpBitmap.LoadBitmap(IDB_BITMAPBRUSH);
11:   CBrush brBitmapBrush(&bmpBitmap);
12:
13:   // Größe der Zeichenbereiche berechnen
14:   CRect rRect;
15:   GetClientRect(rRect);
16: ...
17:   // Schleife durch alle Pinsel und Stifte
18:   for (int i = 0; i < 8; i++)
19:   {
20:     switch (i)
21:     {
22: ...
23:         pdc->SelectObject(&brVertBrush);
24:         break;
25:       case 7:    // Null - Bitmap
26:         // Passenden Stift und Pinsel auswählen
27:         pdc->SelectObject(&pnNullPen);
28:         pdc->SelectObject(&brBitmapBrush);
29:         break;
30:       }
31:       // Welches Werkzeug wird verwendet?
32: ...
33:   pdc->SelectObject(pOldBrush);
34:   pdc->SelectObject(pOldPen);
35: }
```

Tag 10

Quiz-Antworten

1. Single Document Interface - Einzeldokumentschnittstelle

2. Die Ansichtsklasse ist für die Anzeige des Dokuments verantwortlich.
3. Um das Dokument neu zu zeichnen, wird die Funktion OnDraw in der Ansichtsklasse aufgerufen.
4. Um das aktuelle Dokument zu löschen, schreibt man den entsprechenden Code in die Funktion DeleteContents der Dokumentklasse.
5. Die Dokumentklasse verwaltet und manipuliert die Daten. Sie verwaltet die abstrakte Repräsentation des zu bearbeitenden Dokuments.
6. Die fünf Basisklassen in MDI-Anwendungen sind die von CWinApp abgeleitete Klasse, die von CMDIFrameWnd abgeleitete Klasse, die von CMDIChildWnd abgeleitete Klasse, die von CDocument abgeleitete Klasse und die von CView abgeleitete Klasse.
7. Der in der Titelzeile angezeigte Dokumentname sollte als Hinweis genügen. :-)

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Markieren Sie im Arbeitsbereich auf der Registerkarte Klassenansicht die Klasse CLine. Klicken Sie mit der rechten Maustaste und wählen Sie aus dem Kontextmenü den Befehl Hinzufügen / Variable hinzufügen.

2. Legen Sie den Variablentyp mit UINT, die Deklaration mit m_nWidth und den Zugriffsstatus mit private fest. Klicken Sie auf Fertig stellen, um die Variable hinzuzufügen.
3. Klicken Sie mit der rechten Maustaste im Baum der Klassenansicht auf den Konstruktor von CLine. Wählen Sie aus dem Kontextmenü den Befehl Gehe zu Definition.
4. Fügen Sie im Konstruktor das vierte Argument hinzu.
5. Setzen Sie das Element m_nWidth auf das neue Argument, wie es Listing 10.A zeigt.

Listing 10.A: Der modifizierte Konstruktor von CLine

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo,
2:              COLORREF crColor, UINT nWidth)
3: {
4:   // Anfangs- und Endpunkte initialisieren
5:   m_ptFrom = ptFrom;
6:   m_ptTo = ptTo;
7:   m_crColor = crColor;
8:   m_nWidth = nWidth;
9: }
```

6. Klicken Sie mit der rechten Maustaste im Baum der Klassenansicht auf den Konstruktor von CLine. Wählen Sie aus dem Kontextmenü den Befehl Gehe zu Deklaration.
7. Fügen Sie UINT nWidth als viertes Argument in die Konstruktordeklaration ein.
8. Gehen Sie nach unten zur Funktion Draw und nehmen Sie die Änderungen gemäß Listing 10.B vor.

Listing 10.B: Die modifizierte Funktion Draw

```
 1: void CLine::Draw(CDC * pDC)
 2: {
 3:   // Stift erzeugen
 4:   CPen lpen (PS_SOLID, m_nWidth, m_crColor);
 5:
 6:   // Neuen Stift als Zeichenobjekt festlegen
 7:   CPen* pOldPen = pDC->SelectObject(&lpen);
 8:   // Linie zeichnen
 9:   pDC->MoveTo(m_ptFrom);
10:   pDC->LineTo(m_ptTo);
11:   // Auf vorherigen Stift zurücksetzen
12:   pDC->SelectObject(pOldPen);
13: }
```

9. Gehen Sie nach unten zur Funktion Serialize und modifizieren Sie die Funktion gemäß Listing 10.C.

Listing 10.C: Die modifizierte Funktion Serialize

```
1: void CLine::Serialize(CArchive &ar)
2: {
3:   CObject::Serialize(ar);
4:
5:   if (ar.IsStoring())
6:     ar << m_ptFrom << m_ptTo << (DWORD) m_crColor << m_nWidth;
7:   else
8:     ar >> m_ptFrom >> m_ptTo >> (DWORD) m_crColor >> m_nWidth;
9: }
```

10. Markieren Sie im Arbeitsbereich auf der Registerkarte Klassenansicht die Klasse CSDISquigDoc. Klicken Sie mit der rechten Maustaste und wählen Sie aus dem Kontextmenü den Befehl Hinzufügen / Variable hinzufügen.
11. Legen Sie den Variablentyp mit UINT, die Deklaration mit m_nWidth und den Zugriffsstatus als private fest. Klicken Sie auf Fertig Stellen, um die Variable hinzuzufügen.
12. Öffnen Sie den Quellcode von CSDISquigDoc (SDISquigDoc.cpp). Gehen Sie nach unten zur Funktion OnNewDocument und bearbeiten Sie die Funktion entsprechend Listing 10.D.

Listing 10.D: Die modifizierte Funktion OnNewDocument

```
 1: BOOL CSDISquigDoc::OnNewDocument()
 2: {
 3:   if (!CDocument::OnNewDocument())
 4:     return FALSE;
 5:
 6:   // TODO: Hier Code zur Reinitialisierung einfügen
 7:   // (SDI-Dokumente verwenden dieses Dokument)
 8:   // Farbe mit Schwarz initialisieren
 9:   m_nColor = 0;
10:   // Breite mit Sehr dünn initialisieren
11:   m_nWidth = ID_WIDTH_VTHIN - ID_WIDTH_VTHIN;
12:
13:   return TRUE;
14: }
```

13. Gehen Sie nach unten zur Funktion AddLine und nehmen Sie die Änderungen gemäß Listing 10.E vor.

Listing 10.E: Die modifizierte Funktion AddLine

```
 1: CLine * CDay10Doc::AddLine(CPoint ptFrom, CPoint ptTo)
 2: {
 3:   static UINT nWidths[5] = {1, 8, 16, 24, 32};
 4:   CLine* pLine = NULL;
 5:
 6:   try
 7:   {
 8:     // Ein neues Cline-Objekt erzeugen
 9:     pLine = new CLine(ptFrom, ptTo, m_crColors[m_nColor],
10:                                          nWidths[m_nWidth]);
11:     // Die neue Linie in das Objekt-Array einfügen
12:     m_oaLines.Add(pLine);
13:     // Dokument als bearbeitet markieren
14:     SetModifiedFlag();
15:   }
16:   // Ist Speicherausnahme aufgetreten?
17:   catch (CMemoryException* perr)
18:   {
19:     // Meldung für Benutzer, schlechte Neuigkeiten mitteilen
20:     AfxMessageBox("Zu wenig Arbeitsspeicher",
21:                   MB_ICONSTOP | MB_OK);
22:     // Wurde Linienobjekt erzeugt?
23:     if (pLine)
24:     {
25:       // Objekt löschen
26:       delete pLine;
27:       pLine = NULL;
28:     }
29:     // Ausnahmeobjekt löschen
30:     perr->Delete();
31:   }
32:   return pLine;
33: }
```

14. Fügen Sie eine neue Member-Funktion in die Klasse CSDISquigDoc ein. Legen Sie den Funktionstyp als UINT, den Namen als GetWidth und den Zugriffsstatus als public fest.
15. Fügen Sie den Code aus Listing 10.F in die Funktion GetWidth ein.

Listing 10.F: Die Funktion GetWidth

```
1: UINT CSDISquigDoc::GetWidth()
2: {
3:    // Aktuelle Breite zurückgeben
4:    return ID_WIDTH_VTHIN + m_nWidth;
5: }
```

16. Gehen Sie im Arbeitsbereich auf die Registerkarte Ressourcenansicht. Erweitern Sie den Baum, sodass Sie den Inhalt des Ordners Menu sehen. Doppelklicken Sie auf die Menüressource.
17. Klicken Sie auf den leeren Menüeintrag in der obersten Ebene (am rechten Ende der Menüleiste). Geben Sie Brei&te als Beschriftung ein.
18. Ziehen Sie den neuen Menübefehl (Breite) nach links und legen Sie ihn vor dem Menü Ansicht ab.
19. Fügen Sie Menübefehle unter dem Menüeintrag Breite in der Reihenfolge und mit den Beschriftungen und IDs gemäß Tabelle 10.A hinzu.

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	ID_WIDTH_VTHIN
	Beschriftung	Sehr d&ünn
Menübefehl	ID	ID_WIDTH_THIN
	Beschriftung	&Dünn
Menübefehl	ID	ID_WIDTH_MEDIUM
	Beschriftung	&Mittel
Menübefehl	ID	ID_WIDTH_THICK
	Beschriftung	D&ick
Menübefehl	ID	ID_WIDTH_VTHICK
	Beschriftung	&Sehr dick

Tabelle 10.A: Einstellungen der Menüeigenschaften

20. Öffnen Sie die Klassenansicht. Wählen Sie CSDISquigDoc in der Klassenansicht.
21. Wählen Sie im Eigenschaftenfenster die Ansicht Ereignisse aus.
22. Fügen Sie Funktionen für die Nachrichten COMMAND und UPDATE_COMMAND_UI für alle Menübefehle des Menüs Breite hinzu.
23. Nachdem Sie den letzten Menübefehl hinzugefügt haben, bearbeiten Sie die Funktionen des Menübefehls Sehr dünn entsprechend Listing 10.G.

Listing 10.G: Die Menüfunktionen für Sehr Dünn

```
 1: void CSDISquigDoc::OnWidthVthin()
 2: {
 3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
 4:
 5:    // Aktuelle Breite auf Sehr dünn setzen
 6:    m_nWidth = ID_WIDTH_VTHIN - ID_WIDTH_VTHIN;
 7: }
 8:
 9: void CSDISquigDoc::OnUpdateWidthVthin(CCmdUI* pCmdUI)
10: {
11:    // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-UI-
12:    // Behandlungscode ein.
```

```
13:
14:    // Ist Menüpunkt Sehr dünn mit Kontrollhäkchen zu versehen?
15:    pCmdUI->SetCheck(GetWidth() == ID_WIDTH_VTHIN ? 1 : 0);
16: }
```

24. Bearbeiten Sie die Funktionen des Menübefehls Dünn entsprechend Listing 10.H. Gehen Sie bei den restlichen Menüfunktionen in der gleichen Weise vor, wobei Sie die jeweiligen Menü-IDs anstelle von ID_WIDTH_THIN einsetzen.

Listing 10.H: Die Menüfunktionen für Dünn

```
 1: void CSDISquigidthThin()
 2: {
 3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
 4:
 5:    // Aktuelle Breite auf Dünn setzen
 6:    m_nWidth = ID_WIDTH_THIN - ID_WIDTH_VTHIN;
 7: }
 8:
 9: void CSDISquigpdateWidthThin(CCmdUI* pCmdUI)
10: {
11:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
12:
13:    // Ist Menüpunkt Dick mit Kontrollhäkchen zu versehen?
14:    pCmdUI->SetCheck(GetWidth() == ID_WIDTH_THIN ? 1 : 0);
15: }
```

Tag 11

Quiz-Antworten

1. Vergeben Sie für die Schaltfläche der Symbolleiste dieselbe Objekt-ID wie für den Menübefehl.

2. Bei beiden muss das Andocken auf den gleichen Seiten in der Funktion OnCreate der Rahmenklasse aktiviert sein (mithilfe der Funktion EnableDocking).
3. Löschen Sie den Eintrag ID_INDICATOR_NUM aus der Tabelle für die Statusleistenanzeige am Anfang der Quellcode-Datei des Hauptrahmens.
4. In die Symbolleiste ist eine Trennlinie als Platzhalter einzufügen. Der Symbolleisten- Designer hindert Sie in gut gemeinter Absicht daran, Trennlinien einzufügen, weil er hier einen Fehler vermutet.

Lösungen zu den Übungen

1. Fügen Sie einen Eintrag in die Zeichenfolgentabelle mit der ID ID_INDICATOR_WIDTH und dem Titel SEHR DÜNN ein.

Nehmen Sie einen weiteren Eintrag in die Tabelle der Statusleistenanzeige am Anfang der Datei CMainFrame.cpp auf:

```
1: static UINT indicators[] =
2: {
3:     ID_SEPARATOR,           // Statusleistenanzeige
4:     ID_INDICATOR_WIDTH,
5:     ID_INDICATOR_COLOR,
6:     ID_INDICATOR_CAPS,
7:     ID_INDICATOR_NUM,
8:     ID_INDICATOR_SCRL,
9: };
```

Fügen Sie der Klasse CSDISquigDoc eine neue Member-Funktion hinzu. Legen Sie den Funktionstyp als afx_msg void und den Funktionsnamen als OnUpdateIndicatorWidth fest und fügen Sie einen Parameter vom

Typ CCmdUI* namens pCmdUI ein. Legen Sie den Zugriffsstatus als protected fest. In die Funktion übernehmen Sie den folgenden Code:

```
 1: void CSDISquigDoc::OnUpdateIndicatorWidth(CCmdUI *pCmdUI)
 2: {
 3:   CString strWidth;
 4:
 5:   // Wie heißt die aktuelle Breite?
 6:   switch (m_nWidth)
 7:   {
 8:     case 0:    // Sehr dünn
 9:       strWidth = "SEHR DÜNN";
10:       break;
11:     case 1:    // Dünn
12:       strWidth = "DÜNN";
13:       break;
14:     case 2:    // Mittel
15:       strWidth = "MITTEL";
16:       break;
17:     case 3:    // Dick
18:       strWidth = "DICK";
19:       break;
20:     case 4:    // Sehr Dick
21:       strWidth = "SEHR DICK";
22:       break;
23:   }
24:   // Statusleisten-Ausschnitt aktivieren
25:   pCmdUI->Enable(TRUE);
26:   // Text im Statusleistenausschnitt auf aktuelle Breite setzen
27:   pCmdUI->SetText(strWidth);
28: }
```

Fügen Sie der Nachrichtenzuordnungstabelle von CSDISquigDoc einen Eintrag für die Behandlungsroutine der Nachricht ON_UPDATE_COMMAND_UI wie folgt hinzu:

Listing 11.A: Die angepasste Message-Map

```
 1: // CSDISquigDoc
 2:
 3: IMPLEMENT_DYNCREATE(CSDISquigDoc, CDocument)
 4:
 5: BEGIN_MESSAGE_MAP(CSDISquigDoc, CDocument)
 6:     ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
 7:     ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
 8:     ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
 9:     ON_UPDATE_COMMAND_UI(ID_COLOR_BLUE, OnUpdateColorBlue)
10:     ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
11:     ON_UPDATE_COMMAND_UI(ID_COLOR_GREEN, OnUpdateColorGreen)
12:     ON_COMMAND(ID_COLOR_CYAN, OnColorCyan)
13:     ON_UPDATE_COMMAND_UI(ID_COLOR_CYAN, OnUpdateColorCyan)
14:     ON_COMMAND(ID_COLOR_RED, OnColorRed)
15:     ON_UPDATE_COMMAND_UI(ID_COLOR_RED, OnUpdateColorRed)
16:     ON_COMMAND(ID_COLOR_MAGENTA, OnColorMagenta)
17:     ON_UPDATE_COMMAND_UI(ID_COLOR_MAGENTA, OnUpdateColorMagenta)
18:     ON_COMMAND(ID_COLOR_YELLOW, OnColorYellow)
19:     ON_UPDATE_COMMAND_UI(ID_COLOR_YELLOW, OnUpdateColorYellow)
20:     ON_COMMAND(ID_COLOR_WHITE, OnColorWhite)
21:     ON_UPDATE_COMMAND_UI(ID_COLOR_WHITE, OnUpdateColorWhite)
22:     ON_COMMAND(ID_WIDTH_VERYTHIN, OnWidthVerythin)
23:     ON_UPDATE_COMMAND_UI(ID_WIDTH_VERYTHIN,
24:                                  OnUpdateWidthVerythin)
25:     ON_COMMAND(ID_WIDTH_THIN, OnWidthThin)
26:     ON_UPDATE_COMMAND_UI(ID_WIDTH_THIN, OnUpdateWidthThin)
```

```
27:     ON_COMMAND(ID_WIDTH_MEDIUM, OnWidthMedium)
28:     ON_UPDATE_COMMAND_UI(ID_WIDTH_MEDIUM, OnUpdateWidthMedium)
29:     ON_COMMAND(ID_WIDTH_THICK, OnWidthThick)
30:     ON_UPDATE_COMMAND_UI(ID_WIDTH_THICK, OnUpdateWidthThick)
31:     ON_COMMAND(ID_WIDTH_VERYTHICK, OnWidthVerythick)
32:     ON_UPDATE_COMMAND_UI(ID_WIDTH_VERYTHICK,
33:                                 OnUpdateWidthVerythick)
34:     ON_UPDATE_COMMAND_UI(ID_INDICATOR_COLOR,
35:                                 OnUpdateIndicatorColor)
36:     ON_UPDATE_COMMAND_UI(ID_INDICATOR_WIDTH,
37:                                 OnUpdateIndicatorWidth)
38: END_MESSAGE_MAP()
```

2. Öffnen Sie die Symbolleiste IDR_MAINFRAME im Symbolleisten-Designer. Zeichnen Sie ein Symbol auf die leere Schaltfläche am Ende der Symbolleiste. Markieren Sie die Schaltfläche, um die zugehörigen Eigenschaften zu öffnen. Legen Sie die ID mit ID_VIEW_COLORBAR fest und geben Sie eine passende Aufforderung ein. Kompilieren Sie die Anwendung erneut und führen Sie sie aus. Auf der Hauptsymbolleiste sollte nun die Umschaltung der Farbsymbolleiste funktionieren.

Tag 12

Quiz-Antworten

1. DECLARE_SERIAL und IMPLEMENT_SERIAL

2. Rufen Sie dazu die Funktion IsStoring oder IsLoading auf.
3. Den Klassennamen, den Namen der Basisklasse und die Versionsnummer
4. CFormView
5. CFile

Lösung zur Übung

Im Fenster-Designer fügen Sie zwei Optionsfelder und das Textfeld mit der Aufforderung hinzu. Legen Sie die Eigenschaften der Steuerelemente gemäß Tabelle 12.A fest.

Objekt	Eigenschaft	Einstellung
Text	Beschriftung	Geschlecht:
Optionsfeld	ID	IDC_RMALE
	Beschriftung	&männlich
	Gruppe	Eingeschaltet
Optionsfeld	ID	IDC_RFEMALE
	Beschriftung	wei&blich

Tabelle 12.A: Einstellungen der Steuerelementeigenschaften

Verbinden Sie mit den neuen Optionsfeldern wie folgt eine Variable:

Objekt	Name	Kategorie	Typ
IDC_RMALE	m_iSex	Value	int

Inkrementieren Sie die Versionsnummer im Makro IMPLEMENT_SERIAL in der Klasse CPerson. Fügen Sie eine neue Member-Variable in die Klasse CPerson ein. Legen Sie den Typ als int, den Namen als m_iSex und den Zugriffsstatus als private fest. Die Konstruktorfunktion von CPerson sollte mit der zusätzlichen Variable m_iSex in der Initialisierung aktualisiert worden sein, wie es Listing 12.A zeigt.

Listing 12.A: Der modifizierte Konstruktor von CPerson

```
1: CPerson::CPerson(void)
2: : m_bEmployed(false)
3: , m_iAge(0)
4: , m_sName(_T(""))
5: , m_iMaritalStatus(0)
6: , m_iSex(0)
7: {
8: }
```

Fügen Sie entsprechend Listing 12.B die Inline-Funktionen in die Klassendeklaration von CPerson ein, um den Wert der neuen Variablen zu setzen und abzurufen.

Listing 12.B: Die modifizierte Klassendeklaration von CPerson

```
 1: class CPerson :
 2:     public CObject
 3: {
 4:     DECLARE_SERIAL (CPerson)
 5: public:
 6:     // Funktionen zum Setzen der Variablen
 7:     void SetEmployed(BOOL bEmployed) {m_bEmployed = bEmployed;}
 8:     void SetMaritalStatus(int iStat) {m_iMaritalStatus = iStat;}
 9:     void SetAge(int iAge) {m_iAge = iAge;}
10:     void SetSex(int iSex) {m_iSex = iSex;}
11:     void SetName(CString sName) {m_sName = sName;}
12:     // Funktionen zum Holen der aktuellen Variableneinstellungen
13:     BOOL GetEmployed() {return m_bEmployed;}
14:     int GetMaritalStatus() {return m_iMaritalStatus;}
15:     int GetAge() {return m_iAge;}
16:     int GetSex() {return m_iSex;}
17:     CString GetName() {return m_sName;}
18:     CPerson(void);
19:     ~CPerson(void);
20: private:
21:     // Ist die Person erwerbstätig?
22:     BOOL m_bEmployed;
23:     // Das Alter der Person
24:     int m_iAge;
25:     // Der Name der Person
26:     CString m_sName;
27:     // Der Familienstand der Person
28:     int m_iMaritalStatus;
29: public:
30:     // Die Serialisierungsfunktion
31:     virtual void Serialize(CArchive& ar);
32: private:
33:     // Das Geschlecht der Person
34:     int m_iSex;
35: };
```

Aktualisieren Sie die Funktion Serialize in der Klasse CPerson, um die Variable m_iSex entsprechend Listing 12.C einzubinden.

Listing 12.C: Die modifizierte Funktion CPerson.Serialize

```
 1: void CPerson::Serialize(CArchive& ar)
 2: {
 3:   // Call the ancestor function
 4:   CObject::Serialize(ar);
 5:
 6:   // Are we writing?
```

```
 7:   if (ar.IsStoring())
 8:     // Write all of the variables, in order
 9:     ar << m_sName << m_iAge << m_iMaritalStatus << m_bEmployed 10: << m_iSex;
11:   else
12:     // Read all of the variables, in order
13:     ar >> m_sName >> m_iAge >> m_iMaritalStatus >> m_bEmployed
14:        >> m_iSex;
15: }
```

Aktualisieren Sie die Versionsnummer im Makro IMPLEMENT_SERIAL.

```
IMPLEMENT_SERIAL (CPerson, CObject, 2)
```

Modifizieren Sie die Funktion PopulateView im Ansichtsobjekt, um die Variable m_iSex in den Datenaustausch aufzunehmen, wie es Listing 12.D zeigt.

Listing 12.D: Die modifizierte Funktion CSerializeView.PopulateView

```
 1: void CSerializeView::PopulateView(void)
 2: {
 3:   // Einen Zeiger auf das aktuelle Dokument holen
 4:   CSerializeDoc* pDoc = GetDocument();
 5:   if (pDoc)
 6:   {
 7:     // Aktuelle Datensatzposition in der Menge anzeigen
 8:     m_sPosition.Format("Datensatz %d von %d",
 9:                        pDoc->GetCurRecordNbr(),
10:                        pDoc->GetTotalRecords());
11:   }
12:   // Ist Datensatzobjekt gültig?
13:   if (m_pCurPerson)
14:   {
15:     // Ja, alle Werte des Datensatzes holen
16:     m_bEmployed = m_pCurPerson->GetEmployed();
17:     m_iAge = m_pCurPerson->GetAge();
18:     m_sName = m_pCurPerson->GetName();
19:     m_iMaritalStatus = m_pCurPerson->GetMaritalStatus();
20:     m_iSex = m_pCurPerson->GetSex();
21:   }
22:   // Anzeige aktualisieren
23:   UpdateData(FALSE);
24: }
```

Fügen Sie eine Behandlungsroutine für das Klickereignis der beiden neuen Optionsfelder hinzu. Nutzen Sie für beide Behandlungsroutinen dieselbe Funktion. Aktualisieren Sie das Feld des Datensatzobjekts mittels der Set-Funktion wie in Listing 12.E.

Listing 12.E: Die Funktion OnSex

```
 1: void CSerializeView::OnBnClickedRSex()
 2: {
 3:   // TODO: Fügen Sie hier Ihren Kontrollbehandlungscode für die
 4:   // Benachrichtigung ein.
 5:   // Daten im Formular mit den Variablen synchronisieren
 6:   UpdateData(TRUE);
 7:   // Wenn gültiges Person-Objekt vorhanden, Daten an das Objekt
 8:   // übergeben
 9:   if (m_pCurPerson)
10:     m_pCurPerson->SetSex(m_iSex);
11:   // Get a pointer to the document
12:   CSerializeDoc* pDoc = GetDocument();
13:   if (pDoc)
```

```
14:     // Bearbeitet-Flag im Dokument setzen
15:     pDoc->SetModifiedFlag();
16: }
```

Tag 13

Quiz-Antworten

1. ActiveX Data Objects

2. OLE DB
3. Connection, Command, Parameter, Error, Recordset und Field
4. ::CoInitialize(NULL)
5. pCmd->ActiveConnection = pConn
6. Die erste Möglichkeit:

```
_RecordsetPtr ptrRs;
ptrRs = pCmd->Execute();
```

Die zweite:

```
_RecordsetPtr ptrRs;
ptrRs.CreateInstance(__uuidof(Recordset));
ptrRs->PutRefSource(pCmd);
// variant NULL erzeugen
_variant_t vNull;
vNull.vt = VT_ERROR;
vNull.scode = DISP_E_PARAMNOTFOUND;
// Recordset öffnen
ptrRs->Open(vNull, vNull, adOpenDynamic, adLockOptimistic, adCmdUnknown);
```

Lösung zur Übung

1. Fügen Sie der Dokumentklasse Behandlungsfunktionen für die Nachricht UPDATE_ COMMAND_UI der Navigationsmenü-Befehle hinzu.

2. Bearbeiten Sie diese Funktionen und fügen Sie den Code aus Listing 13.A in die Funktionen für die Menübefehle Erster und Vorheriger sowie den Code aus Listing 13.B in die Funktionen für die Menübefehle Letzter und Nächster ein.

Listing 13.A: Die Funktion CAdoDatabaseDoc.OnUpdateDatenErster

```
 1: void CAdoDatabaseDoc::OnUpdateDatenErster(CCmdUI* pCmdUI)
 2: {
 3:   // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-UI-
 4:   // Behandlungscode ein.
 5:   // Existiert der Recordset?
 6:   if (m_ptrRs)
 7:   {
 8:     // Sind wir am BOF?
 9:     if (m_ptrRs->BOF)
10:       pCmdUI->Enable(FALSE);
11:     else
12:       pCmdUI->Enable(TRUE);
13:   }
14: }
```

Listing 13.B: Die Funktion CAdoDatabaseDoc.OnUpdateDataLast

```
 1: void CAdoDatabaseDoc::OnUpdateDatenLetzter(CCmdUI* pCmdUI)
```

```
 2: {
 3:   // TODO: Add your command update UI handler code here
 4:   // Does the recordset exist?
 5:   if (m_ptrRs)
 6:   {
 7:     // Are we at the EOF?
 8:     if (m_ptrRs->EndOfFile)
 9:       pCmdUI->Enable(FALSE);
10:     else
11:       pCmdUI->Enable(TRUE);
12:   }
13: }
```

Tag 14

Quiz-Antworten

1. Eine erweiterte MFC-DLL

2. Das Makro AFX_EXT_CLASS in der Klassendeklaration
3. Eine Standard-DLL
4. Normalerweise nicht. Allerdings müssen Sie die Anwendungen, die auf die DLL zurückgreifen, neu kompilieren, wenn Sie in der exportierten Schnittstelle der DLL Änderungen vorgenommen haben.
5. Die LIB-Datei enthält Funktionsrümpfe der Funktionen in der DLL und den erforderlichen Code, um den Funktionsaufruf zu lokalisieren und an die eigentliche Funktion in der DLL weiterzuleiten.

Lösungen zu den Übungen

1. Folgen Sie diesen Schritten:

1. Erstellen Sie ein neues Projekt. Geben Sie an, dass es sich um ein MFC-DLL- Projekt handelt und geben Sie ihm einen passenden Namen wie zum Beispiel LineDll.
2. Geben Sie an, dass es sich bei der DLL um eine Erweiterungs-MFC-DLL handelt.
3. Nachdem Sie das Projektgerüst erstellt haben, kopieren Sie die Quellcode- und Header-Dateien für die Linienklasse in das Projektverzeichnis. Nehmen Sie diese Dateien in das Projekt auf.
4. Fügen Sie der Deklaration der Klasse CLine das Makro AFX_EXT_CLASS hinzu.
5. Kompilieren Sie die DLL. Kopieren Sie die DLL in das Debug-Verzeichnis für die Testanwendung.
6. Öffnen Sie das Projekt der Standard-DLL. Löschen Sie in der Datei-Ansicht des Arbeitsbereichs die Quellcode- und Header-Dateien der Linienklasse aus dem Projekt. Fügen Sie die Linien-DLL-LIB-Datei in das Projekt ein. Ändern Sie in der Quellcode-Datei die Zeichenfunktionalität, indem Sie wie folgt den Header der Linienklasse in die Version aus dem Projektverzeichnis der CLine-DLL einbinden:

   ```
   #include "StdAfx.h"
   #include "modart.h"
   #include "..\LineDll\Line.h"
   ```

7. Kompilieren Sie das Projekt. Kopieren Sie die DLL in das Debug-Verzeichnis des Projekts für die Testanwendung.
8. Starten Sie die Testanwendung.
2. Folgen Sie diesen Schritten:
1. Öffnen Sie das Projekt der Linienklasse-DLL, das Sie in der vorherigen Übung erstellt haben. Ersetzen Sie im Konstruktor der Klasse die Initialisierung der Variablen m_nWidth wie folgt durch einen konstanten Wert:

   ```
   CLine::CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor, UINT nWidth)
   {
     m_ptFrom = ptFrom;
     m_ptTo = ptTo;
     m_nWidth = 1;
     m_crColor = crColor;
   }
   ```

2. Kompilieren Sie die DLL. Kopieren Sie sie in das Debug-Verzeichnis des Projekts für die Testanwendung. Starten Sie die Testanwendung.

Tag 15

Quiz-Antworten

1. Eigenschaften, Methoden und Ereignisse

2. Um dem Benutzer die Möglichkeit zu geben, die Eigenschaften des Steuerelements festzulegen
3. Ambient, erweitert, vordefiniert und benutzerdefiniert
4. Die Schnittstellenparameter werden zu einer standardisierten, maschinenunabhängigen Struktur gepackt und über Prozessgrenzen hinweg gesendet. Diesen Vorgang bezeichnet man als Marshalling.
5. Mit dem Testcontainer für ActiveX-Steuerelemente

Lösungen zu den Übungen

1. Fügen Sie der Schnittstelle _DSquiggle eine neue Methode hinzu. Legen Sie den Methodennamen als GenNewDrawing und den Rückgabetyp als void fest. Klicken Sie auf Fertig stellen, um die Methode hinzuzufügen. In die Methode übernehmen Sie den Code aus Listing 15.A.

Listing 15.A: Die Funktion CSquiggleCtrl.GenNewDrawing

```
1: void CSquiggleCtrl:: GenNewDrawing()
2: {
3:    // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
4:    // und/oder rufen Sie die Basisklasse auf.
5:    // Flag setzen, sodass neue Zeichnung generiert wird
6:    m_bGenNewDrawing = TRUE;
7:    // Steuerelement ungültig machen, um Funktion OnDraw auszulösen
8:    Invalidate();
9: }
```

2. Fügen Sie der Schnittstelle _DSquiggle eine neue Methode hinzu. Legen Sie den Methodennamen als SaveDrawing und den Rückgabetyp als VARIANT_BOOL fest, damit Sie der Containeranwendung mitteilen können, ob eine Zeichnung geladen wurde. Nehmen Sie zuletzt einen einzelnen Parameter in die Parameterliste auf. Legen Sie seinen Namen als sFileName und seinen Typ als BSTR fest. Klicken Sie auf Fertig stellen, um die Methode hinzuzufügen. In die Methode schreiben Sie den Code aus Listing 15.B.

Listing 15.B: Die Funktion CSquiggleCtrl.SaveDrawing

```
 1: VARIANT_BOOL CSquiggleCtrl::SaveDrawing(LPCTSTR sFileName)
 2: {
 3:    // TODO: Fügen Sie hier Ihren Dispatchhandlercode ein.
 4:    try
 5:    {
 6:      // Ein CFile-Objekt erzeugen
 7:      CFile lFile(sFileName, CFile::modeWrite);
 8:      // Ein CArchive-Objekt zum Speichern der Datei erzeugen
 9:      CArchive lArchive(&lFile, CArchive::store);
10:      // Datei speichern
11:      m_maDrawing.Serialize(lArchive);
12:    }
13:    catch (CFileException* perr)
14:    {
15:      // Fehler berichten
16:      perr->ReportError();
17:      // Ausnahmeobjekt löschen
18:      perr->Delete();
19:      return VARIANT_FALSE;
```

```
20:    }
21:    return VARIANT_TRUE;
22: }
```

Tag 16

Quiz-Antworten

1. Den Webbrowser Internet Explorer

2. Mit der Funktion GetLocationURL
3. IDOK
4. GoBack und GoForward
5. Mit der Funktion Stop

Lösungen zu den Übungen

1. Nehmen Sie einen Menübefehl in das Menü Wechseln zu auf. Legen Sie die Eigenschaften des Menübefehls nach Tabelle 16.A fest.

Objekt	ID	Einstellung
Menübefehl	ID	IDM_NAVIGATE_SEARCH
	Beschriftung	Suchen im &Web
	Eingabeaufforderung	Öffnet die Seite "Suchen im Web"\nSuchen

Tabelle 16.A: Einstellungen des Menübefehls Suchen im Web

Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsroutine für die ID IDM_NAVIGATE_SEARCH für die Nachricht COMMAND hinzu. In die Funktion schreiben Sie den Code entsprechend Listing 16.A.

Listing 16.A: Die Funktion CWebBrowseView.OnNavigateSearch

```
1: void CWebBrowseView::OnNavigateSearch()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Gehe zur Seite "Suchen im Web"
5:    GoSearch();
6: }
```

Fügen Sie eine Symbolleisten-Schaltfläche für die Menü-ID IDM_NAVIGATE_SEARCH hinzu.

2. Nehmen Sie einen Menübefehl in das Menü Wechseln zu auf. Legen Sie die Eigenschaften des Menübefehls entsprechend Tabelle 16.B fest.

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_NAVIGATE_START
	Beschriftung	&Startseite
	Aufforderung	Öffnet die Startseite\nStartseite

Tabelle 16.B: Einstellungen des Menübefehls Startseite

Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsfunktion für die ID IDM_NAVIGATE_START für die Nachricht COMMAND hinzu. Übernehmen Sie den Code aus Listing 16.B in die Funktion.

Listing 16.B: Die Funktion CWebBrowseView.OnNavigateStart

```
1: void CWebBrowseView::OnNavigateStart()
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsbehandlungscode ein.
4:    // Gehe zur Startseite
5:    GoHome();
6: }
```

Fügen Sie eine Symbolleisten-Schaltfläche für die Menü-ID IDM_NAVIGATE_START hinzu.

3. Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsroutine für die Objekt-ID IDM_NAVIGATE_STOP für die Nachricht UPDATE_COMMAND_UI hinzu. Übernehmen Sie den Code aus Listing 16.C in diese Funktion.

Listing 16.C: Die Funktion CWebBrowseView.OnUpdateNavigateStop

```
1: void CWebBrowseView::OnUpdateNavigateStop(CCmdUI* pCmdUI)
2: {
3:    // TODO: Fügen Sie hier Ihren Befehlsaktualisierungs-UI-
4:    // Behandlungscode ein.
5:    // Schaltfläche aktivieren, wenn eine Seite geladen wird
6:    pCmdUI->Enable(GetBusy());
7: }
```

Tag 17

Quiz-Antworten

1. Wenn die Anwendung im Leerlauf arbeitet und keine Nachrichten in der Nachrichtenwarteschlange stehen.

2. Die Rückgabe des Wertes TRUE bewirkt, dass die Funktion OnIdle wiederholt aufgerufen wird, solange die Anwendung im Leerlauf bleibt.
3. Ein OnIdle-Task wird nur ausgeführt, wenn die Anwendung im Leerlauf ist und keine Nachrichten in der Nachrichtenwarteschlange stehen. Ein Thread läuft dagegen vollkommen unabhängig von der übrigen Anwendung.
4. Kritische Abschnitte, Mutexe, Semaphoren und Ereignisse
5. Die verbleibenden Threads und Prozesse, die auf dem Computer laufen, erhalten wesentlich weniger Prozessorzeit.

Lösungen zu den Übungen

1. Bearbeiten Sie die Funktion OnIdle gemäß Listing 17.A.

Listing 17.A: Die modifizierte Funktion CTaskingApp.OnIdle

```
 1: BOOL CMultitaskingApp::OnIdle(LONG lCount)
 2: {
 3:    // TODO: Fügen Sie hier Ihren spezialisierten Code ein,
 4:    // und/oder rufen Sie die Basisklasse auf.
 5:    // Idle-Verarbeitung des Vorfahren aufrufen
 6:    BOOL bRtn = CWinApp::OnIdle(lCount);
 7:    // Position der ersten Dokumentvorlage ermitteln
 8:    POSITION pos = GetFirstDocTemplatePosition();
 9:    // Ist die Position gültig?
10:    if (pos)
```

```
11:   {
12:     // Zeiger auf die Dokumentvorlage holen
13:     CDocTemplate* pDocTemp = GetNextDocTemplate(pos);
40:               // Ersten Idle-Thread drehen?
41:               if (pView->m_bOnIdle1)
42:               {
43:                 // Ersten Idle-Thread drehen
44:                 pDocWnd->DoSpin(0);
45:                 bRtn = TRUE;
46:               }
47:               // Zweiten Idle-Thread drehen?
48:               if (pView->m_bOnIdle2)
49:               {
50:                 // Zweiten Idle-Thread drehen
51:                 pDocWnd->DoSpin(2);
52:                 bRtn = TRUE;
53:               }
54:             }
55:           }
56:         }
57:       }
58:     }
59:   }
60:   return bRtn;
61: }
```

2. Beim Starten der unabhängigen Threads geben Sie einem Thread die Priorität THREAD_PRIORITY_NORMAL und dem anderen die Priorität THREAD_PRIORITY_LOWEST.

Bearbeiten Sie die Funktion SuspendSpinner entsprechend Listing 17.B.

Listing 17.B: Die modifizierte Funktion CTaskingDoc.SuspendSpinner

```
 1: void CTaskingDoc::SuspendSpinner(int iIndex, BOOL bRun)
 2: {
 3:   // Wenn der Thread angehalten wird
 4:   if (!bRun)
 5:   {
 6:     // Ist der Zeiger auf den Thread gültig?
 7:     if (m_pSpinThread[iIndex])
 8:     {
 9:       // Handle für den Thread holen
10:       HANDLE m_hThread = m_pSpinThread[iIndex]->m_hThread;
11:       // Auf Beenden des Threads warten
12:       ::WaitForSingleObject (m_hThread, INFINITE);
13:     }
14:   }
15:   else    // Thread läuft
16:   {
17:     int m_iSpnr;
18:     int m_iPriority;
19:     // Which spinner to use?
20:     switch (iIndex)
21:     {
22:       case 0:
23:         m_iSpnr = 1;
24:         m_iPriority = THREAD_PRIORITY_NORMAL;
25:         break;
26:       case 1:
27:         m_iSpnr = 3;
28:         m_iPriority = THREAD_PRIORITY_LOWEST;
29:         break;
30:     }
```

```
31:     // Start the thread, passing a pointer to the spinner
32:     m_pSpinThread[iIndex] = AfxBeginThread(ThreadFunc,
33:               (LPVOID)&m_cSpin[m_iSpnr], m_iPrioritys);
34:   }
35: }
```

Tag 18

Quiz-Antworten

1. Die Netzwerkadresse (oder den Namen) des Computers und den Port, auf dem der Server lauscht

2. Listen
3. OnReceive
4. OnConnect.
5. Send

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Nehmen Sie eine Member-Variable in die Dialogfeldklasse (CSockDlg) auf. Legen Sie den Variablentyp als BOOL, den Namen als m_bConnected und den Zugriffsstatus als private fest.

2. Initialisieren Sie die Variable in der Funktion OnInitDialog als FALSE.
3. Setzen Sie die Variable in der Dialogfeldfunktion OnAccept auf TRUE, nachdem die Verbindung angenommen wurde.
4. Setzen Sie die Variable in der Dialogfeldfunktion OnClose auf FALSE.
5. Modifizieren Sie die Dialogfeldfunktion OnAccept gemäß Listing 18.A.

Listing 18.A: Die modifizierte Funktion CSockDlg.OnAccept

```
 1: void CSockDlg::OnAccept()
 2: {
 3:   if (m_bConnected)
 4:   {
 5:     // Rejection-Socket erzeugen
 6:     CAsyncSocket sRjctSock;
 7:     // Zu sendende Meldung erzeugen
 8:     CString strMsg =
 9:     "Zu viele Verbindungen, bitte später noch einmal versuchen";
10:     // Verbindung über den Rejection-Socket akzeptieren
11:     m_sListenSocket.Accept(sRjctSock);
12:     // Rejection-Meldung senden
13:     sRjctSock.Send(LPCTSTR(strMsg), strMsg.GetLength());
14:     // Socket schließen
15:     sRjctSock.Close();
16:   }
17:   else
18:   {
19:     // Verbindungsgesuch entgegennehmen
20:     m_sListenSocket.Accept(m_sConnectSocket);
21:     // Socket als verbunden kennzeichnen
22:     m_bConnected = TRUE;
23:     // Text- und Nachrichtensteuerelemente aktivieren
24:     GetDlgItem(IDC_EMSG)->EnableWindow(TRUE);
25:     GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
26:     GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
27:   }
28: }
```

Tag 19

Quiz-Antworten

1. Fügen Sie vor der Klassendeklaration das Schlüsselwort __gc ein.

2. Convert::ToInt32() und Int32::Parse()
3. Fügen Sie vor der Deklaration der Klasse oder Struktur das Schlüsselwort __value ein.
4. Cons ole::WriteLine() oder Console::Write()
5. #using <somedll.dll>

Lösung zur Übung

Modifizieren Sie die AddNumbers-Funktion der Klasse CAddNumbers gemäß Listing 19.A.

Listing 19.A: Die modifizierte Funktion AddNumbers

```
 1: void CAddNumbers::AddNumbers(void)
 2: {
 3:   int iResult;
 4:   String* strMsg;
 5:   bool bAdd;
 6:
 7:   // Benutzer nach der durchzuführenden Aktion fragen
 8:   Console::Write(S"Geben Sie + ein, um die Zahlen zu addieren,
                      oder -, um sie zu subtrahieren: ");
10:   // Antwort des Benutzers lesen
11:   strMsg = Console::ReadLine();
12:   // Plus-Zeichen eingegeben?
13:   if (strMsg->Equals("+"))
14:   {
15:     // Die beiden Zahlen addieren
16:     iResult = m_iNumber1 + m_iNumber2;
17:     bAdd = true;
18:   }
19:   else
20:   {
21:     // Die beiden Zahlen subtrahieren
22:     iResult = m_iNumber1 - m_iNumber2;
23:     bAdd = false;
24:   }
25:   // Die erste eingegebene Zahl auf das Anzeigen vorbereiten
26:   strMsg = String::Concat(S"Erste eingegebene Zahl: ",
27:             Convert::ToString(m_iNumber1));
28:   // Erste eingegebene Zahl anzeigen
29:   Console::WriteLine(strMsg);
30:   // Die zweite eingegebene Zahl auf das Anzeigen vorbereiten
31:   strMsg = String::Concat(S"Zweite eingegebene Zahl: ",
32:             Convert::ToString(m_iNumber2));
33:   // Zweite eingegebene Zahl anzeigen
34:   Console::WriteLine(strMsg);
35:   // Ergebnis auf das Anzeigen vorbereiten
36:   if (bAdd)
37:     strMsg = String::Concat(S"Ergebnis der Addition: ",
38:                 iResult.ToString());
39:   else
40:     strMsg = String::Concat(S"Ergebnis der Subtraktion: ",
41:                 iResult.ToString());
42:   // Ergebnis der Berechnung anzeigen
43:   Console::WriteLine(strMsg);
44: }
```

Tag 20

Quiz-Antworten

1. Vorlagen und Makros für ganze Klassen

2. Die Standard Template Library (STL)
3. Die Business Logic-Klassen müssen von den Schnittstellenklassen getrennt sein.
4. Weil die vorkompilierten Header in einem Binärformat vorliegen, das nicht von verwaltetem und nicht verwaltetem Code gleichzeitig verwendet werden kann

Lösung zur Übung

Um diese Übung fertig zu stellen, folgen Sie diesen Schritten:

1. Fügen Sie der Klasse CTaxCalculator eine neue Funktion hinzu.

2. Legen Sie den Rückgabetyp der neuen Funktion als BOOL, den Namen als GetTaxRate und den Zugriff als public fest und fügen Sie zwei Parameter ein. Legen Sie den Typ des ersten Parameters als wchar_t* und seinen Namen als wszCategory fest. Legen Sie den Typ des zweiten Parameters als float* und seinen Namen als pfTaxRate fest. Klicken Sie auf Fertig stellen, um die Methode in Ihre Klasse einzufügen.
3. Bearbeiten Sie die neue Methode und fügen Sie den Code aus Listing 20.A ein.

Listing 20.A: Die Funktion GetTaxRate

```
 1: BOOL TaxCalcs::CTaxCalculator::GetTaxRate(wchar_t* wszCategory,
 2:                                           float* pfTaxRate)
 3: {
 4:   // Array mit den Steuerkategorien deklarieren
 5:   static const wchar_t *wszCategories[] = {L"Nahrungsmittel",
 6:                              L"Kleidung", L"Musik"};
 7:   // Array mit den Steuersätzen deklarieren
 8:   static const float fRates[] = {0.0725, 0.0835, 0.081};
 9:   // Schleife durch die Kategorien
10:   for (int i = 0; i < 3; i++)
11:   {
12:     // Wurde diese Kategorie gekauft?
13:     if (wcscmp(wszCategory, wszCategories[i]) == 0)
14:     {
15:       // Ja, Steuersatz zurückgeben
16:       *pfTaxRate = fRates[i];
17:       // TRUE zurückgeben
18:       return TRUE;
19:     }
20:   }
21:   // Gekaufte Kategorie wurde nicht gefunden, FALSE zurückgeben
22:   return FALSE;
23: }
```

4. Sie müssen die freigelegte Methode hinzufügen, mithilfe derer Clients auf die Business Logic-Klasse in Ihrer Komponente zugreifen werden. Fügen Sie eine neue Methode zu der Schnittstelle IGetTaxForPurchase hinzu. Achten Sie darauf, die Methode zur Schnittstelle hinzuzufügen, nicht zur Klasse.
5. Legen Sie den Namen der Methode als GetTaxRate fest. Sie fügen zwei Parameter ein, die den beiden an die Business Logic-Klasse zu übergebenden entsprechen. Für den ersten Parameter wählen Sie BSTR* als Parametertyp, geben Sie bstrCategory als Namen ein, aktivieren Sie das Kontrollkästchen in und klicken Sie auf Hinzufügen. Für den zweiten Parameter wählen Sie FLOAT* als Parametertyp, geben Sie pfTax als Namen ein, aktivieren Sie das Kontrollkästchen out und klicken Sie auf Hinzufügen. Klicken Sie auf Fertig stellen, um die Methode zur Schnittstelle hinzuzufügen.
6. Bearbeiten Sie die Methode und fügen Sie den Code aus Listing 20.B ein.

Listing 20.B: Die Methode GetTaxRate

```
 1: STDMETHODIMP CGetTaxForPurchase::GetTaxRate(BSTR* bstrCategory,
 2:                                             FLOAT* pfTax)
 3: {
 4:   // TODO: Fügen Sie hier Ihren Implementierungscode ein.
 5:   // Haben wir einen gültigen Zeiger auf die Variable, in der
 6:   // die zu zahlende Steuer platziert wird?
 7:   if (!pfTax)
 8:     return S_FALSE;
 9:   // Zu zahlende Steuer berechnen
10:   if (m_pTaxCalc->GetTaxRate((wchar_t*)*bstrCategory, pfTax))
11:     return S_OK;
12:   else
13:     return S_FALSE;
14: }
```

7. Öffnen Sie die Quellcode-Datei TaxCalcClient.cpp und fügen Sie zur main-Funktion den fett gedruckten Code aus Listing 20.C hinzu.

Listing 20.C: Die Hauptfunktion des COM-Clients

```
 1: int _tmain(int argc, _TCHAR* argv[])
 2: {
 3:   // COM-Umgebung starten
 4:   CoInitialize(NULL);
 5:   {
 6:     // Namespace des Steuerberechnungsmoduls verwenden
 7:     using namespace TaxCalculations;
...
31:       // Summe anzeigen
32:       printf("    Summe = %9.2f\n", fPurchaseAmt + fTaxAmt);
33:     }
34:     // Berechneten Steuersatz ermitteln
35:     if (pGetTax->GetTaxRate(&bstrCat1, &fTaxRate) == S_OK)
36:       // Berechneten Steuersatz anzeigen
37:       printf("Steuersatz = %f\n", fTaxRate);
38:   }
39:   // COM-Umgebung stoppen
40:   CoUninitialize();
41:   return 0;
42: }
```

8. Markieren Sie in der Projektmappen-Explorer-Ansicht das Projekt TaxCalcClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.
9. Kompilieren Sie die Anwendung und führen Sie sie unter Verwendung der COM- Schnittstelle aus.
10. Fügen Sie der verwalteten Wrapper-Klasse (MgdGetTaxForPurchase) eine neue Funktion hinzu. Öffnen Sie die Header-Datei und fügen Sie die Funktionsdeklaration gemäß Listing 20.D ein.

Listing 20.D: Die modifizierte Header-Datei von MgdGetTaxForPurchase

```
 1: #pragma once
 2: #include "TaxCalculator.h"
 3: #using <mscorlib.dll>
 4: using namespace System;
 5:
 6: public __gc class MgdGetTaxForPurchase
 7: {
 8: private:
 9:     TaxCalcs::CTaxCalculator* m_pTaxCalc;
10: public:
11:     MgdGetTaxForPurchase(void);
12:     BOOL GetPurchaseTaxes(float fPurchaseAmt,
```

```
13:             System::String* pstrCategory, float* pfTax);
14:     BOOL GetTaxRate(System::String* pstrCategory, float* pfTax);
15: };
```

11. Öffnen Sie die Quellcode-Datei der Klasse MgdGetTaxForPurchase noch einmal und fügen Sie die Definition für die neue Funktion entsprechend Listing 20.E ein.

Listing 20.E: Die Funktionsdefinition von GetTaxRate

```
 1: BOOL MgdGetTaxForPurchase::GetTaxRate(System::String*
 2:                            pstrCategory, float* pfTax)
 3: {
 4:   wchar_t *pCat;
 5:   int iLen;
 6:
 7:   // Haben wir einen gültigen Zeiger auf die Variable,
 8:   // in der die zu zahlende Steuer platziert wird?
 9:   if (!pfTax)
10:     return FALSE;
11:   // Kategorie von BSTR in wchar_t umwandeln
12:   __wchar_t pCategory __gc[] = pstrCategory->ToCharArray();
13:   iLen = pstrCategory->get_Length();
14:   pCat = new wchar_t[iLen + 1];
15:   for (int i = 0; i < iLen; i++)
16:   {
17:     pCat[i] = pCategory[i];
18:   }
19:   pCat[i] = NULL;
20:   // Steuersatz ermitteln
21:   if (m_pTaxCalc->GetTaxRate(pCat, pfTax))
22:     return TRUE;
23:   else
24:     return FALSE;
25: }
```

12. Öffnen Sie die Quellcode-Datei MgdTaxCalcClient.cpp und fügen Sie die fett gedruckten Zeilen aus Listing 20.F in die main-Funktion des verwalteten Clients ein.

Listing 20.F: Die Hauptfunktion des verwalteten Clients

```
 1: // Dies ist der Einstiegspunkt für die Anwendung
 2: int _tmain(void)
 3: {
 4:   // TODO: Ersetzen Sie den Beispielcode durch Ihren eigenen
 5:   // Code. Schnittstellenzeiger für Steuerberechnungsmodul holen
 6:   MgdGetTaxForPurchase* pGetTax = new MgdGetTaxForPurchase();
 7:
 8:   float fPurchaseAmt, fTaxAmt, fTotalDue, fTaxRate;
 9:
10:   // String für die gekaufte Kategorie erzeugen
11:   String* pstrCategory = L"Kleidung";
12:   // Kaufpreis angeben
13:   fPurchaseAmt = 45.0;
14:   // Zu berechnende Steuer abfragen
15:   if (pGetTax->GetPurchaseTaxes(fPurchaseAmt, pstrCategory,
16:                                 &fTaxAmt))
17:   {
18:     // Summe berechnen
19:     fTotalDue = fPurchaseAmt + fTaxAmt;
20:     // Gekaufte Kategorie anzeigen
21:     Console::WriteLine(pstrCategory);
22:     // Kaufpreis anzeigen
23:     Console::Write("Kaufpreis = ");
```

```
24:     Console::WriteLine(fPurchaseAmt.ToString("c"));
25:     // Zu zahlende Steuer anzeigen
26:     Console::Write("   Steuer = ");
27:     Console::WriteLine(fTaxAmt.ToString("c"));
28:     // Summe anzeigen
29:     Console::Write("    Summe = ");
30:     Console::WriteLine(fTotalDue.ToString("c"));
31:   }
32:   // Berechneten Steuersatz ermitteln
33:   if (pGetTax->GetTaxRate(pstrCategory, &fTaxRate))
34:   {
35:     // Berechneten Steuersatz anzeigen
36:     Console::Write("Steuersatz = ");
37:     Console::WriteLine(fTaxRate.ToString());
38:   }
39:   return 0;
40: }
```

13. Markieren Sie in der Projektmappen-Explorer-Ansicht das Projekt MgdTaxCalcClient und klicken Sie mit der rechten Maustaste darauf. Wählen Sie Als Startprojekt festlegen aus dem Kontextmenü.
14. Kompilieren Sie die Anwendung und führen Sie sie unter Verwendung der verwalteten C++-Wrapper-Schnittstelle aus.

Tag 21

Quiz-Antworten

1. Fügen Sie dem Projekt einen Verweis für die Komponente hinzu.

2. Fügen Sie eine #using-Direktive hinzu, um die Komponente zu verwenden. Modifizieren Sie das vorerstellte Ereignis für das Projekt und kopieren Sie die Komponenten-DLL in das Ausgabeverzeichnis des Projekts. Fügen Sie eine Anweisung zur Auflösung von #using-Direktiven für das Ausgabeverzeichnis das Projekts ein.
3. wchar_t*
4. Nein

Lösung zur Übung

Um diese Übung durchzuführen, folgen Sie diesen Schritten:

1. Wenn Sie Ihre Komponente mit VB erstellt haben, fügen Sie die Funktion aus Listing 21.A ein. Wenn Sie Ihre Komponente mit C# erstellt haben, fügen Sie eine neue Methode zur Klasse CTaxCalculations hinzu.

Listing 21.A: Die VB-Funktion GetTaxRate

```
 1: Public Function GetTaxRate(ByVal strCategory As String) As
 2:                                                          Double
 3:   ' Array mit den Steuerkategorien deklarieren
 4:   Dim strCategories() As String = {"Nahrungsmittel",
 5:                                    "Kleidung", "Musik"}
 6:   ' Array mit den Steuersätzen deklarieren
 7:   Dim dbRates() As Double = {0.0725, 0.0835, 0.081}
 8:   Dim i As Int32
 9:
10:   ' Voreinstellung des Standard-Rückgabewerts
11:   GetTaxRate = 0
12:   ' Schleife durch die Kategorien
13:   For i = 0 To 2
14:     ' Wurde diese Kategorie gekauft?
15:     If (strCategory = strCategories(i)) Then
16:       ' Ja, zu zahlende Steuer berechnen
```

```
17:       GetTaxRate = dbRates(i)
18:     End If
19:   Next
20: End Function
```

2. Legen Sie im Assistenten zum Hinzufügen von C#-Methoden den Zugriff auf die Methode als public, den Rückgabewert als double und den Namen als GetTaxRate fest und fügen Sie einen Parameter ein. Legen Sie den Modifizierer für den Parameter als Keine, den Parametertyp als string und den Parameternamen als strCategory fest. Klicken Sie auf Fertig stellen, um diese Methode hinzuzufügen.
3. Bearbeiten Sie die gerade erstellte Methode und fügen Sie den Code aus Listing 21.B ein.

Listing 21.B: Die C#-Funktion GetTaxRate

```
 1: public double GetTaxRate(string strCategory)
 2: {
 3:   // Array mit den Steuerkategorien deklarieren
 4:   String[] strCategories = {"Nahrungsmittel",
 5:                             "Kleidung", "Musik"};
 6:   // Array mit den Steuersätzen deklarieren
 7:   double[] dbRates = {0.0725F, 0.0835F, 0.081F};
 8:   // Schleife durch die Kategorien
 9:   for (int i=0; i < 3; i++)
10:   {
11:     // Wurde diese Kategorie gekauft?
12:     if (strCategory == strCategories[i])
13:     {
14:       // Ja, zu zahlende Steuer berechnen
15:       return dbRates[i];
16:     }
17:   }
18:   // Gekaufte Kategorie wurde nicht gefunden, 0 zurückgeben
19:   return 0;
20: }
```

4. Modifizieren Sie die verwaltete C++-Client-Anwendung und fügen Sie den fett gedruckten Code aus Listing 21.C ein.

Listing 21.C: Die modifizierte Hauptfunktion des verwalteten C++-Clients

```
 1: // Dies ist der Einstiegspunkt für die Anwendung
 2: int tmain(void)
 3: {
 4:   // TODO: Ersetzen Sie den Beispielcode durch Ihren eigenen
 5:   // Code.
 6:   VBTaxCalc::CTaxCalculations* pGetTax =
 7:                               new VBTaxCalc::CTaxCalculations();
 8:
 9:   double dbPurchaseAmt, dbTaxAmt, dbTotalDue, dbTaxRate;
10:
11:   // String für die gekaufte Kategorie erzeugen
12:   String* pstrCategory = S"Kleidung";
13:
14:   // Kaufpreis angeben
15:   dbPurchaseAmt = 45.0;
16:   // Zu berechnende Steuer abfragen
17:   dbTaxAmt = pGetTax->CalculateTaxes(dbPurchaseAmt,
18:                                      pstrCategory);
19:   if (dbTaxAmt > 0.0)
20:   {
21:     // Summe berechnen
22:     dbTotalDue = dbPurchaseAmt + dbTaxAmt;
23:     // Gekaufte Kategorie anzeigen
24:     Console::WriteLine(pstrCategory);
```

```
25:     // Gekaufte Menge anzeigen
26:     Console::Write("Kaufpreis = ");
27:     Console::WriteLine(dbPurchaseAmt.ToString("c"));
28:     // Zu zahlende Steuer anzeigen
29:     Console::Write("   Steuer = ");
30:     Console::WriteLine(dbTaxAmt.ToString("c"));
31:     // Summe anzeigen
32:     Console::Write("    Summe = ");
33:     Console::WriteLine(dbTotalDue.ToString("c"));
34:     // Berechneten Steuersatz ermitteln
35:     dbTaxRate = pGetTax->GetTaxRate(pstrCategory);
36:     if (dbTaxRate > 0.0)
37:     {
38:       // Berechneten Steuersatz anzeigen
39:       Console::Write("Steuersatz = ");
40:       Console::WriteLine(dbTaxRate.ToString());
41:     }
42:   }
43:
44:   return 0;
45: }
```

Stichwortverzeichnis

Symbole

Numerics

A

E

F

H

I

J

K

L

M

N

O

Q

R

T

U

V

W

X

Z