

brought to you by

FS:Company



Inhaltsübersicht



Teil 1	Die Entwicklungsumgebung	19
1	Visual C++ und Visual Studio	21
1.1	Visual Studio: Überblick über die Entwicklungsumgebung	22
1.2	Die grundlegenden Arbeitsschritte	28
1.3	Kommandozeilen-Hilfsmittel	33
1.4	Neuerungen in der Version 6.0.	38
1.5	Zusammenfassung	39
2	Arbeitsbereiche und Projekte	41
2.1	Projekte und Arbeitsbereiche	42
2.2	Das Arbeiten mit Unterprojekten	43
2.3	Projekte erstellen und bearbeiten	45
2.4	Projekte konfigurieren	46
2.5	Zusammenfassung	49
3	Die Assistenten	51
3.1	Der Anwendungsassistent	52
3.2	Weitere Anwendungsassistenten und Projekttypen	56
3.3	Der Klassen-Assistent.	58

3.4	Die Klasseninformationsdatei	71
3.5	Zusammenfassung.	72
4	Browser und Debugger	75
4.1	Der Quellcode-Browser	75
4.2	Der Debugger	77
4.3	Weitere Debug-Techniken und -Tools	88
4.4	Zusammenfassung.	94
5	Optimierung.	97
5.1	Der Compiler	97
5.2	Der Profiler	98
5.3	Der Visual Studio Analyzer	103
5.4	Zusammenfassung.	103
6	Wiederverwenden von Programmcode mit der Komponentensammlung	105
6.1	Die Komponentensammlung	105
6.2	Zusammenfassung.	108
Teil 2	Windows-Grundlagen und API	109
7	Betriebssystemübersicht.	111
7.1	Fenster und Nachrichten	112
7.2	Nachrichten und Multitasking	120
7.3	Windows-Funktionsaufrufe	123
7.4	Plattformunterschiede.	132
7.5	Zusammenfassung.	133
8	Das API-Anwendungsgerüst.	135
8.1	Das »wahre« Hello-World-Programm	135
8.2	Eine einfache Nachrichtenschleife: Senden und Hinterlegen von Nachrichten .	137
8.3	Fensterfunktionen	139

8.4	Mehrere Nachrichtenschleifen und Fensterfunktionen	143
8.5	Zusammenfassung	147
9	Fenster, Dialogfelder und Steuerelemente	149
9.1	Die Fensterhierarchie	150
9.2	Fensterverwaltung.	153
9.3	Zeichnen der Inhalte eines Fensters	158
9.4	Fensterverwaltungsnachrichten	160
9.5	Fensterklassen.	162
9.6	Dialogfelder	170
9.7	Standarddialoge.	174
9.8	Steuerelemente	182
9.9	Zusammenfassung	188
10	Ressourcen	191
10.1	Elemente einer Ressourcendatei.	192
10.2	Kompilieren und Verwenden von Ressourcenskripten	202
10.3	Lokalisation von Ressourcen.	203
10.4	Ressourcenvorlagen	204
10.5	Zusammenfassung	204
11	Zeichnen und Gerätekontexte	207
11.1	Das GDI, Gerätetreiber und Ausgabegeräte	207
11.2	Gerätekontexte	208
11.3	Koordinaten	211
11.4	Zeichenobjekte	222
11.5	Clipping.	231
11.6	Zeichenfunktionen.	234
11.7	Hinweise zum Drucken	240
11.8	Zusammenfassung	241

12	Threads und Prozesse	243
12.1	Multitasking in der Win32-Umgebung	244
12.2	Programmierung mit Prozessen und Threads	249
12.3	Zusammenfassung	261
13	DLLs – Dynamische Bibliotheken	263
13.1	Arten von Bibliotheken	263
13.2	Programmieren mit DLLs	264
13.3	Zusammenfassung	269
14	Speicherverwaltung	271
14.1	Prozesse und der Speicher	271
14.2	Von 16- zu 32-Bit	276
14.3	Einfache Speicherverwaltung	279
14.4	Virtueller Speicher und erweiterte Speicherverwaltung	281
14.5	Threads und Speicherverwaltung	290
14.6	Zugriff auf den physikalischen Speicher und die E/A-Schnittstellen	292
14.7	Zusammenfassung	292
15	Dateiverwaltung	295
15.1	Übersicht über das Dateisystem	296
15.2	Win32-Dateiobjekte	297
15.3	Low-Level-Ein-/Ausgabe	305
15.4	Ein-/Ausgabestrom	307
15.5	Spezielle Geräte	309
15.6	Zusammenfassung	311
16	Die Windows-Zwischenablage	313
16.1	Die Formate der Zwischenablage	313
16.2	Zwischenablageoperationen	316

16.3	Eine einfache Implementierung	320
16.4	Zusammenfassung	324
17	Die Registrierung	325
17.1	Die Struktur der Registrierung	325
17.2	Manuelle Bearbeitung der Registrierung	329
17.3	Allgemein verwendete Registrierungsschlüssel	330
17.4	Anwendungen und die Registrierung	333
17.5	Zusammenfassung	338
18	Ausnahmebehandlung	341
18.1	Ausnahmebehandlung in C und C++.	341
18.2	C- und C++-Ausnahmefehler	349
18.3	Zusammenfassung	353
Teil 3	Die MFC.	355
19	Microsoft Foundation Classes: Eine Übersicht	357
19.1	MFC und Anwendungen.	357
19.2	MFC-Grundlagen.	359
19.3	Fensterunterstützungsklassen.	364
19.4	Anwendungsarchitekturklassen	370
19.5	Verschiedene Klassen.	373
19.6	Zusammenfassung	377
20	Das MFC-Anwendungsgerüst.	379
20.1	Ein einfaches MFC-Anwendungsgerüst	379
20.2	Hinzufügen von Programmcode zur Anwendung.	398
20.3	Zusammenfassung	402

21	Die Arbeit mit Dokumenten und Ansichten	403
21.1	Die CDocument-Klasse	403
21.2	Die CView-Klasse.	414
21.3	Zusammenfassung.	420
22	Dialoge und Registerdialoge	423
22.1	Erstellen von Dialogen.	424
22.2	Dialog-Datenaustausch	434
22.3	Dialoge und Nachrichtенbearbeitung	438
22.4	Registerdialoge	439
22.5	Zusammenfassung.	448
23	MFC-Unterstützung für Standarddialoge und Standardsteuerelemente	451
23.1	Standarddialoge	452
23.2	Standardsteuerelemente	460
23.3	Zusammenfassung.	473
24	Gerätekontext und GDI-Objekte	477
24.1	Gerätekontexte.	478
24.2	Unterstützung von GDI-Objekten in der MFC	493
24.3	Zusammenfassung.	499
25	Serialisierung: Datei- und Archivobjekte	501
25.1	Die CFile-Klasse	501
25.2	Die CArchive-Klasse	507
25.3	Serialisierung in MFC-Applikationsrahmen-Anwendungen	513
25.4	Zusammenfassung.	515
26	Container-Klassen	517
26.1	CObject-Container	517
26.2	Weitere Listen-Container	524

26.3	Weitere Array-Container	525
26.4	Zuordnungen.	526
26.5	Auf Templates basierende Objekt-Container	531
26.6	Zusammenfassung	539
27	Ausnahmen, Multithreading und andere MFC-Klassen.	541
27.1	Verwenden von Ausnahmen in MFC-Anwendungen	542
27.2	MFC und Multithreading	551
27.3	Weitere MFC-Klassen	556
27.4	Zusammenfassung	561
Teil 4	Die Daten	563
28	OLE, ActiveX und das Komponentenobjektmodell	565
28.1	OLE-Grundlagen und das Komponentenobjektmodell.	565
28.2	OLE und Verbunddokumente.	572
28.3	Anwendung von COM und OLE.	576
28.4	Ein einfaches Beispiel.	578
28.5	Zusammenfassung	586
29	OLE-Server	589
29.1	Server-Konzepte	589
29.2	Erstellen einer Server-Anwendung mit der MFC	590
29.3	Bearbeiten eines Server-Gerüsts	599
29.4	Zusammenfassung	605
30	OLE-Container	607
30.1	Erstellen einer Container-Anwendung mit dem Anwendungsassistenten	607
30.2	Bearbeiten der Anwendung	620
30.3	Zusammenfassung	627

31	OLE-Drag&Drop	629
31.1	Drag&Drop-Grundlagen	629
31.2	Erstellen einer Container-Anwendung	630
31.3	Drag&Drop-Unterstützung hinzufügen	634
31.4	Zusammenfassung.	643
32	Automatisierung	645
32.1	Erstellen eines Automatisierungs-Servers.	645
32.2	Standardmethoden und Standardeigenschaften	659
32.3	Zusammenfassung.	664
33	Erstellen von ActiveX-Steuerelementen mit der MFC	667
33.1	Erstellen eines Steuerelementgerüsts mit dem Anwendungsassistenten	669
33.2	Bearbeiten des Steuerelements	682
33.3	Hinzufügen eines Eigenschaftendialogs.	691
33.4	Zusammenfassung.	696
34	Verwenden der ActiveX-Templatebibliothek	697
34.1	Warum ATL?	697
34.2	Erstellen eines ActiveX-Steuerelements mit der ATL	699
34.3	Zusammenfassung.	716
35	ActiveX-Steuerelemente verwenden	717
35.1	Hinzufügen von ActiveX-Steuerelementen zu Ihrer Anwendung	719
35.2	Visual-C++-ActiveX-Steuerelemente	727
35.3	Zusammenfassung.	728
Teil 5	Datenbankprogrammierung	729
36	Datenbankprogrammierung mit ODBC	731
36.1	ODBC im Einsatz	732
36.2	Der SQL-Standard und ODBC.	742

36.3	ODBC in MFC-Anwendungen	746
36.4	Zusammenfassung	760
37	DAO – Datenzugriffsobjekte.	761
37.1	DAO-Übersicht	761
37.2	Erstellen einer DAO-Anwendung	763
37.3	DAO-Klassen	775
37.4	Zusammenfassung	779
38	OLE DB und ADO	781
38.1	OLE DB	782
38.2	Ein OLE-DB-SDK-Arbeitsbeispiel	784
38.3	Ein OLE-DB-MFC-Anwendungsbeispiel	789
38.4	ActiveX-Datenobjekte	797
38.5	Übersicht der ADO-Objekte	797
38.6	Ein Arbeitsbeispiel	799
38.7	Zusammenfassung	802
39	Datenbank- und Abfragedesign, SQL-Debugging	803
39.1	Visual-Datenbankwerkzeuge	803
39.2	Arbeiten mit einer Datenbank	807
39.3	SQL Server anwenden	811
39.4	Zusammenfassung	823
Teil 6	Internet- und Netzwerkprogrammierung	825
40	Anwenden der WinInet-API	827
40.1	Internet-Protokolle	828
40.2	Die WinInet-Bibliothek	833
40.3	Zusammenfassung	839

41	MFC-Internet-Klassen	841
41.1	Internet-Unterstützungsklassen	841
41.2	Die MFC-Internet-Klassenarchitektur	841
41.3	Aufbau von Internet-Verbindungen	842
41.4	MFC-Internet-Klassen in Anwendungen verwenden	847
41.5	Zusammenfassung.	850
42	Nachrichtenfähige Anwendungen mit MAPI	851
42.1	Die Architektur von MAPI	852
42.2	MAPI-APIs.	856
42.3	MAPI-Unterstützung in MFC.	864
42.4	Zusammenfassung.	865
43	TCP/IP-Programmierung mit WinSock	867
43.1	TCP/IP-Netzwerke und OSI	867
43.2	Die WinSock-API	874
43.3	Ein einfaches Beispiel mit WinSock	881
43.4	Programmieren mit Sockets und die Microsoft Foundation Classes	883
43.5	Weiterführende Informationen	886
43.6	Zusammenfassung.	888
44	Telefonie-Anwendungen mit TAPI	891
44.1	Übersicht zu TAPI	891
44.2	TAPI-Software-Architektur	896
44.3	TAPI-Dienste	900
44.4	Beispiel einer Datenkommunikation.	905
44.5	Zusammenfassung.	911

45	Netzwerkprogrammierung mit Pipes und Aufruf von Remote Procedures	913
45.1	Kommunizieren mit Pipes.	913
45.2	Ein Arbeitsbeispiel.	917
45.3	Microsoft Remote Procedure Calls.	919
45.4	Zusammenfassung	929
Teil 7	Multimedia	931
46	Multimedia-Anwendungen	933
46.1	Videos abspielen mit einem Funktionsaufruf.	934
46.2	Grundlagen der Multimedia-Programmierung	936
46.3	Programmieren mit MCIWnd	938
46.4	Die Mediensteuerschnittstelle	945
46.5	Fortgeschrittene Schnittstellen	951
46.6	Zusammenfassung	953
47	Die Grafikbibliothek OpenGL	955
47.1	Übersicht zu OpenGL	956
47.2	Erstellen von OpenGL-Windows-Anwendungen in C.	961
47.3	OpenGL in MFC-Anwendungen.	965
47.4	Zusammenfassung	970
48	Hochleistungsgrafik und Ton	973
48.1	Die APIs von DirectX.	974
48.2	Ein einfaches Beispiel.	982
48.3	Zusammenfassung	990
Teil 8	Anhänge	993
A	Erzeugen eigener Anwendungsassistenten	995
A.1	Wie funktioniert der Anwendungsassistent?	996

A.2	Ein Beispiel: der HelloWizard	997
A.3	Weitere Eigenschaften des Anwendungsassistenten	1009
A.4	Zusammenfassung.	1013
B	Übersicht zu C/C++	1015
B.1	Der Präprozessor.	1015
B.2	Compiler	1019
B.3	Kompilieren und Programmausführung	1035
C	Die Standard-Laufzeitbibliothek.	1037
C.1	Zugriff auf Argumente	1037
C.2	Manipulieren von Puffern	1037
C.3	Klassifizieren der Bytes	1038
C.4	Klassifizieren der Zeichen.	1038
C.5	Datenumwandlung.	1038
C.6	Debug-Unterstützung	1038
C.7	Verzeichniskontrolle.	1038
C.8	Ausnahmebehandlung	1039
C.9	Dateibehandlung.	1039
C.10	Unterstützung für Fließkommazahlen	1039
C.11	Eingabe und Ausgabe	1040
C.12	Internationalisierung	1041
C.13	Speicherzuweisung	1041
C.14	Steuerung der Prozesse und der Umgebung	1041
C.15	Suchen und Sortieren	1042
C.16	Strings manipulieren	1042
C.17	Systemaufrufe	1042
C.18	Zeitverwaltung	1042
C.19	Die ANSI-C-Laufzeitbibliothek	1043

D	Die Standard-C++-Bibliothek	1057
D.1	Die C++-Laufzeitbibliothek	1057
D.2	STL und MFC im Vergleich	1091
E	Zur CD-ROM.	1095
F	Literatur	1097
	Stichwortverzeichnis	1099

Die Entwicklungs- umgebung



1. Visual C++ und Visual Studio
2. Arbeitsbereiche und Projekte
3. Die Assistenten
4. Browser und Debugger
5. Optimierung
6. Wiederverwenden von Programmcode
mit der Komponenten-
sammlung



Visual C++ und Visual Studio

Kapitel

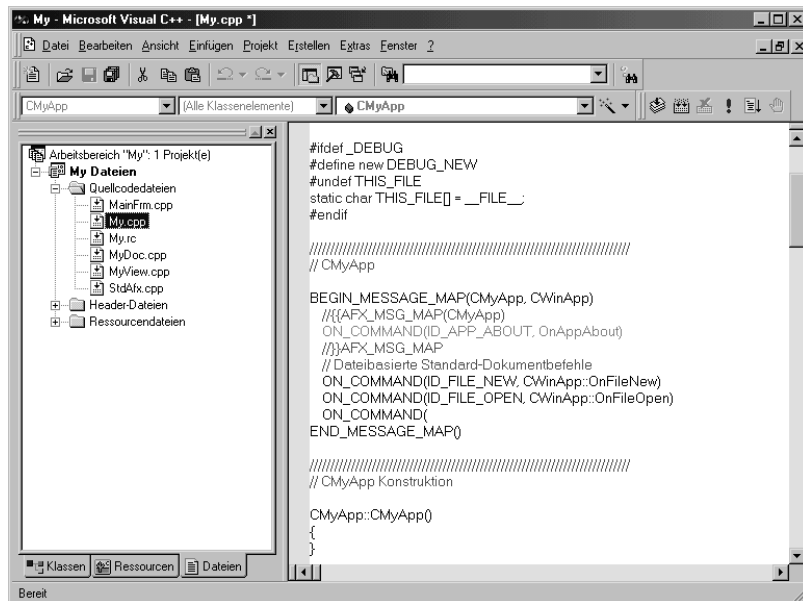
1

Das Visual Studio 6.0, die aktuelle Version des Microsoft Developer Studios, stellt eine integrierte Entwicklungsumgebung dar, in der die verschiedenen Visual-C++-Tools zur Anwendungsentwicklung (Quelltexteditor, Ressourceneditoren, Compiler, Linker, Debugger etc.) eingebettet sind. Der Vorteil für Sie besteht darin, daß Sie bei der Anwendungserstellung nicht zwischen mehreren Dienstprogrammen mit eigenen Hauptfenstern hin- und herspringen müssen, sondern alle anfallenden Aufgaben direkt innerhalb der IDE erledigen können. Dabei kann das Visual Studio nicht nur Visual C++, sondern auch anderen MS-Entwicklertools, wie z.B. Visual J++ als Front-End dienen – was insbesondere den Programmierern zugute kommt, die in mehreren Programmiersprachen gleichzeitig entwickeln.

An die Arbeit im Visual Studio gewöhnt man sich recht schnell, doch kann die komplexe Umgebung für den Einsteiger etwas verwirrend sein. Dieses Kapitel bietet Ihnen daher eine kurze Übersicht über den Aufbau des Visual Studios, beschreibt den grundlegenden Ablauf einer Arbeitssitzung mit dem Visual Studio und stellt Ihnen die interessantesten Neuerungen der 6.0-Version vor.

1.1 Visual Studio: Überblick über die Entwicklungsumgebung

Abbildung 1.1:
Das Visual
Studio



Windows-Programme bestehen selten aus einer einzigen Datei. Meist wird der Code auf mehrere Quelltextdateien verteilt, und je umfangreicher das Programm, um so mehr Quelltextdateien umfaßt es. Zu einer leistungsfähigen Entwicklungsumgebung gehört daher neben Compiler und Editor auch eine Orientierungshilfe, die es dem Programmierer erlaubt, die Übersicht über die Dateien seines Programms zu behalten. Im Visual Studio ist dies das Arbeitsbereichfenster, das standardmäßig für alle Projekte, die Sie im Visual Studio bearbeiten, angezeigt wird. (In Abbildung 1.1 sehen Sie links das Arbeitsbereichfenster und rechts ein maximiertes Editorfenster mit dem Inhalt der Quelldatei *My.cpp*. Geöffnet wurde die Datei durch Doppelklick auf den Dateinamen im Arbeitsbereichfenster.)

Außer dem Arbeitsbereich- und den Editorfenstern gibt es noch weitere Fenster, die Ihnen im Visual Studio als Schnittstelle zu den in die Entwicklungsumgebung integrierten Tools dienen.

Das Arbeitsbereichfenster – Projekte verwalten



Abbildung 1.2:
Das Arbeits-
bereichfenster

Das Arbeitsbereichfenster dient der Verwaltung von Projekten und Arbeitsbereichen (siehe Kapitel 2). Gleichzeitig ist es die zentrale Schaltstelle, über die man Quelltext- und Ressourcendateien zur Bearbeitung in die jeweiligen Editoren laden kann. Haben Sie es einmal aus Versehen geschlossen, können Sie es über den Befehl ANSICHT/ARBEITSBEREICH wieder einblenden lassen.

Das Arbeitsbereichfenster – welches per Voreinstellung in den linken Rahmen des Visual Studios integriert ist, aber auch frei im Visual Studio verschoben werden kann – verfügt über mehrere Registerseiten. (Welche Registerseiten konkret angezeigt werden, hängt von der Art des Projekts ab.)

Registerseite	Beschreibung
DATEIEN	Zeigt Ihnen die Arbeitsbereich-Hierarchie mit den zugehörigen Projekten und Quelldateien an. Mit den Befehlen aus den Kontextmenüs der verschiedenen Knoten können Sie die Quelldateien verwalten, Unterverzeichnisse für die Anzeige einrichten, die Projekte erstellen, Knoten konfigurieren. Per Doppelklick auf eine Datei können Sie diese in einen passenden Editor laden. Neue Dateien können Sie über den Befehl DATEI/NEU in ein Projekt aufnehmen. Bestehende Dateien können Sie über den Befehl BEARBEITEN/LÖSCHEN aus einem Projekt entfernen.
RESSOURCEN	Zeigt Ihnen die in den Ressourcendateien des Projekts abgelegten Ressourcen – nach Ressourcentypen geordnet – an. Mit den Befehlen aus den Kontextmenüs der Knoten können Sie neue Ressourcen anlegen, die Ressourcen-IDs bearbeiten, etc.

Tabelle 1.1:
Die Register-
seiten

Registerseite	Beschreibung
	Per Doppelklick auf eine Ressource können Sie diese in einen passenden Editor laden. Neue Ressourcen können Sie über den Befehl EINFÜGEN aus dem Kontextmenü aufnehmen. Bestehende Ressourcen können Sie über den Befehl BEARBEITEN/LÖSCHEN entfernen.
KLASSEN	Gibt Ihnen einen Überblick über die in Ihrem Programm deklarierten Klassen (einschließlich der Klassenelemente) und globalen Symbole. Mit den Befehlen aus den Kontextmenüs der verschiedenen Knoten können Sie sich weitere Informationen anzeigen lassen, in Deklarationen und Definitionen springen, Klassenelemente hinzufügen, Haltepunkte setzen, etc. Ein Doppelklick auf einen Klassennamen öffnet die zugehörige Quelltextdatei, die die Klassendeklaration enthält, und setzt die Schreibmarke auf den Anfang der Klassendeklaration. Gleiches gilt sinngemäß für Doppelklicke auf Klassenelemente und globale Bezeichner. Die Anzeige der Klassen und Klassenelemente wird ständig gemäß den Änderungen an Ihren Quelltexten aktualisiert. Sowie Sie also ein Klasselement in eine Klassendeklaration aufnehmen oder aus dieser entfernen, übernimmt das Visual Studio diese Änderung in die Klassenansicht.



In früheren Versionen verfügte das Arbeitsbereichfenster zusätzlich über die Seite InfoView, die die Schnittstelle zum Hilfesystem bildete. In der 6.0-Version wurde die Integration des Hilfesystems in die Entwicklungsumgebung aufgegeben. Die Hilfe wird zwar weiterhin über das Hilfe-Menü des Visual Studio aufgerufen, erscheint dann aber in einem eigenen Hauptfenster.

Die Editorfenster – Quelltexte und Ressourcen erstellen

Die Editorfenster dienen dem Aufsetzen und Bearbeiten von Quelltexten und Ressourcen. Um eine Datei in ein Editorfenster zu laden, bedient man sich üblicherweise des Arbeitsbereichfensters, indem man in der Dateien-Ansicht einfach auf den Knoten der zu öffnenden Datei doppelklickt. Möchte man gezielt zur Deklaration einer Klasse oder der Definition einer Elementfunktion einer Klasse springen, kann man diese in der Klassen-Ansicht des Arbeitsbereichfensters anklicken.

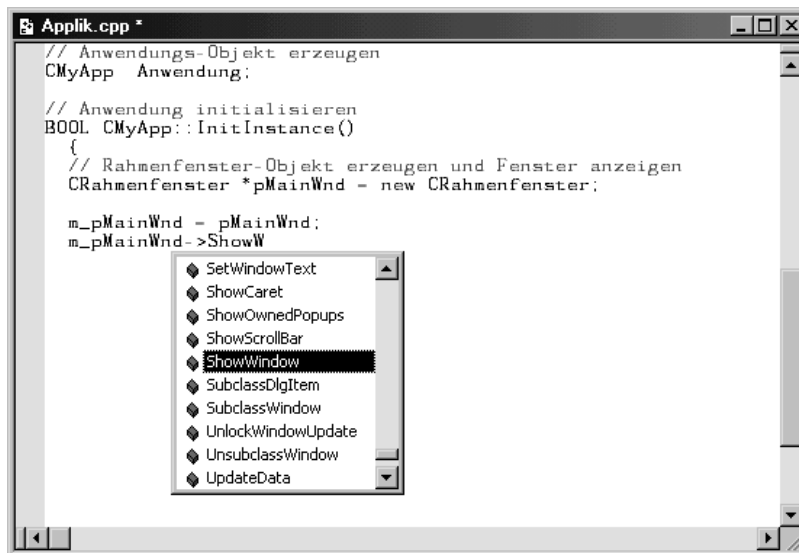
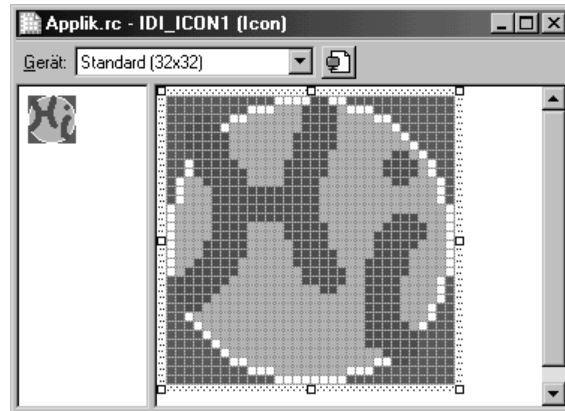


Abbildung 1.3:
Anweisungsvervollständigung

Der Quelltexteditor verfügt über eine übersichtliche Syntaxhervorhebung sowie neuerdings eine Anweisungsvervollständigung, d.h., der Editor kann Ihnen während des Eintippens Ihres Quellcodes Vorschläge für anzusprechende Klassen/Strukturelemente oder Aufrufparameter machen (siehe Abbildung 1.3).

- ➡ Wenn Sie nach dem Namen einer Klasseninstanz einen Zugriffsoperator (., ->) eintippen, springt ein Listenfeld auf, in dem die verschiedenen Elemente der Klasse aufgeführt werden. Wenn Sie weitertippen, wird das Listenfeld zu dem Eintrag gescrollt, der Ihrer bisherigen Buchstabenfolge am besten entspricht. Durch Drücken der Eingabetaste können Sie das aktuell ausgewählte Listenelement in den Quelltext einfügen lassen, wobei etwaige Tippfehler in Ihrer Buchstabenfolge korrigiert werden.
- ➡ Wenn Sie nach einem Funktionsnamen eine öffnende Klammer eingeben, springt ein Listenfeld auf, in dem Ihnen die zu der Funktion gehörenden Parameter angezeigt werden – eine Option, die Ihnen ab und an das Nachschauen in der Online-Hilfe ersparen kann. Die Parameteranzeige unterstützt auch überladene Funktionen.
- ➡ Zur Konfiguration des Quelltexteditors rufen Sie den Befehl EXTRAS/OPTIONEN auf.

Abbildung 1.4:
Ressourcenedi-
tor für Symbole

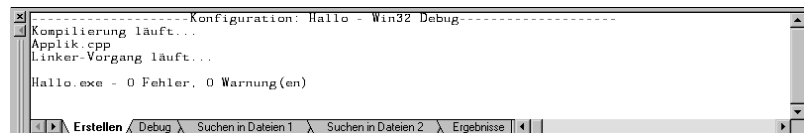


Wenn Sie eine bestimmte Ressource erstellen oder zur Bearbeitung öffnen, wird automatisch der zu dem jeweiligen Ressourcentyp passende Ressourceneditor geladen (siehe Abbildung 1.4).

Je nach Art und Konfiguration des aufgerufenen Ressourceneditors wird die Menüstruktur des Ressourceneditors in die Menüleiste des Visual Studios integriert (für den Symboleditor beispielsweise die Popup-Menüs EINFÜGEN und BILD) und es werden die zugehörigen Werkzeugleisten angezeigt.

Das Ausgabefenster – der Compiler meldet sich

Abbildung 1.5:
Das Ausgabe-
fenster



Das Ausgabefenster wird von verschiedenen integrierten Tools zur Ausgabe von Meldungen verwendet. Für die verschiedenen Tools werden jeweils eigene Seiten verwendet. Die Ausgaben des Compilers und des Linkers erscheinen beispielsweise auf der Seite ERSTELLEN, die Debug-Ausgaben werden auf die Seite DEBUG umgeleitet, etc.

Per Voreinstellung ist das Ausgabefenster in den unteren Rahmen des Visual Studios integriert.

Die Debug-Fenster – Status eines Programms kontrollieren

Der Debugger verfügt über eine ganze Reihe von Ausgabefenstern, die Sie bei der Überwachung des debuggten Programms unterstützen und in denen Sie jeweils verschiedene Informationen zum Status Ihres Programms abfragen können.

Die einzelnen Fenster können über den Befehl ANSICHT/DEBUG-FENSTER aufgerufen werden und werden im Kapitel 4 besprochen. Die Menü-Befehle des Debuggers finden Sie im Popup-Menü DEBUG, das kurz nach Beginn einer Debug-Sitzung (Befehl ERSTELLEN/DEBUG STARTEN) eingeblendet wird.

Auch der Quelltexteditor arbeitet mit dem Debugger zusammen. Während einer Debug-Sitzung können Sie beispielsweise den Inhalt von Variablen abfragen, indem Sie den Mauszeiger einfach auf ein Vorkommen des entsprechenden Variablennamens bewegen. (Voraussetzung ist, daß die Variable in dem Gültigkeitsbereich, in dem das Programm angehalten wurde, gültig ist.)

Die Assistentenleiste



Abbildung 1.6:
Die Assistentenleiste

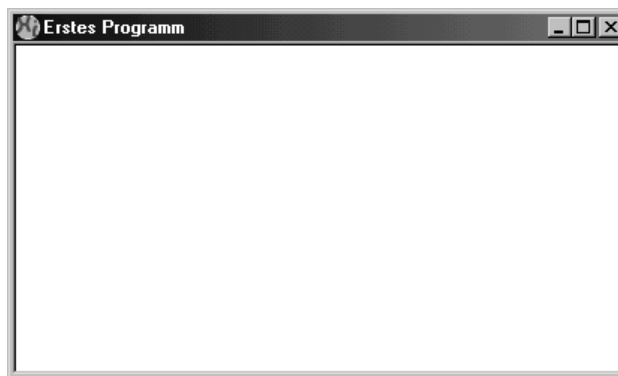
Die Assistentenleiste gehört zu den Symbolleisten des Developer Studios. Sie besteht aus drei Listenfeldern, die der Auswahl einer Klasse oder einer Klassenmethode dienen, und einer Befehlsliste (Pfeilsymbol am rechten Ende der Leiste). Mit den Befehlen dieser Liste können Sie zur Deklaration oder Definition der ausgewählten Klassenmethode oder Klasse springen, Klassen neu anlegen oder Klassen um Methoden erweitern.

1. Lassen Sie die Assistentenleiste anzeigen. Die Assistentenleiste aktivieren Sie über das Kontextmenü des Developer Studios (klicken Sie beispielsweise in den Hintergrund einer der angezeigten Symbolleisten).
2. Markieren Sie eine Klasse. Wählen Sie die Klasse im ersten Listenfeld (C++-Klasse der Assistentenleiste) aus.
3. Markieren Sie eine Methode. Wählen Sie eine Methode im dritten Listenfeld (C++-Elemente der Assistentenleiste) aus.
4. Rufen Sie einen passenden Befehl auf. Klicken Sie zum Aufruf der Befehlsliste auf das rechts gelegene Pfeilsymbol.

1.2 Die grundlegenden Arbeitsschritte

Der Umstieg von einem Compiler auf einen anderen ist stets mit einer gewissen Eingewöhnungszeit verbunden – die sich um so länger hinzieht, je deutlicher sich die Arbeitsumgebung des bis dato verwendeten Compilers von der neuen Arbeitsumgebung unterscheidet. Um all denjenigen Lesern, die zum ersten Mal mit dem Visual-C++-Compiler arbeiten, den Einstieg zu erleichtern, sollen in diesem Abschnitt anhand der Erstellung eines kleinen Beispielprogramms die grundlegenden Arbeitsschritte und die Einbindung der verschiedenen Entwicklertools des Visual Studios in den Erstellungsprozeß demonstriert werden.

Abbildung 1.7:
Das Fenster des
Beispiel-
programms



Bei dem im folgenden zu erstellenden Beispielprogramm handelt es sich um ein Windows-Programm, das aus wenig mehr als einem Hauptfenster besteht und mit Hilfe der MFC (aber ohne Assistentenunterstützung, siehe Kapitel 3) implementiert wird.

1. Schritt: Projekt anlegen

Die Arbeit an einem neuen Programm beginnt immer mit dem Anlegen eines Projekts. In dem Projekt werden die verschiedenen Dateien des Programms (Quelltextdateien (.cpp), Header-Dateien (.h), Ressourcedateien (.res, etc.) u. a.) verwaltet. Über die Projekteinstellungen wird festgelegt, wie die Dateien des Projekts zu einer ausführbaren Datei kompiliert und gelinkt werden sollen. Für jede ausführbare Datei (.exe oder .dll) benötigt man ein eigenes Projekt. Projekte selbst werden in Arbeitsbereichen verwaltet – was vor allem dann interessant ist, wenn zu einem Programm mehrere ausführbare Dateien gehören.

Als Ausgangspunkt für das Beispielprogramm werden zuerst ein Anwendungsbereich und ein leeres Projekt erstellt:

1. Rufen Sie den Befehl DATEI/NEU auf, und markieren Sie auf der Seite Projekte den Eintrag WIN32-ANWENDUNG.
2. Geben Sie auf der rechten Seite des Dialogfensters einen Titel für das Projekt ein (bspw. *Hallo*), wählen Sie das übergeordnete Verzeichnis aus, und lassen Sie einen zugehörigen, neuen Arbeitsbereich erstellen.
3. Lassen Sie von dem Assistenten ein leeres Projekt erstellen und wechseln Sie dann in die DATEIEN-Ansicht des Arbeitsbereichsfensters.
4. Über den Menübefehl PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU (oder alternativ DATEI/NEU) legen Sie innerhalb des Projekts eine C++-Quellcodedatei (namens *Applik.cpp*) und eine C/C++-Header-Datei (namens *Applik.h*) an.

2. Schritt: MFC einbinden

Standardmäßig werden Win32-Anwendungs-Projekte ohne Einbindung der MFC erstellt. Da aber kein API-Programm, sondern ein MFC-Programm erstellt werden soll, muß für die Einbindung der MFC gesorgt werden:

1. Rufen Sie das Dialogfenster PROJEKTEINSTELLUNGEN auf (Befehl PROJEKT/EINSTELLUNGEN), und wählen Sie im Feld MICROSOFT FOUNDATION CLASSES auf der Seite ALLGEMEIN eine der Optionen zur Verwendung der MFC aus. (Links im Dialogfeld muß der Projektknoten HALLO ausgewählt sein.)
2. Führen Sie diese Einstellung für die Debug- und die Release-Version durch.

3. Schritt: Quellcode aufsetzen

Das Grundgerüst der Anwendung besteht aus einem Anwendungs- und einem Rahmenfensterobjekt.

Das Anwendungsobjekt. Zuerst müssen die benötigten Header-Dateien per Include-Anweisung eingebunden und ein Objekt für die Anwendung erstellt werden.

1. Doppelklicken Sie im Arbeitsbereichfenster auf den Knoten der Header-Datei (*Applik.h*).
2. Nehmen Sie per Include-Anweisung die Deklarationen der MFC-Klassen auf.

3. Leiten Sie eine eigene Anwendungsklasse von der MFC-Klasse `CWinApp` ab. Überschreiben Sie in dieser Klasse die Methode `InitInstance()`.

```
// Header-Datei Applik.h
#include <afxwin.h>

class CMyApp : public CWinApp {
public:
    virtual BOOL InitInstance();
};
```

4. In der Quelltextdatei müssen Sie ein Objekt Ihrer Anwendungsklasse erzeugen und für die Implementierung der überschriebenen Elementfunktion `InitInstance()` sorgen.

```
// Quelltextdatei Applik.cpp
#include "Applik.h"

// Anwendungs-Objekt erzeugen
CMyApp Anwendung;

// Anwendung initialisieren
BOOL CMyApp::InitInstance()
{
    return TRUE;
}
```

Das Hauptfenster. Der nächste Schritt besteht darin, ein Fenster als Schnittstelle zum Anwender einzurichten. Wir begnügen uns hier mit einem Rahmenfenster (ohne untergeordnete View-Fenster).

1. In der Header-Datei wird von `CFrameWnd` eine eigene Rahmenfensterklasse abgeleitet und ein Konstruktor deklariert:

```
// Header-Datei Applik.h
#include <afxwin.h>

class CRahmenfenster : public CFrameWnd {
public:
    CRahmenfenster();
};
```

2. In der Quelltextdatei wird die Definition des Konstruktors aufgesetzt und in der `CMyApp`-Methode `InitInstance()` für die Erzeugung und Anzeige des Fensters gesorgt.

Im Konstruktor wird die Methode `Create()` aufgerufen, die für die Anmeldung und Einrichtung des Fensters unter Windows sorgt:

```
CRahmenfenster::CRahmenfenster()
{
    LPCTSTR classname = NULL;

    // Fenster erzeugen
    Create(classname, // 0 für MFC-Vorgabe
           "Erstes Programm", // Titel
           WS_OVERLAPPEDWINDOW, // Stil
           rectDefault, // keine def. Groesse
           0, // kein übergeordn. Fenster
```

```

        0,                // kein Menü
        0,                // kein erw. Stil
        0);              // kein Doc/View
    }

```

3. In der CMyApp-Methode `InitInstance()` wird der Konstruktor der Rahmenfensterklasse aufgerufen und somit das Rahmenfensterobjekt erzeugt.

Damit die Anwendung zusammen mit dem Hauptfenster geschlossen wird, muß der zurückgelieferte Zeiger an das CMyApp-Datenelement `m_pMainWnd` übergeben werden (beachten Sie, daß damit eine Umwandlung in ein Objekt der Klasse `CWinThread` verbunden ist). Zum Anzeigen des Fenster wird die Methode `ShowWindow()` aufgerufen.

```

// Anwendung initialisieren
BOOL CMyApp::InitInstance()
{
    // Rahmenfenster-Objekt erzeugen und Fenster anzeigen
    CRahmenfenster *pMainWnd = new CRahmenfenster;

    m_pMainWnd = pMainWnd;
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}

```

4. Schritt: Ressource bearbeiten

Unter Windows verfügt üblicherweise jedes Programm über ein Symbol (Ikone) zur grafischen Präsentation. Windows verwendet dieses Symbol in verschiedenen Kontexten (Titel des Hauptfensters, Anzeige in Explorer, Task-Leiste) in jeweils verschiedenen Größen. Aufgabe jeder ordentlichen Windows-Anwendung ist es daher, ein entsprechendes Symbol bereitzustellen.

Erstellen Sie zuerst die Bitmap für das Anwendungssymbol

1. Über den Menübefehl PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU legen Sie innerhalb des Projekts ein Ressourcenskript (namens *Applik.rc*) an.
2. In die Ressourcenskriptdatei fügen Sie über den Befehl EINFÜGEN aus dem Kontextmenü des *Applik.rc*-Knotens eine neue Icon-Ressource ein.
3. Zeichnen Sie Ihr Symbol.
4. Speichern Sie die Ressource und die Ressourcenskriptdatei. Der Ressourceneditor legt daraufhin automatisch eine *.ico*-Datei für das Symbol und eine *resource.h*-Datei mit der Deklaration der Ressourcen-IDs an.

Danach wird das Symbol mit der Anwendung verbunden

1. Machen Sie die Ressourcen-IDs in Ihrem Programm bekannt, indem Sie eine entsprechende Include-Anweisung in die Header-Datei *Applik.h* aufnehmen. (Wenn Sie möchten, können Sie die Header-Datei *resource.h* zudem über den Befehl PROJEKT/DEM-PROJEKT HINZUFÜGEN/DATEIEN in Ihre Projektverwaltung aufnehmen.)
2. Im Konstruktor der Rahmenfensterklasse muß das Symbol mit dem Fenster verbunden werden. Zu diesem Zweck wird zuerst
 - die Icon-Ressource geladen (`AfxFindResourceHandle` und `LoadIcon`),
 - dann wird mit Hilfe der Funktion `PreCreateWindow` die Windows-Klasse abgefragt, die das MFC-Gerüst bereits für das Hauptfenster der Anwendung vorgesehen hat,
 - schließlich wird eine Kopie dieser Fensterklasse erzeugt, mit dem Icon verbunden und mit Hilfe der Funktion `AfxRegisterWndClass` registriert.

Beim Aufruf der Funktion `Create()` wird das Rahmenfenster nunmehr nach Maßgabe dieser Fensterklasse – und somit auch mit dem für diese Fensterklasse definierten Symbol – erzeugt.

```
CRahmenfenster::CRahmenfenster()
{
    LPCTSTR classname = NULL;

    HINSTANCE hInst = AfxFindResourceHandle(IDI_ICON1,
                                           RT_GROUP_ICON);
    HICON hIcon = ::LoadIcon(hInst,
                             MAKEINTRESOURCE(IDI_ICON1));
    if(hIcon != NULL)
    {
        CREATESTRUCT cs;
        memset(&cs, 0, sizeof(CREATESTRUCT));
        PreCreateWindow(cs);

        WNDCLASS wndcls;
        if (cs.lpszClass != NULL &&
            GetClassInfo(AfxGetInstanceHandle(), cs.lpszClass,
                        &wndcls) )
        {
            // register a very similar WNDCLASS
            classname = AfxRegisterWndClass(wndcls.style,
                                           wndcls.hCursor,
                                           wndcls.hbrBackground, hIcon);
        }
    }

    // Fenster erzeugen
    Create(classname, "Erstes Programm (ohne Doc/View)");
}
```

Über weitere Möglichkeiten zur Konfiguration eines Fensters können Sie sich unter den Stichwörtern `PreCreateWindow`, `CREATESTRUCT`, `LoadFrame`, `SetClassLong` in der Online-Hilfe informieren.



5. Schritt: Kompilieren

Kompilieren Sie das Projekt, und führen Sie das Programm aus.

1. Rufen Sie dazu einfach den Befehl ERSTELLEN/AUSFÜHREN VON... (`Strg` + `F5`) auf.

6. Schritt: Debuggen

Sollten Fehler bei der Ausführung des Programms auftreten, versuchen Sie es zu debuggen (siehe Kapitel 4). Zur Ausführung des Programms unter Kontrolle des Debuggers rufen Sie den Befehl ERSTELLEN/DEBUG STARTEN/AUSFÜHREN auf (`F5`).

7. Schritt: Fertige Version erstellen

Ist das Programm fertig und läuft fehlerfrei, sollten Sie eine Release-Version erstellen.

Standardmäßig werden vom Visual Studio für jedes Projekt zwei Konfigurationen angelegt: eine Debug- und eine Release-Version. Die Debug-Version ist so eingestellt, daß bei der Kompilation spezielle Debug-Informationen mit in die Zieldatei (.exe, .dll) aufgenommen werden. Bei der Release-Version wird dagegen auf die Debug-Informationen verzichtet. Statt dessen wird der Code vom Compiler optimiert.

1. Rufen Sie den Befehl ERSTELLEN/AKTIVE KONFIGURATION FESTLEGEN auf, und wählen Sie im erscheinenden Dialogfeld die Release-Konfiguration.
2. Lassen Sie das Projekt danach neu erstellen. Die resultierende EXE-Datei wird standardmäßig in dem Unterverzeichnis *Release* abgelegt.

1.3 Kommandozeilen-Hilfsmittel

Obwohl Visual Studio die Schnittstelle für den Zugriff auf die Features von Visual C++ bildet, können der C/C++-Compiler und andere Komponenten auch über die Kommandozeile gesteuert werden. Diese Vorgehensweise ist bisweilen mit einem geringeren Aufwand verbunden als der Weg über die integrierte Entwicklungsumgebung, so z.B., wenn einfache Testprogramme kompiliert werden sollen. Dann ist

man mit der Eingabe einiger einfacher Kommandozeilenanweisungen wie z.B. `cl myprog.c` wesentlich schneller am Ziel und kann sich das Einrichten und Konfigurieren eines Visual-C++-Projekts, das Hinzufügen von Dateien zu dem Projekt sowie das Kompilieren und Ausführen in der Entwicklungsumgebung sparen.

Wann jedoch würden Sie Kommandozeilen-Hilfsmittel verwenden? Ich möchte Ihnen dazu gerne einige Anregungen geben: Ist Ihr Projekt derart komplex, daß eine Make-Datei erforderlich ist, verwenden Sie die Entwicklungsumgebung mit der integrierten Projektverwaltung. Möchten Sie Ihr Programm interaktiv debuggen oder verfügt Ihr Programm über eine umfangreiche Ressourcendatei, sollten Sie ebenfalls die Entwicklungsumgebung verwenden.

Wenn Sie jedoch lediglich ein Beispiel mit zehn Zeilen aus einem Buch eingeben und testen, sind die Kommandozeilen-Hilfsmittel völlig ausreichend. Viele Beispiele dieses Buches können bequemer über die Kommandozeile kompiliert werden (natürlich können diese auch in ein Visual-C++-Projekt geladen werden).

Die Ausführung einiger Kommandozeilen-Hilfsmittel ist von dem Pfad der entsprechenden Visual-C++-Verzeichnisse sowie von korrekt eingerichteten Umgebungsvariablen abhängig. Verwenden Sie Windows NT als Entwicklungsplattform, bietet Ihnen das Installationsprogramm die Registrierung der Umgebungsvariablen an, so daß diese automatisch in dem Fenster der Eingabeaufforderung aufgeführt werden. Arbeiten Sie mit Windows 95, müssen Sie die Batch-Datei `VCVARS32.BAT` ausführen lassen (Verzeichnis `PROGRAMME\DEVSTUDIO\VC\BIN`), um die Variablen zu registrieren und mit den Kommandozeilen-Hilfsmitteln arbeiten zu können. Beachten Sie bitte, daß Sie möglicherweise die Umgebungswerte für das DOS-Fenster vergrößern müssen (selektieren Sie dazu den Eintrag `EIGENSCHAFTEN` aus dem Systemmenü des DOS-Fensters, und öffnen Sie dort das Register `SPEICHER`), bevor `VCVARS32.BAT` erfolgreich ausgeführt werden kann.

Der C/C++-Compiler

Der Visual-C++-Compiler wird mit der Anweisung `cl` über die Kommandozeile aufgerufen. Werden dem Compiler lediglich der Name der Quelldatei und keine weiteren Parameter übergeben, kompiliert er die Datei und ruft anschließend den Linker auf, um die ausführbare Datei zu erstellen.

Wenn Sie in der Kommandozeile die Namen von Objektdateien oder Bibliothekdateien angeben, werden diese dem Linker übergeben.

```
CL HELLO.C MYFUNC.OBJ MYLIB.LIB
```

Geben Sie diese Zeile ein, kompiliert der Visual-C++-Compiler *HELLO.C* und ruft anschließend den Linker mit den Dateien *HELLO.OBJ* und *MYFUNC.OBJ* auf. Außerdem übergibt der Compiler den Namen der Bibliotheksdatei *MYLIB.LIB* an den Linker, der die darin enthaltene Bibliothek und alle Standardbibliotheken verwenden wird, um nach Bibliotheksfunktionen zu suchen.

Möchten Sie lediglich eine Datei kompilieren, ohne daraus eine ausführbare Datei erzeugen zu lassen, verwenden Sie die Option */c*:

```
CL /C HELLO.C
```

Beachten Sie bitte, daß Sie sowohl den Schrägstrich als auch den Bindestrich verwenden können, um Kommandozeilen-Optionen anzugeben.

Weitere nützliche Optionen sind:

- ➡ */MT* (mit der Multithread-Version der Laufzeitbibliothek binden)
- ➡ */MD* (mit der DLL-Version der Laufzeitbibliothek binden)
- ➡ */LD* (erstellt eine DLL)
- ➡ Weitere Optionen können Sie durch Eingabe von *cl /?* anzeigen lassen

Möchten Sie komplexe Optionen bestimmen, sollten Sie von Visual Studio aus kompilieren.

Die von dem Visual-C++-Compiler generierten Objektdateien werden im COFF-Dateiformat abgelegt (Common Object File Format).

Der Linker

Der Linker *LINK.EXE* ist ein Programm, dem Dateien im COFF-Format, 32-Bit-Objektmodulformat (OMF), Bibliotheksdateien und andere Dateien übergeben werden können. Er erzeugt daraus ausführbare Win32-Dateien oder DLLs. Die folgende Zeile zeigt einen einfachen Aufruf des Linkers mit einer Objektdatei:

```
LINK HELLO.OBJ
```

Der Linker akzeptiert viele Kommandozeilen-Optionen. Eine dieser Optionen ist */subsystem*, die den Typ der zu erzeugenden ausführbaren Datei angibt. Bestimmen Sie beispielsweise die Option */subsystem:windows*, wird eine ausführbare Windows-Datei erstellt. Gewöhnlich muß diese Option jedoch nicht angegeben werden. Die Voreinstellung des Linkers lautet entweder */subsystem:console*, wenn

die Objektdatei eine Definition der *Main*-Funktion (oder der Unicode-Zeichenversion *WMain*) enthält, oder */subsystem:windows*, wenn eine *WinMain*- oder *wWinMain*-Funktion vorhanden ist.

Möchten Sie eine DLL anstelle einer ausführbaren Datei erstellen, benutzen Sie die Option */DLL*. Diese Option wird automatisch verwendet, wenn der Linker von dem Compiler aufgerufen und diesem die Option */MD* übergeben wurde.

Andere Optionen steuern, wie die übergebenen Dateien bearbeitet werden, welche Standardbibliotheken verwendet werden sollen und bestimmen weiterhin den Typ und den Inhalt der Ausgabedateien. Zusätzlich zu den ausführbaren Dateien kann der Linker Debug-Dateien (wie z.B. Map-Dateien) erzeugen. Sie können außerdem bestimmen, ob die ausführbare Datei Debug-Informationen enthalten soll oder nicht.

Der Bibliothekmanager

Der Bibliothekmanager *LIB.EXE* wird zur Erstellung von Bibliotheken aus COFF-Objektdateien verwendet. Er kann außerdem zur Erzeugung von Exportdateien und Importbibliotheken für DLLs genutzt werden.

LIB.EXE wird gewöhnlich mit mehreren Objektdateinamen über die Kommandozeile aufgerufen. Das Programm verwendet den Namen der ersten Objektdatei als Dateiname für die Bibliothekdatei und erstellt (oder aktualisiert) eine Bibliothek, die aus den angegebenen Objektdateien besteht. Der Name der Ausgabedatei kann mit Hilfe der Option */OUT* überschrieben werden.

Mit der Option */REMOVE* können Sie später Objektdateien aus der Bibliothek entfernen. Die Option */EXTRACT* wird zum Extrahieren der Inhalte einer Objektdatei in eine gesonderte Datei verwendet.

Um mit dem Bibliothekmanager eine Ausgabedatei und eine Eingabebibliothek zu erstellen, verwenden Sie die Option */DEF*. Beachten Sie bitte, daß Sie diese Option nur gelegentlich nutzen werden, da der Linker die Exportdatei und Eingabebibliothek gewöhnlich automatisch erzeugt. Der Bibliothekmanager wird überwiegend in Situationen eingesetzt, die das Exportieren aus sowie das Importieren in derselben Bibliothek verlangen.

NMAKE

Das Hilfsmittel zur Programmüberprüfung, das kurz Make-Werkzeug genannt wird (*NMAKE.EXE*), überprüft die Abhängigkeiten in Make-Dateien und führt Befehle zur Programmgenerierung aus. Eine einfache Make-Datei könnte wie folgt aufgebaut sein:

```
test.exe: test.obj  
    link test.obj  
test.obj: test.c  
    cl /c test.c
```

In allen Make-Dateien überprüft das Make-Werkzeug die generierten Dateien in der Reihenfolge ihrer Abhängigkeiten und aktualisiert die Dateien, wenn diese älter als deren Abhängigkeiten sind.

Zusätzlich zu den Zieldateien und Abhängigkeiten können Make-Dateien weitere Features enthalten, wie z.B. Makros und Ableitungsregeln. Diese Features machen Make-Dateien zu leistungsfähigen und flexiblen Werkzeugen.

Wichtige Optionen von *NMAKE.EXE* sind */a* (alle Zieldateien uneingeschränkt erstellen), */n* (das Make-Werkzeug zeigt Befehle lediglich an und führt diese nicht aus) und */f* (bestimmt den Namen der Make-Datei).

Andere Kommandozeilen-Hilfsmittel

Weitere Kommandozeilen-Hilfsmittel, die mit Visual C++ ausgeliefert werden, sind *rc.exe*, *bscmake.exe*, *dumpbin.exe*, *aviedit.exe* und *editbin.exe*.

Der Ressource-Compiler *rc.exe* kompiliert Ressource-Dateien und bereitet diese für die Anbindung an die Objektdateien Ihres Projekts vor. Sie übergeben dem Ressource-Compiler in der Kommandozeile den Namen der Ressource-Datei (mit der Endung *.rc*) und optionale Schalter. Ihnen stehen die Schalter */d* (definiert ein Symbol für den Präprozessor), */fo* (spezifiziert den Namen der Ausgabedatei) und */v* (für die ausführliche Ausgabe) zur Verfügung. Sie erhalten eine vollständige Liste der Kommandozeilen-Optionen, indem Sie *rc.exe* mit der Option */?* ausführen lassen.

Das Browse-Informationswerkzeug *bscmake.exe* wird verwendet, um Browse-Informationsdateien (BSC) aus SBR-Dateien zu erzeugen, die wiederum während des Kompilierens erstellt werden. Visual Studio kann Browse-Informationsdateien anzeigen.

Der Binärdateimonitor *dumpbin.exe* zeigt Informationen über COFF-Objektdateien an.

aviedit.exe ist ein einfaches Programm zur Bearbeitung von AVI-Dateien. Verwenden Sie diese Anwendung, um eine Animationsdatei aus mehreren Bitmaps zu generieren.

Der Binärdatei-Editor *editbin.exe* wird verwendet, um bestimmte Eigenschaften von COFF-Objektdateien einzusehen und zu modifizieren.

Einige Optionen dieses Programms sind das Verändern der Basisadresse einer Datei sowie das Ändern der Standard-Heap-Größe und der Standard-Stack-Größe.

1.4 Neuerungen in der Version 6.0

Die 6.0-Version des Visual C++-Compilers wurde in mancherlei Hinsicht verbessert und mit einer Vielzahl zusätzlicher Optionen ausgestattet. Statt die neuen Optionen und Möglichkeiten einzeln aufzulisten, möchte ich Sie gezielt auf einige ausgesuchte Neuerungen hinweisen.

Compiler

- Der Compiler ist schneller geworden: laut Hersteller gegenüber Visual C++ 5.0 sogar um bis zu 30%.
- Mit dem Schlüsselwort `__forceinline` kann der Programmierer die Inline-Kompilierung von Funktionen erzwingen, sofern eine Inline-Kompilierung nur irgendwie möglich ist (siehe Online-Hilfe). Ausgesuchte »Funktionen« können damit beispielsweise effektiver vor Crackern geschützt werden.

Editor und Arbeitsbereich

- Der Quelltexteditor bietet eine interaktive Hilfe zur Anweisungsvervollständigung.
- Die Klassen-Ansicht basiert fortan nicht mehr auf den kompilierten Dateien, sondern zieht sich ihre Informationen direkt aus den Quelltexten. Die Anzeige wird daher während der Bearbeitung der Quelltexte ständig aktualisiert.

Debugger

- Der Programmierer kann nun während des Debuggens Code abändern und dann ohne erneute Kompilierung den geänderten Code austesten.
- Die Optionen zur Darstellung und Formatierung verschiedener Variablen (Varianten, GUIDs, Funktionszeiger etc.) wurden ausgeweitet und verbessert.

Assistenten

- Anwendungs- und Klassenassistent wurden ausgebaut.
- Neue Assistenten und Projekttypen sind hinzugekommen.

MFC

Die MFC wurde um eine Reihe von Klassen erweitert, insbesondere um Klassen für

- ➡ die Internet-Explorer-4.0-Steuerelemente
- ➡ eine HTML-View
- ➡ OLE DB-Klassen

Tools

- ➡ Die Enterprise-Edition wird mit dem »Visual Modeler« ausgeliefert. Der Visual Modeler ist ein CASE-Tool für OOD (objektorientiertes Design), verfügt im Vergleich zu professionellen Tools allerdings nur über einen sehr eingeschränkten Leistungsumfang.

Hilfe

- ➡ Die Online-Hilfe wurde ganz auf MSDN umgestellt.

Wenn Sie sich detailliert über die Neuerungen in der 6.0-Version des Visual C++-Compilers informieren wollen, schlagen Sie bitte in der Online-Hilfe (MSDN) unter dem Ordner zu Visual C++ nach.



1.5 Zusammenfassung

Visual Studio ist der Kern des Visual-C++-Entwicklungssystems. Es bildet eine grafische Schnittstelle zu den wichtigsten Tools der Anwendungsentwicklung, beispielsweise:

- ➡ der Projektverwaltung (Arbeitsbereichfenster und Menü PROJEKT)
- ➡ dem Quelltexteditor (Editorfenster und Menü BEARBEITEN)
- ➡ den Ressourceneditoren (Editorfenster und verschiedene Menüs)
- ➡ dem Compiler (Ausgabefenster und Menü ERSTELLEN)
- ➡ dem Debugger (Debug-Fenster und Menü DEBUG)

Eine große Anzahl verschiedener Visual-C++-Komponenten können auch über die Kommandozeile aufgerufen werden. Ein einfaches Programm mit der Bezeichnung HELLO kann beispielsweise mit der Anweisung `cl hello.c` kompiliert werden. Weitere Kommandozeilen-Hilfsmittel sind der Linker, der Bibliothekmanager und das Hilfsmittel zur Programmüberprüfung (Make-Werkzeug).

Arbeitsbereiche und Projekte

Kapitel 2

Größere Programme sind zumeist in mehrere Module aufgeteilt, d.h. Quelltexteinheiten, die jeweils für sich kompiliert werden und dann erst vom Linker zu einer ausführbaren Datei zusammengebunden werden. Die Erstellung eines solchen Programms kann sehr aufwendig sein:

- ➡ Die einzelnen Quelltexteinheiten müssen einzeln kompiliert werden,
- ➡ unter Umständen bedürfen die einzelnen Einheiten individueller Compiler-Einstellungen,
- ➡ verschiedene Quelltexte bedürfen unterschiedlicher Compiler (z.B. Ressourcen-Compiler für Windows-Ressourcen),
- ➡ beim Linken müssen alle einzubindenden Objekdateien angegeben werden etc.

Um sich die wiederholte Tipparbeit zu sparen, arbeiten die meisten Compiler mit Make-Dateien, in denen die Einstellungen festgehalten werden. Die Projektverwaltung von Visual C++ geht noch darüber hinaus, indem Ihnen die Erstellung dieser Make-Dateien abgenommen wird. Sämtliche Informationen zu dem Projekt werden in der .dsp-Datei des Projekts abgespeichert.

*Projektdateien
enthalten die
Informationen
zur Programm-
erstellung*

Der Aufbau des Projekts wird im Arbeitsbereichfenster, Seite DATEIEN, grafisch angezeigt. Über die Befehle in den Menüs DATEI und PROJEKT sowie die Kontextmenüs der Seite DATEIEN des Arbeitsbereichfensters können Sie Ihre Projekte bearbeiten, Dateien hinzufügen oder entfernen, die Kompilierung beeinflussen, das Projekt konfigurieren, etc. und die entsprechenden Änderungen werden automatisch in der .dsp-Datei festgehalten und zur Erstellung des Projekts herangezogen. Die

Erstellung und Bearbeitung umfangreicher Programme gestaltet sich damit ebenso einfach wie die Erstellung einfacher Programme aus einem Modul.

2.1 Projekte und Arbeitsbereiche

In Visual C++ werden Projekte immer in Arbeitsbereichen verwaltet. Dies zählt sich vor allem dann aus, wenn mehrere zusammengehörende Projekte in einem Arbeitsbereich untergebracht werden (beispielsweise die im Laufe der Zeit immer weiterentwickelten Versionen eines Programms oder Programme, die aus einer EXE mit mehreren DLLs bestehen).

Arbeitsbereiche mit einem Projekt

Wenn es Ihnen lediglich darum geht, ein Projekt für ein Programm zu erstellen (und Sie nicht beabsichtigen, später weitere gleichberechtigte Projekte in den Arbeitsbereich aufzunehmen), brauchen Sie nicht zwischen Projekt und Arbeitsbereich zu unterscheiden:

Legen Sie den Arbeitsbereich implizit im Zuge der Projekterstellung an, indem Sie auf der Seite PROJEKTE im Dialogfeld NEU (Aufruf über DATEI/NEU) die Option NEUEN ARBEITSBEREICH ERSTELLEN aktivieren.

Arbeitsbereich und Projekt tragen dann den gleichen Namen und teilen sich auf der Festplatte ein gemeinsames Verzeichnis.

Arbeitsbereiche mit mehreren Projekten

Wenn Sie in einem Arbeitsbereich mehrere gleichberechtigte Projekte verwalten wollen, sollten Sie vor dem Anlegen des ersten Projekts zuerst einen leeren Arbeitsbereich einrichten:

1. Legen Sie zuerst auf der Seite ARBEITSBEREICH im Dialogfeld NEU (Aufruf über DATEI/NEU) einen leeren Arbeitsbereich an.
2. Danach legen Sie auf der Seite PROJEKTE des Dialogfelds das oder die Projekte an (wobei Sie darauf achten, daß die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH markiert ist).

Arbeitsbereich und Projekte haben dann unterschiedliche Namen, und die Verzeichnisse der Projekte werden standardmäßig dem Verzeichnis des Arbeitsbereichs untergeordnet.

Ein Projekt aktivieren Von den verschiedenen Projekten eines Arbeitsbereichs ist immer nur eines aktiv (durch Fettschrift im Arbeitsbereichfenster hervorgehoben). Auf dieses Projekt beziehen sich die Menübefehle (beispielsweise zur Kompilation und Erstellung).

Über den Menübefehl PROJEKT/AKTIVES PROJEKT FESTLEGEN können Sie die verschiedenen Projekte eines Arbeitsbereichs aktivieren.

Arbeitsbereiche konfigurieren

Um die allgemeinen Einstellungen für die Arbeitsbereiche festzulegen, rufen Sie den Befehl EXTRAS/OPTIONEN auf, und wechseln Sie zur Seite ARBEITSBEREICH. Hier können Sie beispielsweise festlegen,

- ➡ welche Fenster beim Öffnen automatisch in den Rahmen integriert werden sollen,
- ➡ ob die bei der letzten Sitzung geöffneten Fenster automatisch mit dem Arbeitsbereich geöffnet werden sollen oder
- ➡ ob und wie viele der zuletzt bearbeiteten Dateien und Arbeitsbereiche im Menü Datei (entweder direkt oder in Untermenüs) aufgelistet werden sollen.

2.2 Das Arbeiten mit Unterprojekten

Unterprojekte verwendet man üblicherweise dann, wenn die Zielfeile eines Projekts B als Quelldatei eines anderen Projekts A dienen soll. In solchen Fällen würde man Projekt B als Unterprojekt von Projekt A einrichten.

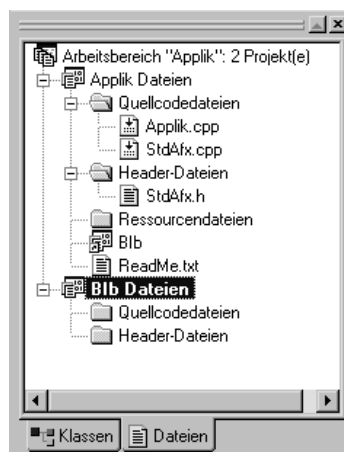


Abbildung 2.1:
Projekt mit
Unterprojekt

Ein sinnvolles Beispiel für die Verwendung von Unterprojekten sind statische Bibliotheken. Angenommen, Sie haben für ein größeres Kalkulationsprogramm eine umfangreiche Sammlung von Funktionen für mathematische Berechnungen zusammengestellt, die Sie zur besseren Wiederverwertung in Form einer statischen Bibliothek (Extension .lib) zur Verfügung stellen wollen. In einem solchen Fall bietet es sich an, die statische Bibliothek als Unterprojekt des Kalkulationsprogramms (oder anderer Programme, die die Funktionen der Bibliothek nutzen sollen) einzurichten.

Beispiel Um beispielsweise eine statische Bibliothek »Bib« als Unterprojekt zu einem Hauptprogramm »Applik« einzurichten, kann man wie folgt vorgehen:

1. Erstellen Sie über den Befehl DATEI/NEU ein ganz normales Win32-Projekt (hier *Applik*), vgl. Kapitel 1.2.
2. Erstellen Sie ein abhängiges Unterprojekt (hier *Bib*). Rufen Sie den Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU auf, wechseln Sie auf die Seite PROJEKTE, und nehmen Sie dort folgende Einstellungen vor:
 - Wählen Sie WIN32-BIBLIOTHEK (STATISCHE) als Zieltyp aus.
 - Geben Sie einen Projektnamen an (hier *Bib*).
 - Aktivieren Sie die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH.
 - Aktivieren Sie die Option ABHÄNGIGKEIT VON und wählen Sie *Applik* als übergeordnetes Projekt aus.
3. Setzen Sie die Quelltexte für beide Projekte auf.
4. Stellen Sie die nötigen externen Abhängigkeiten her. Nehmen wir an, daß im Quelltext von *Applik.cpp* auf eine in *Bib.cpp* definierte Funktion zugegriffen werden soll. Dafür, daß die Definition dieser Funktion im Programm *Applik* zur Verfügung steht, sorgt die Einrichtung von *Bib* als Unterprojekt. Allerdings muß die Funktion noch mit Hilfe einer Deklaration im Modul *Applik.cpp* bekanntgemacht werden. Nehmen Sie daher eine Include-Anweisung zur Einbindung der Header-Datei *Bib.h* in *Applik.h* auf.
5. Erstellen Sie das übergeordnete Projekt. Danach werden die externen Abhängigkeiten im Arbeitsbereichfenster angezeigt. (Die angezeigten Abhängigkeiten beruhen immer auf dem letzten Erstellungsprozeß.)

Bei der Erstellung eines übergeordneten Projekts werden zuerst alle eingerichteten Unterprojekte erstellt.

Um Unterprojekte auszuschließen, ohne sie gleich aus dem Arbeitsbereich löschen zu müssen, rufen Sie den Befehl PROJEKTE/ABHÄNGIGKEITEN auf, und deaktivieren Sie das Kontrollkästchen des entsprechenden Unterprojekts.

2.3 Projekte erstellen und bearbeiten

Wegen der zentralen Bedeutung der Projektverwaltung für die Anwendungsentwicklung im Visual Studio finden Sie in der folgenden Tabelle noch einmal die wichtigsten Projektbefehle zusammengefaßt.

Aktion	Befehl
Neues Projekt anlegen	Rufen Sie den Befehl DATEI/NEU auf. Ist aktuell noch ein Arbeitsbereich geöffnet, können Sie in dem erscheinenden Dialogfeld festlegen, ob das Projekt dem aktuellen Arbeitsbereich hinzugefügt werden soll.
Bestehendes Projekt öffnen	Rufen Sie den Befehl DATEI/ARBEITSBEREICH ÖFFNEN auf. Legen Sie innerhalb des Arbeitsbereichs das aktive Projekt fest.
Aktives Projekt festlegen	Wählen Sie das Projekt über den Befehl PROJEKT/AKTIVES PROJEKT FESTLEGEN aus.
Projekt erstellen	Rufen Sie einen der Befehle ERSTELLEN/»PROJEKTNAME« ERSTELLEN oder ERSTELLEN/ALLES NEU ERSTELLEN auf, je nachdem, ob Sie das Projekt aktualisieren oder komplett neu kompilieren lassen wollen.
Projekt löschen	Markieren Sie den Projektknoten im Arbeitsbereichfenster, und rufen Sie den Befehl BEARBEITEN/LÖSCHEN auf (Entf).
Neue Datei hinzufügen	Rufen Sie den Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU auf.
Bestehende Datei hinzufügen	Rufen Sie den Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN auf. (Der Befehl DATEI/ÖFFNEN öffnet eine Datei ohne sie dem Projekt einzugliedern.)

Tabelle 2.1:
Projekte
bearbeiten

Aktion	Befehl
Datei in Editor laden	Doppelklicken Sie im Arbeitsbereichfenster auf den Knoten der Datei.
Datei löschen	Markieren Sie die Datei im Arbeitsbereichfenster, und rufen Sie den Befehl BEARBEITEN/LÖSCHEN auf (<u>Entf</u>).

2.4 Projekte konfigurieren

Die Konfiguration Ihrer Projekte erfolgt auf verschiedenen Ebenen.

- Beim Anlegen des Projekts wählen Sie bereits Plattform und Art der zu erstellenden Zielfeile aus (Seite PROJEKTE im Dialogfeld NEU).
- Durch Hinzufügen von Dateien (Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN) richten Sie die einzelnen Module des Projekts ein.
- Über das Dialogfeld PROJEKTEINSTELLUNGEN können Sie auf die Kompilation der einzelnen Quelldateien und die Erstellung des gesamten Projekts einwirken.

2.4.1 Das Dialogfeld Projekteinstellungen

Über den Befehl PROJEKT/EINSTELLUNGEN rufen Sie das Dialogfeld PROJEKTEINSTELLUNGEN auf. Alle Einstellungen, die Sie in diesem Fenster vornehmen, beziehen sich auf die Konfiguration (siehe unten), die im Feld EINSTELLUNGEN FÜR angezeigt wird, und auf den Knoten, der in der darunter gelegenen Baumansicht ausgewählt ist.

Tabelle 2.2:
Einstellungen für
Projekte

Registerseite	Beschreibung
Allgemein	Auf dieser Seite können Sie entscheiden, ob und wie die MFC eingebunden werden soll, und in welche Verzeichnisse die im Laufe des Erstellungsprozesses erzeugten temporären und binären Dateien geschrieben werden sollen.
Debug	Über diese Seite können Sie den Debugger konfigurieren (beispielsweise durch Angabe von Kommandozeilenargumenten zu dem zu debuggenden Programm). Interessant ist auch das Feld AUSFÜHRBARES PROGRAMM FÜR DEBUG-SITZUNG. Sie werden sich nun bestimmt fragen, wieso Sie in dieses Feld eine andere Datei eintragen sollten, als die des aktuellen Programms? Die Antwort ist sehr einfach: Sie können hier auch DLLs oder andere Komponenten angeben, die nicht direkt von der Kommandozeile aus aufgerufen werden können.

Registerseite	Beschreibung
	Verwenden Sie Visual C++ beispielsweise, um einen MAPI-Transport-Provider zu entwickeln, geben Sie den Namen des MAPI-Spooler <i>MAPISP32.EXE</i>, als ausführbare Datei an, da dieses Programm Ihre Transport-Provider-DLL lädt. Erstellen Sie ein OLE-Steuerelement (OCX-Datei), bestimmen Sie eine ActiveX-Steuerelement-Container-Applikation, wie z. B. die mit Visual C++ ausgelieferte Anwendung <i>TSTCON32.EXE</i>, als ausführbare Datei.
C/C++	Über diese Seite konfigurieren Sie den Compiler. Gehen Sie die verschiedenen Kategorien nacheinander durch, um die Übersetzung des Projekts durch den Compiler anzupassen. Die Umsetzung Ihrer Einstellungen in Compiler-Optionen wird Ihnen im Feld PROJEKT OPTIONEN angezeigt. Eine interessante Kategorieseite ist mit VORKOMPILIERTER HEADER bezeichnet. Dort geben Sie an, wie der Compiler vorkompilierte Header-Dateien (PCH) während des Kompilierens erstellen und verwenden soll. Das Verwenden vorkompilierter Header ist sehr wichtig, da auf diese Weise die Performance des Compilers erhöht wird. Die Kategorieseite bietet zwei wesentliche Optionen. Selektieren Sie VORKOMPILIERTER HEADER AUTOMATISCH VERWENDEN, sucht und erkennt der Compiler die Header in Ihrer Projektdatei und erzeugt daraus die entsprechenden vorkompilierten Header. Projekte, die von einem Anwendungs-Assistenten generiert wurden, verwenden jedoch keine automatisch vorkompilierten Header. Für solche Projekte müssen Sie die Erstellung und Verwendung vorkompilierter Header explizit vornehmen. Sie haben jedoch die Möglichkeit, einen vorkompilierten Header nur einmal erstellen zu lassen und allen anderen Dateien zuzuweisen. Dazu verwenden Sie eine beliebige Datei Ihres Projekts (z. B. <i>STDAFX.CPP</i>), aus der der Compiler den vorkompilierten Header erstellt. Alle anderen Quelltextdateien werden so eingerichtet, daß sie diesen vorkompilierten Header verwenden.
Linker	Über diese Seite konfigurieren Sie den Linker. Gehen Sie die verschiedenen Kategorien nacheinander durch, um die Bindung der Module zum ausführbaren Programm anzupassen. Die Umsetzung Ihrer Einstellungen in Linker-Optionen wird Ihnen im Feld PROJEKT OPTIONEN angezeigt.
Ressourcen	Über diese Seite konfigurieren Sie den Ressourcen-Compiler. Die Umsetzung Ihrer Einstellungen in Compiler-Optionen wird Ihnen im Feld PROJEKT OPTIONEN angezeigt.

Registerseite	Beschreibung
Midl	Optionen für den MIDL-Compiler.
Browse-Informationen	Optionen zur Erstellung von Informationen für den Browser.
Benutzerdefiniertes Erstellen	Hier haben Sie die Möglichkeit, eigene Befehle für die Erstellung der Zielformatvorlage vorzugeben (beispielsweise das Anlegen einzelner Sicherungsdateien oder das Aufrufen externer Tools). Enthält Ihr Projekt beispielsweise eine Datei zur grammatikalischen Spezifizierung, die mit <i>GRAM.Y</i> bezeichnet ist und von dem <i>yacc</i>-Parser bearbeitet werden muß, möchten Sie möglicherweise ein benutzerdefiniertes Verfahren für diese Datei definieren.
Linker-Vorstufe	Hier können Sie Befehle eingeben, die vor Aufruf des Linkers ausgeführt werden.
Bearbeiten nach dem Erstellen	Hier können Sie Befehle eingeben, die nach Abschluß des Erstellungsvorgangs ausgeführt werden.

2.4.2 Arbeiten mit Konfigurationen

Damit Sie nicht ständig im Dialogfeld PROJEKTEINSTELLUNGEN von Seite zu Seite springen müssen, um die Projekterstellung an Ihre augenblicklichen Bedürfnisse anzupassen, haben Sie die Möglichkeit, alle im Dialogfeld PROJEKTEINSTELLUNGEN vorgenommenen Einstellungen als »Konfiguration« abzuspeichern.

Indem Sie sich verschiedene Konfigurationen für Standardaufgaben anlegen, brauchen Sie statt der Anpassung der Optionen nur noch die gewünschte Konfiguration auszuwählen.

- Konfigurationen einrichten*
1. Legen Sie eine neue Konfiguration an. Rufen Sie dazu den Befehl ERSTELLEN/KONFIGURATIONEN auf, und klicken Sie in dem erscheinenden Dialogfeld auf den Schalter HINZUFÜGEN.
 2. Geben Sie einen Namen für die neue Konfiguration an. Zusätzlich können Sie noch eine bereits bestehende Konfiguration auswählen, auf deren Einstellungen Sie aufbauen wollen.
 3. Schicken Sie die Dialogfelder ab.
 4. Passen Sie die neue Konfiguration an. Rufen Sie dazu den Befehl PROJEKT/EINSTELLUNGEN auf, und wählen Sie im Feld EINSTELLUNGEN für Ihre neue Konfiguration aus. Überarbeiten Sie dann die Optionen.

Standardmäßig wird jedes Projekt mit den beiden Konfigurationen Win32 Release (erzeugt optimierten Code ohne Debug-Informationen) und Win32 Debug (erzeugt Debug-Informationen) ausgestattet.



2.5 Zusammenfassung

Ein Arbeitsbereich kann ein oder mehrere Hauptprojekt(e) enthalten, die wiederum aus mehreren untergeordneten Projekten bestehen können. Visual Studio führt zu jedem Projekt verschiedene Konfigurationen. Eine Projekt-Konfiguration setzt sich aus Einstellungen zusammen, die zur Erstellung des Projekts verwendet werden. Erstellen Sie beispielsweise mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt, erzeugt der Assistent zwei Konfigurationen: eine für den Debug-Vorgang und eine für den Vorgang des Kompilierens.

Die Projekteinstellungen können nach Aufruf des Befehls EINSTELLUNGEN aus dem Menü PROJEKT modifiziert werden. Der Dialog PROJEKT-EINSTELLUNGEN ist ein komplexer Dialog mit mehreren Eigenschaftenseiten und verschiedenen untergeordneten Seiten. Die vorgenommenen Einstellungen beziehen sich jeweils auf den ausgewählten Knoten (Projekt oder Datei) sowie die ausgewählte Konfiguration.

Die Assistenten

Kapitel

3

Statt Ihre Programme von Grund auf selbst zu schreiben, können Sie für die am häufigsten benötigten Anwendungstypen sogenannte Assistenten zu Hilfe nehmen. Bei diesen Assistenten handelt es sich um dienstbare Geister, die als einzige Schnittstelle zum Anwender eines oder mehrere Dialogfelder anzeigen, über die Sie auf die Arbeit des Assistenten einwirken können. Nach dem Abschicken Ihrer Angaben erstellt der Assistent für Sie ein passendes Projekt, legt die ersten Dateien an und setzt ein Code-Gerüst auf, das Sie von den formelleren Programmieraufgaben (Anlegen eines Doc/View-Gerüsts für MFC-Anwendungen oder MFC-DLLs, Einrichten einer Datenbankankbindung oder Typlibibliothek, Registrierung für ActiveX-Steuererelemente, etc.) befreit, so daß Sie gleich mit der kreativen Arbeit beginnen können.

Projekte, die mit einem der MFC-Anwendungsassistenten erstellt wurden, können darüber hinaus mit dem Klassenassistenten vielfach weiterverarbeitet werden.

Dieses Kapitel beschreibt kurz den MFC-Anwendungsassistenten und den Klassenassistenten und gibt Ihnen einen Überblick über die verschiedenen Projekttypen, die Ihnen im Dialogfeld DATEI/NEU angeboten werden.

Ein praktisches Beispiel für den Einsatz des MFC-Anwendungsassistenten und des Klassenassistenten finden Sie im Kapitel zur Erstellung eines MFC-Anwendungsgerüsts.

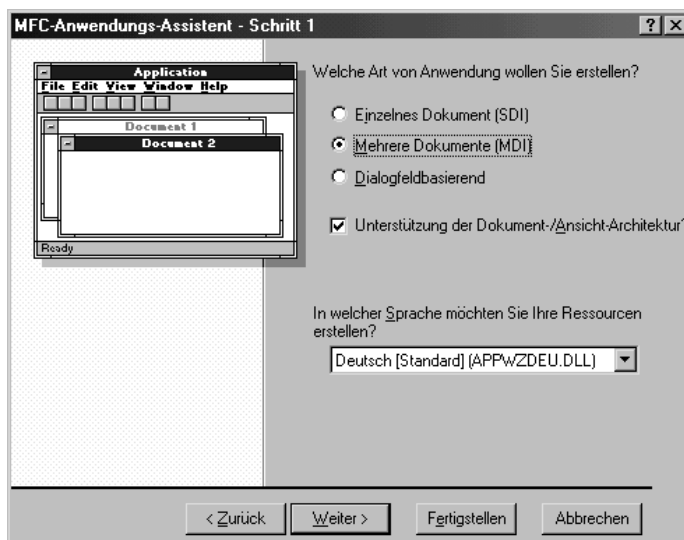




Grundsätzlich gilt, daß die Verwendung eines Assistenten Sie nicht von der Aufgabe befreit, sich mit dem generierten Code detailliert auseinanderzusetzen. Ohne ein gründliches Verständnis des erzeugten Anwendungsgerüsts ist jeder Versuch, auf diesem Gerüst aufzubauen oder es abzuwandeln, zumindest als gefährlich einzustufen.

3.1 Der Anwendungsassistent

Abbildung 3.1:
Der MFC-Anwendungsassistent



Der MFC-Anwendungsassistent ist unzweifelhaft der leistungsfähigste der angebotenen Assistenten, da er nicht nur ein direkt kompilier- und ausführbares MFC-Projekt anlegt, sondern auf dem Weg über eine Reihe von Dialogfeldern es dem Programmierer sogar erlaubt, das zu erzeugende Projekt in vielfältiger Weise an seine Bedürfnisse anzupassen und beispielsweise mit

- ➔ Doc/View-Unterstützung
- ➔ Datenbankbindung
- ➔ OLE- und ActiveX-Features
- ➔ verschiedenen Fensterdekorationen
- ➔ etc.

auszustatten.

Den Grundtyp Ihrer Anwendung legen Sie dabei gleich auf der ersten Dialogseite fest, wo Sie sich für eine der folgenden Anwendungen entscheiden können:

- ➔ EINZELNES DOKUMENT (SDI), eine Anwendung mit einem Rahmenfenster, in dem immer nur ein Dokumentfenster zur Zeit angezeigt werden kann (vgl. Notepad-Editor)
- ➔ MEHRERE DOKUMENTE (MDI), eine Anwendung mit einem Rahmenfenster, in dem mehrere Dokumentfenster verwaltet werden können (vgl. Word für Windows)
- ➔ DIALOGFELDBASIEREND, eine Anwendung mit einem Dialogfeld als Rahmenfenster (vgl. Visual-C++-Assistenten)

Alle Anwendungsassistenten werden über Datei/Neu aufgerufen

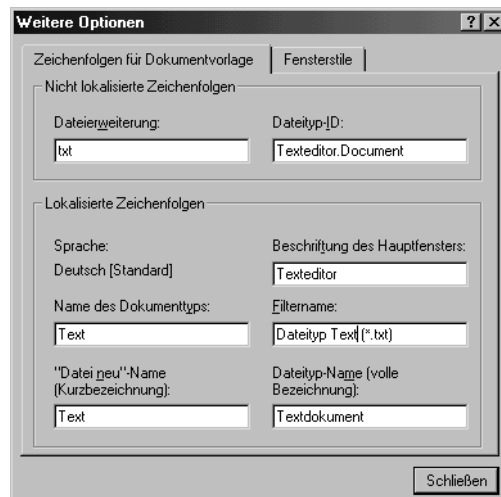
Seite	Beschreibung
1	Auf der ersten Seite legen Sie den grundlegenden Typ Ihrer Anwendung fest (SDI, MDI oder Dialog – siehe oben) und wählen eine Sprache für die anzulegenden Ressourcen. Seit VC 6.0 gibt es die Möglichkeit, auf Doc/View-Unterstützung zu verzichten. Doc/View ist ein Programmiermodell, das sich rein auf die Implementierung eines Programms bezieht und die Idee propagiert, daß für bestimmte Anwendungen die saubere Trennung der Daten (Doc) und deren Darstellung (View) Vorteile bringt (insbesondere dann, wenn ein und dieselben Daten auf unterschiedliche Weise angezeigt werden sollen). Anmerkung! Meinen Erfahrungen nach ist der Assistent nicht in der Lage, ausführbare Projekte ohne Doc/View zu erstellen.
2	Auf der zweiten Seite können Sie Ihre Anwendung mit einer Datenbank verbinden. Im oberen Teil wählen Sie die Art der Datenbankunterstützung und ob Befehle zum Laden und Speichern von Dokumentdateien in das Menü DATEI aufgenommen werden sollen. (Die Dokumentklassen von Datenbank-Applikationen müssen häufig kurzfristig lediglich den Inhalt einer Datenbank darstellen, ohne diese zu speichern. Verwenden Sie in diesem Fall die Option DATENBANKANSICHT OHNE DATEIUNTERSTÜTZUNG.) Wenn Sie sich für eine Datenbankbindung entschieden haben, können Sie im unteren Teil des Dialogfelds über den Schalter DATENQUELLE eine Datenbank auswählen. Seit VC 6.0 haben Sie hier die Möglichkeit, die Datenbankunterstützung nicht nur durch ODBC oder DAO, sondern auch durch OLE DB aufzubauen. (Wobei OLE DB am fortschrittlichsten und für ambitionierte Programmierer am empfehlenswertesten ist, siehe Kapitel zur Datenbankprogrammierung.)

*Tabelle 3.1:
Die Seiten des
MFC-Anwendungsassistenten*

Seite	Beschreibung
3	Die dritte Seite führt die OLE-Möglichkeiten und ActiveX-Features auf. Geben Sie hier an, ob Ihre Applikation OLE-Verbindokumentfunktionalität als Server, Container, Mini-Server oder Container/Server unterstützen soll. Außerdem können Sie Ihrem Projekt Unterstützung für Automations-Server sowie für ActiveX-Steuerelement-Container hinzufügen. Selektieren Sie die letzte Option, wenn Sie ActiveX-Steuerelemente in den Dialogen Ihrer Applikation verwenden möchten. Wenn Sie die Unterstützung für Verbunddokumente selektieren, können Sie ebenfalls die Unterstützung für Verbunddateien aktivieren. Nach Auswahl dieser Option werden die Objekte Ihrer Applikation im Verbunddateiformat gespeichert, welches das Laden nach Aufforderung und sukzessives Speichern ermöglicht (aber größere Dateien bedingt).
4	Die nächste Dialogseite des Anwendungsassistenten (Abbildung 2.6) enthält unterschiedliche Optionen. Die meisten Optionen sind selbsterklärend oder verfügen über eine ausreichende Online-Hilfe. Setzen Sie die Option KONTEXTABHÄNGIGE HILFE, werden Ihrer Applikation die Grundstruktur einer Hilfeprojektdatei sowie Hilfethemendateien hinzugefügt. Außerdem wird die Batch-Datei <i>Makehelp.bat</i> erzeugt, die zur erneuten Generierung überarbeiteter Hilfedateien verwendet werden kann. Selektieren Sie das Kontrollkästchen <i>MAPI (MESSAGING API)</i>, werden die MAPI-Bibliotheken an Ihre Applikation gebunden, und dem Menü DATEI wird der Eintrag SENDEN hinzugefügt. Auf diese Weise gewährleisten Sie eine minimale MAPI-Unterstützung für die Kompatibilität zu Windows-95-Applikationsanforderungen. Das Aktivieren der Option WINDOWS-SOCKETS führt dazu, daß Ihrem Projekt WinSock-Bibliotheken und Header-Dateien hinzugefügt werden. Die WinSock-Funktionalität müssen Sie jedoch selbst dem Projekt hinzufügen. Ihr besonderes Interesse sollte dem Dialog WEITERE OPTIONEN gelten, der nach einem Klick auf die gleichlautende Schaltfläche aufgerufen wird. In diesem Dialog bestimmen Sie einige zusätzliche Optionen, die das Äußere sowie die Ausführung Ihrer Applikation betreffen.

Seite	Beschreibung

	Der Dialog WEITERE OPTIONEN besteht aus zwei Registern.



Das erste Register mit der Bezeichnung ZEICHENFOLGEN FÜR DOKUMENTVORLAGE (siehe Abbildung) ermöglicht Ihnen die Angabe verschiedener Zeichenfolgen, die für den Datei-öffnen-Dialog, die Dokumentvorlagen und den Fenstertitel (MDI) relevant sind. Die eingegebenen Zeichenfolgen werden in der Stringtabelle der Ressource-Datei der Anwendung unter *IDR_MAINFRAME* gespeichert.

Das zweite Register des Dialogs ist mit FENSTERSTILE bezeichnet und wird zur Konfiguration der Rahmenfensterstile Ihrer Applikation verwendet.

- 5 Auf dieser Seite legen Sie fest,
 - ob Sie ein normales oder ein Explorer-ähnliches Fenster haben möchten,
 - ob ausführliche Kommentare angelegt werden sollen,
 - ob die MFC statisch oder als DLL eingebunden werden soll.
- 6 Auf der letzten Seite werden die zu generierenden Klassen angezeigt, und Sie haben die Möglichkeit, zu den einzelnen Klassen passende Basisklassen auszuwählen und die Dateinamen zu ändern.

3.2 Weitere Anwendungsassistenten und Projekttypen

Neben dem MFC-Anwendungsassistenten existieren noch eine Reihe weiterer Anwendungsassistenten und Projekttypen, die Sie auf der Seite PROJEKTE des Dialogfelds NEU (Aufruf über DATEI/NEU) auswählen können.



Möglicherweise werden nicht alle Projekttypen in Ihrer Visual-C++-Version angezeigt. Einige Projekttypen sind lediglich in der Professional- und Enterprise-Edition des Produkts enthalten.

Add-in-Assistent für DevStudio

Assistent zur Erstellung von Add-In-DLLs, die der Erweiterung und Automatisierung der Developer-Studio-IDE dienen (können sowohl auf das Visual-Studio-Objektmodell wie auch auf die Ressourcen des Computers zugreifen).

Assistent für erw. gespeicherte Prozeduren

Legt ein Projekt zur Erstellung von gespeicherten Funktionen (Stored Procedures) für einen Microsoft SQL-Server an.

Assistent für ISAPI-Erweiterungen

Dieser Assistent unterstützt Sie bei der Erstellung von API-Erweiterungen für Internet Server (beispielsweise den Microsoft Internet Information Server (WinNT) oder den Microsoft Personal Web Server (wird mit Microsoft FrontPage ausgeliefert)). Als Erweiterungen können auch Filter implementiert werden, die in den Datentransfer vom und zum Server eingeschaltet werden.

ATL-COM-Anwendungs-Assistent

Mit diesem Assistenten können Sie ActiveX-Steuerelemente auf der Grundlage der ATL (Active Template Library) erstellen.

Benutzerdefinierter Anwendungsassistent

Dieser Assistent ermöglicht Ihnen die Erstellung eigener Anwendungsassistenten. Ein solcher Anwendungsassistent kann auf dem Standard-Anwendungsassistenten für MFC-Applikationen oder DLLs basieren. Er kann außerdem auf einem bestehenden Projekt basieren oder benutzerdefinierte Schritte enthalten, die Sie definieren.

Wenn Sie festlegen, daß Ihr benutzerdefinierter Anwendungsassistent auf einem der Standard-Anwendungsassistenten basieren soll, bestimmen Sie im nächsten Schritt den gewünschten Anwendungsassistenten. Soll der Assistent auf einem bestehenden Projekt basieren, spezifizieren Sie anschließend den Projektpfad.

Cluster-Ressourcentyp-Assistent

Erzeugt zwei Projekte für Microsoft Cluster Server (MSCS)-Ressourcen. Die Cluster-Architektur verbindet mehrere verbundene Systeme zu einem nach außen einheitlich erscheinenden Netzwerk. Ausfälle in einzelnen Untersystemen können von anderen Systemen abgefangen werden.

Der Microsoft Cluster Server wird mit dem Windows NT-Server, Enterprise Edition, ausgeliefert.

Datenbank-Assistent

Die Option NEW DATABASE WIZARD ermöglicht Ihnen die Erstellung einer neuen SQL-Server-Datenbank. Zu dieser Datenbank wird ein Projekt generiert. Beachten Sie bitte, daß Sie über einen Zugriff auf den Microsoft-SQL-Server verfügen müssen, der entweder auf Ihrem Computer oder auf einem Netzwerkrechner vorhanden sein muß, um die Datenbank erstellen lassen zu können.

Datenbankprojekt

Ein Datenbankprojekt verweist in die Tabellen einer Datenbank. Die Tabellen müssen dem Projekt manuell hinzugefügt werden, nachdem dieses erzeugt wurde.

Dienstprogramm-Projekt

Leeres Programmgerüst für Dateien, die nur kompiliert, aber nicht gelinkt werden sollen.

Makefile

Verwenden Sie diese Option, um ein Projekt zu generieren, das auf einer Make-Datei basiert. Geben Sie anschließend den Namen der externen Make-Datei oder anderer ausführbarer Programme an, die zur Erzeugung Ihres Projekts ausgeführt werden sollen.

MFC-ActiveX-Steuerelement-Assistent

Der MFC-ActiveX-Steuerelement-Assistent führt Sie zur Erstellung eines Projekts, das aus einem oder mehreren ActiveX-Steuerelementen besteht, die auf der MFC-Steuerelement-Implementierung basieren. Ein ActiveX-Steuerelement-Projekt erzeugt eine spezielle DLL-Datei,

die eine OCX-Datei bildet. Die beiden Dialogseiten des Anwendungsassistenten dienen der Angabe bestimmter Eigenschaften sowie der Konfiguration einzelner Steuerelemente Ihres ActiveX-Steuerelement-Projekts.

MFC-Anwendungsassistent (dll)

Assistent zur Erstellung von MFC-Programmgerüsten für DLLs (Dynamische Linkbibliotheken).

MFC-Anwendungsassistent (exe)

Assistent zur Erstellung von MFC-Programmgerüsten für Windows-Anwendungen.

Win32 Dynamic-Link Library

Projekt für dynamische Bibliotheken.

Per Voreinstellung für API-Projekte, die MFC kann aber nachträglich über die Projektkonfiguration eingebunden werden.

Win32-Anwendung

Projekt für Windows-Anwendungen.

Per Voreinstellung für API-Projekte, die MFC kann aber nachträglich über die Projektkonfiguration eingebunden werden.

Win32-Bibliothek (statische)

Projekt für statische Bibliotheken.

Per Voreinstellung für API-Projekte, die MFC kann aber nachträglich über die Projektkonfiguration eingebunden werden.

Win32-Konsolenanwendung

Projekt für textbildschirmbasierte Konsolenanwendungen (ohne MFC).

3.3 Der Klassen-Assistent

Aufruf über Ansicht/Klassen-Assistent Der Klassen-Assistent dient der komfortablen, semi-automatischen Bearbeitung von Projekten mit Klassen, die von `CComTarget` abgeleitet sind und über eine `clw`-Datei verfügen (beispielsweise die vom MFC-Anwendungsassistenten generierten Projekte).

Insbesondere folgende Maßnahmen werden unterstützt:

- ➔ Neue Klassen erstellen
- ➔ Antwortmethoden zur Botschaftsverarbeitung einrichten

- ➔ Virtuelle Methoden einrichten
- ➔ Dialogklassen erstellen
- ➔ Elementvariablen für Steuerelemente aus Dialogen anlegen
- ➔ Ereignisse für ActiveX-Steuerelemente definieren
- ➔ Automatisierung von Klassen
- ➔ Klassen aus Typbibliotheken erstellen

Der Klassen-Assistent erscheint als fünfseitiges Dialogfeld:

- ➔ NACHRICHTENZUORDNUNGSTABELLEN. Auf dieser Seite können Sie sich darüber informieren, welche Antwortmethoden in welchen Klassen zu welchen Botschaften deklariert sind (soweit diese vom Klassen-Assistenten verwaltet werden). Sie können Methoden zur Botschaftsverarbeitung einrichten, bearbeiten oder löschen, Sie können virtuelle Methoden überschreiben, und Sie können in den Quelltext der aufgeführten Methoden springen.
- ➔ MEMBER-VARIABLEN. Auf dieser Seite können Sie Elementvariablen zum Datenaustausch zwischen einer Anwendung und den Steuerelementen eines Dialogs oder eines Datenbankformulars einrichten.
- ➔ AUTOMATISIERUNG. Auf dieser Seite können Sie Eigenschaften und Methoden automatisieren.
- ➔ ACTIVEX-EREIGNISSE. Auf dieser Seite können Sie Aktionen definieren, die Ereignisse in ActiveX-Steuerelementen auslösen (nur für die Entwicklung, nicht für die Verwendung von ActiveX-Steuerelementen gedacht).
- ➔ KLASSEN-INFO. Auf dieser Seite können Sie den Nachrichtenfilter einer Klasse so einstellen, daß er feststellt, welche Nachrichten der Klassen-Assistent bereitstellt, um die Behandlungsroutinen in Ihrer Klasse zuzuordnen. Sie können auch ein Fremdobjekt anzeigen oder setzen, das mit der Formularansichts- oder Datensatzansichts-Klasse Ihres Dialogs verbunden ist.

Die Informationen zur Bearbeitung des Projekts entnimmt der Klassen-Assistent einer Datenbankdatei (Extension .clw), die von den Anwendungsassistenten standardmäßig erstellt wird, die aber auch nachträglich angelegt oder aktualisiert werden kann.



3.3.1 Erstellen einer neuen Klasse

Mit einem Klick auf die Schaltfläche **KLASSE HINZUFÜGEN** erstellen Sie eine neue Klasse für Ihre Applikation. Der Klassen-Assistent ermöglicht Ihnen,

- eine Klasse unter Verwendung einer bestehenden Implementierung hinzuzufügen,
- eine Klasse anhand einer Typenbibliothek zu erstellen,
- eine vollständig neue Klasse zu erstellen.

Klassen importieren Möchten Sie eine Klasse aus einer existierenden Implementierung erzeugen, müssen Sie die Klassendaten in den Klassen-Assistenten importieren. Gewöhnlich ist die Klasse bereits ein Teil Ihres Projekts. Diese Vorgehensweise ist sehr geeignet, wenn Sie eine neue Klasse in Ihr Projekt importiert haben, indem Sie die Header- und Implementierungsdateien in das Projektverzeichnis kopierten und manuell dem Projekt hinzufügten. Das Importieren der Klassendaten ist auch dann sinnvoll, wenn Sie die Klasseninformationsdatei (CLW-Datei) Ihrer Applikation erneut erstellen müssen, weil diese beschädigt wurde. In dieser Datei verwahrt der Klassen-Assistent alle relevanten Klasseninformationen.

Klassen aus Typenbibliotheken Das Erzeugen einer Klasse aus einer Typenbibliothek erfordert das Generieren einer neuen Klasse, die die in der Typenbibliothek beschriebene Schnittstelle umfaßt. Sie können dieses Feature beispielsweise verwenden, um eine Klasse zu erstellen, die ein ActiveX-Steuerelement oder ein Automatisierungsobjekt repräsentiert.

Neue Klassen Wenn Sie eine Klasse von Grund auf neu anlegen lassen wollen, geben Sie zuerst den Namen der Klasse ein und wählen dann eine Basisklasse aus.

Repräsentiert die selektierte Basisklasse eine Dialogvorlage (z. B. einen Dialog, eine Formularansicht oder Eigenschaftenseite), wird der Zugriff auf das Feld **DIALOGFELD-ID** freigegeben. Hier wählen Sie einen Bezeichner für eine Dialogvorlage aus der Liste der Bezeichner aus, die in der Ressource-Datei Ihrer Applikation vermerkt sind.

Unterstützt die Basisklasse die Automatisierung, können Sie die entsprechenden Optionen im unteren Bereich des Dialogs selektieren. Beachten Sie bitte, daß einige Klassen Automatisierung, aber nicht das Erstellen von Objekten unterstützen. Ein Zugriff auf die Option **ERSTELLBAR NACH TYP-ID** ist für solche Klassen nicht möglich.

3.3.2 Nachrichtenzuordnungstabellen

MFC implementiert einen speziellen Mechanismus für die Verwaltung von Windows-Nachrichten. Nachdem die Nachrichtenschleife einer Applikation die Nachrichten empfangen hat, werden diese von Klassenobjekten, die sich von der Applikationsklasse *CCmdTarget* ableiten, gemäß spezifischer Regeln bearbeitet. Empfängt ein Objekt eine Nachricht, wird diese entweder verarbeitet oder weitergeleitet.

Ein Objekt verarbeitet lediglich solche Nachrichten, deren Typ in der Nachrichtenzuordnungstabelle aufgeführt und mit einer Bearbeitungsfunktion verbunden ist. Sie greifen über den Klassen-Assistenten auf Nachrichtenzuordnungstabellen für MFC-Klassen zu.

Abbildung 3.2 zeigt die Nachrichtenzuordnungstabelle für die Eigenschaftenseite eines ActiveX-Steuerelements. (Ich habe ein einfaches ActiveX-Steuerelementprojekt für die Abbildungen dieses Kapitels verwendet, da ActiveX-Steuerelemente alle erforderlichen Klassen besitzen, die zur Demonstration der Features des Assistenten benötigt werden.)

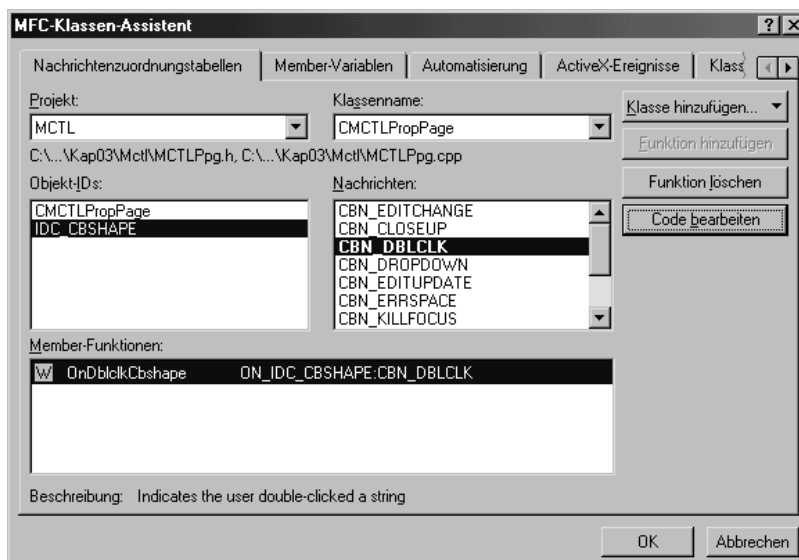


Abbildung 3.2:
Bearbeiten von
Nachrichtenzu-
ordnungstabel-
len mit dem
Klassen-Assi-
stenten

Die Selektion in dem Listenfeld **KLASSENNAME** gibt an, daß Sie die Einträge der Nachrichtenzuordnungstabelle für die Klasse *CMCTL-PropPage* einsehen und festlegen wollen. Im Listenfeld **OBJEKT-IDS** werden die ausgewählte Klasse und alle zugehörigen Objekte aufgeführt, die Nachrichten generieren können (Menübefehle, Steuerelemente, etc.). In Abbildung 3.2 sehen Sie beispielsweise neben der

Klasse *CMCTLPropPage* noch die ID *IDC_CBSHAPE*, die sich auf ein Kombinationsfeld bezieht, das in der Dialogvorlage für *CMCTLPropPage* enthalten ist. (Wenn die Klasse eines Dialogs oder einer Eigenschaftenseite ausgewählt ist, führt der Klassen-Assistent die Bezeichner aller Steuerelemente der Dialogvorlage auf, die mit der in dem Listenfeld OBJEKT-IDS selektierten Klasse verknüpft sind.)

Bearbeitungsfunktionen einrichten

Um sich über die Bearbeitungsfunktionen zu einer Klasse, einem Objekt oder einer Botschaft zu informieren oder eine solche Funktion einzurichten oder zu bearbeiten, gehen Sie wie folgt vor:

1. Wählen Sie im Feld **KLASSENNAME** die Klasse aus, in der die Antwortmethode deklariert werden soll.
2. Wählen Sie im Feld **OBJEKT-IDS** das Objekt aus, dessen Bearbeitungsfunktionen Sie einsehen möchten oder für das Sie eine Botschaftsbearbeitungsfunktion einrichten wollen.
Die bereits eingerichteten Bearbeitungsfunktionen werden jetzt im Feld **MEMBER-FUNKTIONEN** angezeigt.

Wenn Sie eine bereits eingerichtete Bearbeitungsfunktion bearbeiten wollen,

- 3a. Wählen Sie die Bearbeitungsfunktion im Feld **MEMBER-FUNKTIONEN** aus, und klicken Sie auf den Schalter **CODE BEARBEITEN**.

Wenn Sie eine neue Bearbeitungsfunktion für eine Botschaft einrichten wollen,

- 3b. Wählen Sie im Feld **NACHRICHTEN** eine Botschaft aus.
Für IDs von Menübefehlen stehen Ihnen beispielsweise die Nachrichten **COMMAND** (zur Behandlung des eigentlichen Menübefehls) und **UPDATE_COMMAND_UI** (zur Aktualisierung des Menübefehls im Menü, beispielsweise durch Anzeigen eines Häkchens oder Deaktivierung) zur Verfügung. Für Klassen stehen Ihnen die virtuellen Methoden, die vordefinierten Bearbeitungsfunktionen (**On...**) und die Windows-Botschaften (**WM_...**) zur Verfügung (die Auswahl hängt von dem Typ der Klasse ab und kann über das Feld **FILTER FÜR NACHRICHTEN** auf der Seite **KLASSEN-INFO** verändert werden).
Nachrichten (oder Methoden), für die bereits Implementierungen vorliegen, werden im Feld **NACHRICHTEN** durch Fettschrift hervorgehoben.

4. Klicken Sie auf den Schalter **FUNKTION HINZUFÜGEN**.

Änderungen im Quelltext

Wenn Sie eine Bearbeitungsfunktion einrichten, aktualisiert der Klassen-Assistent automatisch Ihren Quelltext. Fügen Sie beispielsweise die in Abbildung 3.2 dargestellte Member-Funktion der Klasse *CMCTLPropPage* hinzu, wird die Deklaration der Nachrichtenzuordnungstabelle in der Header-Datei wie folgt modifiziert:

```
// Message maps
protected:
    ///{AFX_MSG(CMCTLPropPage)
    afx_msg void OnDbclckCbshape();
    ///}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

Der Klassen-Assistent ermittelt die Position der Nachrichtenzuordnungstabelle in Ihrem Programmcode anhand der speziellen Kommentare, die die Tabelle umschließen. Deklarationen von Nachrichtenzuordnungstabellen sind beispielsweise durch Kommentare gekennzeichnet, die mit `///{AFX_MSG` beginnen. Sie sollten Programmcode, der mit solchen Kommentaren versehen ist, nicht verändern. Das Auffinden dieser spezifischen Programmabschnitte in Ihren Quelldateien ist sehr einfach. Der Developer-Studio-Editor zeigt die Abschnitte in einer besonderen Farbe an.

Die Deklaration der Nachrichtenzuordnungstabelle ist nicht der einzige Abschnitt, der von dem Klassen-Assistent modifiziert wird. Die Definition der Nachrichtenzuordnungstabelle in der Implementierungsdatei der Klasse *CMCTLPropPage* wird ebenfalls verändert.

```
////////////////////////////////////
// Message map

BEGIN_MESSAGE_MAP(CMCTLPropPage, C01ePropertyPage)
    ///{AFX_MSG_MAP(CMCTLPropPage)
    ON_CBN_DBLCLK(IDC_CBSHAPE, OnDbclckCbshape)
    ///}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Beachten Sie bitte, daß der Klassen-Assistent eine Grundstruktur für die Implementierung der neuen Funktion erstellt, die mit *OnDbclckCbshape* bezeichnet ist. Diese Implementierung wird der Datei einfach angehängt. Sie können den entsprechenden Abschnitt an die gewünschte Position im Programmcode verschieben:

```
////////////////////////////////////
// CMCTLPropPage message handlers

void CMCTLPropPage::OnDbclckCbshape()
{
    // TODO: Add your control notification handler code here
}
```

Virtuelle Elementfunktionen überschreiben

1. Wählen Sie im Feld KLASSENNAME die Klasse aus, in der die Elementfunktion überschrieben werden soll. Danach müssen Sie die Klasse noch einmal im Feld OBJEKT-IDS auswählen.
2. Wählen Sie im Feld NACHRICHTEN die zu überschreibende Elementfunktion aus.
3. Klicken Sie auf den Schalter FUNKTION HINZUFÜGEN.

Die eingerichtete Elementfunktion wird im Feld MEMBER-FUNKTIONEN angezeigt. Virtuelle Elementfunktionen werden dabei durch ein vorangestelltes »V« gekennzeichnet.

Member-Funktionen löschen

1. Um eingerichtete Member-Funktionen zu löschen, markieren Sie die Funktion im Feld MEMBER-FUNKTIONEN, und drücken Sie den Schalter FUNKTION LÖSCHEN.

Haben Sie für eine eingerichtete Member-Funktion bereits Code ausgesetzt, kann der Klassen-Assistent die Definition nicht mehr selbst entfernen. In solchen Fällen werden Sie darauf hingewiesen, daß Sie die Definition der Member-Funktion selbst löschen müssen.

Member-Funktionen bearbeiten

Um in die Definition einer eingerichteten Member-Funktion zu springen und den passenden Code einzugeben, markieren Sie die Funktion im Feld MEMBER-FUNKTIONEN, und drücken Sie den Schalter CODE BEARBEITEN.

3.3.3 Member-Variablen und Datenaustausch

In dem zweiten Register des Klassen-Assistenten-Dialogs können Sie Member-Variablen definieren und modifizieren.

Wenn Sie einen Dialog implementieren, reicht es nicht, die Ressource aufzusetzen, eine Klasse mit der Ressource zu verbinden und den Dialog in der Anwendung aufzurufen. Nachdem der Anwender den Dialog beendet hat, müssen Sie die Eingaben des Anwenders in den Steuerelementen des Dialogs abfragen (und üblicherweise auch darauf reagieren).

Die MFC verfügt über einen speziellen internen Mechanismus, der Ihnen die Abfrage der Eingaben in den Steuerelementen wesentlich erleichtert – DDX (Dialogdatenaustausch) und DDV (Dialogdatenüberprüfung).

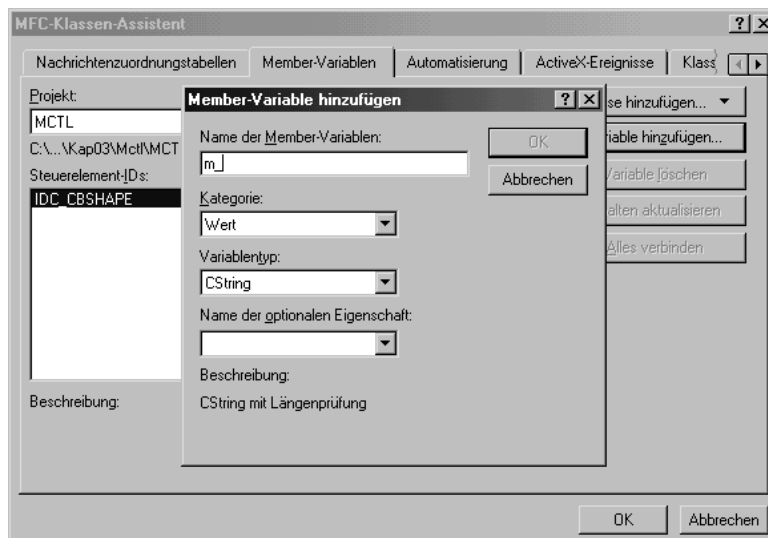


Abbildung 3.3:
Bearbeiten von
Member-Variablen mit dem
Klassen-Assistenten

Um DDX nutzen zu können, brauchen Sie nur mit Hilfe des Klassen-Assistenten für jedes Steuerelement Ihres Dialogs eine Elementvariable einzurichten. Die Implementierung sorgt dann dafür, daß beim Aufrufen des Dialogs die Steuerelemente mit den Werten der zugehörigen Elementvariablen initialisiert werden und daß beim Abschicken des Dialogs die Elementvariablen umgekehrt mit den Eingaben aus den Steuerelementen aktualisiert werden.

Elementvariablen für Dialogelemente einrichten:

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und wechseln Sie zur Seite MEMBER-VARIABLEN.
2. Wählen Sie im Feld KLASSENNAME die Dialogklasse aus. Wenn Sie noch keine Dialogklasse für Ihre Dialogressource eingerichtet haben, können Sie dies durch Klick auf den Schalter KLASSE HINZUFÜGEN/NEU nachholen.
3. Wählen Sie im Feld STEUERELEMENT-IDS ein Steuerelement aus.
4. Klicken Sie auf den Schalter VARIABLE HINZUFÜGEN. Im Dialog MEMBER-VARIABLE HINZUFÜGEN legen Sie neben dem Namen auch noch Kategorie und Variablentyp der übertragenen Daten fest.
 - Im Feld KATEGORIE entscheiden Sie, ob der Inhalt des Steuerelements (Wert) oder eine Referenz auf das Steuerelement (Control) übertragen werden soll.

- Im Feld VARIABLENTYP wählen Sie den genauen Datentyp aus (üblicherweise ist dieser bereits durch den Typ des Steuerelements festgelegt).

Wenn Sie eine neue Member-Variable definieren, aktualisiert der Klassen-Assistent den Quellcode Ihrer Applikation an verschiedenen Positionen. Das Hinzufügen einer `int`-Variablen `m_nShape` führt beispielsweise zu der folgenden Änderung in der *CMCTLPropPage*-Header-Datei:

```
// Dialog Data
//{{AFX_DATA(CMCTLPropPage)
enum { IDD = IDD_PROPPAGE_MCTL };
int m_nShape;
//}}AFX_DATA
```

Der Klassen-Assistent verändert auch die Implementierungsdatei der Klasse. Dem Klassen-Konstruktor wurden Initialisierungen für die neuen Variablen hinzugefügt:

```
CMCTLPropPage::CMCTLPropPage() :
COlePropertyPage(IDD, IDS_MCTL_PPG_CAPTION)
{
    //{{AFX_DATA_INIT(CMCTLPropPage)
    m_nShape = -1;
    //}}AFX_DATA_INIT
}
```

Eine weitere Funktion, die von dem Klassen-Assistenten modifiziert wurde, ist die `DoDataExchange`-Member-Funktion. In dieser Funktion werden Informationen zwischen dem Steuerelementobjekt selbst und der Variable ausgetauscht, die den Wert des Objekts repräsentiert. Für *CMCTLPropPage* wird die Funktion wie folgt modifiziert:

```
////////////////////////////////////
// CMCTLPropPage::DoDataExchange - Moves data between page and properties

void CMCTLPropPage::DoDataExchange(CDataExchange* pDX)
{
    //{{AFX_DATA_MAP(CMCTLPropPage)
    DDP_CBIndex(pDX, IDC_CBSHAPE, m_nShape, _T("Shape"));
    DDX_CBIndex(pDX, IDC_CBSHAPE, m_nShape);
    //}}AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}
```

Zur Erleichterung des Datenaustausches zwischen dem Steuerelementobjekt und der Member-Variablen wird nicht nur eine DDX-Funktion (Dialog Data Exchange), sondern ebenfalls eine DDP-Funktion (Eigenschaftenseite) aufgerufen. Diese Funktion transferiert Daten zwischen der Member-Variablen und dem ActiveX-Steuerelementobjekt.

Sollten die Klassen bestimmte Datensätze in Datenbanken repräsentieren, nimmt der Klassen-Assistent möglicherweise Veränderungen an weiteren Funktionen vor (z.B. an der `DoFieldExchange`-Member-Funktion der Klasse, die von `CRecordSet` abgeleitet wird).

3.3.4 Automatisierung

Das Register AUTOMATISIERUNG im Dialog des Klassen-Assistenten ermöglicht Ihnen das Hinzufügen und Modifizieren von Automatisierungseigenschaften und -Methoden (zuvor OLE-Automatisierung). Eine Eigenschaft ist eine Member-Variable, auf die über die Automatisierungsschnittstelle zugegriffen wird. Eine Methode ist eine Member-Funktion. Klassen, die ActiveX-Steuerelemente repräsentieren, können zusätzlich zur Automatisierung Eigenschaften und Methoden besitzen.

Nicht alle Klassen in Ihrer Applikation unterstützen die Automatisierung. Unterstützung für Automatisierung können Sie beispielsweise vorsehen, indem Sie bei der Erstellung der Klasse im MFC-Anwendungsassistenten die Option AUTOMATISIERUNG aktivieren.

Automatisierungs-Methode hinzufügen

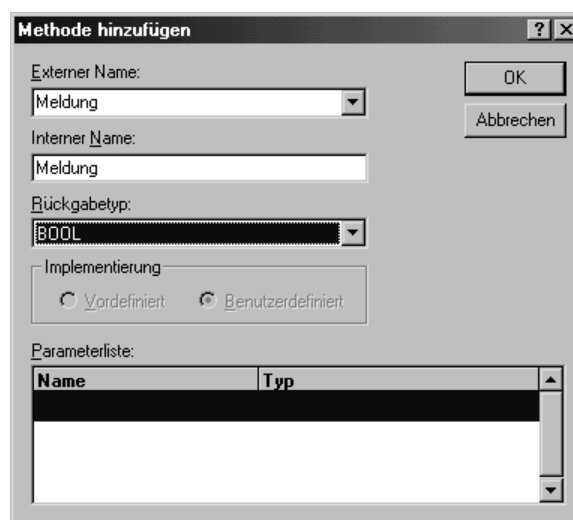


Abbildung 3.4:
Hinzufügen
einer Automati-
sierungs-
Methode

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und wechseln Sie zur Seite AUTOMATISIERUNG.
2. Wählen Sie im Feld KLASSENNAME die zu automatisierende Klasse aus. Wenn Sie noch keine entsprechende Klasse für die Automatisierung eingerichtet haben, können Sie dies durch Klick auf den Schalter KLASSE HINZUFÜGEN/NEU nachholen.
3. Klicken Sie auf den Schalter METHODE HINZUFÜGEN. Im Dialog METHODE HINZUFÜGEN machen Sie folgende Angaben:

- Den externen Namen der Methode. Unter diesem Namen können Automatisierungs-Clients auf die Methode zugreifen. Repräsentiert die Klasse ein ActiveX-Steuerelement, können Sie den Namen einer Standardmethode ebenfalls aus dem Kombinationsfeld unter EXTERNER NAME auswählen.
- Den internen Namen. Unter diesem Namen wird die Methode in der C++-Klasse des Servers deklariert und definiert.
- Einen Rückgabebetyp.
- Den Implementierungstyp.
- Etwaige Parameter. Doppelklicken Sie dazu einfach in die leere Schablone, und geben Sie den Parameternamen ein. Den Typ wählen Sie aus einer Liste aus.

Eigenschaften automatisieren

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und wechseln Sie zur Seite AUTOMATISIERUNG.
2. Wählen Sie im Feld KLASSENNAME die zu automatisierende Klasse aus. Wenn Sie noch keine entsprechende Klasse für die Automatisierung eingerichtet haben, können Sie dies durch Klick auf den Schalter KLASSE HINZUFÜGEN/NEU nachholen.
3. Klicken Sie auf den Schalter EIGENSCHAFT HINZUFÜGEN. Im Dialog EIGENSCHAFT HINZUFÜGEN machen Sie folgende Angaben:
 - Den externen Namen der Eigenschaft. Unter diesem Namen können Automatisierungs-Clients auf die Methode zugreifen.
 - Einen Typ. Den Datentyp der Eigenschaft wählen Sie aus der Liste aus.
 - Den Implementierungstyp. Entscheiden Sie sich hier zwischen Member-Variable und Get/Set-Methoden.

Falls Sie Member-Variable gewählt haben:

- Den Variablennamen. Geben Sie hier den Namen des zugehörigen Datenelements in der C++-Klasse ein.
- Eine Benachrichtigungsfunktion. Diese Funktion wird aufgerufen, wenn sich der Wert der Member-Variablen geändert hat.
- Etwaige zusätzliche Parameter.

Falls Sie Get/Set-Methoden gewählt haben:

- Eine Get-Funktion. Über diese Methode wird der Wert der Eigenschaft zurückgeliefert.

Für ActiveX-Steuerelemente können Sie den Grad der Unterstützung für die Datenverbindung bestimmen, die Sie Ihrem Steuerelement hinzufügen möchten. Datenverbindung ist ein Begriff, der für die Möglichkeit steht, ActiveX-Steuerelement-Eigenschaften an bestimmte Felder einer Datenbank zu binden. Betätigen Sie die Schaltfläche Datenverbindung, um mit Hilfe des gleichlautenden Dialogs den Grad der Datenverbindung zu bestimmen, den die Eigenschaft unterstützen soll, und um eine Eigenschaft zu definieren, die anschließend gebunden werden kann.



3.3.5 ActiveX-Ereignisse

ActiveX-Ereignisse werden von ActiveX-Steuerelementen generiert. Ereignisse sind ein Medium für die Kommunikation zwischen dem Steuerelement und dessen Container.

Die ActiveX-Ereignisse einer ActiveX-Steuerelementklasse können in dem Register ACTIVEX-EREIGNISSE des Klassen-Assistenten definiert oder modifiziert werden.

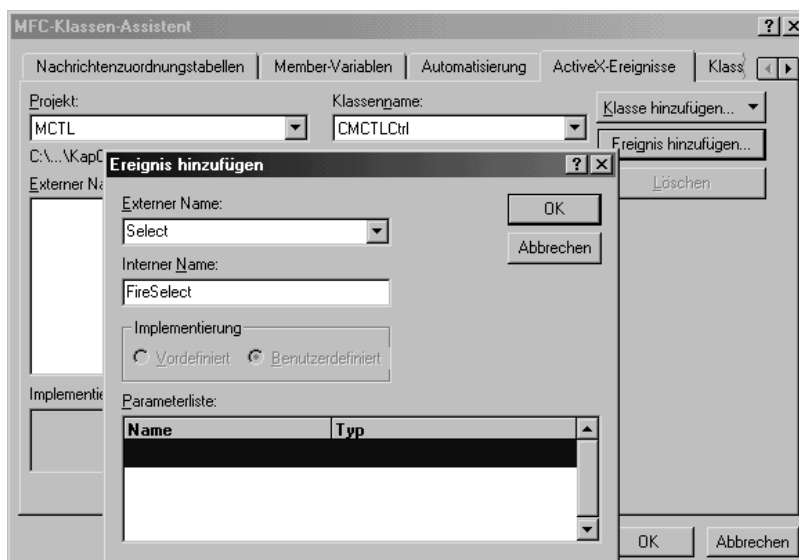


Abbildung 3.5:
Erstellen eines
ActiveX-Ereig-
nisses

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und wechseln Sie zur Seite ACTIVE-X-EREIGNISSE.
2. Wählen Sie im Feld KLASSENNAME die Klasse des ActiveX-Steuer-elements aus. Wenn Sie noch keine entsprechende Klasse für die Automatisierung eingerichtet haben, können Sie dies durch Klick auf den Schalter KLASSE HINZUFÜGEN/NEU nachholen.
3. Klicken Sie auf den Schalter EREIGNIS HINZUFÜGEN. Im Dialog EREIGNIS HINZUFÜGEN machen Sie folgende Angaben:
 - Den externen Namen des Ereignisses. Sie können entweder ein eigenes Ereignis definieren oder ein vordefiniertes Ereignis auswählen.
 - Der interne Name ist der Name der Member-Funktion, die das Ereignis auslöst. Der Name ist eine Kombination aus dem Wort *Fire* und dem externen Namen. Für ein Ereignis mit der Bezeichnung *Select* gibt der Klassen-Assistent somit den Funktionsnamen *FireSelect* vor.
 - Den Implementierungstyp. Für vordefinierte Ereignisse kann eine vordefinierte Implementierung bestimmt werden oder eine benutzerdefinierte Implementierung vorgesehen werden.
 - Die Parameter der auslösenden Funktion.

Nachdem ein neues Ereignis definiert wurde, modifiziert der Klassen-Assistent die Header-Datei Ihrer Klasse, indem er das Ereignis in die Ereignistabelle einträgt. Das Ereignis wird der Klasse in Form seiner auslösenden Funktion und seiner Inline-Implementierung hinzugefügt. Für das benutzerdefinierte Ereignis *Select* modifiziert der Klassen-Assistent die Ereignistabelle wie folgt:

```
// Event maps
//{{AFX_EVENT(CMCTLCtrl)
void FireSelect(BOOL IsSelected)
{FireEvent(eventidSelect,EVENT_PARAM(VTS_BOOL), IsSelected);}
//}}AFX_EVENT
DECLARE_EVENT_MAP()
```

Das Ereignis wird ausgeführt, wenn Sie die auslösende Funktion aus anderen Funktionen in Ihrem Programmcode aufrufen.

3.3.6 Klassen-Info

Das letzte Register des Klassen-Assistenten ist mit KLASSEN-INFO bezeichnet. Hier werden verschiedene Klasseneigenschaften aufgeführt. Des weiteren können in dem Register bestimmte erweiterte Optionen modifiziert werden.

- ➡ Die erste der erweiterten Optionen, die mit FILTER FÜR NACHRICHTEN bezeichnet ist, ermöglicht Ihnen zu bestimmen, welche Nachrichten der Klassen-Assistent in der Nachrichtenzuordnungstabelle aufführen soll. Beachten Sie bitte, daß sich das Verändern des Nachrichtenfilters nicht auf den Programmcode Ihrer Applikation auswirkt. Die hier vorgenommene Einstellung betrifft lediglich die Anzeige der Nachrichten.
- ➡ Die Option FREMDKLASSE ist für Datenbank-Applikationen relevant. Die Ansichtsklasse einer Datenbank-Applikation, die von *CRecordView* oder *CDaoRecordView* abgeleitet wird, ist mit einer Datensatzklasse verknüpft, die sich wiederum von *CRecordSet* oder *CDaoRecordSet* ableitet. In diesem Fall trägt die Fremdklasse die Bezeichnung der Datensatzklasse. Die Fremdklasse ist ein Zeiger in die Ansichtsklasse Ihrer Applikation, die ein Fremdklassen-Objekt repräsentiert. Der Klassen-Assistent muß die Fremdklasse identifizieren können, da er auf die Member-Variablen der Datensatzklasse verweisen und deren Member-Funktion *OnFieldExchange* aktualisieren können muß.

3.4 Die Klasseninformationsdatei

Der Klassen-Assistent kann nicht den gesamten Quellcode Ihrer Applikationen analysieren. Die Informationen, die nicht aus der Quelldatei ermittelt werden können, werden in einer besonderen Datei, der Klasseninformationsdatei, gespeichert. Diese Datei trägt dieselbe Bezeichnung wie Ihr Projekt. Die Dateiendung lautet *.CLW*.

Die Informationen für die *.clw*-Datei können nur dann aus dem Quelltext ausgelesen werden, wenn im Quelltext entsprechende Makros zur Unterstützung des Klassen-Assistenten verwendet werden (Codeabschnitte außerhalb dieser Bereiche werden vom Klassen-Assistenten nicht angetastet). Sofern Sie einen Anwendungs-Assistenten oder den Klassen-Assistenten zur Bearbeitung Ihres Quelltextes nutzen, werden diese Makros automatisch eingefügt. Die gleichen Markierungen verwendet der Klassen-Assistent, um Code in Ihr Projekt einzufügen.

Hierzu gehört beispielsweise, daß die Deklarationen für Antwortfunktionen zu Botschaften zwischen die Makros `//{{AFX_MSG(CFensterklasse)}` und `//}}AFX_MSG` geschrieben werden und als `afx_msg` deklariert werden:

```
//{{AFX_MSG(CMyWnd)
    afx_msg void OnPaint();
//}}AFX_MSG
```

Änderungen, die Sie mit Hilfe des Klassen-Assistenten vorgenommen haben, werden automatisch in der .clw-Datei eingetragen.

Wenn Sie einen Quelltext von Hand bearbeitet haben und die Informationen für den Klassen-Assistenten danach aktualisieren wollen, löschen Sie die .clw-Datei im Windows Explorer und lassen die Datei dann vom Klassen-Assistenten neu erstellen, wobei Sie darauf achten sollten, daß alle Header- und Quelltextdateien des Projekts im Feld Dateien im Projekt aufgeführt werden.

3.5 Zusammenfassung

Anwendungsassistenten befreien Sie von lästigen Routearbeiten – von dem Anlegen eines Projekts mit Quelldateien bis zur Implementierung eines lauffähigen Grundgerüsts.

Über DATEI/NEU gelangen Sie zu einem Dialogfeld, in dem Sie zwischen verschiedenen Anwendungsassistenten und Projekttypen wählen können.

Am leistungsfähigsten ist der MFC-Anwendungsassistent, mit dem Sie Windows-MFC-Anwendungen mit oder ohne Doc/View als SDI, MDI oder dialogbasierte Anwendung erstellen lassen können. Der Anwendungsassistent kann zudem für eine Datenbankunterstützung, ActiveX- und OLE-Features, Fensterdekorationen, kontextabhängige Hilfe und vieles mehr sorgen.

Der Klassen-Assistent ist ein weiteres wesentliches Werkzeug des Visual-C++-Entwicklungssystems. Mit Hilfe des Assistenten können Sie Projekte, die mit dem MFC-Anwendungsassistenten generiert wurden, weiter bearbeiten.

Der Klassen-Assistent wird in einem Dialog mit fünf Registern ausgeführt.

- ➔ Das erste dieser Register, das mit NACHRICHTENZUORDNUNGSTABELLE bezeichnet ist, ermöglicht Ihnen die Definition von Bearbeitungsfunktionen für verschiedene Ereignisse.
- ➔ In dem Register MEMBER-VARIABLEN können Sie Member-Variablen für den Datenaustausch mit Dialog-Steuerelementen oder Datenbankformularen einrichten.
- ➔ Das Register AUTOMATISIERUNG dient der Einrichtung von verschiedenen Automatisierungs-Methoden und -Eigenschaften für Klassen, die Automatisierung unterstützen. Für ActiveX-Steuerelemente können Sie vordefinierte sowie benutzerdefinierte Methoden und Eigenschaften bestimmen.

- ➔ Das Register ACTIVE-X-EREIGNISSE dient der Definition von Active-X-Ereignissen, die von einem Active-X-Steuerelement generiert werden können.
- ➔ In dem Register KLASSEN-INFO werden einige allgemeine Informationen zu bestimmten Klassen angezeigt.
- ➔ Neue Klassen können von Grund auf aus Typenbibliotheken und aus bestehenden Header- und Implementierungsdateien generiert werden.

Browser und Debugger

Kapitel

4

Browser und Debugger sind zwei Entwicklertools, die einem ähnlichen Zweck dienen, aber unterschiedlichen Einsatzgebieten angehören.

Beide Tools dienen dazu, uns detailliert über den Zustand eines Programms zu informieren.

Der Browser vermittelt dabei allerdings nur ein statisches Bild des Programms. Er führt uns zu Definitionen und Referenzen, zeichnet Klassenhierarchien und Funktionsaufrufe nach.

Der Debugger erlaubt uns, ein Programm während der Ausführung zu kontrollieren. Mit seiner Hilfe können wir zur Laufzeit die Programmausführung gezielt anhalten und uns die augenblicklichen Werte von Variablen, den Zustand des Stacks und andere aktuelle Informationen anzeigen lassen.

Beide Tools sind auf spezielle Informationen angewiesen, die während der Kompilierung erzeugt werden müssen.

4.1 Der Quellcode-Browser

Der Quellcode-Browser dient – anders als Assistentenleiste und Klassen-Ansicht – rein der Analyse des Quelltextes, kennt also keine Befehle zum Einfügen von Klassen oder Klassenmethoden. Dafür kann Ihnen der Quellcode-Browser Informationen zusammenstellen, die Assistentenleiste und ClassView nicht kennen. So

- ➡ unterstützt der Quellcode-Browser auch lokale Variablen,
- ➡ kann der Quellcode-Browser die in einer Datei verwendeten Bezeichner auflisten,

- steht der Quellcode-Browser hinter verschiedenen Befehlen, die auch in den Kontextmenüs der Klassen-Ansicht verfügbar sind, beispielsweise die Darstellung von Klassenhierarchien.

4.1.1 Hinzufügen von Browser-Informationen zu einem Projekt

Der Quellcode-Browser ist zur Ausführung seiner Befehle auf spezielle Browser-Informationen angewiesen. Um dem Projekt Browser-Informationen hinzuzufügen, müssen Sie Ihre Projekteinstellungen ändern. Alternativ dazu können Sie das Programm *BSCMAKE.EXE* aufrufen.

Projekteinstellungen ändern

Um Browser-Informationen zu erzeugen, aktivieren Sie im Dialogfeld PROJEKTEINSTELLUNGEN (Aufruf über PROJEKT/EINSTELLUNGEN) für den Projektknoten die Optionen

- BROWSE-INFO GENERIEREN auf der Seite C/C++, Kategorie ALLGEMEIN,
- BROWSER-INFORMATIONSDATEI ERSTELLEN auf der Seite BROWSER-INFORMATION,

und lassen Sie das Projekt neu erstellen.



Sollten Sie versuchen, Browser-Features zu nutzen, wenn keine Browser-Informationsdatei vorhanden ist, gibt das Developer Studio eine Warnung aus und bietet Ihnen an, das Projekt einschließlich der erforderlichen Browser-Informationen erneut zu erstellen.

Browser-Informationsdatei löschen

Die Browser-Informationen werden in einer – äußerst umfangreichen – Datei (Extension .bsc) abgelegt. Löschen Sie diese, wenn Sie die Browser-Informationen nicht mehr benötigen. Sie können die .bsc-Datei für den Browser nur dann löschen, wenn der Browser den Zugriff auf sie freigegeben hat. Rufen Sie dazu den Menübefehl EXTRAS/QUELLCODE-BROWSER-DATEI SCHLIESSEN auf.

4.1.2 Mit dem Quellcode-Browser arbeiten

1. Rufen Sie den Quellcode-Browser auf (Befehl QUELLCODE-BROWSER im Menü EXTRAS). Es erscheint das Dialogfeld DURCHSUCHEN.
2. Geben Sie den Bezeichner ein, über dessen Verwendung Sie sich informieren wollen.

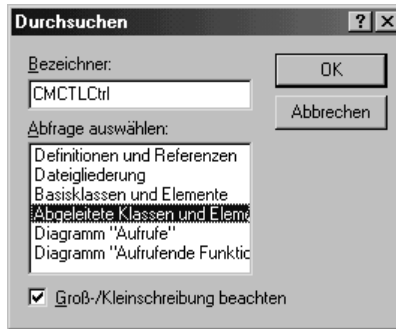


Abbildung 4.1:
Das Fenster des
Browsers

Wenn Sie möchten, können Sie den Bezeichner (Klasse, Methode, lokale oder globale Variable etc.) auch vor dem Aufruf des Quellcode-Browsers im Quelltext oder der Klassen-Ansicht markieren. Der Name wird dann direkt in das Feld BEZEICHNER übertragen, und Sie vermeiden Tippfehler.

3. Rufen Sie einen der Befehle im Listenfeld ABFRAGE AUSWÄHLEN auf.

4.2 Der Debugger

Eines der leistungsfähigsten Features von Visual C++ ist der integrierte Symbol-Debugger. Mit Hilfe dieses Werkzeugs können Sie Ihre Programme während der Ausführung überwachen. Sie können das Programm mittels Haltepunkten an definierten Stellen anhalten, es schrittweise ausführen lassen, sich Informationen über die Werte von Variablen, den Zustand des Stacks, etc. anzeigen lassen.

*Erstellen/Debug
starten führt eine
Anwendung im
Debugger aus*

Zudem unterstützt der Visual-C++-Compiler Just-In-Time-Testläufe (das Testen von abgestürzten Programmen, auf die außerhalb der Entwicklungsumgebung zugegriffen wurde) und Remote-Testläufe. Der Debugger ist ebenso wie andere Features (Quellcode-Browser oder Quellcode-Editor) vollständig in die Visual-Studio-Umgebung integriert.

4.2.1 Vorbereiten einer Anwendung auf den Testlauf

Damit der Debugger korrekt arbeiten kann, ist er auf entsprechende Debug-Informationen in den .obj- und .exe-Dateien angewiesen. Damit diese erstellt werden, sind folgende Projekteinstellungen nötig:

Auf der Seite C/C++, Kategorie ALLGEMEIN der Projekteinstellungen:

- ➔ PROGRAMMDATENBANK oder PROGRAMMDATENBANK FÜR »BEARBEITEN UND FORTFAHREN« im Feld DEBUG-INFO. Letztere Einstellung unterstützt das Debuggen editierten Codes ohne Neukompilation (siehe unten). (Kommandozeilenoption `/ZI`)
- ➔ OPTIMIERUNGEN (DEAKTIVIEREN (DEBUG)), um eine möglichst vollständige Übereinstimmung zwischen Ihrem Quelltext und dem debuggten Objektcode zu erreichen. (Kommandozeilenoption `/Od`)

Auf der Seite C/C++, Kategorie CODE GENERATION der Projekteinstellungen:

- ➔ Wählen Sie unter LAUFZEIT-BIBLIOTHEK eine passende Debug-Bibliothek aus, um eine Debug-Version der C-Laufzeitbibliothek in Ihr Projekt einzubinden. (Kommandozeilenoptionen `/MDd` (DLL debuggen), `/MLd` (Single-Thread debuggen) oder `/MTd` (Multi-thread debuggen))

Auf der Seite LINKER, Kategorie ALLGEMEIN der Projekteinstellungen:

- ➔ DEBUG-INFO GENERIEREN setzen. (Kommandozeilenoption `/debug`)

Auf der Seite LINKER, Kategorie ANPASSEN der Projekteinstellungen:

- ➔ PROGRAMMDATENBANK VERWENDEN setzen.

Auf der Seite LINKER, Kategorie DEBUG der Projekteinstellungen:

- ➔ Die Option DEBUG-INFO setzen.



Ist Ihre Applikation eine MFC-Anwendung, die mit dem Anwendungs-Assistenten erstellt wurde, brauchen Sie vermutlich keine Änderungen vorzunehmen. Der Anwendungs-Assistent generiert gewöhnlich eine Debug-Konfiguration zu Ihrem Projekt und deklariert diese als Standardkonfiguration.

4.2.2 Ausführen einer Anwendung im Debugger

Nachdem Sie Ihre Anwendung mit den Debug-Einstellungen kompiliert haben, können Sie die Anwendung im Debug-Modus ausführen lassen, indem Sie einen der Einträge aus dem Untermenü DEBUG STARTEN auswählen, das wiederum in dem Menü ERSTELLEN enthalten ist:

- ➔ ERSTELLEN/DEBUG STARTEN/AUSFÜHREN (**F5**). Führt Programm ganz bzw. bis zum nächsten Haltepunkt aus.

- ➡ ERSTELLEN/DEBUG STARTEN/IN AUFRUF SPRINGEN (**F11**). Startet das Programm und stoppt es zu Beginn der Eintrittsfunktion (`main()`, `WinMain()`). Ansonsten führt dieser Befehl das Programm schrittweise aus.
- ➡ ERSTELLEN/DEBUG STARTEN/AUSFÜHREN BIS CURSOR (**Strg** + **F10**). Führt das Programm bis zur aktuellen Cursorposition aus.
- ➡ ERSTELLEN/DEBUG STARTEN/VERBINDEN MIT PROZESS. Verbindet den Debugger nachträglich mit einem bereits in der Ausführung befindlichen Programm (wird über ein Dialogfeld ausgewählt).

DLLs debuggen

Für DLLs ist das Feld *Ausführbares Programm für Debug-Sitzung* interessant. Verwenden Sie dieses Eingabefeld, um DLL-Projekte zu debuggen. Geben Sie jedoch nicht den Namen der DLL, sondern den Namen des Programms an, das die DLL lädt und testet. Um beispielsweise ein ActiveX-Steuerelement zu debuggen (das ein besonderer DLL-Typ ist), verwenden Sie die Anwendung `tstcon32.exe`.



Wenn Sie eine Debug-Sitzung starten, erscheinen abhängig von Ihren Visual-Studio-Einstellungen verschiedene Debug-Fenster. In der Menüleiste von Visual Studio wird das Menü ERSTELLEN durch das Menü DEBUG ersetzt.

Der nächste Schritt besteht darin, die Anwendung gezielt anzuhalten, um sich dann über den aktuellen Zustand des Programms informieren zu können.

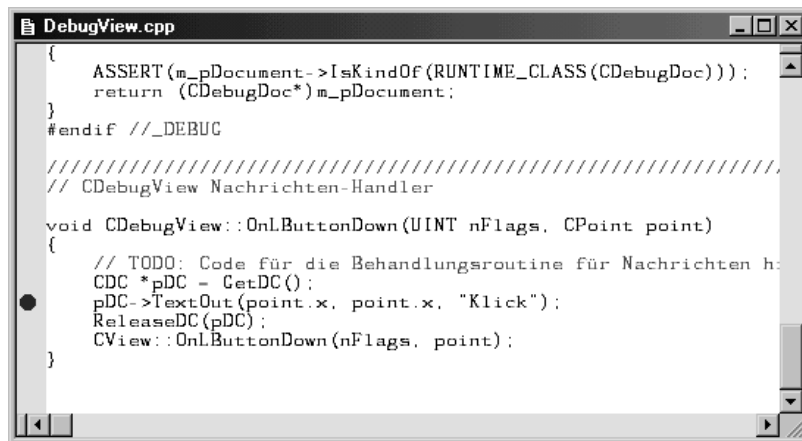
4.2.3 Haltepunkte und Einzelschrittausführung

Die beiden wesentlichen Features des Debuggers sind das Einfügen von Haltepunkten in den Programmcode und die Einzelschrittausführung des Programms.

Haltepunkte setzen

Die einfachste Möglichkeit, einen Haltepunkt zu setzen, besteht darin, die Schreibmarke an die gewünschte Position im Quellcode zu bewegen und dort die Taste **F9** zu drücken. Ein Haltepunkt ist durch einen großen roten Punkt links von der Programmzeile gekennzeichnet (siehe Abbildung 4.2).

Abbildung 4.2:
Haltepunkt
setzen



Beachten Sie, daß Sie auch in dem Fenster Disassemblierung Haltepunkte setzen können.

Haltepunkte deaktivieren oder löschen

Um einen Haltepunkt zu löschen, setzen Sie die Schreibmarke auf die Zeile mit dem Haltepunkt und drücken erneut die Taste **[F9]**.

Wollen Sie den Haltepunkt gesetzt lassen (für spätere Verwendung), ihn aber im Moment beim Debuggen nicht berücksichtigen, können Sie ihn im Haltepunktfenster deaktivieren. Rufen Sie dazu das Haltepunktfenster auf (Befehl BEARBEITEN/HALTEPUNKTE) und deaktivieren Sie ihn, indem Sie auf das Häkchen neben dem Haltepunkt klicken.

Spezielle Haltepunkte

Im Dialogfeld HALTEPUNKTE (Aufruf über BEARBEITEN/HALTEPUNKTE) können Sie zwischen drei verschiedenen Haltepunkttypen wählen.

- ➡ **Pfad-Haltepunkte** unterbrechen die Programmausführung an einer bestimmten Stelle im Programmcode. Dies sind die Haltepunkte, die Sie mit Hilfe der Taste **[F9]** setzen. Sie können einem Pfad-Haltepunkt eine Bedingung hinzufügen. Das Programm wird in diesem Fall nur dann unterbrochen, wenn die angegebene Bedingung erfüllt ist.
- ➡ **Daten-Haltepunkte** unterbrechen die Programmausführung, wenn sich der Wert des angegebenen Ausdrucks verändert. Daten-Haltepunkte implizieren eine Speicherüberwachung und können den Debug-Prozeß verlangsamen.

- ➡ Der NACHRICHTEN-Haltepunkt unterbricht die Programmausführung, wenn die angegebene Nachricht von einer Ihrer Windows-Prozeduren empfangen wurde. Für API-Programme sind dies die von Ihnen definierten Fensterfunktionen; in MFC-Programmen können Sie die entsprechenden botschaftsverarbeitenden Methoden der MFC angeben.

Programmausführung

Die Programmausführung kann ebenfalls unterbrochen werden, indem Sie aus dem Menü DEBUG den Eintrag ANHALTEN auswählen.

Interessant ist aber nicht nur das Anhalten des Programms im Debugger, sondern auch die schrittweise Ausführung:

Debug-Befehl	Kürzel	Beschreibung
In Aufruf springen	F11	Führt die jeweils nächste Programmzeile Ihrer Quelldatei oder die nächste Anweisung in dem Fenster DISASSEMBLIERUNG aus. Ist in dieser Programmzeile ein Funktionsaufruf enthalten, verzweigt die Einzelschrittausführung in diese Funktion.
Aufruf als ein Schritt	F10	Ähnlich wie IN AUFRUF SPRINGEN. Die Einzelschrittausführung verzweigt jedoch nicht in Funktionen. Statt dessen werden Funktionen vollständig ausgeführt.
Ausführen bis Rücksprung	⇧ + F11	Führt die aktuelle Funktion aus. Bisweilen ist es möglich, daß das Programm während der Ausführung einer Systemfunktion unterbrochen wird. Wählen Sie in solchen Situationen wiederholt diesen Befehl, bis Sie sich wieder in Ihrem Programmcode befinden.
Ausführen bis Cursor	Strg + F10	Führt das Programm bis zu der Position in dem Quellcode- oder Assembler-Fenster aus, an der sich ein Haltepunkt oder die Schreibmarke befindet.

*Tabelle 4.1:
Befehle zur
schrittweisen
Ausführung*

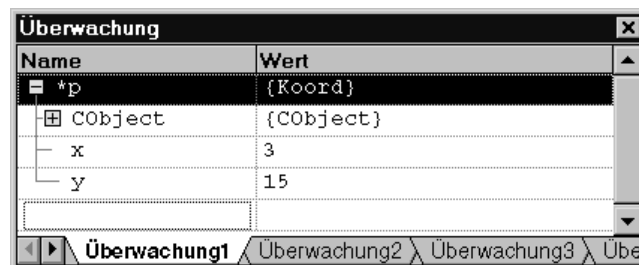
4.2.4 Die Debug-Fenster

Die Debug-Fenster werden über das Menü Ansicht aufgerufen

Während eines Testlaufs führt Visual Studio Debug-Informationen in verschiedenen Debug-Fenstern auf. Diese Fenster werden über die entsprechenden Einträge des Menüs ANSICHT geöffnet, sofern sie nicht bereits angezeigt werden. Die Fenster können in der gewohnten Ansicht oder gedockt dargestellt werden. In der gedockten Ansicht werden sie ebenfalls in den Kontextmenüs der Symbolleisten aufgeführt. (Sie öffnen das Kontextmenü einer Symbolleiste, indem Sie den Mauszeiger auf einen freien Bereich der Leiste bewegen und mit der rechten Maustaste klicken.)

Das Fenster Überwachung

Abbildung 4.3:
Das Fenster
Überwachung



In diesem Fenster können die Werte ausgewählter Variablen überwacht werden.

- ➡ Um eine neue Variable zur Überwachung einzurichten, doppelklicken Sie einfach in die leere Schablone in der Spalte NAME, und geben Sie den Namen der zu überwachenden Variablen (oder einen Ausdruck) ein. Alternativ können Sie einen Namen auch per Drag&Drop aus Ihrem Quelltext in das Feld ziehen.
- ➡ In der Spalte WERT wird der aktuelle Inhalt der Variablen angezeigt (für Ausdrücke wird der Wert des berechneten Ausdrucks angezeigt).
Wenn Sie hier einen Wert ändern, wird die Änderung an das Programm weitergereicht. Sie können auf diese Weise das Programm schnell für verschiedene Variablenwerte austesten.
- ➡ Um eine Variable aus der Überwachung zu streichen, markieren Sie den Eintrag der Variablen, und drücken Sie die **[Entf]**-Taste.
- ➡ Zur übersichtlicheren Verwaltung der zu überwachenden Ausdrücke stellt Ihnen das Fenster vier einzelne Seiten zur Verfügung.

Um sich schnell über den Inhalt einer aktuellen Variablen zu informieren, können Sie statt dieses Fensters auch die Debugger-Unterstützung des Texteditors nutzen und einfach den Mauszeiger für eine Weile auf den Bezeichner der Variablen rücken.



Das Fenster Aufrufliste

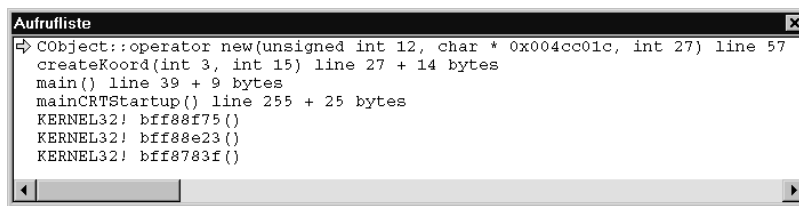


Abbildung 4.4:
Das Fenster
Aufrufliste

Dieses Fenster zeigt Ihnen an, welche Funktionen bis zum Erreichen der aktuellen Ausführungsposition aufgerufen (und noch nicht beendet) wurden. Sie können in diesem Fenster also die Aufrufabfolge der Funktionen (einschließlich der Parameterwerte) und den aktuellen Zustand des Programm-Stacks kontrollieren.

Das Fenster Speicher

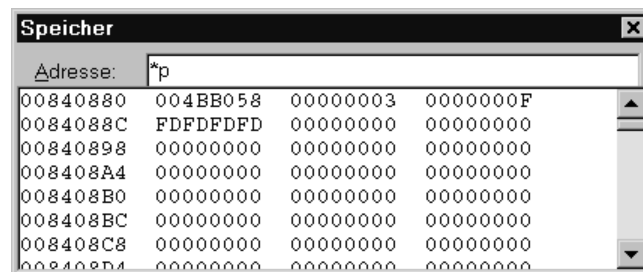


Abbildung 4.5:
Das Fenster
Speicher

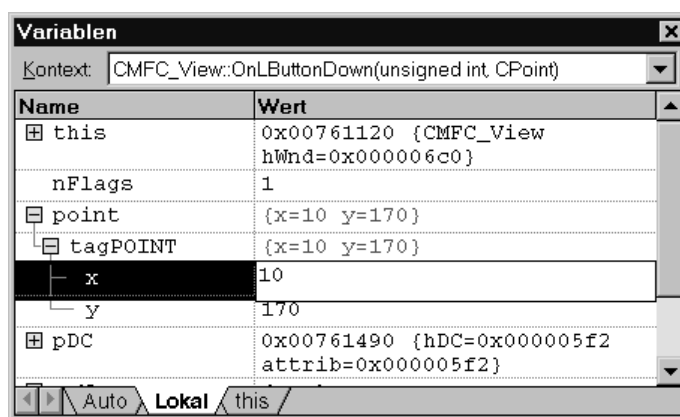
Dieses Fenster liefert Ihnen verschiedene Ansichten Ihres Speichers. In der Abbildung ist `p` beispielsweise ein Zeiger auf eine von `CObject` abgeleitete Klasse mit zwei Integer-Datenelementen `x` und `y`, die die Werte 3 (hexadezimal 0000 0003) und 15 (hexadezimal 0000 000F) haben.

- ➡ Um zwischen den verschiedenen Darstellungen des Speicherinhalts zu wechseln, klicken Sie mit der rechten Maustaste in das Fenster, um das zugehörige Kontextmenü aufzurufen, und wählen Sie einen der folgenden Befehle: Byte-Format (In dieser Ansicht wird zusätzlich versucht, den Speicherinhalt als Text zu interpretieren.), Kurzes Hex-Format (Short), Langes Hex-Format (Long).

- ➡ Um zu einer bestimmten Adresse zu springen, geben Sie die Adresse in das gleichnamige Feld ein (beispielsweise im Hexadezimalformat (0x00840885) oder als Zeigervariable), oder ziehen Sie eine Adresse (als direkte Adresse oder als Variablennamen) per Drag&Drop aus dem Quelltext oder einem anderen Debug-Fenster in das Speicher-Fenster.

Das Fenster Variablen

Abbildung 4.6:
Das Fenster
Variablen



Mit Hilfe dieses Fensters können Sie sich schnell und bequem über die Inhalte der aktuellen Variablen informieren.

Das Fenster verfügt über drei verschiedene Seiten:

- ➡ Die Registerkarte AUTO zeigt Informationen über die Variablen der aktuellen Anweisung sowie der vorangegangenen Anweisung an.
- ➡ Die Registerkarte LOKAL zeigt die Namen und Werte aller lokalen Variablen der aktuellen Funktion an. Wenn Sie durch das Programm gehen, werden je nach Kontext andere Variablen angezeigt.
- ➡ Die Registerkarte THIS zeigt Namen und Inhalt des Objekts an, auf das der `this`-Zeiger gerichtet ist. Alle Basisklassen des Objekts werden automatisch eingeblendet.

Sie selbst haben keine Möglichkeit, Variablen zur Überwachung in diesem Fenster einzurichten – benutzen Sie dazu das Fenster ÜBERWACHUNG.

Das Fenster Register

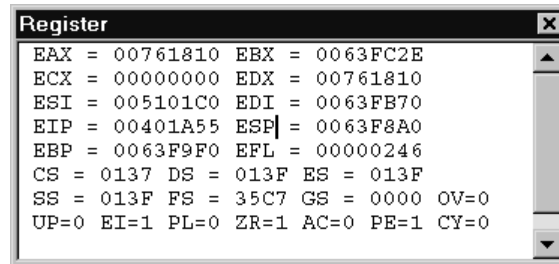


Abbildung 4.7:
Das Fenster
Register

Dieses Fenster zeigt die Namen und aktuellen Werte der plattformspezifischen CPU-Register und Attribute sowie den Fließkomma-Stack an.

Das Fenster Disassemblierung

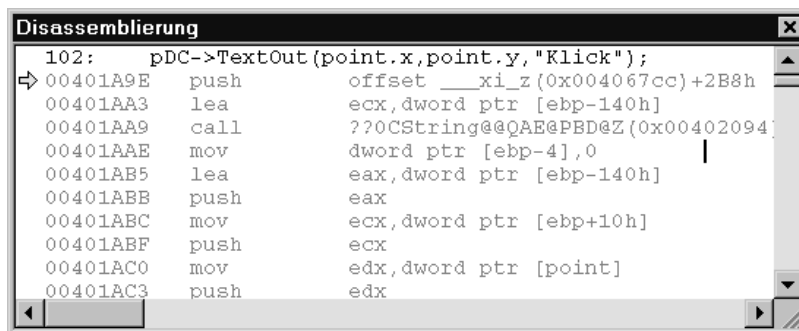


Abbildung 4.8:
Das Feld Dis-
assemblierung

Dieses Fenster zeigt die Assembleranweisungen an, die der Compiler für den Quellcode generiert.

- ➔ Wenn Sie im Kontextmenü des Fensters den Befehl QUELLCODE-ANMERKUNG gesetzt haben, wird über den Assembleranweisungen der zugehörige Quelltext angezeigt.
- ➔ Ein spezielles Feature des Fensters DISASSEMBLIERUNG ist in dem Kontextmenü aufgeführt, das nach einem Klick auf die rechte Maustaste geöffnet wird. Die Option NÄCHSTE ANWEISUNG SETZEN ermöglicht Ihnen, den Befehlszeiger zu verändern. Dieser wird nach einer Auswahl der Option auf die Adresse der Anweisung gesetzt, die sich unter der Schreibmarke befindet. Sie können dieses Feature verwenden, um bestimmte Abschnitte Ihres Programmcodes zu überspringen. Sie sollten mit dieser Option jedoch sehr vorsichtig umgehen. Setzen Sie den Befehlszeiger mit NÄCHSTE ANWEISUNG SETZEN niemals auf die Anweisung einer anderen Funktion, und achten Sie darauf, daß der Stack die korrekten Wer-

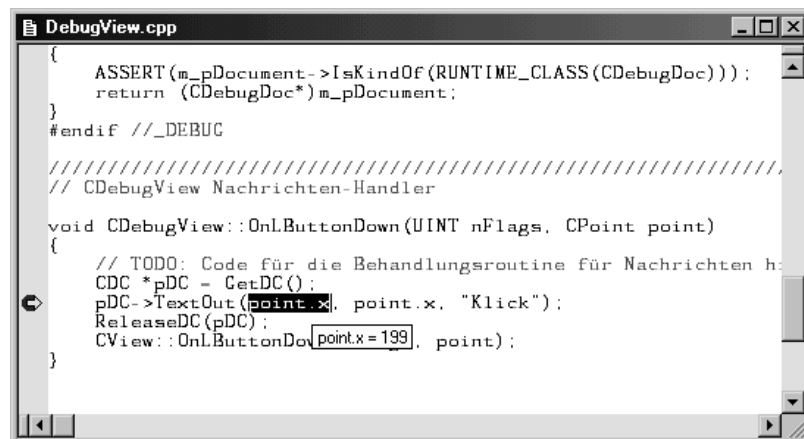
te enthält. Sollten Sie diese Hinweise nicht berücksichtigen, sind die Ergebnisse unberechenbar, so daß die im Testlauf befindliche Applikation möglicherweise abstürzt.

4.2.5 Schnellüberwachung im Editorfenster

Bisweilen ist es erforderlich, den Wert bestimmter Symbole zu ermitteln, die nicht in den Fenstern VARIABLEN und ÜBERWACHUNG aufgeführt sind. Der Visual-C++-Debugger stellt zu diesem Zweck zwei Hilfsmittel zur Verfügung: die SCHNELLÜBERWACHUNG und DATENINFO.

DatenInfo

Abbildung 4.9:
DatenInfo im
Debug-Modus



DatenInfo ist ähnlich dem bekannten QuickInfo, das Hinweise zu Schaltflächen oder anderen Anwenderschnittstellen anzeigt, wenn der Mauszeiger für eine kurze Zeit darüber angeordnet wird. Bewegen Sie den Mauszeiger während des Debug-Vorgangs auf den Namen eines Symbols, das ausgewertet werden kann, wird der Wert des Symbols in einem kleinen gelben Kästchen angezeigt (siehe Abbildung 4.9).

Schnellüberwachung

Genügen Ihnen die Informationen des DatenInfos nicht, können Sie den Dialog SCHNELLÜBERWACHUNG aufrufen (siehe Abbildung 4.10). Wählen Sie dazu aus dem Menü DEBUG den Eintrag SCHNELLÜBERWACHUNG aus. Befindet sich die Schreibmarke auf einem Symbolnamen, erscheint das Symbol automatisch in dem Dialog. Ist statt dessen ein Ausdruck markiert, wird dieser in dem Dialog aufgeführt.

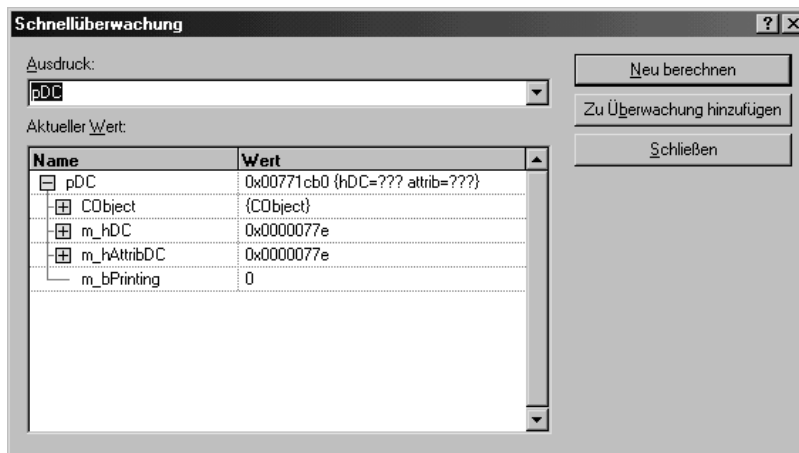


Abbildung 4.10:
Schnellüberwachung

Die Funktion und der Aufbau des Dialogs SCHNELLÜBERWACHUNG gleicht der Funktion und dem Aufbau des Fensters ÜBERWACHUNG. Sie verwenden dieses Fenster, um die Werte der Variablen eines einfachen Datentyps zu verändern.

4.2.6 Bearbeiten und Fortfahren

Kompilierte Programme haben üblicherweise den Nachteil, daß sie recht unbequem zu debuggen sind. Wurde während einer Debug-Sitzung ein Fehler entdeckt, mußte der Programmierer bislang die Debug-Sitzung beenden, den Fehler im Editor beseitigen, die Anwendung neu kompilieren und dann erneut im Debugger austesten.

Bequemer debuggen

Seit VC 6.0 gibt es nun die Möglichkeit, ein Programm während des Debuggens zu bearbeiten und dann mit der aktuellen Debug-Sitzung fortzufahren.

Haben Sie während des Debuggens einen Fehler ausfindig gemacht:

1. Ändern Sie den Quelltext
2. Rufen Sie den Befehl **DEBUG/CODE-ÄNDERUNGEN ZUWEISEN** auf, und warten Sie bis der Code automatisch neu kompiliert wurde. (Wenn Sie die Debug-Sitzung nach einem Haltepunkt fortsetzen, können Sie sich den Befehlsaufruf sparen.)
3. Fahren Sie mit Ihrer Debug-Sitzung fort.

4.3 Weitere Debug-Techniken und -Tools

Der integrierte Visual-C++-Debugger und die Debug-Features in den MFC-Klassen stellen noch verschiedene weitere Debug-Techniken zur Verfügung.

4.3.1 Meldungsfenster

Manchmal ist der Einsatz des integrierten Debuggers störend oder übertrieben aufwendig. In solchen Fällen kann Sie ein gut platziertes Meldungsfenster bei der Suche nach einem schwer auffindbaren Fehler unterstützen. Mußten Sie beispielsweise feststellen, daß eine Funktion mit der Bezeichnung `foo`, der zwei Parameter übergeben werden, nicht korrekt in Ihrer Anwendung arbeitet, können Sie sehr einfach ermitteln, ob die Funktion die gewünschten Parameter erhält. Dazu fügen Sie der Funktion einen `AfxMessageBox`-Aufruf wie folgt hinzu:

```
...
char temp[100];
wsprintf(temp, "Calling foo(x = %d, y = %d)", x, y);
AfxMessageBox(temp);
foo(x, y);
```

4.3.2 TRACE-Diagnosemakros

Wenn eine MFC-Applikation mit Debug-Bibliotheken kompiliert wird, erzeugen einige MFC-Funktionen eine Debug-Ausgabe. Sie können die Debug-Ausgabe auch in Ihrem eigenen Programmcode verankern, indem Sie die Makros `TRACE`, `TRACE0`, `TRACE1`, `TRACE2` oder `TRACE3` verwenden. Die Debug-Ausgabe wird an ein vordefiniertes Objekt vom Typ `CDumpContext` gesendet, das mit `afxDump` bezeichnet ist. Außerdem erscheint die Ausgabe gewöhnlich in dem Ausgabefenster von Visual Studio (vorausgesetzt, die Anwendung wird im Debugger ausgeführt). Um Debug-Informationen einsehen zu können, müssen Sie das Register `DEBUG` in diesem Fenster öffnen.

Um beispielsweise die an die Funktion `foo` übergebenen Werte zu überprüfen, könnten Sie den folgenden Programmcode schreiben:

```
...
TRACE2("Calling foo(x = %d, y = %d)", x, y);
foo(x, y);
```

Beachten Sie bitte, daß diese Art der Debug-Ausgabe nur dann erfolgen kann, wenn Sie Ihre Applikation für den Debug-Vorgang kompiliert haben. Ihre Applikation muß außerdem auch dann über den Debugger gestartet worden sein, wenn Sie keine anderen Debug-Features nutzen möchten.

Sie sollten die Makros `TRACE0`, `TRACE1`, `TRACE2` und `TRACE3` verwenden, wenn Sie Speicherkapazität benötigen, da diese weniger Speicher als `TRACE` beanspruchen.



Die `TRACE`-Makros erfüllen so lange keine Funktion, bis die Applikation für den Debug-Vorgang erstellt wurde.

4.3.3 Das ASSERT-Makro

Das Makro `ASSERT` unterbricht die Programmausführung, wenn eine bestimmte Bedingung falsch (`false`) ist. Dieses Makro kann in Debug-Versionen Ihrer Applikation verwendet werden, um beispielsweise zu prüfen, ob eine Funktion die korrekten Parameter erhalten hat:

```
void foo(int x)
{
    ASSERT (x >= 0 && x < 100);
    ...
}
```

Das `ASSERT_VALID`-Makro prüft, ob ein Zeiger auf ein zulässiges, von `CObject` abgeleitetes Objekt verweist. Verwenden Sie beispielsweise eine Funktion mit der Bezeichnung `GetDocument`, die ein Zeiger auf ein `CMyDoc`-Objekt zurückgibt, prüfen Sie den Zeiger wie folgt:

```
CMyDoc *pDoc;
pDoc = GetDocument();
ASSERT_VALID(pDoc);
...
```

`ASSERT`-Makros unterbrechen die Programmausführung, nachdem eine Meldung ausgegeben wurde, die die Nummer der Zeile angibt, in der die Assertion nicht erfüllt wurde.

In Programmen, die nicht für den Debug-Vorgang erstellt wurden, haben Assertionsmakros keine Funktion.

4.3.4 Objekt-Dump

Die `CObject`-Klasse enthält eine Member-Funktion mit der Bezeichnung `Dump`, die den Inhalt eines Objekts an die Debug-Ausgabe übergibt (vorausgesetzt, die Anwendung wird im Debugger ausgeführt). Sollten Sie beabsichtigen, dieses Feature zu nutzen, implementieren Sie die `Dump`-Member-Funktion für Klassen, die Sie direkt oder indirekt von `CObject` ableiten.

Enthält Ihre Applikation beispielsweise eine von `CDocument` abgeleitete Klasse mit der Bezeichnung `CMyDoc`, welche die beiden Member-Variablen `m_x` und `m_y` verwendet, könnte Ihre `CMyDoc::Dump`-Implementierung wie folgt aufgebaut sein:

```

#ifdef _DEBUG
void CMyDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
    dc << "m_x = " << m_x << '\n';
    dc << "m_y = " << m_y << '\n';
}
#endif // _DEBUG

```

4.3.5 Ermitteln von Speicherverlust mit der Klasse CMemoryState

Die Klasse CMemoryState ermittelt den Speicherverlust, der entsteht, wenn die C++-Operatoren New oder Delete unsachgemäß eingesetzt werden. Erstellen Sie ein CMemoryState-Objekt, und rufen Sie dessen Member-Funktion Checkpoint auf, um ein Abbild des reservierten Speichers zu erstellen. Später können Sie die Member-Funktion DumpAllObjectsSince aufrufen, um die Inhalte aller Objekte seit dem letzten Aufruf von Checkpoint zu löschen. Um beispielsweise jedes von foo reservierte Objekt zu löschen, dessen Reservierung nicht vollständig von der Funktion rückgängig gemacht werden konnte, verwenden Sie den folgenden Programmcode:

```

...
CMemoryState memState;
memState.Checkpoint();
foo(x, y);
memState.DumpAllObjectsSince();

```



Um die Debug-Ausgabe im Ausgabefenster angezeigt zu bekommen, muß die betreffende Anwendung im Debugger ausgeführt werden (**F5**).

Wenn Sie sich nicht den vollständigen Dump der reservierten Objekte anzeigen lassen möchten, können Sie die Member-Funktion DumpStatistics verwenden. DumpStatistics wird aufgerufen, nachdem die Differenz zwischen zwei Speicherzuständen mit Hilfe der Member-Funktion Difference ausgewertet wurde. Dieses Verfahren erfordert drei CMemoryState-Objekte. Die ersten beiden Objekte erstellen die Abbilder der beiden Speicherzustände, während das dritte Objekt die Differenz ermittelt. Betrachten Sie dazu auch das folgende Beispiel:

```

// Speicher.cpp  Konsolen-Anwendung mit MFC-Einbindung
#include <afxwin.h>
#define new DEBUG_NEW

CMemoryState msOld, msNew, msDif;
class Koord : public CObject
{
public:
    int x,y;
    Koord(int X, int Y) {x = X; y = Y;}
};

```

```

Koord* createKoord(int x, int y)
{
    // Achtung!!
    // p_tmp ist lokal und wird mit Fkt. aufgelöst
    // dyn. erzeugtes Koord-Objekt bleibt in Speicher
    Koord *p_tmp = new Koord(3,15);
    return p_tmp;
}

void main()
{
    msOld.Checkpoint();
    Koord *p = createKoord(3,15);
    msNew.Checkpoint();

    msOld.DumpAllObjectsSince(); //gibt allokiertes Objekt aus
    msNew.DumpAllObjectsSince(); //gibt keine Objekte aus

    //Statistik ausgeben
    msDif.Difference( msOld, msNew );
    msDif.DumpStatistics();
    // bis hierher alles ok

    // Zeiger p wird umgesetzt!
    // Koord-Objekt (3,15) aus Fkt-Aufruf bleibt in Speicher
    p = new Koord(3,14);
    delete p;                // löscht nur (3,14)-Objekt
}

```

CMemoryState-Objekte können Speicherverluste, die durch inkorrekte Aufrufe von malloc/free, GlobalAlloc/GlobalFree oder LocalAlloc/LocalFree verursacht wurden, nicht ermitteln.



4.3.6 Remote-Testlauf

Remote-Testläufe ermöglichen das Debuggen von Programmcode, der auf einem anderen Rechner ausgeführt wird. Der lokale PC und der Remote-Rechner sind über ein serielles Kabel oder ein lokales Netzwerk miteinander verbunden. Der lokale Rechner führt Visual Studio und den integrierten Debugger aus, während die zu testende Applikation auf dem Remote-PC mit dem Visual-C++-Debug-Monitor ausgeführt wird.

Damit ein Remote-Testlauf durchgeführt werden kann, muß der Remote-Rechner den Visual-C++-Debug-Monitor *msvcmon.exe* ausführen. Damit dieses Programm korrekt ausgeführt wird, müssen die folgenden DLLs auf dem Remote-Rechner installiert werden: *msvcrt.dll*, *dm.dll*, *msvcp60.dll* und *msdis110.dll*. Kopieren Sie diese Dateien in ein Verzeichnis, das als Systempfad vermerkt ist (z.B. in das Windows-Systemverzeichnis).

Abbildung 4.11:
Remote-
Debuggen

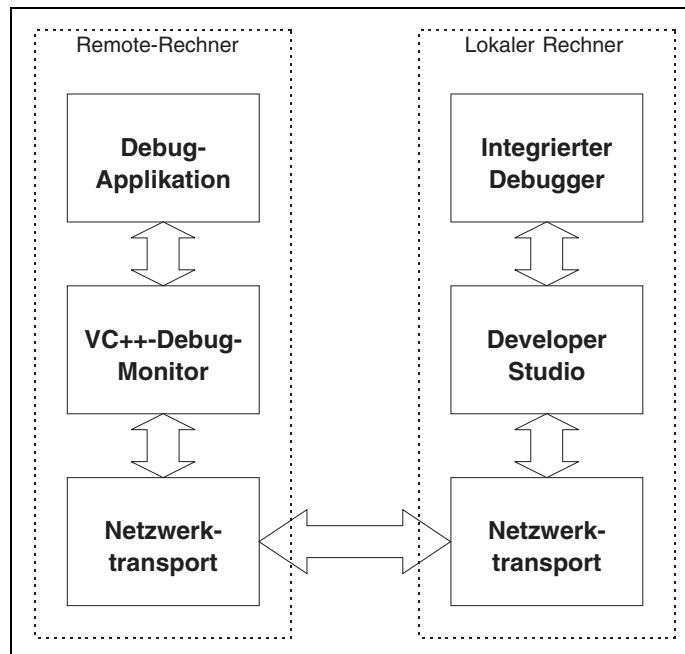
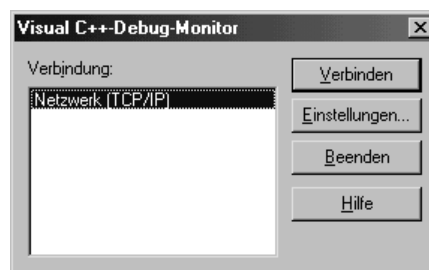


Abbildung 4.12:
Der Visual-C++-
Debug-Monitor



Starten Sie zunächst den Debug-Monitor auf dem Remote-Computer. Der Debug-Monitor wird in einem Dialog dargestellt (siehe Abbildung 4.12), in dem Sie die erforderlichen Einstellungen vornehmen. Sie können den Remote-Testlauf über eine TCP/IP-Netzwerkverbindung ausführen. Drücken Sie die Schaltfläche **EINSTELLUNGEN**, um die Details der Verbindung zu bestimmen, nachdem Sie den gewünschten Verbindungstyp ausgewählt haben.

Nachdem ein Remote-Testlauf vollständig konfiguriert wurde, drücken Sie bitte die Schaltfläche **VERBINDEN**. Der Debug-Monitor wartet nun auf eine eingehende Verbindung.

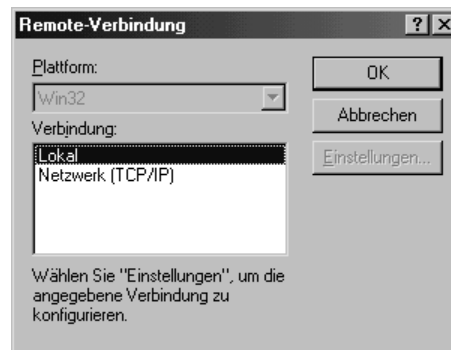


Abbildung 4.13:
Der Dialog
Remote-Ver-
bindung

Sie müssen Visual Studio auch auf dem lokalen Rechner für den Remote-Testlauf konfigurieren. Wählen Sie dazu den Eintrag REMOTE-VERBINDUNG DES DEBUGGERS aus dem Menü ERSTELLEN aus. Daraufhin wird der Dialog REMOTE-VERBINDUNG (siehe Abbildung 4.13) angezeigt, in dem Sie den Verbindungstyp angeben und die Verbindungsoptionen nach einem Klick auf die Schaltfläche EINSTELLUNGEN setzen.

Wenn Sie für den Debug-Vorgang eine TCP/IP-Verbindung einrichten, werden Sie aufgefordert, ein Paßwort anzugeben. Stellen Sie sicher, daß dieses Paßwort für beide Rechner, den lokalen PC und den Remote-Rechner, gleich ist.



Der wesentliche Vorteil des Remote-Debuggens besteht darin, daß die Applikation auf einem Computer ausgeführt wird, der nicht von dem Debugger beeinflusst wird. Remote-Testläufe sind beispielsweise für Applikationen geeignet, die den gesamten Windows-Bildschirm benötigen und über die Tastatur gesteuert werden – wie Full-Screen-Spiele.

4.3.7 Just-in-Time-Testläufe

Just-in-Time-Debuggen beschreibt die Fähigkeit des Visual-C++-Debuggers, ein Programm zu debuggen, dessen Ausführung durch einen Ausnahmefehler unterbrochen wurde. Just-in-Time-Testläufe sind überwiegend für den Debug-Vorgang solcher Applikationen geeignet, die nicht aus dem integrierten Debugger heraus gestartet wurden.

Just-in-Time-Testläufe werden in dem Dialog OPTIONEN eingeschaltet, den Sie aus dem Menü EXTRAS auswählen. Selektieren Sie dort das Register DEBUG, und aktivieren Sie das Kontrollkästchen JUST-IN-TIME-DEBUGGING.

4.4 Zusammenfassung

Der Quellcode-Browser ist ein sehr wichtiges Entwicklungswerkzeug von Visual C++, mit dem Sie sich über

- Definitionen und Referenzen von ausgewählten Symbolen
- den Inhalt von Quelltextdateien
- Klassenhierarchien
- und Funktionsaufrufe

informieren können.

Um den Quellcode-Browser verwenden zu können, müssen Ihrem Projekt Browse-Informationen hinzugefügt werden. Sie können die dazu notwendige Option in dem Dialog PROJEKT-EINSTELLUNGEN, Register C/C++ und BROWSE-INFORMATION, vornehmen.

Das Fenster des Quellcode-Browsers wird über den Eintrag QUELLCODE-BROWSER im Menü EXTRAS oder über verschiedene Kontextmenüeinträge aufgerufen.

Der in Visual Studio integrierte Debugger wird ausgeführt, wenn Sie eine Applikation über eine der Debug-Anweisungen im Menü ERSTELLEN starten. Der Debugger verfügt über verschiedene Ansichtsfenster, über die Sie den Zustand Ihrer Anwendung überwachen können. Dazu zählen Quellcode-Fenster sowie die Fenster VARIABLEN, ÜBERWACHUNG, REGISTER, AUFRUFE, SPEICHER und DISASSEMBLIERUNG.

Sie bereiten eine Anwendung auf den Debug-Vorgang vor, indem Sie sie mit den erforderlichen Flags kompilieren und binden. Dies geschieht für MFC-Applikationen, die mit dem Anwendungsassistenten erzeugt wurden, automatisch. Für solche Applikationen erstellt der Anwendungsassistent eine Debug-Konfiguration und deklariert diese als Standardkonfiguration.

Während des Debuggens einer Anwendung kann deren Ausführung unterbrochen werden. Setzen Sie dazu Haltepunkte, und nutzen Sie die verschiedenen Debug-Befehle zur schrittweisen Ausführung.

C++ kennt viele Debug-Techniken, die Sie zum Testen von MFC- und anderen Applikationen verwenden können. Dazu gehören beispielsweise die TRACE- und ASSERT-Makros sowie die Funktion `CObject::Dump`. Mit Objekten vom Typ `CMemoryState` können Sie feststellen, ob es bei der Ausführung Ihrer Anwendung zu Speicherverlusten kommt.

Eine besondere Debug-Technik ist das Remote-Debuggen. Remote-Testläufe ermöglichen Ihnen, eine Anwendung auf einem entfernten Rechner mit Hilfe des Debuggers auf dem lokalen Computer zu debuggen.

Visual C++ ermöglicht Just-In-Time-Debuggen. Auf diese Weise können Sie Anwendungen testen, die außerhalb der Umgebung des integrierten Debuggers abgestürzt sind.

Optimierung

Kapitel

5

Sollten Sie auch nach dem Debuggen noch nicht die Lust am Programmieren verloren haben, können Sie darangehen, Ihr Programm nach verschiedenen Kriterien zu optimieren. Visual C++ unterstützt Sie dabei mit drei verschiedenen Werkzeugen:

- ➡ dem Compiler
- ➡ dem Profiler
- ➡ dem Analyzer

5.1 Der Compiler

Die einfachste Form der Optimierung ist, die ganze Arbeit dem Compiler zu überlassen.

1. Öffnen Sie Ihr Projekt und rufen Sie das Fenster für die Projekteinstellungen auf (Befehl PROJEKT/EINSTELLUNGEN).
2. Wählen Sie auf der Seite C/C++ eine der Einstellungen im Feld OPTIMIERUNGEN.
3. Deaktivieren Sie die Debug-Einstellungen, die den Umfang der Zieldateien unnötig vergrößern.

Die Assistenten des Visual Studios legen neue Projekte standardmäßig mit zwei Konfigurationen an: Debug und Release. Die Release-Version ist für die Endversion des Programms gedacht und bewirkt, daß das Projekt optimiert und ohne Debug-Informationen kompiliert wird.

:~)
TIP

5.2 Der Profiler

Aufruf über Erstellen/Profil Der Visual-C++-Profiler ist ein Hilfsmittel zur Analyse der Laufzeit-Performance. Der Profiler kann zur Erstellung von Funktionsanalysen und Zeilenanalysen eingesetzt werden. Funktionsanalysen ermitteln, wie Funktionen in Ihrem Programm ausgeführt werden. Zeilenanalysen führen eine ähnliche Untersuchung für einzelne Zeilen des Quellcodes durch. Sie müssen Ihren Programmcode mit Debug-Informationen kompilieren lassen, um eine Zeilenanalyse durchführen zu können. Für Funktionsanalysen werden Debug-Informationen ignoriert.



Haben Sie die benutzerdefinierte Installation von Visual C++ ausgeführt, ist der Profiler möglicherweise nicht auf Ihrem System vorhanden. In diesem Fall können Sie nicht auf die Features des Profilers zugreifen. Wiederholen Sie das Installationsprogramm, um den Profiler einzurichten.

5.2.1 Vorbereiten einer Anwendung für den Profiler

Um einen Profilerlauf durchführen zu können, müssen Sie

- Im Dialog PROJEKT-EINSTELLUNGEN. (Aufruf PROJEKT/EINSTELLUNGEN) auf der Registerseite LINKER, Kategorie ALLGEMEIN, die Option PROFILER-LAUF ERMÖGLICHEN setzen. (Kommandozeilenoption/PROFILE).

Um zuverlässige Ergebnisse zu erhalten, sollten Sie folgende Empfehlungen beherzigen:

Vermeiden Sie Eingaben durch den Benutzer Werden innerhalb einer Funktion Eingaben vom Benutzer verlangt, sei es durch `scanf`, `cin` oder ein Dialogfeld, wird die für die Eingabe erforderliche Zeit der Funktion hinzuaddiert. Der dadurch zustande kommende Wert kann weit über dem tatsächlich von den Anweisungen benötigten Zeitverbrauch liegen und eine realistische Einschätzung des Bereichs unmöglich machen. Vermeiden Sie daher Eingaben durch den Benutzer, indem Sie

- Eingabedaten im Programm vorgeben (zum Beispiel mit `random`),
- Eingabedaten aus einer Datei einlesen,
- Beim Programmablauf Verzweigungen mit Eingaben meiden,
- Anweisungen, in denen Eingaben vom Benutzer verlangt werden, aus der Erfassung ausgrenzen (PREP-Optionen `/EXCALL` `/INC`).

Ausgaben sind ebenfalls immer sehr zeitraubend und können die Analyse verfälschen.

Vermeiden Sie Ausgaben

Funktionen, in denen nur wenig Zeit verbracht wird, liefern ungenaue Werte und sind nur schlecht untereinander zu vergleichen. Stellen Sie also sicher, daß in den interessierenden Funktionen genügend Zeit verbracht wird.

Sorgen Sie für umfangreiches statistisches Material

Konfrontieren Sie Ihr Programm nicht mit zu trivialen Aufgaben. (Wenn Sie beispielsweise Schleifen berechnen lassen, sorgen Sie für ausreichend viele Iterationen.)

Lassen Sie das Programm mehrmals hintereinander ablaufen, wobei Sie ab dem zweiten Lauf den Profiltyp VEREINIGEN wählen, um die Ergebnisse der einzelnen Läufe aufzaddieren zu lassen

5.2.2 Der Profiler-Vorgang

Wie arbeitet der Profiler? Der Profiler besteht aus drei untergeordneten Programmen, die über die Kommandozeile aufgerufen werden. Die Programme sind mit *PREP*, *PROFILE* und *PLIST* bezeichnet. Eine einfache Darstellung der Arbeitsweise dieser Hilfsmittel ist in Abbildung 5.1 dargestellt.

PREP wird während des Profiler-Vorgangs zweimal ausgeführt. Das Programm liest zunächst die ausführbare Datei der Anwendung ein und erstellt eine PBI- sowie eine PBT-Datei. Die PBI-Datei bildet die Eingabequelle des Hilfsmittels *PROFILE*. Dieses Programm startet und analysiert die Applikation. Die Ergebnisse werden in eine PBO-Datei geschrieben. Die PBO- und PBT-Dateien dienen der anschließend ausgeführten Anwendung *PREP* als Eingabequelle. *PREP* erzeugt eine neue PBT-Datei. Schließlich wird *PLIST* zur Erstellung der Ergebnisse verwendet.

Der Profiler wird gewöhnlich über Batch-Dateien gestartet, die *PREP*, *PROFILE* und *PLIST* in der erforderlichen Reihenfolge aufrufen (beispielsweise *LCover.bat* und *FCover.bat* aus dem VisualC++-Unterverzeichnis *BIN*).

Die Ausgabe von *PLIST* ist eine schriftliche Zusammenfassung der Ergebnisse der Analyse.

5.2.3 Profiltypen

Der Profiler kann unterschiedliche Analysen durchführen. Einige dieser Analysen werden über den Dialog PROFIL ausgewählt (Aufruf über ERSTELLEN/PROFIL), andere erfordern die Verwendung von Batch-Dateien, die sich gewöhnlich in dem Verzeichnis *\bin* befinden.

Abbildung 5.1:
Arbeitsweise
des Profilers

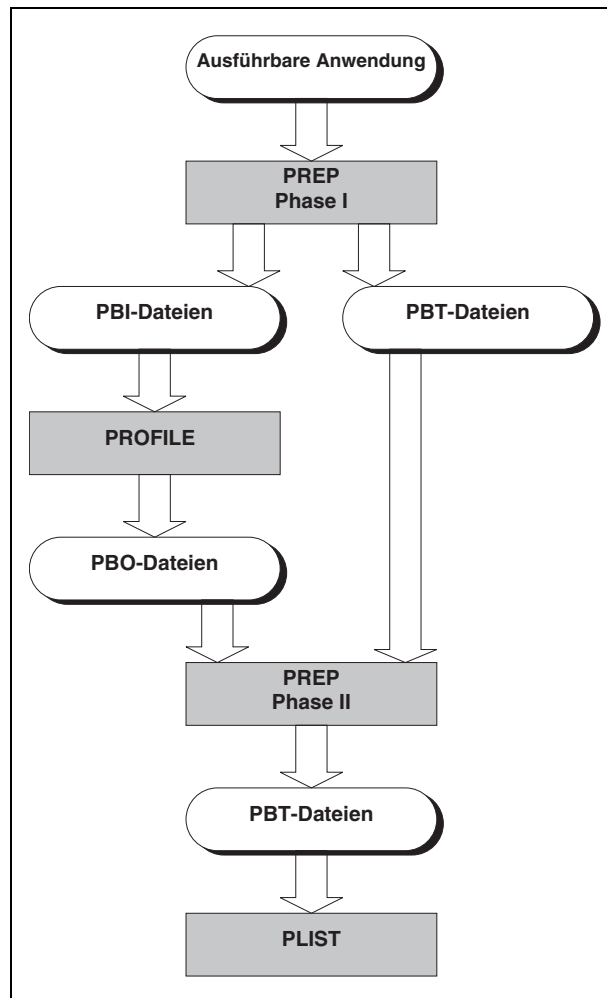
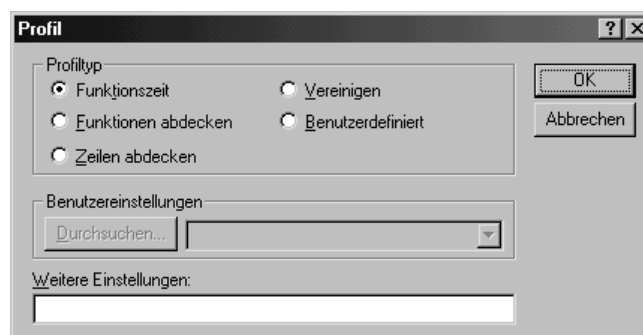


Abbildung 5.2:
Profiler starten



Profiltyp	Beschreibung
Funktionszeit	Zeigt für die überwachten Funktionen die Anzahl der Aufrufe und die verbrauchte Zeit an (einmal mit und einmal ohne die Zeit, die in Unterfunktionen verbraucht wurde). (Batch-Datei <i>Ftime.bat</i>)
Funktionen abdecken	Zeigt an, welche Funktionen überhaupt aufgerufen werden. Sie verwenden diese Option, um beispielsweise zu prüfen, ob bestimmte Abschnitte Ihres Programmcodes ausgeführt wurden. (Batch-Datei <i>FCover.bat</i>)
Zeilen abdecken	Zeigt an, welche Zeilen ausgeführt werden. (Batch-Datei <i>LCover.bat</i>)
Vereinigen	Addiert zu dem Ergebnis der neuen Analyse das Ergebnis der letzten Analyse (zur statistischen Absicherung; Voraussetzung ist, daß die Analysen von der gleichen Art sind).
Benutzerdefiniert	Zum Aufruf eigener Batch-Dateien (werden im Feld BENUTZEREINSTELLUNGEN eingegeben). Zur Aufstellung eigener Batch-Dateien laden Sie am besten eine der vorgegebenen Batch-Dateien aus dem VisualC++-Verzeichnis <code>\bin</code> , und überarbeiten diese, indem Sie andere Optionen für die Aufrufe der Profiler-Befehlszeilenprogramme setzen (siehe unten).

Tabelle 5.1:
Profiltypen

Nach der Einstellung des Profiltyps starten Sie den Profilerlauf durch Drücken des OK-Schalters.

Das Ergebnis der Analyse wird nach einiger Zeit im Ausgabefenster auf einer eigenen Seite angezeigt.

Das Programm *PLIST* generiert gewöhnlich eine Ausgabe, die von Ihnen im Ausgabefenster gelesen werden kann. Sie können jedoch mit dem Schalter */T* bestimmen, daß die Ausgabe in einem Datenbankformat generiert werden soll, das von anderen Applikationen importiert werden kann. Mit Hilfe des Analyse-Makros *PROFILER.XLM* für Microsoft Excel, können Sie die Profiler-Ausgabe unter Excel bearbeiten.



5.2.4 Erweiterte Profiler-Einstellungen

Die Analyse Ihrer gesamten Applikation ist nicht immer sinnvoll. Gewöhnlich ist eine Analyse bestimmter Abschnitte Ihres Programmcodes ausreichend. Außerdem wird die Programmausführung während des Profiler-Vorgangs auf diese Weise beschleunigt.

Ihnen stehen drei Methoden zur Auswahl, um erweiterte Optionen zu dem Profiler zu bestimmen.

- ➡ Sie können die Datei *PROFILER.INI* modifizieren.
- ➡ Sie können unter Visual Studio in dem Dialog PROFIL weitere Einstellungen vornehmen.
- ➡ Sie können eigene Batch-Dateien für die Analyse schreiben.

Die Datei Profiler.ini

Die Datei *PROFILER.INI* führt Einstellungen zu dem Profiler in dem Abschnitt *[profiler]* auf. Dort können Sie Bibliothek- (LIB) und Objektdateien (OBJ) angeben, die von dem Profiler-Vorgang ausgeschlossen werden sollen. Ihre Datei *profiler.ini* könnte die folgenden Zeilen enthalten:

```
[profiler]
exclude:user32.lib
exclude:gdi32.lib
```

Die Datei *PROFILER.INI* befindet sich in dem VisualC++-Verzeichnis *\BIN*.

Einstellungen im Dialog Profil

Im Dialog PROFIL haben Sie die Möglichkeit, über das Feld WEITERE EINSTELLUNGEN zusätzliche Parameter anzugeben, die dem Programm *PREP* während des ersten Aufrufs übergeben werden.

Einzelnes Modul überwachen Möchten Sie beispielsweise lediglich ein einzelnes Modul analysieren (beispielsweise das Modul der Datei *myapp.cpp*), können Sie den folgenden Eintrag in dem Feld WEITERE EINSTELLUNGEN vornehmen:

```
/EXCALL /INC MyApp.cpp
```

Einzelne Zeilen überwachen Möchten Sie lediglich einen bestimmten Programmzeilenbereich in der Datei *myapp.cpp* analysieren lassen, können Sie schreiben:

```
/EXCALL /INC MyApp.cpp(30-67)
```



Eine Aufzählung der möglichen Argumente finden Sie in der Online-Hilfe unter dem Indexeintrag *Prep*.

Eigene Batch-Dateien

Schließlich können Sie eigene Batch-Dateien zur Analyse entwickeln. Verwenden Sie die Batch-Dateien, die in dem VisualC++-Verzeichnis *\BIN* installiert sind (*FCOUNT.BAT*, *FCOVER.BAT*, *FTIME.BAT*, *LCOUNT.BAT* und *LCOVER.BAT*) als Vorlage für die Entwicklung eigener Profiler-Batch-Dateien.

5.3 Der Visual Studio Analyzer

Der Visual Studio Analyzer dient speziell zur Optimierung verteilter Anwendungen. Mit der Hilfe des Analyzers können Sie

- den Aufbau einer verteilten Anwendung nachvollziehen
- aufzeigen, was in Ihrer verteilten Anwendung geschieht
- die Laufzeit detailliert aufschlüsseln (beispielsweise auch um die Auswirkung verschieden starker Auslastungen eines Servers zu simulieren)

Der Visual Studio Analyzer wird nur zusammen mit der Enterprise Edition ausgeliefert.



5.4 Zusammenfassung

Der erste Schritt zur Optimierung einer Anwendung besteht darin, das Projekt mit passenden Compiler-Einstellungen zur Optimierung neu erstellen zu lassen (Release-Version).

Der zweite Schritt besteht im Einsatz des Visual-C++-Profilers, der über den Befehl ERSTELLEN/PROFIL aufgerufen wird. (Bevor Sie das Programm im Profiler ausführen können, müssen Sie die Linker-Option PROFILER-LAUF ERMÖGLICHEN setzen.)

Verwenden Sie den Profiler, um Zeilen- oder Funktionsanalysen durchzuführen. Die Zeilenanalyse ermittelt, welche und wie oft bestimmte Zeilen aufgerufen wurden. Die Funktionsanalyse wird verwendet, um Funktionszeiten und Funktionsaufrufe zu ermitteln. Um zuverlässigere Daten zu erhalten, können Sie Profiler-Läufe mit Hilfe des Profiltyps VEREINIGEN wiederholen und zusammenaddieren lassen.

Zusätzliche Profiler-Einstellungen können in dem Eingabefeld WEITERE EINSTELLUNGEN oder in benutzerdefinierten Batch-Dateien vorgenommen werden. Sie können die Analyse beispielsweise auf bestimmte Funktionen oder Quellcode-Zeilen begrenzen.

Mit Hilfe des Visual Studio Analyzers können verteilte Anwendungen analysiert werden.

Wiederverwenden von Programmcode mit der Komponentensammlung

Kapitel 6

Eines der Leitworte der objektorientierten Programmierung lautet Wiederverwenden von Programmcode. Das Visual-C++-Entwicklungssystem realisiert diese Devise, indem es ein Depot für wiederverwendbare Code-Komponenten, die sogenannte Komponentensammlung, zur Verfügung stellt.

In der Komponentensammlung sind Elemente enthalten, die auf den Einsatz in Ihrer Applikation warten. So können Sie Ihren Anwendungen beispielsweise Zwischenablage- und Palettenfunktionen, einen Begrüßungsbildschirm, einen Paßwort-Dialog, und viele andere Features hinzufügen.

6.1 Die Komponentensammlung

Die Visual-C++-Komponentensammlung ist ein Sammelbehälter für unterschiedliche Standard- und benutzerdefinierte Komponenten. Was aber ist eine Komponente? Eine Komponente kann eine Klasse einschließlich der Header-, Implementierungs- und Ressourcen-Dateien sein. Komponenten können ebenfalls von Microsoft oder Drittanbietern zur Verfügung gestellt werden.

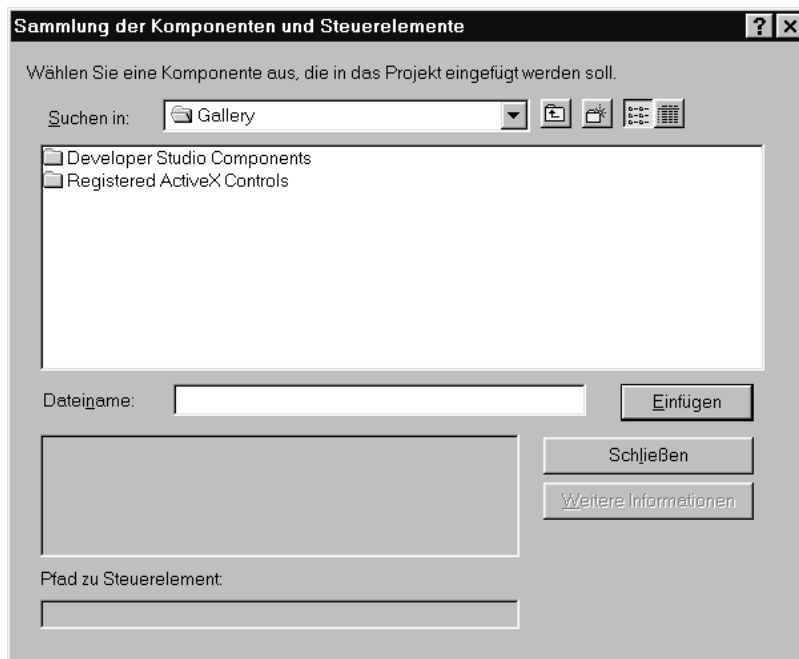
Die Komponentensammlung bietet die Möglichkeit, Komponenten zu speichern und zu verwalten.

6.1.1 Einfügen einer Komponente in ein Projekt

Die Komponentensammlung wird über den Befehl KOMPONENTEN UND STEUERELEMENTE des Eintrags DEM PROJEKT HINZUFÜGEN im Menü PROJEKT aufgerufen. Sie erscheint in Form des Dialogs SAMMLUNG DER KOMPONENTEN UND STEUERELEMENTE (siehe Abbildung 6.1).

Die Komponenten, die installiert werden können, sind in verschiedenen Ordnern gespeichert. Zwei dieser Ordner werden während der Installation von Visual C++ erzeugt: der Ordner DEVELOPER STUDIO COMPONENTS sowie der Ordner REGISTERED ACTIVEX CONTROLS. Weitere Ordner können erstellt werden, um benutzerdefinierte Komponenten aufzunehmen.

Abbildung 6.1:
Die Komponenten- und Steuerelementsammlung



Um bestimmte Komponenten Ihrem Projekt hinzuzufügen,

1. öffnen Sie den Ordner, der die Komponente enthält,
2. selektieren die Komponente und
3. klicken auf die Schaltfläche EINFÜGEN.

Viele Standardkomponenten zeigen Konfigurationsdialoge an, wenn Sie diese zum Einfügen auswählen. Einige Komponenten überprüfen sogar den Quellcode Ihrer Anwendung, um zu ermitteln, ob die selektierte Komponente mit der Anwendung kompatibel ist.

6.1.2 Erstellen eigener Komponenten

Während Sie mit Visual C++ arbeiten, entwickeln Sie einige Komponenten, die für die Wiederverwendung geeignet sind. Vielleicht möchten Sie einen Begrüßungsdialog, eine von `CDocItem` abgeleitete Klasse eines Dokument-Objekts oder einen benutzerdefinierten Info-Dialog mit dem Firmenlogo Ihres Unternehmens erstellen. Jede der soeben genannten Komponenten ist wiederverwendbar.

Um eine Komponente der Komponentensammlung hinzuzufügen,

1. Öffnen Sie die Klassen-Ansicht und klicken mit der rechten Maustaste auf die Klasse, die in eine wiederverwendbare Komponente konvertiert werden soll.
2. Wählen Sie aus dem anschließend dargestellten Kontextmenü den Eintrag ZUR KOMPONENTENSAMMLUNG HINZUFÜGEN aus.

Wenn Sie der Komponentensammlung eine Klasse hinzufügen, wird eine OGX-Datei erstellt, die in den selektierten Ordner kopiert wird. Microsoft empfiehlt, benutzerdefinierte Komponenten nicht in den bereits vorhandenen Ordnern DEVELOPER STUDIO COMPONENTS und REGISTERED ACTIVEX CONTROLS abzulegen.

Die Funktion Zur Komponentensammlung hinzufügen wird optimal ausgeführt, wenn die Klassen in individuellen Dateien gespeichert werden. Damit ist gemeint, daß nicht mehrere Klassen in derselben Quell- oder Header-Datei deklariert oder definiert sind.



Wenn der Komponentensammlung die Klassen eines Projekts über die Funktion ZUR KOMPONENTENSAMMLUNG HINZUFÜGEN hinzugefügt werden, erstellt Visual C++ automatisch einen neuen Ordner, der die Bezeichnung des Quellprojekts trägt. Sie können die Ordner manipulieren, Komponenten zwischen den Ordnern verschieben und der Sammlung neue Ordner hinzufügen oder bestehende Ordner und Komponenten löschen respektive umbenennen. Dies geschieht in dem Dialog SAMMLUNG DER KOMPONENTEN UND STEUERELEMENTE.

Mit Hilfe dieses Dialogs können Sie außerdem Komponenteneigenschaften ermitteln und setzen. Ein Klick mit der rechten Maustaste auf eine benutzerdefinierte Komponente öffnet deren Kontextmenü. Wählen Sie dort den Eintrag EIGENSCHAFTEN aus, um den EIGENSCHAFTEN-Dialog dieser Komponente zu öffnen

6.2 Zusammenfassung

Die Komponentensammlung ist ein Depot für Standard- und benutzerdefinierte Komponenten sowie ActiveX-Steuerelemente.

Um Ihrer Applikation eine Komponente oder ein Steuerelement hinzuzufügen, wählen Sie das gewünschte Objekt in dem Dialog SAMMLUNG DER KOMPONENTEN UND STEUERELEMENTE aus und klicken auf die Schaltfläche EINFÜGEN.

Sie können Ihre eigenen Klassen der Sammlung als Komponenten hinzufügen. Klicken Sie dazu in der Klassen-Ansicht mit der rechten Maustaste auf die gewünschte Klasse, und wählen Sie aus dem anschließend dargestellten Kontextmenü den Eintrag ZUR KOMPONENTENSAMMLUNG HINZUFÜGEN aus.

Die Konfiguration der Komponentensammlung ermöglicht Ihnen, neue Ordner zu erzeugen und Komponenten darin zu speichern. Benutzerdefinierte Komponenten werden in OGX-Dateien gespeichert. Da diese dem Verbunddokumentformat entsprechen, können Sie deren Eigenschaften einsehen und verändern.

Das Visual-C++-Entwicklungssystem stellt Ihnen einige Standardkomponenten zur Verfügung. So können Sie Ihren Anwendungen beispielsweise Zwischenablagefunktionen, eine Dialogfeldleiste, Paletten, Pop-up-Menüs, einen Begrüßungsbildschirm und einen Tips-und-Tricks-Dialog sowie viele andere Features hinzufügen.

Windows- Grundlagen und API



- 
7. Betriebssystemübersicht
 8. Das API-Anwendungsgerüst
 9. Fenster, Dialogfelder und Steuerelemente
 10. Ressourcen
 11. Zeichnen und Gerätekontexte
 12. Threads und Prozesse
 13. DLLs – Dynamische Bibliotheken
 14. Speicherverwaltung
 15. Dateiverwaltung
 16. Die Windows-Zwischenablage
 17. Die Registrierung
 18. Ausnahmebehandlung
- 

Betriebssystem- übersicht

Kapitel

7

Die 32-Bit-Edition von Visual C++ kann zur Entwicklung von Programmen für zwei Win32-Plattformen verwendet werden: Windows NT (auch mit mehreren Prozessoren) und Windows 95/98. Die 32-Bit-Erweiterung für Windows 3.1, Win32s, wird nicht mehr unterstützt. Microsoft hat diese Option bereits aus Visual C++ Version 4.2 entfernt.

Windows NT ist Microsofts High-End-Server-Betriebssystem. Es ist ein echtes 32-Bit-Multitasking-Betriebssystem mit einer integrierten Grafikumgebung und erweiterter Server-Funktionalität. Die Entwicklung von Windows NT war auf maximale Portierbarkeit, Stabilität und Sicherheit ausgerichtet. Das Betriebssystem ist vollständig kompatibel zu stabiler MS-DOS- und Windows-3.1-Software. Es bildet jedoch kein Substitut für Ihr altes MS-DOS-System. Möchten Sie beispielsweise ein komplexes Spiel starten, müssen Sie Ihren Rechner möglicherweise erneut unter DOS booten.

Windows 95/98 ist das Betriebssystem, das beide Welten in sich vereint. Im Gegensatz zu Windows NT bildet die Abwärtskompatibilität von Windows 95/98 eines der wesentlichen Kriterien. Trotz dieses Umstands und der Tatsache, daß für Windows 95/98 sehr viel Programmcode aus Windows 3.1 übernommen wurde, weist das neue Betriebssystem nur wenige Unzulänglichkeiten auf.

Obwohl bedeutende Unterschiede zwischen diesen Plattformen bestehen, verwenden sie die gleichen wesentlichen Features. Die überwiegende Anzahl einfacher Anwendungen ist sowohl zu Windows NT als auch zu Windows 95/98 kompatibel. Compiler- oder Betriebssystem-Features werden daher in diesem Buch unabhängig von dem verwendeten Betriebssystem beschrieben. Bedeutende Plattformunterschiede werden natürlich erwähnt.

7.1 Fenster und Nachrichten

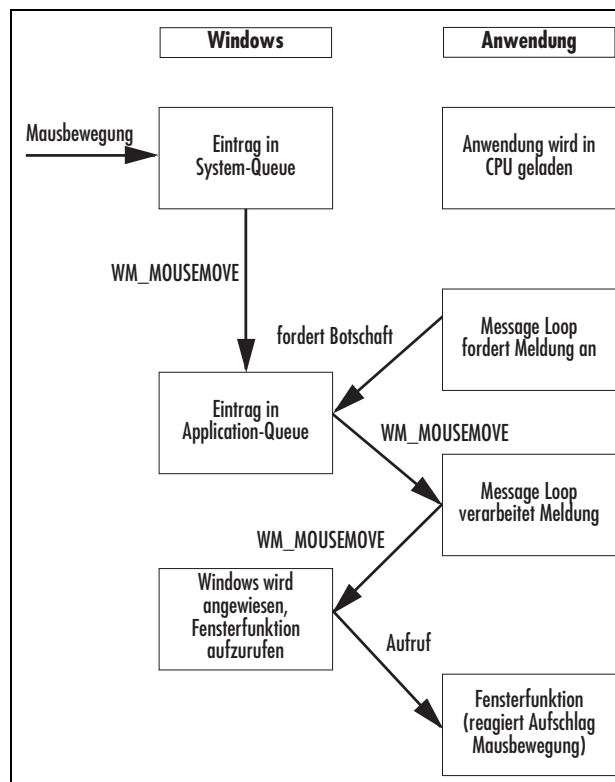
Windows wird häufig als ein *nachrichtengesteuertes* Betriebssystem bezeichnet. Nachrichten dienen der Kommunikation zwischen dem Betriebssystem und den laufenden Anwendungen, aber auch der Kommunikation zwischen den Anwendungen.

Von DOS zu Windows Unter DOS kann im Prinzip immer nur ein Programm gleichzeitig ausgeführt werden. Dieses Programm verfügt dann uneingeschränkt über alle Ressourcen des Systems.

Unter Windows laufen meist mehrere Anwendungen gleichzeitig. Aufgabe des Windows-Betriebssystems ist es daher, dafür zu sorgen, daß die Ressourcen ordnungsgemäß verteilt werden. Dies betrifft nicht nur Ressourcen wie Speicher und Handles, sondern auch die Aus- und Eingabegeräte.

7.1.1 Ereignisbehandlung über die Nachrichtenwarteschlange

Abbildung 7.1:
Ereignis-
behandlung



Betrachten wir zum Beispiel den einfachen Fall, daß der Anwender mit der Maus in das Hauptfenster einer Anwendung klickt. Windows fängt dieses Ereignis auf, übersetzt es in eine Nachricht.

Eine solche Nachricht besteht aus mehreren Teilen, unter anderem: *Nachrichten*

- ➡ einem Fenster-Handle, der das Fenster identifiziert, an das die Nachricht gerichtet ist
- ➡ einem Nachrichtencode (beispielsweise `WM_LBUTTONDOWN` für das Niederdrücken der linken Maustaste.)
- ➡ und zwei 32-Bit-Parameter (`wParam` und `lParam`) für zusätzliche Informationen (beispielsweise den Koordinaten des Mausklicks).

```
typedef struct tagMSG { // Datentyp für Windows-Nachrichten
    HWND    hwnd;
    UINT    message;
    WPARAM  wParam;
    LPARAM  lParam;
    DWORD   time;
    POINT   pt;
} MSG;
```

Diese Nachricht schickt Windows an die Nachrichtenwarteschlange der Anwendung (Application-Queue). Jede Anwendung (genauer gesagt, sogar jeder Thread) bekommt von Windows eine solche Nachrichtenwarteschlange zugeteilt.

Die Anwendung muß folglich eine Schleife implementieren, die ständig die Nachrichtenwarteschlange nach eingetroffenen Nachrichten abfragt. Dies ist die sogenannte *Message Loop*. *Die Message Loop*

```
// MessageLoop
while (GetMessage (&Message, NULL, 0, 0) )
{
    TranslateMessage(&Message);
    DispatchMessage(&Message);
}
```

Sie ist allerdings nicht die Endstation der Botschaftsverarbeitung, sondern lediglich eine Zwischenstation. Ihr Vorteil ist die chronologische Bearbeitung der einkommenden Botschaften (First In, First Out). Dies ist insbesondere für die Bearbeitung von Benutzereingaben wichtig (wenn der Anwender beim Aufsetzen eines Textes in einen bestimmten Absatz klickt und dort ein neues Wort einfügt, möchte er auch, daß zuerst der Mausklick und dann die Tastatureingaben bearbeitet werden und nicht umgekehrt). Botschaften, die die Anwendung über vom Anwender ausgelöste Ereignisse informieren, laufen daher stets über die MessageLoop.



Zudem beschleunigt die Einschaltung einer Warteschlange die Ausführung von Windows, das auf diese Weise am schnellsten auf Ereignisse reagieren und die weitere Verarbeitung an die Anwendungen verteilen kann.

In der Message Loop werden die Nachrichten eingelesen und an die Fenster der Anwendung verteilt. (Ziel einer Nachricht ist ja ein Fenster).

Wie werden die Nachrichten an die Fenster geschickt? Um Nachrichten empfangen zu können, definiert jedes Fenster eine Fensterfunktion. Nach der Verarbeitung in der Message Loop wird die Nachricht wieder an Windows übergeben, das die entsprechende Fensterfunktion aufruft und dieser die Nachricht, in verschiedene Parameter aufschlüsselt, übergibt.

Die Fensterfunktion Die Fensterfunktion empfängt die Nachricht und sorgt für eine adäquate Verarbeitung. Zu diesem Zwecke definiert jede Fensterfunktion eine mehr oder weniger umfangreiche `switch`-Anweisung. In dieser `switch`-Anweisung wird festgestellt, welche Botschaft empfangen wurde, und eine entsprechende Funktion zur Beantwortung der Botschaft aufgerufen.

```
// Fensterfunktion WndProc
LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMessage,
                          WPARAM wParam, LPARAM lParam)
{
    // beantworte Botschaften mit entsprechenden Aktionen
    switch(uiMessage)
    {
        case WM_DESTROY: PostQuitMessage(0);
                        return(0);
        default: return(DefWindowProc(hWnd, uiMessage,
                                      wParam, lParam));
    }
}
```

Nachrichtenübertragung ohne MessageLoop

Eine Vielzahl von Nachrichten haben nur indirekt etwas mit Benutzeraktionen zu tun und werden intern von Windows verschickt (beispielsweise, wenn Windows ein Fenster darüber informiert, daß es gerade verschoben, neu dimensioniert oder geschlossen wird). In diesen Fällen umgeht Windows meist die Nachrichtenwarteschleifen der einzelnen Anwendungen und schickt die Nachricht statt dessen direkt an die Fensterfunktion des betroffenen Fensters.

Nachrichten selbst abschicken

Anwendungen können nicht nur Botschaften empfangen, sie können auch selbst Botschaften verschicken – an sich selbst oder an andere Anwendungen. Auf diese Weise kann eine Anwendung beispielsweise:

- ➔ eines ihrer Fenster durch Neuzeichnen aktualisieren lassen,
- ➔ die Funktionsweise von Windows-Steuerelementen verändern,
- ➔ mit anderen Anwendungen Informationen austauschen.

Um eine Botschaft abzuschicken, stehen einer Anwendung ebenso wie Windows zwei Möglichkeiten offen: entweder über die MessageLoop oder direkt an eine Fensterfunktion.

1. Zum Abschicken von Botschaften über die MessageLoop verwendet man die Funktionen der `PostMessage`-Familie. Diese Funktionen tragen die zu verschickende Botschaft in die gewünschte Message Queue und kehren dann direkt zurück (im Erfolgsfall mit einem Wert ungleich NULL). Nach Rückkehr der `PostMessage`-Funktion kann man also nicht davon ausgehen, daß die Botschaft in der empfangenden Anwendung bereits bearbeitet wurde!

```
BOOL PostMessage( HWND hWnd, UINT message,  
                 WPARAM wParam = 0, LPARAM lParam = 0 );
```

2. Botschaft direkt verschicken. Um Botschaften direkt an bestimmte Fenster zu schicken, verwendet man die Funktionen der `SendMessage`-Familie. Diese Funktionen senden die zu übermittelnde Botschaft direkt an die Fensterfunktion eines Fensters. Im Gegensatz zu den Funktionen der `PostMessage`-Familie kehrt `SendMessage` allerdings danach nicht zurück, sondern wartet darauf, daß die Fensterfunktion die Botschaft verarbeitet hat. Die `SendMessage`-Funktionen sind daher vor allem für die schnelle und synchronisierte Verarbeitung von Botschaften bestens geeignet!

```
LRESULT SendMessage( HWND hWnd, UINT message,  
                    WPARAM wParam = 0, LPARAM lParam = 0 );
```

7.1.2 Fenster und Fensterklassen

Was sich dem Anwender als Fenster mit Rahmen, Titel, verschiedenen Dekorationen und einem Arbeitsbereich darstellt, das er mit der Maus aktivieren und verschieben kann, ist für den Programmierer nur ein spezieller Typ eines Fensters.

Aus Sicht des Programmierers ist ein Fenster ein Teil der grafischen Schnittstelle seiner Applikation zum Anwender, die sich dadurch auszeichnet, daß sie Nachrichten verarbeiten kann. Ein Fenster verfügt daher über einen Handle, der von Windows zur eindeutigen Identifizie-

ung des Fensters vergeben wird, und eine Fensterfunktion, in der die an das Fenster gerichteten Nachrichten verarbeitet werden.

Hauptfenster, Clientfenster, Dialogfelder, Steuerelemente wie Schalter, Editierfelder, etc. sind aus Windows-Sicht Fenster.

Wie wird ein Fenster mit einer Fensterfunktion verbunden?

Aus Sicht von Windows ist ein Fenster ein Objekt. Windows ist allerdings nicht objektorientiert implementiert. Wäre dies der Fall, würde man annehmen, daß ein Fenster eine Instanz einer Fensterklasse ist. In dieser Fensterklasse wäre dann die Fensterfunktion als Elementfunktion definiert.

Nun, Windows ist zwar nicht objektorientiert implementiert, aber das verwendete Prinzip ist ganz ähnlich.

Fensterklassen Jedes Fenster wird auf der Grundlage einer Fensterklasse erzeugt. Allerdings ist diese »Klasse« in Wirklichkeit eine Struktur (WNDCLASSEX). In dieser Fensterklasse werden Informationen über Erscheinungsbild und Verhalten für die Fenster festgelegt, die später auf der Grundlage dieser Fensterklasse erzeugt werden. Unter anderem gehört hierzu auch ein Zeiger auf die Fensterfunktion.

Mehrere Fenster können auf der Grundlage ein und derselben Fensterklasse erzeugt werden. Beispielsweise gehen alle Schalter-Steuerelemente auf eine gemeinsame Fensterklasse zurück. Voraussetzung ist allerdings, daß die Fensterklasse zuvor unter Windows registriert wurde. Im Falle einiger Standard-Fensterklassen (wie zum Beispiel auch der Fensterklasse für die Schalter-Steuerelemente) braucht sich der Programmierer darum allerdings nicht zu kümmern, da diese Fensterklassen bereits als fester Teil des Betriebssystems registriert sind.

Windows bietet sehr viele Standard-Fensterklassen an. Diese globalen Systemklassen implementieren die Funktionalität einiger allgemeiner Steuerelemente. Jede Anwendung kann diese Klassen für die eigenen Fenster verwenden. So ist es beispielsweise möglich, daß eine Anwendung Eingabefelder implementiert, indem sie die Fensterklasse Edit verwendet.

Eigene Fensterklassen registrieren Anwendungen können außerdem ihre eigenen Fensterklassen über die Funktion `RegisterClassEx` definieren. Diese Funktion ermöglicht dem Programmierer, ein Fensterverhalten zu implementieren, das die vom Betriebssystem unterstützten globalen Klassen nicht bieten. Eine Anwendung kann beispielsweise die Funktionalität des eigenen Hauptfensters implementieren und das Symbol und Menü des Hauptfensters registrieren lassen.

Windows ermöglicht auch das Erstellen von Subklassen und Superklassen aus bestehenden Klassen.

- ➔ Eine Subklasse ist eine Klasse, deren Fensterprozedur durch die einer anderen Klasse ersetzt wurde.
- ➔ Superklassen sind neue Klassen, die auf einer bestehenden Klasse basieren und deren Fensterprozedur übernehmen. Um aus einer Fensterklasse eine Superklasse erstellen zu können, ermittelt die Anwendung einige Klasseninformationen mit Hilfe der Funktion `GetClassInfo`. Anschließend wird die ermittelte `WNDCLASS`-Struktur modifiziert und in dem Aufruf von `RegisterClass` verwendet. Die Anwendung ermittelt über `GetClassInfo` ebenfalls die Adresse der originalen Fensterprozedur. Nachrichten, die die neue Fensterprozedur nicht bearbeitet, werden an diese Funktion weitergeleitet (siehe Beispielprogramm aus Kapitel 1).

7.1.3 Nachrichtentypen

Windows kennt derzeit an die 400 verschiedene Nachrichten. Glücklicherweise muß der Programmierer nicht selbst für die korrekte Bearbeitung all dieser Nachrichten sorgen. Worauf es letztendlich ankommt, ist die für ein Programm wirklich interessanten Nachrichten abzufangen und mit dem Aufruf passender Antwortfunktionen zu verbinden. Alle anderen Nachrichten können einer Standardverarbeitung zugeführt werden (durch Aufruf der API-Funktion `DefWindowProc`).

Nachrichtentyp	Beschreibung
Windows-Nachrichten	Das Gros der Nachrichten. Hierzu gehört die Gruppe der WindowsManager-Nachrichten die alle mit <code>WM_</code> beginnen (mit der Ausnahme von <code>WM_COMMAND</code>). Die Gruppe ist in weitere Kategorien unterteilt. Diese Kategorien enthalten DDE- (Dynamic Data Exchange), Zwischenablage-, Maus-, Tastatur-, Nicht-Client-Bereich- (Nachrichten, die sich auf die Titelzeile, den Rand und den Menübereich eines Fensters beziehen. Diese werden von dem Betriebssystem verwaltet und nicht von der Anwendung.), MDI- (MultipleDocument Interface) und viele weitere Nachrichtentypen. Die Definition der Kategorien erfolgt nach keiner streng vorgegebenen Norm. Sie soll dem Programmierer lediglich helfen, eine Übersicht über die große Anzahl verschiedener Manager-Nachrichten zu erhalten. Die <code>WM_</code> -Nachrichten sind nicht begrenzt. Werden dem Betriebssystem neue Features hinzugefügt, kann auch die Anzahl der Nachrichten steigen.

Tabelle 7.1:
Nachrichtentypen

Nachrichtentyp	Beschreibung
	Andere Nachrichtengruppen beziehen sich auf spezifische Fenstertypen wie Textfeld-Steuerelemente, Schaltflächen, Listenfelder, Kombinationsfelder, Bildlaufleisten, Listenansichten und andere Elemente.
Benachrichtigungen	Dies sind Nachrichten, die von Steuerelementen oder Kindfenstern an ihre Elternfenster geschickt werden.
Command-Nachrichten	WM_COMMAND-Nachrichten, die von Elementen der Benutzeroberfläche (Menübefehl, Schalter der Symbolleiste, Tastaturkürzel) ausgelöst werden. (In MFC-Anwendungen werden diese Botschaften üblicherweise nicht von Fensterobjekten, sondern von Dokumenten, Dokumentvorlagen oder dem Anwendungsobjekt abgefangen.)

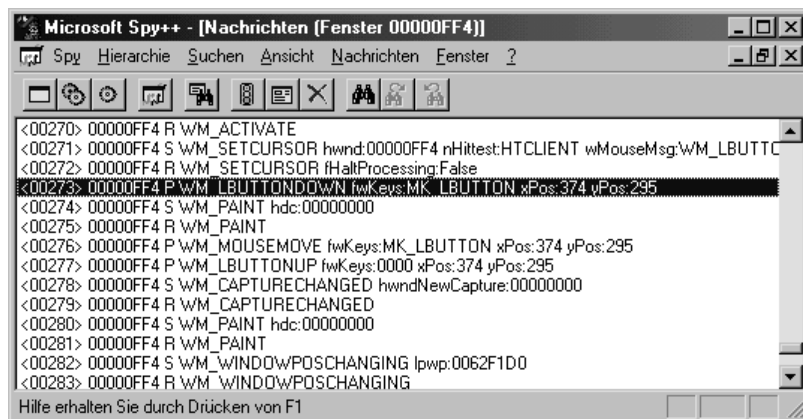
Eigene Nachrichten

Anwendungen können auch eigene Nachrichten definieren. Dies geschieht unter Zuhilfenahme der Funktion `RegisterWindowMessage`.

Das Verwenden eigener Nachrichtentypen ermöglicht bestimmten Bereichen einer Anwendung miteinander zu kommunizieren. Separate Anwendungen können ebenfalls auf diese Weise Informationen austauschen.

Den Nachrichten auf der Spur

Abbildung 7.2:
Spy++



Wer sich einmal darüber informieren möchte, welche Nachrichten so bei einer seiner Anwendungen oder speziell einem Fenster eingehen, der kann dazu das mit Visual C++ ausgelieferte Diagnose-Tool Spy++ verwenden.

Verwenden Sie SPY++ beispielsweise, um den Info-Dialog von Word für Windows zu überwachen.

1. Rufen Sie Spy++ auf (Befehl EXTRAS/SPY++).
2. Öffnen Sie den Info-Dialog von Word und arrangieren Sie Word und Spy++ nebeneinander auf Ihrem Desktop.
3. Richten Sie Spy++ für die Überwachung des Info-Dialogs ein.
 - Rufen Sie den Befehl SPY/FENSTER SUCHEN auf.
 - Nehmen Sie mit der Maus das Zielscheibensymbol auf und klicken Sie mit diesem Symbol in das zu überwachende Fenster.
 - Setzen Sie danach im Dialogfeld FENSTER SUCHEN die Option NACHRICHTEN, und klicken Sie auf OK.
4. Bewegen Sie die Maus über die Schaltfläche OK und betätigen Sie diese.

Im Spy++-Fenster werden die darauf folgenden Nachrichtenströme angezeigt (siehe Tabelle 7.1).
5. Beenden Sie die Nachrichtenüberwachung (Befehl NACHRICHTEN/PROTOKOLLIERUNG BEENDEN).

Symbolbezeichner	Beschreibung
WM_LBUTTONDOWN	Die linke Maustaste wurde gedrückt.
WM_PAINT	Die Schaltfläche OK wird während ihrer Betätigung erneut gezeichnet.
WM_LBUTTONUP	Die linke Maustaste wurde losgelassen.
WM_PAINT	Die Schaltfläche OK wird erneut gezeichnet, nachdem Sie losgelassen wurde.
WM_WINDOWPOSCHANGING	Die Position des Fensters wird geändert.
WM_WINDOWPOSCHANGED	Die Position des Fensters wurde geändert.
WM_NCACTIVATE	Die Titelseite des Fensters wurde aktiviert.
WM_ACTIVATE	Der Client-Bereich des Fensters wurde aktiviert.
WM_WINDOWPOSCHANGING	Die Position des Fensters wird geändert.
WM_KILLFOCUS	Das Fenster verliert den Fokus.

*Tabelle 7.2:
Nachrichten, die
nach einem Klick
auf OK an den
Info-Dialog von
Word für Win-
dows gesendet
werden*

Symbolbezeichner	Beschreibung
WM_DESTROY	Das Fenster wird zerstört.
WM_NCDESTROY	Die Titelzeile des Fensters wird zerstört.

7.2 Nachrichten und Multitasking

In Windows 3.1 hatte die Nachrichtenschleife eine für die Interaktion der Anwendung mit dem Betriebssystem wichtige Aufgabe: Sie ermöglichte der Anwendung die Abgabe der Steuerung des Prozessors. Da Windows 3.1 kein präemptives Multitasking-Betriebssystem ist, besitzt es keine Möglichkeit, einer nicht kooperierenden Anwendung die Steuerung des Prozessors abzunehmen. Ein stabiles System ist unter Windows 3.1 daher von dem kooperativen Verhalten einer Anwendung abhängig.

Windows-95/98- und Windows-NT sind dagegen nicht auf die Kooperation der Anwendungen angewiesen – beide Betriebssysteme sind in der Lage, laufenden Anwendungen jederzeit die Kontrolle über die CPU zu entziehen.



Systemabstürze unter Windows 95/98

Diese Besonderheit der 32-Bit-Betriebssysteme ist unter anderem die Grundlage dafür, daß ein abstürzender Prozeß nicht das gesamte Betriebssystem mitzieht. Unter Win95/98 können abstürzende Prozesse aber auch weiterhin das System lahmlegen. Dies liegt daran, daß Windows 95/98 große Teile alten Win16-Codes verwendet, der immer nur von einer Anwendung gleichzeitig ausgeführt werden kann (i.G: zu 32-Bit-Code, der reentrant ist). Stürzt eine Anwendung während der Ausführung von 16-Bit-Betriebssystemcode ab, gibt sie diesen nicht mehr frei, und der Code kann nicht mehr von anderen Anwendungen ausgeführt werden.

7.2.1 Multithreading und Multitasking

Multithreading

Unter Win16 bezeichnete man in Ausführung befindlichen Code als Task. Da man unter Windows 3.x ein Programm mehrfach aufrufen kann, sind die Bezeichnungen Programm und Task nicht identisch. Statt dessen spricht man von Instanzen eines Programms und jeder solchen Instanz würde dann eine Task entsprechen.

In Win32 spricht man dagegen von Prozessen und Threads. Jede Instanz eines Programms entspricht nun einem Prozeß, und jeder Prozeß verfügt automatisch über einen Thread, der den eigentlichen auszuführenden Handlungsfaden bezeichnet. Unter Win32 werden Botschaften an Threads gesendet, und Threads sind es, die sich die Kontrolle über die CPU teilen.

Bis dahin gibt es noch keinen wesentlichen Unterschied zwischen Threads und Tasks, aber Threads haben den Vorzug, daß sie selbst neue Threads erzeugen können (wobei erzeugter und erzeugender Thread dem gleichen Prozeß angehören). Da alle erzeugten Threads am Multitasking (siehe unten) teilnehmen, hat eine Anwendung damit die Möglichkeit, zeitaufwendige Routinen (beispielsweise das Ausdrucken eines Textes) als Thread abzuspalten, so daß die Anwendung, genauer gesagt ihr Hauptthread, während des Druckens weiter ausgeführt werden kann.

Multitasking und Nachrichtenwarteschlangen

Lassen Sie mich zunächst das Nachrichtensystem und die Task-Architektur der 16-Bit-Version von Windows beschreiben. Diese verwendet lediglich eine Nachrichtenwarteschlange. Die von verschiedenen Betriebssystemereignissen generierten Nachrichten – wie z.B. Tastatur- oder Maus-Interrupts – werden von Windows in diese Warteschlange gestellt. Windows gibt der zugehörigen Anwendung die Kontrolle über die CPU, damit diese die Nachricht mit Hilfe der Funktionen `GetMessage` oder `PeekMessage` auslesen kann.

Kann eine Anwendung die Nachricht nicht auslesen, blockiert die Anwendung das ganze System. Nachrichten werden jedoch weiterhin in der Warteschlange abgelegt. Da die Warteschlange lediglich eine begrenzte Kapazität besitzt, entsteht nach einem Absturz möglicherweise ein Überlauf. Immer dann, wenn eine Nachricht nicht in der Warteschlange abgelegt werden kann, erzeugt Windows einen Fehlerton. Das Ergebnis ist ein abgestürztes System, das bei der geringsten Mausbewegung unangenehme Signaltöne erzeugt.

Unter Win32-Systemen (Windows 95/98 und Windows NT) weist das Betriebssystem die CPU nacheinander den laufenden Threads zu. Da das Betriebssystem den Threads auch die Kontrolle über die CPU entziehen und weiterreichen kann, könnte es im Falle einer Nachrichtenschleife schnell geschehen, daß einer oder mehrere Threads versuchen, gleichzeitig auf die Nachrichtenwarteschlange zuzugreifen. Da ein Umschalten zwischen Threads nicht wie bisher von der nächsten erhältlichen Nachricht in der Warteschlange abhängig ist, besteht kei-

ne Garantie dafür, daß ein Thread lediglich die Nachrichten erhält, die an ihn adressiert sind. Aus diesem Grund wurde die Warteschlange der 16-Bit-Version von Windows in einzelne Nachrichtenwarteschlangen für jeden Thread unterteilt.

7.2.2 Threads und Nachrichten

Unter Win32 werden Fenster und Nachrichtenschleifen nicht mehr kompletten Anwendungen, sondern den einzelnen Threads zugeordnet.

Worker-Threads Bedeutet dies, daß ein Thread ein Fenster besitzen und eine Nachrichtenschleife enthalten muß? Glücklicherweise nicht. Andernfalls würde der Einsatz von Threads in gewöhnlichen Programmen sehr hinderlich sein. Natürlich können Threads erzeugt werden, die kein Fenster besitzen und über keine Nachrichtenschleife verfügen.

Stellen Sie sich beispielsweise eine hochentwickelte mathematische Anwendung vor, in der eine komplexe Berechnung für jedes Element eines zweidimensionalen Arrays (einer Matrix) durchgeführt werden muß. Die einfachste Möglichkeit, diese Absicht zu realisieren, besteht darin, eine Schleife zu implementieren, in der die Berechnung wiederholt durchgeführt wird. Unter der 16-Bit-Version von Windows war diese Vorgehensweise nicht zulässig. Während der Ausführung der Schleife hätte keine andere Anwendung auf den Prozessor zugreifen können, so daß der Rechner in dieser Zeit blockiert gewesen wäre. In Win32-Systemen ist es möglich, einen separaten Thread einzurichten, in dem die Berechnung durchgeführt wird, während der Haupt-Thread der Anwendung weiterhin die erhaltenen Nachrichten bearbeitet. Der einzige daraus resultierende Nachteil ist ein Performance-Verlust – nichts Unerwartetes während einer komplexen, rechenintensiven Berechnung. Der Thread, der die Berechnung vornimmt, verfügt über keine Fenster, Nachrichtenwarteschlangen oder Nachrichtenschleifen. Er führt lediglich eine Aufgabe aus: die Berechnung.

MFC hat einen Namen für diese Threads. Sie werden als Worker-Threads bezeichnet, im Gegensatz zu den weiterentwickelten User-Interface-Threads, die der Bearbeitung von Nachrichtenwarteschlangen dienen.

7.3 Windows-Funktionsaufrufe

Obwohl die Nachrichtenschleife eine der wesentlichen Eigenschaften einer Windows-Anwendung repräsentiert, ist sie nicht der einzige Mechanismus, über den eine Anwendung mit Windows interagiert. Windows offeriert eine Vielzahl verschiedener Systemaufrufe zur Ausführung unterschiedlicher Aufgaben. Dazu zählen beispielsweise Prozeßsteuerung, Fenster-Management, Dateiverwaltung, Speicher-verwaltung, Grafikdienste und Kommunikation.

Die wesentlichen Windows-Funktionsaufrufe können in drei Kategorien unterteilt werden.

- Kernel-Dienste enthalten Systemaufrufe zur Kontrolle von Prozessen und Threads, zum Ressource-Management sowie zur Datei- und Speicherverwaltung.
- User-Dienste umfassen Systemaufrufe zur Verwaltung der Benutzerschnittstellenelemente, wie z.B. Fenster, Steuerelemente, Dialoge oder Nachrichten.
- GDI-Dienste (Graphics Device Interface) stellen eine geräteunabhängige Grafikausgabefunktionalität zur Verfügung.

Die Funktionen dieser Dienste sind in den Bibliotheken des Betriebssystems implementiert und können über die Funktionen der Windows-API (Application Programming Interface) aufgerufen werden.

Darüber hinaus gibt es eine Reihe weiterer APIs für unterschiedliche Aufgaben. Beispiele hierfür sind MAPI (Messaging API), TAPI (Telephony API) oder ODBC (Open Database Connectivity). Der Umfang, in dem diese APIs in das System integriert wurden, variiert. Das Komponentenobjektmodell (COM, die Grundlage für ActiveX und OLE) ist beispielsweise ein Teil der Windows-Funktionalität, obwohl es in Form mehrerer System-DLLs implementiert wurde. Andere APIs, wie z.B. Winsock, bilden sogenannte Add-Ons.

7.3.1 Kernel-Dienste

Kernel-Dienste lassen sich der Kategorie Dateiverwaltung, Speicherverwaltung, Prozeß- und Thread-Kontrolle sowie Ressourcenverwaltung zuordnen. Trotz ihrer geringen Anzahl beschreiben diese Kategorien die überwiegend verwendeten Kernel-Modulfunktionen.

Die bevorzugte Methode der Dateiverwaltung unterscheidet sich von der, die gewöhnlich in C/C++-Programmen eingesetzt wird. Anstatt über die C++-Klasse `iostream` respektive über die Standard-C-Biblio-

Dateiverwaltung

thekfunktionen für die Stream- oder Low-Level-Ein- und Ausgabe auf Dateien zuzugreifen, sollten Anwendungen das Win32-Dateiobjektkonzept und die entsprechenden Funktionen übernehmen. Dateiobjekte ermöglichen den Zugriff auf Dateien, ohne die C/C++-Bibliothek zu nutzen. Beispiele hierfür sind überlappede Ein-/Ausgabe- und Speicher-Map-Dateien, die für die Intertask-Kommunikation benötigt werden.

*Speicher-
verwaltung* Für die Speicherverwaltung der meisten Anwendungen sind die C-`malloc`-Funktion oder der C++-`new`-Operator vollkommen ausreichend. In einer Win32-Anwendung werden diese Aufrufe automatisch in die entsprechenden Win32-Systemaufrufe zur Speicherverwaltung umgewandelt. Für Anwendungen mit umfangreichen Anforderungen an die Speicherverwaltung bestehen hochentwickelte Funktionen zur Verwaltung des virtuellen Speichers. Diese Funktionen können beispielsweise verwendet werden, um Adreßraum zu manipulieren, für den mehrere hundert Megabyte Speicher reserviert wurden.

*Threadsynchro-
nisierung* Eine der wichtigsten Aspekte des Prozeß- und Thread-Managements betrifft die Synchronisierung. Dieses Problem ist neu in der Windows-Umgebung, da die 16-Bit-Version des Betriebssystem noch nicht damit konfrontiert wurde. Unter der kooperativen Multitasking-Umgebung von Windows 3.1, geben die Anwendungen die Steuerung des Prozessors lediglich zu vordefinierten Zeitpunkten während der Programmausführung ab. Die Ausführung mehrerer Threads geschieht synchron. Im Gegensatz dazu erhalten Prozesse und Threads in der präemptiven Multitasking-Umgebung keine Informationen über den Status der Ausführung anderer Threads. Um zu gewährleisten, daß mehrere Threads in einer korrekten Reihenfolge ausgeführt werden, und um Situationen zu vermeiden, in denen zwei oder mehrere Threads lange Zeit auf andere Threads warten müssen, wird ein Synchronisationsmechanismus verwendet. In der Win32-Umgebung bilden verschiedene Synchronisationsobjekte diesen Mechanismus, der von den Threads genutzt werden kann, um andere Threads über ihren Status zu informieren, sensible Programmcode-Bereiche vor einer erneuten Ausführung zu schützen oder Informationen über andere Threads respektive den Status anderer Objekte zu ermitteln.

Kernel-Objekte Unter Win32 werden viele Kernel-Ressourcen (die nicht mit Anwenderschnittstellen-Ressourcen verwechselt werden sollten) durch Kernel-Objekte repräsentiert. Beispiele hierfür sind Dateien, Threads, Prozesse und Synchronisierungsobjekte. Gewöhnlich verweisen Zugriffsnummern auf diese Objekte. Einige Funktionen dienen der Manipulation aller Objekte, während andere Funktionen lediglich Objekte eines bestimmten Typs bearbeiten. Unter Windows NT besitzen Objekte Eigenschaften, die sich auf die Sicherheit beziehen. Ein Thread kann

beispielsweise ein Dateiojekt so lange nicht bearbeiten, bis er die entsprechende Genehmigung dazu erhält, die sich auf die Sicherheitseigenschaften des Dateiojekts beziehen.

Das Kernel-Modul stellt außerdem Funktionen zur Verfügung, die der Verwaltung von Anwenderschnittstellen-Ressourcen dienen. Diese Ressourcen nehmen Symbole, Mauszeiger, Dialogvorlagen, Zeichenfolgentabellen, Versionsinformationen, Tastaturkürzeltabellen, Bitmaps und benutzerdefinierte Ressourcen auf. Kernel-Systemaufrufe reservieren Speicher für Ressourcen, laden Ressourcen aus einer Datei (gewöhnlich die ausführbare Datei der Anwendung) und entfernen Ressourcen aus dem Speicher. *Ressourcen*

Auch die Funktionalität für 32-Bit-Konsolenanwendungen wird von dem Kernel-Modul zur Verfügung gestellt. Solche Programme erscheinen als einfache, herkömmliche DOS-Programme. Sie sind jedoch vollwertige 32-Bit-Anwendungen, die über die Kommandozeile ausgeführt werden und die grafische Schnittstelle von Windows nicht nutzen. Auch diese Anwendungen verfügen über einen Zugriff auf sehr viele Win32-Systemaufrufe. Eine Konsolenanwendung kann beispielsweise die Funktionen zum virtuellen Speicher verwenden oder ein Multithread-Programm sein. *Konsolenanwendungen*

Windows NT

Einige Bereiche der Kernel-Modulfunktionalität betreffen ausschließlich Windows NT. Das NT-Kernel-Modul enthält beispielsweise verschiedene Funktionen, mit deren Hilfe die Sicherheitsattribute der Kernel-Objekte ermittelt und gesetzt werden können.

Ein weiterer spezifischer NT-Bereich ist die Banddatensicherungsfunktionalität. Einige Aufrufe löschen und formatieren ein Band, andere lesen oder schreiben Daten darauf.

7.3.2 User-Dienste

Wie der Name bereits impliziert, enthält dieser Dienst Systemaufrufe, die Elemente der Benutzerschnittstelle verwalten. Dazu zählen Funktionen, die Fenster, Dialoge, Menüs, Text- und Grafik-Cursor, Steuerelemente, die Zwischenablage und einige weitere Bereiche verwalten.

Erst durch die User-Dienstfunktionen werden die High-Level-Komponenten der Benutzerschnittstelle möglich. Das Kernel-Modul bietet Speicherreservierung, Thread-Management und andere Dienste an, die Windows benötigt, um ausgeführt werden zu können. Das GDI-Modul verfügt über grundlegende Grafikfunktionen. Erst das User-Modul integriert diese beiden Bereiche und enthält beispielsweise das Konzept eines Fensters.

- Fenster* Fenster-Management-Aufrufe umfassen Funktionen zur Verwaltung der Größe, Position, des Aufbaus und der Fensterprozedur, sowie Funktionen zur Freigabe und zum Sperren eines Fensters. Außerdem ermitteln diese Funktionen Informationen über Fenster und verwalten Steuerelemente, wie z.B. Schaltflächen, Bildlaufleisten oder Textfelder. Das Anwender-Modul enthält ebenfalls Funktionen zur Verwaltung von untergeordneten MDI-Fenstern (Multiple Document Interface).
- Menüs* Aufrufe in dem User-Modul, die sich auf Menüs beziehen, stellen eine Funktionalität zur Verfügung, um Menüs, Menüleisten und Pop-up-Menüs zu erstellen, darzustellen und zu manipulieren.
- Mauszeiger* Anwendungen können über User-Funktionen die Form des grafischen Mauszeigers und der Schreibmarke verändern.
- Zwischenablage* Die Verwaltung der Windows-Zwischenablage geschieht ebenfalls über User-Funktionen. Die Zwischenablage ist im wesentlichen ein einfacher Mechanismus, über den Anwendungen Daten austauschen. Eine Anwendung kann Daten in einem der verschiedenen Zwischenablageformate in der Zwischenablage plazieren. Andere Anwendungen können den Inhalt der Zwischenablage ermitteln und die darin enthaltenen Daten verwenden, sofern diese in einem Format vorliegen, das die Anwendung interpretieren kann. Viele Anwendungen bieten Funktionen zur Bearbeitung des Inhalts der Zwischenablage in dem Menü BEARBEITEN an (AUSSCHNEIDEN, KOPIEREN, EINFÜGEN).
- Nachrichten* Auch die Funktionen zur Verwaltung von Nachrichten und Nachrichtenwarteschlangen sind in dem User-Modul enthalten. Anwendungen können die entsprechenden Aufrufe verwenden, um den Inhalt einer Nachrichtenwarteschlange zu überprüfen, Nachrichten entgegenzunehmen. Nachrichten zu versenden und neue Nachrichten zu erstellen.

7.3.3 GDI-Dienste

Die Funktionen des GDI (Graphics Device Interface) werden gewöhnlich dazu verwendet, einfache geräteunabhängige Grafikoperationen auf bestimmten Gerätekontexten auszuführen.

- Gerätekontexte* Ein Gerätekontext ist eine Schnittstelle zu einem spezifischen Grafikgerät. Der Gerätekontext ermittelt Informationen über das Gerät und gibt die Grafiken darauf aus. Die Informationen, die über den Gerätekontext ermittelt werden können, beschreiben das Gerät detailliert. Die Technologie des Geräts (z.B. Vektor oder Raster), sein Typ, Name, die Auflösung sowie Farben und Schriftarten werden mit Hilfe der entsprechenden Gerätekontext-Aufrufe erfaßt.

Die Grafikausgabe erfolgt durch die Übergabe des Gerätekontext-Handles an die entsprechende GDI-Ausgabefunktion. Über den Gerätekontext wird ein allgemeiner geräteunabhängiger Grafikaufruf in mehrere Anweisungen konvertiert, die schließlich die Grafik auf dem Gerät ausgeben. Ruft eine Anwendung beispielsweise die GDI-Funktion `Ellipse` auf, ermittelt der Gerätekontext den Treiber, der den Aufruf ausführt. Der Gerätetreiber wiederum kann den Aufruf an einen Hardware-Beschleuniger weitergeben, sofern das Video-System diesen unterstützt.

GDI-Gerätekontexte können verschiedene Geräte beschreiben. Einige der überwiegend verwendeten Gerätekontexte sind der Darstellungsgerätekontext (für die Ausgabe auf den Bildschirm des Computers), der Speichergerätekontext (für die Ausgabe in eine Bitmap, die in dem Speicher abgelegt wird) und der Druckergerätekontext (für die Ausgabe, die eventuell in den Steuercode des Druckers umgewandelt und an diesen gesendet wird).

Ein besonderer Gerätekontext ist der Meta-Datei-Gerätekontext, der Anwendungen ermöglicht, eine kontinuierliche Aufzeichnung von GDI-Ausgabefunktionen durchzuführen. Diese Aufzeichnung ist geräteunabhängig und kann später somit auf jedem Gerät ausgegeben werden. Meta-Dateien erfüllen eine besondere Funktion in der geräteunabhängigen Darstellung von eingebetteten OLE-Objekten. Sie ermöglichen das Portieren von OLE-Objekten und erlauben Container-Anwendungen, diese Objekte sogar dann darstellen oder ausdrucken zu lassen, wenn die Anwendung selbst nicht ausgeführt wird.

Das Zeichnen in einen Gerätekontext geschieht gewöhnlich über logische Koordinaten. Diese Koordinaten beschreiben Objekte mit realen geräteunabhängigen Maßen. Einem Rechteck kann beispielsweise eine Breite von zwei Zentimetern und eine Höhe von einem Zentimeter zugewiesen werden. Die GDI konvertiert logische Koordinaten in physische Koordinaten.

Das Konvertieren der Koordinaten ist für die Betriebssysteme *Koordinaten* Windows 95/98 und Windows NT unterschiedlich. Wie sein 16-Bit-Vorgänger ist auch Windows 95/98 auf 16-Bit-Koordinaten begrenzt. Diese Begrenzung obliegt dem Umstand, daß Windows 95/98 aus sehr viel Windows-3.1-Programmcode besteht. Windows NT hingegen arbeitet mit 32-Bit-Koordinaten. Daher ist dieses Betriebssystem für komplexe Grafikanwendungen geeignet, wie z.B. CAD-Programme.

Sowohl Windows NT als auch Windows 95/98 unterstützen eine einfache Konvertierung von logischen zu physischen Koordinaten. Dazu werden die Werte der originalen Koordinaten ermittelt. Diese werden auf die logischen und physischen Werte umgerechnet. Die ursprüngli-

chen Koordinaten geben eine horizontale und vertikale Verschiebung an. Die Umrechnung ermittelt die Ausrichtung und Skalierung, der Objekte nach der Konvertierung.

Windows NT bietet außerdem Funktionen zur Transformation. Diese Funktionen können jede lineare Transformation ausführen, um logische Koordinaten in physische Koordinaten umzuwandeln. Zusätzlich zur Transformation und Skalierung kann die Ausgabe gedreht oder zugeschnitten werden.

Zeichenfunktionen Die wohl vorwiegend verwendeten GDI-Funktionen sind die zum Zeichnen verschiedener Objekte, wie z.B. Rechtecke, Ellipsen, Polygone oder Textelemente. (Das sind nur einige Beispiele.)

Andere Zeichenfunktionen sind die Funktionen zur Bit-Verschiebung, die zum schnellen Kopieren von Bitmaps benutzt werden. (Für Anwendungen, wie z.B. Spiele, die sehr hohe Geschwindigkeiten benötigen, sollten die Funktionen zum Manipulieren von Bitmaps in der DirectX-SDK verwendet werden.)

Gerätekontext-Funktionen Andere Funktionen verwalten die Gerätekontexte. Kontexte für verschiedene Geräte können mit diesen Funktionen erstellt und zerstört werden. Außerdem kann der Status der Geräte gespeichert und wieder geladen werden. Mit Hilfe dieser Funktionen ermitteln Sie ebenfalls Informationen über die Geräte.

Koordinatenfunktionen Des weiteren bestehen Funktionen, die Koordinatenumwandlungen bearbeiten. Funktionen für alle Win32-Plattformen werden verwendet, um den Ursprung und die Ausdehnung eines Fensters (die logischen Koordinaten) und den Viewport (die Koordinaten des Zielgeräts) zu setzen oder zu ermitteln. Spezifische NT-Funktionen manipulieren die Transformationsmatrixen.

Paletten GDI-Funktionen können ebenfalls zur Bearbeitung von Paletten verwendet werden. Diese werden vorwiegend von Anwendungen verwendet, die eine besondere Farbqualität für Geräte umsetzen müssen, die lediglich über eine begrenzte Anzahl der gleichzeitig darstellbaren Farben verfügen – 256 Farben beispielsweise. Durch das Bearbeiten der Farbpalette können diese Anwendungen (ein bezeichnendes Beispiel ist ein Programm zur Ansicht von Grafikdateien, wie GIF- oder PCX-Dateien) eine Farbzusammenstellung einrichten, die zu der bestmöglichen Darstellung des anzuzeigenden Bilds führt. Auf diese Weise wird die Abhängigkeit von Dither-Techniken reduziert, wodurch die Bildqualität verbessert wird. Sie haben außerdem die Möglichkeit, die Farbpalette derart zu bearbeiten, daß eine Palettenanimation entsteht. Diese Technik verändert die Farbpalette, um den Eindruck einer Bewegung auf dem Bildschirm zu erzeugen.

Ein weiteres GDI-Feature ist das Erstellen und Verwalten von GDI-Objekten. Pinsel, Stifte, Schriftarten, Bitmaps oder Paletten können in Gerätekontexten erstellt und selektiert werden, um die Form der zu zeichnenden Figuren zu bestimmen.

GDI-Objekte

Das GDI-Modul stellt außerdem Funktionen zum Verwalten von Schriftarten zur Verfügung (einschließlich der TrueType-Schriftarten).

Schriftarten

Andere Funktionen verwalten zwei Arten von Meta-Dateien (die herkömmlichen Windows-Meta-Dateien sowie die neuen erweiterten Meta-Dateien). Meta-Dateien können erstellt, gespeichert, wieder geladen und auf jedem Gerät ausgegeben werden.

Metadateien

Ein weiteres Leistungsmerkmal des GDI-Moduls ist die Verwaltung von Bereichen sowie das Clipping-Management. Clipping ist in der Windows-Umgebung sehr wichtig, da es einer Anwendung ermöglicht, eine Oberfläche (z.B. ein untergeordnetes Fenster) ohne Rücksicht auf deren Begrenzung darzustellen. Außerdem berücksichtigt das Clipping, daß bestimmte Bereiche der Oberfläche von anderen Objekten auf dem Bildschirm überdeckt werden können.

Clipping

7.3.4 Andere APIs

Windows besteht natürlich aus weit mehr Funktionen als denen, die in den drei zuvor beschriebenen Modulen enthalten sind. Darüber hinaus gibt es viele andere Module und viele andere APIs, die alle einen besonderen Bereich oder eine spezifische Funktionalität implementieren. Nachfolgend finden Sie einige der überwiegend verwendeten APIs aufgeführt. Einige dieser APIs werden später in diesem Buch detailliert erläutert.

- ➡ Allgemeine Steuerelementfunktionen werden zur Bearbeitung der neuen allgemeinen Windows-95/98-Steuerelemente benutzt. Diese Funktionen sind natürlich ausschließlich unter Windows 95/98, Windows NT 3.51 und höher erhältlich.
- ➡ Standarddialoge enthalten Systemdialoge zum Öffnen einer Datei, zum Selektieren einer Farbe aus einer Farbpalette, zur Auswahl einer Schrift aus den installierten System-Fonts und zum Bestimmen eines Suchen- respektive Suchen- und Ersetzen-Vorgangs. Diese Dialoge können entweder so verwendet werden, wie sie Ihnen zur Verfügung stehen, oder über neue Dialogvorlagen und Fensterprozeduren modifiziert werden.
- ➡ MAPI (Messaging Applications Programming Interface) gewährt den Anwendungen Zugriff auf die Nachrichtenfunktionen über bestimmte Nachrichtensysteme, wie z.B. Microsoft Mail. Windows

arbeitet mit drei verschiedenen MAPI-Varianten: Die einfache MAPI wird von älteren Anwendungen genutzt, die auf Nachrichten basieren. Dazu zählen Anwendungen, die kein Nachrichtensystem erfordern, dieses jedoch verwenden können, wenn es vorhanden ist. Microsoft Word lässt sich dieser Kategorie zuordnen. Neue Anwendungen, die mit Nachrichten arbeiten (die somit ein bestehendes Nachrichtensystem verwenden), sollten CMC (Common Messaging Calls Interface) nutzen. Komplex auf Nachrichten basierende Workgroup-Anwendungen arbeiten gewöhnlich mit allen MAPI-Diensten (Erweiterte MAPI).

- ➔ MCI (Multimedia Control Interface) ist die Multimedia-Schnittstelle. Über die MCI-Funktionen erhalten Anwendungen Zugriff auf die Video-, Audio- und MIDI-Funktionalität von Windows. Die überwiegende Anzahl der Multimedia-Anwendungen verwendet die MCI-Funktionen für die Medienwiedergabe. Einige Anwendungen nutzen möglicherweise komplexere MCI-Funktionen für die Bearbeitung von Medien-Dateien.
- ➔ Die COM-API ist eine umfangreiche Sammlung von Systemaufrufen, die die gesamte OLE-Funktionalität implementieren. Dazu zählen OLE-Container und Server-Funktionalität für die Inplace-Bearbeitung, das Aktivieren von Objekten, Drag & Drop, Automatisierung und ActiveX-Steuerelemente (früher benutzerdefinierte OLE-Steuerelemente).
- ➔ TAPI ist die Telefonie-API. Anwendungen verwenden die TAPI, um geräteunabhängig auf Telefonie-basierende Ressourcen zuzugreifen (Modems, FAX-Modems, Anrufbeantworter).

Verschiedene Bereiche sind der Netzwerk-Funktionalität zugewiesen. Dazu zählen WinSock, die Windows-Socket-Bibliothek, WinInet, die Windows-Internet-API, RAS (Remote Access Service) und die RPC-Bibliothek (Remote Procedure Call).

7.3.5 Fehlerrückgaben

Viele Windows-Funktionen verwenden einen allgemeinen Mechanismus für die Fehlerrückgabe. Tritt ein Fehler auf, setzen diese Funktionen einen spezifischen Thread-Fehlerwert, der mit einem Aufruf der Funktion `GetLastError` ermittelt werden kann. Die 32-Bit-Werte, die von dieser Funktion zurückgegeben werden, sind in der Header-Datei `WINERROR.H` oder in spezifischen Bibliothek-Headerdateien definiert.

Die Funktionen Ihrer Anwendung können diesen Fehlerwert ebenfalls setzen, indem Sie `SetLastError` aufrufen. Spezifische Anwendungsfehler sind dadurch gekennzeichnet, daß das 29. Bit des Fehlerwerts gesetzt ist. Fehlercodes mit diesem gesetzten Bit sind von dem Betriebssystem für die anwendungsspezifische Verwendung reserviert.

7.3.6 Verwenden von Standard-C/C++-Bibliotheks-Funktionen

Win32-Anwendungen können mit einigen Einschränkungen ebenfalls die Standardfunktionen der C/C++-Bibliothek nutzen.

Eine Windows-Anwendung verfügt gewöhnlich nicht über einen Zugriff auf die traditionellen `stdin`-, `stdout`- oder `stderr`-Ströme, und auch nicht über die entsprechenden DOS-Dateizugriffsnummern (0, 1 und 2) oder C++-Ein- und Ausgabe-Objekte (`cin` und `cout`). Lediglich Konsolanwendungen können diese Standard-Dateizugriffsnummern verwenden. (Windows-Anwendungen verfügen jedoch über einen eigenen Ein- und Ausgabestandard.)

Wie ich bereits erwähnte, sollten Windows-Anwendungen die Win32-Dateiverwaltungsfunktionen für die Dateiverwaltung verwenden. Das bedeutet nicht, daß die Standardfunktionen des C-Stroms und der Low-Level-Ein- und Ausgabe oder die C++-Ein- und Ausgabe-Bibliothek nicht mehr verfügbar wären. Diese Bibliotheken verfügen lediglich nicht über die Möglichkeiten der Win32-API. Die Funktionen der C/C++-Bibliothek berücksichtigen beispielsweise nicht die Sicherheitsattribute eines Dateiobjekts und können auch nicht für die asynchronen, überlappten Ein- und Ausgabe-Operationen verwendet werden.

Anwendungen sollten auch auf die Verwendung der C-Bibliothek verzichten, die MS-DOS-Funktionen zur Steuerung der Prozesse enthält, und statt dessen `CreateProcess` benutzen.

Die meisten anderen C/C++-Bibliotheken können ohne Einschränkungen verwendet werden. Wenngleich die Win32-API eine größere Anzahl verschiedener Funktionen zur Speicherverwaltung anbietet, benötigen die meisten Anwendungen keine Dienste, die komplexer als die von `malloc` oder dem C++-Operator `new` angebotenen Dienste sind. Die C/C++-Routinen zur Bearbeitung von Puffern, Zeichenfolgen, Zeichen- und Byte-Klassifizierungen, Datenkonvertierung und mathematischen Funktionen, um nur einige zu nennen, können ebenfalls uneingeschränkt genutzt werden.

Win32-Anwendungen sollten nicht versuchen, auf den MS-DOS-Interrupt 21 oder auf IBM-PC-BIOS-Funktionen zuzugreifen. Die APIs, die zu diesem Zweck unter der 16-Bit-Version von Windows angeboten wurden, bestehen nicht mehr. Anwendungen, die einen Low-Level-Zugriff auf die System-Hardware benötigen, sollten mit dem entsprechenden DDK (Device Driver Development Kit) entwickelt werden.

7.4 Plattformunterschiede

Die Win32-API wurde entwickelt, um als plattformunabhängige API zu dienen. Dennoch bestehen einige Plattformunterschiede, die sich auf Einschränkungen des zugrundeliegenden Betriebssystems beziehen. Einige dieser Restriktionen wurden zuvor in diesem Kapitel beschrieben. Die folgenden Abschnitte führen eine Übersicht und Zusammenfassung der Windows-95/98- und Windows-NT-Features auf.

Visual C++ kann nicht nur Anwendungen für diese beiden Plattformen erzeugen, sondern auch darauf installiert werden. Nachfolgend finden Sie einige Hinweise zur Programmentwicklung mit Hilfe dieser Betriebssysteme.

7.4.1 Windows NT

Eine beinahe vollständige Implementierung der Win32-API ist in Windows NT enthalten. Seit Version 3.51 bietet Windows NT dieselben neuen benutzerdefinierten Steuerelemente wie Windows 95/98 an.

Windows NT unterstützt Unicode, erweiterte Sicherheits-Features und unterschiedliche System-Level während einer Bandsicherung. Die Server-Plattform Windows NT verfügt über eine komplexere Server-Umgebung als Windows 95/98. Dank einer echten 32-Bit-Implementierung ist Windows NT die stabilere Plattform. Dieses Betriebssystem ist daher als Entwicklungsumgebung sehr geeignet.

Windows NT wurde bisher nachgesagt, Speicher und CPU-Leistung zu sehr zu beanspruchen. Version 4.0 ist hinsichtlich dieses Rufs eine angenehme Überraschung – plötzlich ist NT schneller als Windows 95/98. Obwohl das Betriebssystem weiterhin sehr viel Speicher benötigt, erscheint sogar die Speicherverwaltung effizienter als bisher. Ein gut ausgestattetes Entwicklungssystem sollte unter Windows NT über mindestens 32 Mbyte RAM, 1 Gbyte Festplattenkapazität und einen Pentium-Prozessor verfügen.

7.4.2 Windows 95/98

Obwohl Windows 95/98 einige Features von Windows NT vermissen läßt, ist es diesem bezüglich der Kompatibilität zu älterer Software und Hardware überlegen. Die überwiegende Anzahl der Features, die nicht unter Windows 95/98 realisiert wurden, fehlt den Anwendern nicht.

Doch welche Funktionen fehlen Windows 95/98? Die erweiterten Sicherheits-Features, die Unterstützung von Unicode und unterschiedlichen System-Levels während der Bandsicherung wurden bereits genannt.

Für Grafikprogrammierer, die von der NT-Umgebung kommen, bestehen einige Nachteile. Zwei dieser Nachteile sind die fehlende GDI-Unterstützung für Transformationsfunktionen und die Begrenzung des Koordinatenraums auf 16-Bit-Koordinaten.

Windows 95/98 hingegen ist eine flexible und stabile Multithread-Umgebung, die sich in vielerlei Hinsicht mit Windows NT vergleichen läßt. Windows 95/98 stellt eine umfangreiche Untermenge der Win32-API zur Verfügung. Mit Ausnahme der spezifischen NT-Features, die bereits erwähnt wurden, sind die meisten API-Funktionen erhältlich.

Die Speicherverwaltung von Windows 95/98 ist ein wenig besser als die von Windows NT. Einige Anwender konnten das Betriebssystem auf einem alten 386-System mit 4 Mbyte RAM installieren.

Windows 95/98 bildet eine ausgezeichnete Entwicklungsplattform. Seine Stabilität ist außergewöhnlich, auch wenn diese nicht mit der von Windows NT verglichen werden kann. Alle 32-Bit-Entwicklungswerkzeuge, einschließlich der Konsolenanwendungen, werden korrekt auf dieser Plattform ausgeführt.

7.5 Zusammenfassung

Visual-C++-Anwendungen können in der Win32-Umgebung eingesetzt werden. Dazu zählen die verschiedenen spezifischen Plattformversionen von Windows NT und Windows 95/98. Die 32-Bit-Erweiterung für Windows 3.1, Win32s, wird nicht mehr unterstützt.

Der Kern jeder Windows-Anwendung ist die Nachrichtenschleife. Das Betriebssystem informiert Anwendungen mit Hilfe von Nachrichten über unterschiedliche Ereignisse. Anwendungen wiederum bearbeiten diese Nachrichten nach einem Aufruf der entsprechenden Fensterfunktion. Ein Fenster ist ein rechteckiger Bereich des Bildschirms, aber auch ein abstraktes, eigenständiges Objekt, das Nachrichten empfangen und bearbeiten kann.

Prozesse oder Anwendungen besitzen Threads. Diese wiederum besitzen Fenster. Threads sind parallele Ausführungspfade innerhalb einer Anwendung.

Anwendungen interagieren mit Windows, indem Sie eine der vielen Betriebssystemfunktionen aufrufen, die entweder in Windows selbst oder in verschiedenen Add-Ons implementiert sind. Die Systemfunktionen lassen sich in drei Kategorien unterteilen. Das Kernel-Modul verwaltet den Speicher, Dateien und Prozesse. Das User-Modul verwaltet die Elemente der benutzerdefinierten Schnittstelle (besonders Fenster) und Nachrichten. Das GDI-Modul stellt Grafikdienste zur Verfügung.

Weitere Module implementieren bestimmte Funktionalitäten, wie z.B. COM, MAPI, Netzwerk, allgemeine Steuerelemente und Standarddialoge sowie Multimedia.

Visual-C++-Anwendungen können mit einigen Einschränkungen Standardfunktionen der C/C++-Bibliothek verwenden.

Die drei primären Win32-Plattformen unterscheiden sich in der Implementierung der Win32-API. Eine beinahe vollständige Implementierung bietet Windows NT. Windows 95/98 verfügt über eine große Untermenge der API-Funktionen. Einige spezifische NT-Elemente und erweiterte Komponenten fehlen dem Betriebssystem jedoch.

Das API- Anwendungsgerüst

Kapitel 8

Einige Anwender, die über die Windows-Programmierung sprechen, sind der Meinung, daß mit einem Betriebssystem etwas nicht stimmen kann, das sehr viele Programmzeilen benötigt, um ein einfaches Programm zur Ausgabe der Nachricht »Hello, World!« ausführen zu lassen. Doch ist diese Aussage wirklich wahr oder nur ein weiteres Gerücht über das Unternehmen Microsoft und das Betriebssystem Windows?

8.1 Das »wahre« Hello-World-Programm

Betrachten Sie bitte einmal den Dialog in Abbildung 8.1. Wie viele Zeilen C-Programmcode und Ressource-Code glauben Sie sind notwendig, um eine Anwendung zu erstellen, die lediglich diesen Dialog anzeigt? Keine? Um wie viele Zeilen ist dieses Programm länger als das »originale« Hello-World-Programm, das zu Beginn des ersten Kapitels der C-Bibel von Kernighan-Ritchie erscheint?



Abbildung 8.1:
Hello World
unter Windows

Sie haben richtig geraten. Die Anzahl der Programmzeilen beträgt in beiden Fällen fünf Zeilen (wobei die Leerzeilen zwischen dem eigentlichen Programmcode nicht gezählt werden). Die Windows-Version des Hello-World-Programms ist in Listing 8.1 aufgeführt.

Listing 8.1:
Der Quellcode
des Hello-World-
Programms
hello.c

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
    MessageBox(NULL, "Hello, World!", "", MB_OK);
}
```

Das Kompilieren dieses Programms ist nicht schwieriger als die Kompilierung des originalen Hello-World-Programms über die Kommandozeile. (Nehmen Sie dieses Beispielprogramm als Anlaß, den Visual-C++-Compiler über die Kommandozeile zu verwenden.) Damit der Compiler von der Kommandozeile aus gestartet werden kann, muß zunächst die Batch-Datei *VCVARS32.BAT* ausgeführt werden. Visual C++ erstellt diese Datei während der Installation in dem *\BIN*-Verzeichnis von VC. (Abhängig von Ihrem System und dessen Konfiguration, müssen Sie den Umgebungsspeicher für DOS-Fenster möglicherweise vergrößern.)

Schließlich geben Sie

```
CL HELLO.C USER32.LIB
```

ein. Das Programm *Hello.exe* kann nun ausgeführt werden. Geben Sie dazu *HELLO* in die Kommandozeile ein. Sowohl Windows 95/98 als auch Windows NT können Anwendungen auf diese Weise starten.

Das Verhalten dieser »Anwendung« ist überraschend komplex. Im Gegensatz zu seinem »einfachen« C-Pendant zeigt das Programm nicht nur die Nachricht an. Es interagiert mit dem Anwender. Nachdem die Nachricht ausgegeben wurde, wird die Anwendung weiterhin auf dem Bildschirm angezeigt und kann mit der Maus verschoben werden. Sie können die Schaltfläche OK betätigen, um das Programm zu beenden. Alternativ dazu können Sie die Maustaste über der Schaltfläche drücken und halten. Beobachten Sie, wie sich die Schaltfläche verändert, wenn Sie den Mauszeiger darüber bewegen. Das Fenster verfügt ebenfalls über ein Menü, das mit der Tastenkombination **Alt** + **[]** geöffnet werden kann. Unter Windows 95/98 öffnen Sie dieses Menü, indem Sie den Mauszeiger auf die Titelzeile bewegen und mit der rechten Maustaste klicken. Der Eintrag *VERSCHIEBEN* in diesem Menü wird verwendet, um die Position des Fensters zu verändern. Mit einem Druck auf die Taste **Esc** kann die Anwendung ebenfalls geschlossen werden.

Wie Sie sehen, verfügt die Anwendung trotz der wenigen Code-Zeilen über sehr viele Funktionen. Doch woher kommt diese Komplexität? Die Antwort lautet: Nachrichtenschleife und Fensterfunktion. Leider ist ein fünfzeiliges Programm für die Erläuterung des Verhaltens von Windows-Anwendungen nicht geeignet. Wir werden zu diesem Zweck eine umfangreichere Anwendung verwenden.

8.2 Eine einfache Nachrichtenschleife: Senden und Hinterlegen von Nachrichten

Das Problem unseres ersten Hello-World-Programms ist dessen Schlichtheit. Der Aufruf von `MessageBox` führt zu einer komplexen, für uns nicht erkennbaren Funktionalität. Damit wir diese Funktionalität verstehen können, müssen wir sie sichtbar machen. Wir erstellen daher ein Fenster, das wir selbst verwalten. Die Funktion `MessageBox` wird nicht diese Aufgabe für uns übernehmen.

Die neue Version des Programms *HELLO.C* ist in Abbildung 8.2 dargestellt. Diesmal erscheint der Text »Hello, World!« auf einer Schaltfläche, die den gesamten Client-Bereich einnimmt. (Der Text wird außerdem in der Titelzeile des Fensters aufgeführt.) Die Anwendung ist sehr einfach aufgebaut, so daß wir die wesentlichen Elemente erörtern können, ohne auf irrelevante Details eingehen zu müssen.



Abbildung 8.2:
Hello, World,
mit einer einfachen Nachrichtenschleife

Ein gewöhnliches Windows-Programm registriert zunächst eine Fensterklasse während der Initialisierung, bevor es das Hauptfenster aus der zuvor registrierten Klasse erzeugt. Wir verzichten vorerst auf die Registrierung einer neuen Klasse und die damit verbundenen Umstände (wie z.B. das Schreiben einer Fensterfunktion). Statt dessen verwenden wir eine der bestehenden Fensterklassen, die mit `BUTTON` bezeichnet ist. Die Funktionalität dieser Klasse entspricht nicht dem Verhalten der vorherigen Version von *HELLO.C*. Das ist auch nicht meine Absicht. Mein Ziel besteht darin, Ihnen die Funktion einer einfachen Nachrichtenschleife zu demonstrieren.

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE d2,
                  LPSTR d3, int d4)
{
    MSG msg;
    HWND hwnd;

    hwnd = CreateWindow("BUTTON", "Hello, World!",
                      WS_VISIBLE | BS_CENTER, 100,100,100,80,
                      NULL, NULL, hInstance, NULL);
    while (GetMessage(&msg, NULL, 0, 0))
```

Listing 8.2:
Hello, World mit
einer einfachen
Nachrichtenschleife

```
{
    if (msg.message == WM_LBUTTONUP)
    {
        DestroyWindow(hwnd);
        PostQuitMessage(0);
    }
    DispatchMessage(&msg);
}
return msg.wParam;
}
```

Dieses Beispiel beschreibt eine Nachrichtenschleife. Nachdem das Fenster des Programms erstellt wurde, wird eine `while`-Schleife ausgeführt, die wiederholt die Windows-Funktion `GetMessage` aufruft. Erhält die Anwendung eine Nachricht, gibt `GetMessage` einen Wert zurück. Dieser Wert ist nur dann `FALSE`, wenn die erhaltene Nachricht `WM_QUIT` war. (Diese und andere symbolische Konstanten sind in den Header-Dateien definiert, wie z.B. `WINDOWS.H`.) Das Programm reagiert auf die spezielle Nachricht `WM_LBUTTONUP`. Trifft diese Nachricht ein, wird das Fenster mit einem Aufruf von `DestroyWindow` zerstört. Ein Aufruf der Funktion `PostQuitMessage` gewährleistet, daß die Funktion `GetMessage` auf den Empfang der `WM_QUIT`-Nachricht wartet, woraufhin die Schleife verlassen wird. Andere Nachrichten als `WM_LBUTTONUP` werden mit Hilfe der Funktion `DispatchMessage` weitergeleitet.

Wohin aber werden die Nachrichten mit der `DispatchMessage`-Funktion weitergeleitet? Die Nachrichten werden an die Standard-Fensterfunktion der Klasse `BUTTON` weitergeleitet. Diese Fensterfunktion bleibt uns verborgen, da Sie ein Teil des Betriebssystems ist.

Der Aufruf der Funktion `DestroyWindow` in dem Abschnitt, der die Nachricht `WM_LBUTTONUP` bearbeitet, erstellt eine weitere Nachricht mit der Bezeichnung `WM_DESTROY`. Eine bessere Vorgehensweise als das Zerstören der Anwendung mit einem Aufruf der `PostQuitMessage`-Funktion im `WM_LBUTTONUP`-Handler besteht darin, den Aufruf dieser Funktion in einen anderen Bedingungsblock einzubinden, der sich auf die `WM_DESTROY`-Nachricht bezieht. Dies ist jedoch nicht möglich, da `WM_DESTROY`-Nachrichten gewöhnlich nicht hinterlegt, sondern direkt an die Anwendung gesendet werden.

Wenn eine Nachricht hinterlegt wird, ermittelt die Anwendung diese über einen Aufruf der Funktion `GetMessage` oder `PeekMessage`. (`PeekMessage` wird verwendet, wenn die Anwendung einige Aufgaben ausführen muß und keine Nachrichten zur Bearbeitung vorliegen.) Das Senden einer Nachricht an eine Anwendung impliziert einen direkten und unmittelbaren Aufruf der Fensterfunktion, wobei alle Nachrichtenschleifen umgangen werden. Die Nachricht `WM_DESTROY`, die nach einem Aufruf von `DestroyWindow` erzeugt wird, kann daher nicht von der Nachrichtenschleife empfangen werden. Sie wird der Fensterfunktion der Fensterklasse `BUTTON` direkt übergeben.

Auch dieses Beispiel hat uns nicht geholfen, den Aufbau einer Fensterfunktion zu verstehen. Sie werden daher eine weitere, komplexere Version des Hello-World-Programms kennenlernen.

8.3 Fensterfunktionen

In diesem Abschnitt werden wir für unsere Anwendung ein eigene Fensterklasse registrieren. In der Fensterklasse definieren wir einen Zeiger auf eine Fensterfunktion. Alle Fenster, die auf der Grundlage der Fensterklasse erzeugt werden, sind dann mit dieser Fensterfunktion verbunden.

8.3.1 Mausbehandlung

In dieser Version von Hello.c nutzen wir die Fensterfunktion, um die Nachricht `WM_LBUTTONDOWN` abzufangen. Als Antwort auf diese Nachricht zeichnen wir den Hello-World-Gruß in das Fenster.

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    RECT clientRect;

    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            hdc = GetDC(hwnd);
            if (hdc != NULL)
            {
                GetClientRect(hwnd, &clientRect);
                DPtoLP(hdc, (LPPPOINT)&clientRect, 2);
                DrawText(hdc, "Hello, World!", -1, &clientRect,
                        DT_CENTER|DT_VCENTER|DT_SINGLELINE);
                ReleaseDC(hwnd, hdc);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;
```

*Listing 8.3:
Quellcode von
Hello World mit
einer neuen Fen-
sterklasse*


```

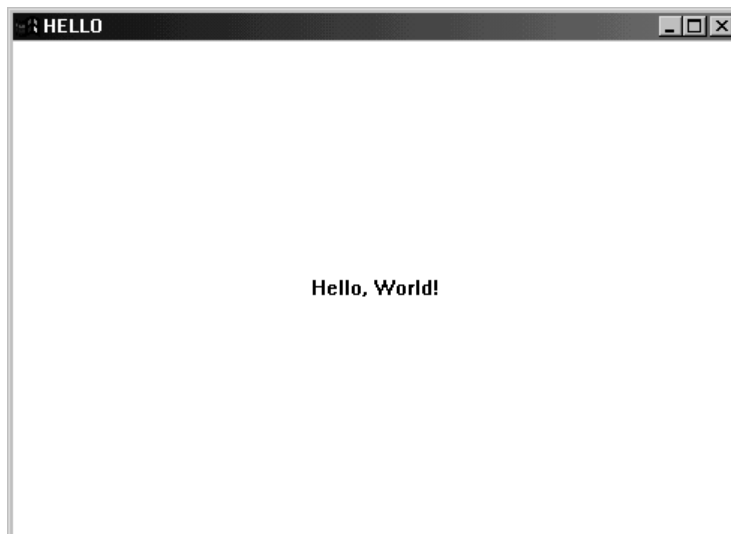
if (hPrevInstance == NULL)
{
    memset(&wndClass, 0, sizeof(wndClass));
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wndClass.lpszClassName = "HELLO";
    if (!RegisterClass(&wndClass)) return FALSE;
}
hwnd = CreateWindow("HELLO", "HELLO",
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                    NULL, NULL, hInstance, NULL);
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
return msg.wParam;
}

```

Um dieses Programm kompilieren zu können, müssen Sie die Bibliothek *gdi32.lib* in der Kommandozeile angeben:

CL HELLO.C USER32.LIB GDI32.LIB

Abbildung 8.3:
Hello World mit
eigener Fenster-
klasse



Dieses Programm demonstriert Einsatz einer Fensterklasse und Unterstützung der Maus durch das Programm. Klickt der Anwender mit der Maus in das Fenster, erscheint der Hello-World-Text. Das Programm verfügt über ein Systemmenü und kann verschoben, vergrößert und verkleinert werden, und es reagiert auf den Menüeintrag SCHLIESSEN oder die Tastenkombination **Alt** + **F4**.

Doch wie arbeitet das Programm?

Zunächst beginnt die Ausführung mit der Funktion `WinMain`. Die Anwendung prüft, ob Kopien des Programms ausgeführt werden. In diesem Fall muß die Fensterklasse nicht erneut registriert werden. Andernfalls wird die Fensterklasse registriert, und ihre Eigenschaften werden in der Struktur `WndClass` definiert. In diese Struktur wird ebenfalls die Adresse der Fensterfunktion `WndProc` bestimmt.

Fensterklasse registrieren

Anschließend wird ein Fenster mit einem Systemaufruf der Funktion `CreateWindow` erstellt. Nachdem das Fenster dargestellt wurde, wird die Nachrichtenschleife ausgeführt. Diese wird verlassen, wenn `GetMessage` die Nachricht `WM_QUIT` erhält und daraufhin den Wert `FALSE` zurückgibt.

Fenster erzeugen

`WndProc` zeigt die Struktur der Fensterfunktion. Eine gewöhnliche Fensterfunktion ist eine umfangreiche `switch`-Anweisung. Abhängig von den erhaltenen Nachrichten, werden unterschiedliche Funktionen aufgerufen, die die erforderlichen Aktionen ausführen. Der Programmcode in Listing 8.3 bearbeitet zwei Nachrichten: `WM_LBUTTONDOWN` und `WM_DESTROY`.

Die Fensterfunktion

Als Antwort auf die `WM_LBUTTONDOWN`-Nachricht holt sich die Fensterfunktion einen Handle zum Gerätekontext des Fensters und gibt den »Hello-World«-Text zentriert im Fenster aus.

`WM_DESTROY`-Nachrichten werden versendet, wenn der Anwender das Zerstören eines Anwendungsfensters verursacht. Unser Programm reagiert darauf mit einem Aufruf der Funktion `PostQuitMessage`. Auf diese Weise wird gewährleistet, daß die `GetMessage`-Funktion in `WinMain` die Nachricht `WM_QUIT` erhält. Die Hauptnachrichtenschleife wird daraufhin verlassen.

Was geschieht mit Nachrichten, die nicht von unserer Fensterfunktion bearbeitet werden? Solche Nachrichten werden an die Standard-Fensterfunktion `DefWindowProc` weitergeleitet. Diese Funktion steuert über ihre Standardbehandlungsroutine das Verhalten des Anwendungsfensters und der Nicht-Client-Bereichskomponenten (wie z.B. die Titelleiste).

Das Dialog-Pendant zu `DefDlgProc` heißt `DefWindowProc`. Die Funktion bearbeitet spezifische Dialog-Nachrichten und verwaltet die Steuerelemente des Dialogs, wenn dieser den Fokus erhält oder verliert.



8.3.2 Die Besonderheit von WM_PAINT

Das letzte Beispiel war schon eine ausgereifte Windows-Anwendung. Allerdings mußte man zur Anzeige des Textes erst die linke Maustaste drücken. Auch verschwand der Text wieder, wenn Windows das Fenster neu aufbauen mußte (beispielsweise, wenn der Anwender das Fenster minimiert und dann wiederherstellt).

Windows kann Fenster mitsamt Inhalt verschieben, aber nicht wieder von Null aufbauen. Statt dessen schickt Windows immer dann, wenn ein Neuaufbau des Fensterinhalts erforderlich wird, eine WM_PAINT-Nachricht an das Fenster. Aufgabe des Fensters ist es, diese Nachricht abzufangen und den Fensterinhalt zu rekonstruieren.

Listing 8.4:
Hello World und
WM_PAINT

```
#include <windows.h>

void DrawHello(HWND hwnd)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT clientRect;

    hDC = BeginPaint(hwnd, &paintStruct);
    if (hDC != NULL)
    {
        GetClientRect(hwnd, &clientRect);
        DPTOLP(hDC, (LPPPOINT)&clientRect, 2);
        DrawText(hDC, "Hello, World!", -1, &clientRect,
            DT_CENTER | DT_VCENTER | DT_SINGLELINE);
        EndPaint(hwnd, &paintStruct);
    }
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_PAINT:
            DrawHello(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
    }
}
```

```

    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wndClass.lpszClassName = "HELLO";
    if (!RegisterClass(&wndClass)) return FALSE;
}
hwnd = CreateWindow("HELLO", "HELLO",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
    NULL, NULL, hInstance, NULL);
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
return msg.wParam;
}

```

Um dieses Programm kompilieren zu können, müssen Sie die Bibliothek *gdi32.lib* in der Kommandozeile angeben (CL HELLO.C USER32.LIB GDI32.LIB).

Beachten Sie, daß in der Behandlung der WM_PAINT-Nachricht der Gerätekontext nicht über GetDC/ReleaseDC, sondern BeginPaint und EndPaint ermittelt und freigegeben wird.

Die Auslagerung des Codes zur Bearbeitung der WM_PAINT-Nachricht in eine eigene Funktion ist nicht nötig, kann aber das Programm übersichtlicher machen.



8.4 Mehrere Nachrichtenschleifen und Fensterfunktionen

Bisher haben wir in unseren *HELLO.C*-Beispielen lediglich eine Nachrichtenschleife verwendet. (In dem ersten Beispiel befand sich die Nachrichtenschleife in der MessageBox-Funktion.) Kann mehr als eine Nachrichtenschleife in einer Anwendung verwendet werden? Und wieso sollten derartige Anwendungen erstellt werden, wenn dies möglich ist?

Die Antwort auf die erste Frage lautet: ja. Anwendungen können mehrere Nachrichtenschleifen einsetzen. Stellen Sie sich die folgende Situation vor: Eine Anwendung, die über eine eigene Nachrichtenschleife verfügt, ruft eine MessageBox-Funktion auf. Dieser Aufruf führt dazu, daß die in der MessageBox-Funktion implizierte Nachrichtenschleife eine gewisse Zeit, nämlich während die Nachricht angezeigt wird, die Bearbeitung der Nachrichten übernimmt.

Anwendungen können mehrere Nachrichtenschleifen einsetzen

Dieses Szenarium gibt die Antwort auf die zweite Frage. Sie implementieren eine zweite (oder dritte, vierte usw.) Nachrichtenschleife, wenn Nachrichten zu einem bestimmten Zeitpunkt während der Programmausführung auf eine andere Weise als bisher bearbeitet werden müssen.

Freihand-zeichnen Stellen Sie sich beispielsweise das Zeichnen mit der Maus vor. Eine Anwendung kann eine Freihand-Zeichenfunktion zur Verfügung stellen, indem Sie auf Mausereignisse reagiert und die Position der Maus ermittelt, wenn die linke Maustaste im Client-Bereich betätigt wird. Während der Ermittlung der Koordinaten wird die Anwendung mit Hilfe separater Nachrichten über jede Mausbewegung informiert. Die Anwendung zeichnet somit eine Freihandlinie, indem Sie der Zeichnung ein neues Segment hinzufügt, wenn der Anwender die Maus bewegt. Das Zeichnen ist beendet, wenn der Anwender die linke Maustaste losläßt.

Die fünfte und letzte Version des Hello-World-Programms verwendet mehrere Nachrichtenschleifen. Sie muß über die Kommandozeile mit der Anweisung

```
CL HELLO.C USER32.LIB GDI32.LIB
```

kompiliert werden. Sie ermöglicht Ihnen, den Text »Hello, World!« mit der Maus zu zeichnen.

Abbildung 8.4:
Grafische Version von Hello World



Der in Listing 8.4 aufgeführte Quellcode der Anwendung enthält zwei `while`-Schleifen, die die `GetMessage`-Funktion aufrufen. Die Hauptnachrichtenschleife in `WinMain` unterscheidet sich nicht von der des letzten Beispiels. Der neue Programmcode befindet sich in der Funktion `DrawHello`.

```
#include <windows.h>

void AddSegmentAtMessagePos(HDC hDC, HWND hwnd, BOOL bDraw)
{
    DWORD dwPos;
    POINTS points;
    POINT point;

    dwPos = GetMessagePos();
    points = MAKEPOINTS(dwPos);
    point.x = points.x;
    point.y = points.y;
    ScreenToClient(hwnd, &point);
    DPTOLP(hDC, &point, 1);
    if (bDraw) LineTo(hDC, point.x, point.y);
    else MoveToEx(hDC, point.x, point.y, NULL);
}

void DrawHello(HWND hwnd)
{
    HDC hDC;
    MSG msg;

    if (GetCapture() != NULL) return;
    hDC = GetDC(hwnd);
    if (hDC != NULL)
    {
        SetCapture(hwnd);
        AddSegmentAtMessagePos(hDC, hwnd, FALSE);
        while(GetMessage(&msg, NULL, 0, 0))
        {
            if (GetCapture() != hwnd) break;
            switch (msg.message)
            {
                case WM_MOUSEMOVE:
                    AddSegmentAtMessagePos(hDC, hwnd, TRUE);
                    break;
                case WM_LBUTTONDOWN:
                    goto ExitLoop;
                default:
                    DispatchMessage(&msg);
            }
        }
    }
ExitLoop:
    ReleaseCapture();
    ReleaseDC(hwnd, hDC);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            DrawHello(hwnd);
    }
}
```

*Listing 8.5:
Quellcode der
grafischen Versi-
on von Hello
World*

```

        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HELLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("HELLO", "HELLO",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                       NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Das vorherige Beispiel gibt »Hello, World!« über eine Zeichenfolge aus. Das aktuelle Beispiel ist sehr viel komplexer. Es prüft zunächst, ob eine andere Anwendung die Maus verwendet und ermittelt den Handle des Gerätekontextes für das Hauptfenster. Anschließend wird die Maus mit Hilfe der Funktion `SetCapture` okkupiert. Windows beginnt daraufhin, der Anwendung `WM_MOUSEMOVE`-Nachrichten zu senden.

Die Funktion `DrawHello` ruft die Hilfsfunktion `AddSegmentAtMessagePos` auf, die die aktuelle Zeichenposition auf die in der letzten Nachricht vermerkte Position setzt, wenn als dritter Parameter der Boolesche Wert `FALSE` übergeben wurde. Dazu wird die `GetMessagePos`-Funktion aufgerufen, die die Position des Mauszeigers während der Erstellung der letzten Nachricht zurückgibt. `AddSegmentAtMessagePos` verwendet außerdem Transformationsfunktionen, um die Bildschirmkoordinaten der Maus in die logischen Koordinaten des Fensters umzuwandeln.

Nach dem Aufruf der Funktion `AddSegmentAtMessagePos` führt die `DrawHello`-Funktion die neue Nachrichtenschleife aus. Während die Maus okkupiert wird, erwarten wir ein besonderes Verhalten von unserer

Anwendung. Sie soll die Mausbewegungen überwachen und der Zeichnung zusätzliche Segmente hinzufügen. Dazu wird erneut die Funktion `AddSegmentAtMessagePos` aufgerufen. Der dritte übergebene Parameter wird auf `TRUE` gesetzt, wenn die Anwendung eine `WM_MOUSEMOVE`-Nachricht empfängt.

Die Nachrichtenschleife wird verlassen, wenn die Maustaste losgelassen wird oder der Anwendung die Maus entzogen wird. In diesem Fall wird die Routine `DrawHello` beendet, und die erste Nachrichtenschleife setzt die Bearbeitung der nachfolgenden Nachrichten fort.

War die Implementierung der beiden Nachrichtenschleifen wirklich notwendig? Hätten wir die `WM_MOUSEMOVE`-Nachrichten nicht ebenfalls in der Fensterfunktion bearbeiten können? Sicherlich wäre diese Vorgehensweise möglich gewesen, doch ein strukturierter Programmcode, wie der des letzten Beispiels, gestalten Programme übersichtlicher.

8.5 Zusammenfassung

Jede Windows-Anwendung verwendet eine Nachrichtenschleife. Nachrichtenschleifen rufen wiederholt eine der Funktionen `GetMessage` oder `PeekMessage` auf und empfangen auf diese Weise Nachrichten, die anschließend über `DispatchMessage` an die Fensterfunktionen weitergeleitet werden.

Fensterfunktionen werden in dem Augenblick für Fensterklassen definiert, in dem die Fensterklasse mit der Funktion `RegisterClass` registriert wird. Eine gewöhnliche Fensterfunktion enthält eine `switch`-Anweisung mit einem Bedingungsblock für alle Nachrichten, die von der Anwendung bearbeitet werden sollen. Andere Nachrichten werden an die Standard-Fensterfunktion `DefWindowProc` oder `DefDlgProc` weitergeleitet.

Nachrichten können an eine Anwendung gesendet oder für diese hinterlegt werden. Hinterlegte Nachrichten werden in der Nachrichtenwarteschlange gespeichert und dort von `GetMessage` oder `PeekMessage` ausgelesen. Das Senden einer Nachricht ruft unmittelbar die Fensterfunktion auf und umgeht auf diese Weise die Nachrichtenwarteschlange und Nachrichtenschleifen.

Eine Anwendung kann über mehrere Nachrichtenschleifen verfügen. Es ist jedoch nicht erforderlich, mehr als eine Schleife zu verwenden. Diese Vorgehensweise dient lediglich einem gut strukturierten, übersichtlichen Programmcode.

Fenster, Dialogfelder und Steuerelemente

Kapitel

9

Ein Windows-Fenster wird als rechteckiger Bereich auf dem Bildschirm definiert. Diese Definition informiert nicht über das Potential der Funktionalität, die sich hinter der abstrakten Vorstellung von einem Fenster verbirgt. Ein Fenster ist die primäre Einheit, über die ein Anwender und eine Windows-Anwendung interagieren.

Ein Fenster ist nicht nur ein Bereich, in dem Anwendungen Ihre Ausgabe darstellen. Es ist das Ziel der Ereignisse und Nachrichten innerhalb der Windows-Umgebung. Obwohl das Windows-Fensterkonzept bereits einige Jahre vor den ersten objektorientierten Programmiersprachen definiert wurde, ist seine Terminologie zeitgemäß. Die Eigenschaften eines Fensters bestimmen dessen Aufbau, während die Methoden definieren, wie es auf die Eingabe des Anwenders oder auf andere Systemereignisse reagieren soll.

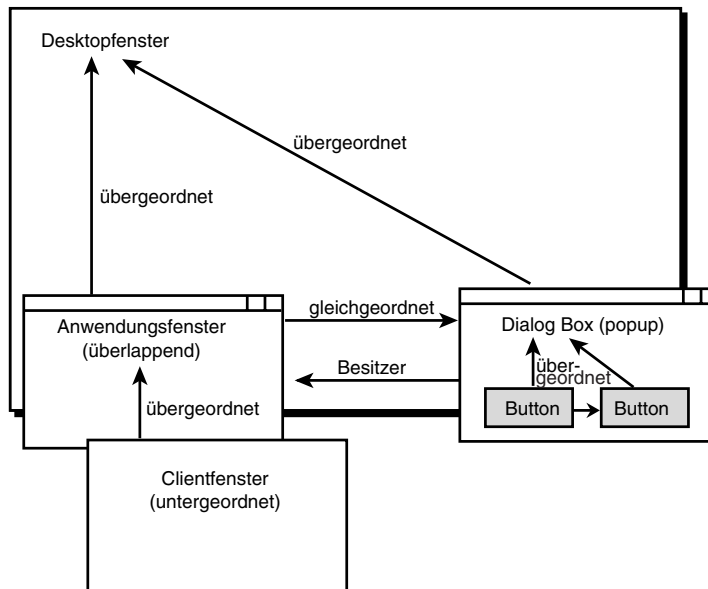
Ein Fenster wird durch einen Handle identifiziert. Dieser Handle (gewöhnlich eine Variable vom Typ `HWND`) bezeichnet eindeutig jedes Fenster im System. Dazu zählen die Anwendungsfenster, Dialogfelder, der Desktop, bestimmte Symbole und Schaltflächen. Benutzerschnittstellen-Ereignisse werden mit dem entsprechenden Handle in Windows-Nachrichten abgelegt und anschließend an die das Fenster besitzende Anwendung (respektive an den Thread, um präzise zu sein) gesendet oder für diese hinterlegt.

Windows bietet sehr viele Funktionen zum Erstellen und Verwalten von Fenstern.

9.1 Die Fensterhierarchie

Windows ordnet Fenster in einer hierarchischen Struktur an. Jedem Fenster ist ein Fenster übergeordnet (das Parent- oder Elternfenster). Mehrere Fenster können sich auf einer Gliederungsstufe befinden. Die Grundlage aller Fenster bildet der Desktop, der nach dem Windows-Start generiert wird. Das übergeordnete Fenster von Top-Level-Fenstern ist somit das Desktop-Fenster. Das übergeordnete Fenster eines untergeordneten Fensters (Child-Fenster oder Dokumentfenster) ist entweder ein Top-Level-Fenster oder ein anderes Child-Fenster, das sich auf einer höheren Hierarchiestufe befindet. Abbildung 9.1 stellt diese Hierarchie an einem gewöhnlichen Windows-Bildschirm dar.

Abbildung 9.1:
Die Hierarchie
der Fenster



Die Windows-NT-Hierarchie ist ein wenig komplexer, da dieses Betriebssystem mit mehreren Desktops gleichzeitig arbeiten kann. Windows NT verwendet gewöhnlich drei Desktops, für den Anmeldebildschirm, für die Anwender-Anwendungen sowie für Bildschirmschoner.

Die sichtbare Fensterhierarchie gibt gewöhnlich die logische Hierarchie wieder. Diese ordnet die Fenster derselben Gliederungsebene in deren Z-Reihenfolge an. Die Z-Reihenfolge ist die Reihenfolge, in der Fenster einer Gliederungsebene geöffnet wurden. Diese Anordnung

kann für Top-Level-Fenster verändert werden. Top-Level-Fenster mit dem erweiterten Fensterstil `WM_EX_TOPMOST` werden über allen anderen Top-Level-Fenstern angeordnet.

Ein Top-Level-Fenster kann ein anderes Top-Level-Fenster besitzen. Ein Fenster, das einem anderen Fenster gehört, wird immer über diesem angeordnet und dann geschlossen, wenn sein Besitzer auf Symbolgröße verkleinert wird. Ein bezeichnendes Beispiel für ein Top-Level-Fenster, das ein anderes Fenster besitzt, ist eine Anwendung, die einen Dialog darstellt. Das Dialogfeld ist kein untergeordnetes Fenster (es ist nicht an den Client-Bereich des Hauptfensters gebunden), aber es ist in dem Besitz des Anwendungsfensters.

Übergeordnete Fenster	Aktive übergeordnete Fenster können nicht von anderen Fenstern überlappt werden (außer diese haben den erweiterten Fensterstil <code>WM_EX_TOPMOST</code>). Hauptfenster von Anwendungen und Dialogfenster sind übergeordnete Fenster.
Untergeordnete Fenster	Untergeordnete Fenster können nur im Client-Bereich ihres übergeordneten Fensters angezeigt werden. Untergeordnete Fenster können nicht ohne ihr übergeordnetes Fenster existieren. Clientfenster sind untergeordnete Fenster; Hauptfenster sind dem Desktop untergeordnet.
Zugeordnete Fenster	Besitzt ein Fenster andere Fenster, werden diese zusammen mit ihrem Besitzer geschlossen.

Dialoge sind zugeordnete Fenster.

Die Fensterhierarchie durchsuchen

Verschiedene Funktionen ermöglichen die Suche nach einem bestimmten Fenster in der Fensterhierarchie. Nachfolgend finden Sie eine Übersicht über einige der überwiegend verwendeten Funktionen:

Tabelle 9.1:
Suchfunktionen
für Fenster

Funktion	Beschreibung
GetDesktopWindow	Ermittelt den Handle des aktuellen Desktop-Fensters.
EnumWindows	Führt alle Top-Level-Fenster auf. Eine benutzerdefinierte Funktion wird einmal für jedes Top-Level-Fenster aufgerufen. Die Adresse der Funktion wird der Funktion <code>EnumWindows</code> übergeben. Top-Level-Fenster, die nach dem Aufruf der Funktion erstellt wurden, werden nicht aufgeführt. Ein neues Fenster wird auch dann nicht aufgeführt, wenn es während der Erstellung der Auflistung erzeugt wurde.
EnumChildWindows	Führt alle untergeordneten Fenster des angegebenen Fensters auf, deren Handle der Funktion <code>EnumChildWindows</code> übergeben wird. Das Auflisten der Fenster geschieht mit einer benutzerdefinierten Funktion, deren Adresse ebenfalls während des Aufrufs der Funktion <code>EnumChildWindows</code> angegeben wird. Die Funktion führt auch untergeordnete Fenster auf, die sich von dem angegebenen Child-Fenster ableiten. Untergeordnete Fenster, die vor ihrer Erfassung zerstört oder nach der Auflistung erstellt wurden, werden nicht in die Liste aufgenommen.
EnumThreadWindows	Ordnet alle Fenster in einer Liste an, die der angegebene Thread besitzt. Dazu wird eine benutzerdefinierte Funktion für jedes Fenster ausgeführt, das der genannten Bedingung entspricht. Der Handle des Threads und die Adresse der benutzerdefinierten Funktion werden <code>EnumThreadWindows</code> als Parameter übergeben. Die Auflistung umfaßt Top-Level-Fenster, untergeordnete Fenster und deren Ableitungen. Fenster, die nach der Auflistung erstellt wurden, werden nicht erfaßt.
FindWindow	Sucht das Top-Level-Fenster, dessen Fensterklassenname oder Fenstertitel der Funktion übergeben wird.
GetParent	Ermittelt das übergeordnete Fenster des angegebenen Fensters.
GetWindow	Diese Funktion bietet eine flexible Möglichkeit zur Manipulation der Fensterhierarchie. Abhängig davon, welcher Wert mit dem zweiten Parameter <code>uCmd</code> übergeben wurde, kann diese Funktion verwendet werden, um den Handle eines übergeordneten, besitzenden, untergeordneten oder gleichgestellten Fensters zu ermitteln.

9.2 Fensterverwaltung

Gewöhnlich kann eine Anwendung ein Fenster in zwei Schritten erstellen. Zuerst wird die Fensterklasse registriert, und anschließend das Fenster selbst. Dazu wird die Funktion `CreateWindow` aufgerufen. Die Fensterklasse bestimmt das Verhalten des neuen Fenstertyps und enthält die Adresse der neuen Fensterfunktion. Mit `CreateWindow` steuert die Anwendung einige Eigenschaften des neuen Fensters, wie z.B. dessen Größe, Position und Aufbau.

9.2.1 Die Funktion `RegisterClass` und die Struktur `WNDCLASS`

Eine neue Fensterklasse wird mit dem folgenden Aufruf registriert:

```
ATOM RegisterClass(CONST WNDCLASS *lpwc);
```

Der einzige Parameter dieser Funktion, `lpwc`, verweist auf eine Struktur vom Typ `WNDCLASS`, die den neuen Fenstertyp beschreibt. Der Rückgabewert ist vom Typ `atom`, ein 16-Bit-Wert, der eine Zeichenfolge in einer Windows-Tabelle bezeichnet.

Die `WNDCLASS`-Struktur ist wie folgt definiert:

```
typedef struct _WNDCLASS {
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpstrMenuName;
    LPCTSTR lpstrClassName;
} WNDCLASS;
```

Die Bedeutung einiger Parameter ist unmißverständlich:

<code>hIcon</code>	Nimmt den Handle des Symbols auf, das zur Darstellung des auf Symbolgröße verkleinerten Fensters dieser Klasse verwendet wird.
<code>hCursor</code>	Ist der Handle des Standard-Mauszeigers, der dargestellt werden soll, wenn die Maus auf das Fenster bewegt wird.
<code>hbrBackground</code>	Nimmt den Handle des GDI-Pinsels auf, der zum Zeichnen des Fensterhintergrunds benutzt wird.

<code>lpszMenuName</code>	Die Zeichenfolge, auf die <code>lpszMenuName</code> verweist, bezeichnet die Menü-Ressource (über den Namen oder über das Makro <code>MAKEINTRESOURCE</code> mit einem Integer-Bezeichner), die das Standardmenü dieser Klasse bildet.
<code>lpszClassName</code>	Nimmt die Bezeichnung der Klasse auf.
<code>CbClsExtra, cbWndExtra</code>	Die Parameter <code>cbClsExtra</code> und <code>cbWndExtra</code> reservieren zusätzlichen Speicher für die Fensterklasse oder für einzelne Fenster. Anwendungen können diesen zusätzlichen Speicher nutzen, um spezifische Anwendungsinformationen zu speichern, die sich auf die Fensterklasse oder ein einzelnes Fenster beziehen.

Die ersten beiden Parameter werden bewußt zuletzt erläutert. Die Eigenschaften, die ein Fenster als individuelle komplexe Einheit erscheinen lassen, werden über die Fensterklasse und Fensterfunktion gesteuert.

WNDPROC lpfnWndProc

Der Parameter `lpfnWndProc` bestimmt die Adresse der Fensterfunktion (siehe Kapitel 7). Diese Funktion bearbeitet jede an das Fenster gerichtete Nachricht. Sie kann die Nachrichten selbst bearbeiten oder die Standard-Fensterfunktion `DefWindowProc` aufrufen. Die Funktion kann über jedes Ereignis benachrichtigt werden: das Verändern der Größe, das Verschieben des Fensters, Mausereignisse, Tastaturereignisse, Anweisungen, Aufforderung zum erneuten Zeichnen, Timer- und andere Hardware-Ereignisse usw.

Klassenstile (UINT style)

Bestimmte globale Kennzeichen der Fensterklasse werden mit Hilfe des Klassenstilparameters `style` gesteuert. Dieser Parameter kann mehrere Werte aufnehmen, die mit dem logischen ODER-Operator (`|`) verknüpft werden.

Stil	Beschreibung
CS_BYTEALIGNCLIENT	Bestimmt beispielsweise, daß der Client-Bereich eines Fensters immer an der Byte-Begrenzung der auf dem Bildschirm dargestellten Bitmap positioniert wird, um die Grafik-Performance zu erhöhen (vor allem dann, wenn Sie leistungsfähige Anwendungen für Grafik-Hardware der unteren Leistungsklasse schreiben).
CS_DBLCLKS	Gibt an, daß Windows Doppelklick-Mausnachrichten generieren soll, wenn der Anwender innerhalb des Fensters einen Doppelklick ausführt.
CS_HREDRAW	Gewährleistet, daß das Fenster vollständig neu gezeichnet wird, nachdem dessen horizontale Größe verändert wurde.
CS_VREDRAW	Gewährleistet, daß das Fenster vollständig neu gezeichnet wird, nachdem dessen vertikale Größe verändert wurde.
CS_SAVEBITS	Führt dazu, daß Windows den Speicher reserviert, der von Unix- und X-Programmierern gewöhnlich als Sicherungsspeicher bezeichnet wird. Dieser Speicherbereich nimmt eine Kopie der Fenster-Bitmap auf, so daß dieses automatisch erneut gezeichnet werden kann, nachdem bestimmte Bereiche überdeckt wurden. (Der Parameter sollte nicht unüberlegt eingesetzt werden, da der erforderliche Speicherbereich sehr groß ist und daher zu Performance-Einbußen führen kann.)

Tabelle 9.2:
Verschiedene
Klassenstile

Mit der 16-Bit-Version von Windows war es möglich, die globale Klasse einer Anwendung über den Stil `CS_GLOBALCLASS` zu registrieren. Auf diese globale Klasse konnten alle anderen Anwendungen und DLLs zugreifen. Win32 unterstützt diese Möglichkeit nicht mehr. Damit eine Anwendung eine globale Klasse verwenden kann, muß diese von einer DLL registriert werden, die jede Anwendung lädt. Eine derartige DLL kann über die Registrierung definiert werden.



9.2.2 Erstellen eines Fensters mit CreateWindow

Möchten Sie ein Fenster erstellen, müssen Sie zunächst eine neue Fensterklasse registrieren. Anschließend erzeugt die Anwendung mit Hilfe der Funktion `CreateWindow` ein Fenster:


```
HWND CreateWindow(  
    LPCTSTR lpClassName,  
    LPCTSTR lpWindowName,  
    DWORD dwStyle,  
    int x,  
    int y,  
    int nWidth,  
    int nHeight,  
    HWND hWndParent,  
    HMENU hMenu,  
    HANDLE hInstance,  
    LPVOID lpParam  
);
```

Die Fensterklasse (lpClassName)

Der erste Parameter `lpClassName` definiert den Namen der Klasse, von der das Fenster das eigene Verhalten ableitet. Die Klasse muß entweder mit `RegisterClass` registriert werden, oder eine der vordefinierten Steuerelementklassen sein. Dazu zählen die Klassen `BUTTON`, `COMBOBOX`, `EDIT`, `LISTBOX`, `SCROLLBAR` und `STATIC`. Außerdem bestehen einige Fensterklassen, die überwiegend von Windows intern genutzt werden und auf die lediglich über Integer-Bezeichner zugegriffen werden kann. Solche Klassen stehen für Menüs, das Desktop-Fenster und Symboltitel zur Verfügung, um nur einige zu nennen.

Fenster (dwStyle)

Der Parameter `dwStyle` bestimmt den Stil des Fensters. Dieser Parameter sollte nicht mit dem Klassenstil verwechselt werden, der `RegisterClass` über die Struktur `WNDCLASS` während der Registrierung der neuen Fensterklasse übergeben wird.

- ➔ Der Klassenstil bestimmt einige fixe Eigenschaften der Fenster, die sich auf die Fensterklasse beziehen.
- ➔ Der Fensterstil hingegen wird `CreateWindow` übergeben, um die variablen Eigenschaften des Fensters zu definieren. `dwStyle` kann beispielsweise verwendet werden, um zu bestimmen, wie das Fenster nach dem ersten Aufruf dargestellt werden soll (minimiert, maximiert, sichtbar oder versteckt).

Wie der Klassenstil, ist auch der Fensterstil eine Kombination einiger Werte (die mit dem bitweisen `ODER`-Operator verknüpft werden). Zusätzlich zu den allgemeinen Stilwerten, die für alle Fenster gleich sind, bestehen einige spezifische Werte für die vordefinierten Fensterklassen. Der Stil `BS_PUSHBUTTON` kann beispielsweise für Fenster der Klasse `BUTTON` verwendet werden, die `WM_COMMAND`-Nachrichten an die übergeordneten Fenster senden, wenn die Schaltfläche betätigt wird.

Auswahl verschiedener dwStyle-Werte

WS_OVERLAPPED	Spezifizieren ein Top-Level-Fenster. Ein WS_OVERLAPPED-Fenster wird immer mit einem Titel dargestellt. Überlappte Fenster werden gewöhnlich als Hauptfenster einer Anwendung verwendet.
WS_POPUP	Spezifizieren ein Top-Level-Fenster. Ein WS_POPUP-Fenster braucht keine Titelzeile. Pop-up-Fenster werden gewöhnlich als Dialogfelder eingesetzt. Wenn ein Top-Level-Fenster erstellt wird, richtet die aufrufende Anwendung das übergeordnete (oder besitzende) Fenster über den Parameter <code>hwndParent</code> ein. Das übergeordnete Fenster eines Top-Level-Fensters ist das Desktop-Fenster.
WS_CHILD	Untergeordnete Fenster werden mit dem Stil WS_CHILD erstellt. Der wesentliche Unterschied zwischen einem untergeordneten und einem Top-Level-Fenster besteht darin, daß ein Child-Fenster lediglich in dem Clientbereich des übergeordneten Fensters dargestellt werden kann.
Kombinationen	Windows definiert einige Stilkombinationen, die zum Erstellen »gewöhnlicher« Fenster sehr nützlich sind. Der Stil WS_OVERLAPPEDWINDOW kombiniert den Stil WS_OVERLAPPED mit WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX und WS_MAXIMIZEBOX, um ein charakteristisches Top-Level-Fenster zu erzeugen. WS_POPUPWINDOW kombiniert zur Darstellung eines gewöhnlichen Dialogfelds WS_POPUP mit WS_BORDER und WS_SYSMENU.

9.2.3 Erweiterte Stile und die Funktion CreateWindowEx

Die Funktion `CreateWindowEx` spezifiziert im Vergleich zur Funktion `CreateWindow` eine Kombination erweiterter Fensterstile. Erweiterte Fensterstile ermöglichen eine detaillierte Steuerung des Fensteraufbaus und der Funktionsweise des Fensters.

Mit Hilfe des Stils WS_EX_TOPMOST kann eine Anwendung beispielsweise ein Fenster über allen anderen Fenstern darstellen lassen. Ein Fenster, dem der Stil WS_EX_TRANSPARENT zugewiesen ist, verdeckt niemals andere Fenster und erhält lediglich dann eine WM_PAINT-Nachricht, wenn alle unter diesem Fenster angeordneten Fenster aktualisiert wurden.

Andere erweiterte Fensterstile beziehen sich auf Windows 95/98 sowie auf die unterschiedlichen Versionen von Windows NT ab Version 3.51. So kann Windows NT 3.51 mit der Windows-95/98-Oberfläche installiert werden. Der Stil `WS_EX_TOOLWINDOW` wird beispielsweise zur Erstellung eines Symbolleistenfensters verwendet. Ein Symbolleistenfenster verfügt über eine schmalere Titelseite. Die Eigenschaften eines derartigen Fensters sind auf dessen Einsatz als frei bewegbare Symbolleiste ausgerichtet.

Eine weitere Gruppe spezifischer Windows-95/98-Stile bestimmt das Verhalten eines Fensters hinsichtlich der verwendeten Sprache. Die Stile `WS_EX_RIGHT`, `WS_EX_RTLREADING` und `WS_EX_LEFTSCROLLBAR` können beispielsweise zusammen mit einer von rechts nach links ausgerichteten Sprache, wie Hebräisch oder Arabisch, verwendet werden.

9.3 Zeichnen der Inhalte eines Fensters

Das Zeichnen in ein Fenster geschieht mit GDI-Zeichenfunktionen. Anwendungen ermitteln die Zugriffsnummer des Anzeige-Gerätekontextes mit einer Funktion, wie `GetDC`. Anschließend ruft die Anwendung GDI-Funktionen auf, wie z.B. `LineTo`, `Rectangle` oder `TextOut`.

Ein Fenster wird neu gezeichnet, nachdem die Nachricht `WM_PAINT` empfangen wurde.

9.3.1 Die Nachricht `WM_PAINT`

Die Nachricht `WM_PAINT` wird einem Fenster gesendet, wenn Bereiche desselben erneut von der Anwendung gezeichnet werden müssen. Außerdem darf keine andere Nachricht in der Nachrichtenwarteschlange des Threads enthalten sein, der das Fenster besitzt. Anwendungen reagieren auf diese Nachricht mit einigen Zeichenanweisungen, die in den Funktionen `BeginPaint` und `EndPaint` eingeschlossen sind.

Die Funktion `BeginPaint` nimmt einige Parameter entgegen, die in der Struktur `PAINTSTRUCT` gespeichert sind:

```
typedef struct tagPAINTSTRUCT {
    HDC  hdc;
    BOOL fErase;
    RECT rcPaint;
    BOOL fRestore;
    BOOL fIncUpdate;
    BYTE rgbReserved[32];
} PAINTSTRUCT;
```

`BeginPaint` löscht den Hintergrund, sofern dies erforderlich ist, indem die Funktion der Anwendung die Nachricht `WM_ERASEBKGD` sendet.

Die Funktion `BeginPaint` sollte lediglich dann aufgerufen werden, wenn eine `WM_PAINT`-Nachricht empfangen wurde. Jedem Aufruf von `BeginPaint` muß ein Aufruf von `EndPaint` folgen.



Anwendungen können den `hDC`-Member der Struktur verwenden, um in den Client-Bereich des Fensters zu zeichnen. Der `rcPaint`-Member repräsentiert das kleinste Rechteck, das alle Bereiche des Fensters umschließt, die aktualisiert werden müssen. Sie beschleunigen den Zeichenprozeß, indem Sie die Aktivitäten der Anwendung auf diesen rechteckigen Bereich begrenzen.

9.3.2 Erneutes Zeichnen eines Fensters durch Löschen des Inhalts

Die Funktionen `InvalidateRect` und `InvalidateRgn` löschen den gesamten Fensterinhalt oder einige Bereiche des Inhalts. Windows sendet einem Fenster die Nachricht `WM_PAINT`, wenn dessen zu aktualisierender Bereich nicht leer ist, und wenn für den Thread, der das Fenster besitzt, keine Nachrichten mehr in der Nachrichtenwarteschlange vorhanden sind. Der zu aktualisierende Bereich ist die Menge aller zu aktualisierenden Bereiche, die in den vorherigen Aufrufen von `InvalidateRect` und `InvalidateRgn` spezifiziert wurden.

Diese Vorgehensweise ist ein sehr effizienter Mechanismus für Anwendungen, die Bereiche ihres Fensters aktualisieren müssen. Das Fenster wird nicht sofort aktualisiert. Statt dessen wird zunächst der entsprechende Bereich gelöscht. Während der Bearbeitung der `WM_PAINT`-Nachrichten können die Anwendungen den zu aktualisierenden Bereich überprüfen (der `rcPaint`-Member der Struktur `PAINTSTRUCT`) und lediglich die Elemente im Fenster aktualisieren, die in diesem Bereich angeordnet sind.

9.4 Fensterverwaltungs- nachrichten

Ein gewöhnliches Fenster reagiert, abgesehen von der `WM_PAINT`-Nachricht, auf viele weitere Nachrichten.

Tabelle 9.3:
Fensterverwal-
tungsnachrich-
ten

Nachricht	Beschreibung
<code>WM_CREATE</code>	Die erste Nachricht, die eine Fensterfunktion eines neu erstellten Fensters erhält. Diese Nachricht wird versendet, bevor das Fenster sichtbar ist und die Ausführung der Funktionen <code>CreateWindow</code> und <code>CreateWindowEx</code> beendet ist. Die Anwendung kann auf diese Nachricht reagieren, indem sie einige initialisierende Funktionen ausführt, bevor das Fenster angezeigt wird.
<code>WM_DESTROY</code>	Wird an ein Fenster gesendet, das bereits von dem Bildschirm entfernt wurde und nun zerstört werden kann.
<code>WM_CLOSE</code>	Gibt an, daß ein Fenster geschlossen werden soll. Die Standardimplementierung in <code>DefWindowProc</code> ruft <code>DestroyWindow</code> auf, wenn diese Nachricht eingeht. Anwendungen können beispielsweise einen Bestätigungsdialog anzeigen und <code>DestroyWindow</code> nur dann aufrufen, wenn der Anwender das Schließen des Fensters bestätigt.
<code>WM_QUIT</code>	Gewöhnlich die letzte Nachricht, die das Hauptfenster einer Anwendung erhält. Nach dieser Nachricht gibt <code>GetMessage</code> 0 zurück, woraufhin die Nachrichtenschleifen der meisten Anwendungen verlassen werden. Diese Nachricht weist darauf hin, daß die Anwendung beendet werden kann. Sie wird nach einem Aufruf von <code>PostQuitMessage</code> generiert.
<code>WM_QUERYENDSESSION</code>	Benachrichtigt die Anwendung, daß die Windows-Sitzung beendet wird. Eine Anwendung kann <code>FALSE</code> als Antwort auf diese Nachricht zurückgeben, um das Herunterfahren des Systems zu verhindern. Nachdem die Nachricht <code>WM_QUERYENDSESSION</code> bearbeitet wurde, sendet Windows allen Anwendungen die Nachricht <code>WM_ENDSESSION</code> mit den Ergebnissen der vorangegangenen Bearbeitung.

Nachricht	Beschreibung
WM_ENDSESSION	Wird an die Anwendungen gesendet, nachdem die Nachricht WM_QUERYENDSESSION bearbeitet wurde. Die Nachricht zeigt an, ob Windows heruntergefahren wird oder ob das Herunterfahren abgebrochen wurde. Steht das Herunterfahren unmittelbar bevor, kann die Windows-Sitzung zu jeder Zeit nach Bearbeitung der Nachricht WM_ENDSESSION durch alle Anwendungen beendet werden. Es ist daher wichtig, daß jede Anwendung alle Aufgaben ausführt, bevor Windows heruntergefahren wird.
WM_ACTIVATE	Zeigt an, daß ein Top-Level-Fenster aktiviert oder deaktiviert wird. Die Nachricht wird zunächst zu einem Fenster gesendet, das aktiviert werden soll. Ein Fenster, das deaktiviert werden soll, erhält die Nachricht zuletzt.
WM_SHOWWINDOW	Wird gesendet, wenn ein Fenster versteckt oder angezeigt werden soll. Sie verstecken ein Fenster, indem Sie die Funktion ShowWindow aufrufen oder die Größe eines anderen Fensters maximieren.
WM_ENABLE	Wird an ein Fenster gesendet, wenn der Zugriff auf dieses freigegeben oder gesperrt wird. Der Zugriff auf ein Fenster wird mit einem Aufruf der Funktion EnableWindow freigegeben oder gesperrt. Ein gesperrtes Fenster kann keine Eingaben von der Maus oder Tastatur entgegennehmen.
WM_MOVE	Zeigt an, daß die Position eines Fensters verändert wurde.
WM_SIZE	Zeigt an, daß die Größe eines Fensters verändert wurde.
WM_SETFOCUS	Zeigt an, daß ein Fenster fokussiert wurde. Eine Anwendung kann beispielsweise die Schreibmarke darstellen, nachdem sie diese Nachricht erhalten hat.
WM_KILLFOCUS	Zeigt an, daß ein Fenster den Fokus verliert. Stellt eine Anwendung eine Schreibmarke dar, sollte diese zerstört werden, nachdem die Nachricht erhalten wurde.
WM_GETTEXT	Gibt zu verstehen, daß der Fenstertext in einen Puffer kopiert werden soll. Für die überwiegende Anzahl der Fenster ist der Fenstertext der Titel. Für Steuerelemente, wie z.B. Schaltflächen, Textfelder, statische Steuerelemente und Kombinationsfelder, ist der Fenstertext der Text, der in dem Steuerelement dargestellt wird. Diese Nachricht wird gewöhnlich von der DefWindowProc-Funktion bearbeitet.
WM_SETTEXT	Zeigt an, daß dem Fenstertext der Inhalt eines Puffers zugewiesen werden soll. Die Funktion DefWindowProc richtet den Fenstertext ein und stellt diesen dar.

Verschiedene Nachrichten betreffen den Nicht-Clientbereich eines Fensters, also die Titelzeile, Fensterränder, Menüs und andere Bereiche, die gewöhnlich nicht durch die Anwendung aktualisiert werden. Eine Anwendung kann diese Nachrichten abfangen, um einen Fensterahmen mit einem angepaßten Aufbau oder Verhalten zu erstellen:

Tabelle 9.4:
NC-Nachrichten

Nachricht	Beschreibung
WM_NCPAINT	Der Nicht-Clientbereich eines Fensters (der Fensterrahmen) muß neu gezeichnet werden. Die Funktion <code>DefWindowProc</code> bearbeitet diese Nachricht, indem sie den Fensterrahmen erneut zeichnet.
WM_NCCREATE	Bevor die <code>WM_CREATE</code> -Nachricht an ein Fenster gesendet wird, erhält dieses die Nachricht <code>WM_NCCREATE</code> . Anwendungen können diese Nachricht verwenden, um spezifische Initialisierungen des Nicht-Clientbereichs durchzuführen.
WM_NCDESTROY	Zeigt an, daß der Nicht-Clientbereich eines Fensters zerstört wird. Diese Nachricht wird einem Fenster im Anschluß an die Nachricht <code>WM_DESTROY</code> gesendet.
WM_NCACTIVATE	Wird an ein Fenster gesendet, um anzuzeigen, daß der Nicht-Clientbereich aktiviert oder deaktiviert wurde. Die Funktion <code>DefWindowProc</code> ändert die Farbe der Titelzeile des Fensters, um die Aktivierung respektive Deaktivierung anzuzeigen.

9.5 Fensterklassen

Jedem Fenster ist eine Fensterklasse zugewiesen. Eine Fensterklasse ist entweder eine von Windows zur Verfügung gestellte Klasse oder eine benutzerdefinierte Klasse, die über die Funktion `RegisterClass` registriert wird.

9.5.1 Die Fensterfunktion

Eine Fensterklasse definiert die Eigenschaften und das Verhalten der Fenster, die auf dieser Klasse basieren. Die wohl wichtigste, aber nicht einzige Eigenschaft einer Fensterklasse ist die Fensterfunktion (siehe Kapitel 7 und 8).

Die Fensterfunktion wird immer dann aufgerufen, wenn eine Nachricht über die Funktion `SendMessage` an das Fenster gesendet oder über die Funktion `DispatchMessage` für das Fenster hinterlegt wird. Die Fenster-

funktion bearbeitet diese Nachrichten. Die Nachrichten, die nicht bearbeitet werden sollen, werden an die Standard-Fensterfunktion (`DefWindowProc` für Fenster und `DefDlgProc` für Dialoge) weitergeleitet.

Das Verhalten eines Fensters wird über die Fensterfunktion implementiert. Indem die Fensterfunktion auf verschiedene Nachrichten eingeht, bestimmt sie, wie ein Fenster auf Maus- und Tastaturereignisse reagiert, und wie der Aufbau des Fensters aufgrund dieser Ereignisse verändert werden muß. Für eine Schaltfläche könnte die Fensterfunktion beispielsweise auf die Nachricht `WM_LBUTTONDOWN` reagieren, indem sie das Fenster erneut zeichnet, so daß die Schaltfläche gedrückt dargestellt wird. Für ein Textfeld könnte die Fensterfunktion eine `WM_SETFOCUS`-Nachricht bearbeiten, indem sie die Schreibmarke in dem Textfeld darstellt.

Windows verfügt über zwei Standard-Fensterfunktionen: `DefWindowProc` und `DefDlgProc`.

*Standard-
verarbeitung*

- ➡ Die Funktion `DefWindowProc` implementiert das Standardverhalten gewöhnlicher Top-Level-Fenster. Sie bearbeitet Nachrichten für den Nicht-Clientbereich und verwaltet den Fensterrahmen. Sie implementiert außerdem einige weitere Aspekte des Verhaltens eines Top-Level-Fensters, z.B. die Reaktion auf Tastaturereignisse. Wird beispielsweise die Taste `[Alt]` betätigt, reagiert die Funktion mit einer Selektion des ersten Menüs in der Menüleiste.
- ➡ Die Funktion `DefDlgProc` wird für Dialogfelder verwendet. Zusätzlich zu dem Standardverhalten eines Top-Level-Fensters verwaltet die Funktion den Fokus innerhalb eines Dialogs. Sie implementiert das Verhalten des Dialogfelds derart, daß der Fokus nach einem Druck auf die Taste `[Tab]` auf das jeweils nächste Dialogfeld-Steuerelement gesetzt wird.

Zusätzlich zu den Standard-Fensterfunktionen stellt Windows verschiedene Fensterklassen zur Verfügung. Diese implementieren das Verhalten von Dialogfeld-Steuerelementen, wie z.B. Schaltflächen, Textfelder, Listen- und Kombinationsfelder und statischen Textfeldern. Diese Klassen werden als globale Systemklassen bezeichnet, ein Relikt aus den Tagen der 16-Bit-Version von Windows. Unter Win32 sind diese Klassen nicht länger global. Eine Änderung, die eine globale Systemklasse betrifft, bezieht sich lediglich auf die von dieser Klasse abgeleiteten Fenster derselben Anwendungen. Die Fenster anderer Anwendungen sind nicht betroffen, da Win32-Anwendungen in separaten Adreßräumen ausgeführt werden, so daß sie von anderen Anwendungen isoliert sind.

Eine Anwendung kann von einer selbst definierten Klasse oder einer von Windows zur Verfügung gestellten Fensterklasse eine neue Klasse ableiten und dieser ein neues oder modifiziertes Verhalten zuweisen. Die mit diesem Verfahren erzeugten Klassen werden als Sub- oder Superklassen bezeichnet.



Eine Anwendung sollte nicht versuchen, eine Sub- oder Superklasse aus einem Fenster zu erzeugen, dessen Besitzer ein anderer Prozeß ist.

9.5.2 Subklassen

Subklassen ersetzen die Fensterfunktion einer Fensterklasse durch die einer anderen Fensterklasse. Dies geschieht durch einen Aufruf der Funktion `SetWindowLong` oder `SetClassLong`.

- Der Aufruf von `SetWindowLong` mit dem Indexwert `GWL_WNDPROC` ersetzt die Fensterfunktion eines bestimmten Fensters.
- Ein Aufruf von `SetClassLong` mit dem Indexwert `GCL_WNDPROC` hingegen ersetzt die Fensterfunktion aller Fenster der Klasse, die nach dem Aufruf von `SetClassLong` erstellt wurde.

Betrachten Sie bitte einmal das einfache Beispiel in Listing 9.2. Sie kompilieren dieses Programm mit der Kommandozeilenanweisung

```
CL SUBCLASS.C USER32.LIB
```

Das Beispiel gibt »Hello, World!« aus. Es verwendet dazu die Systemklasse `BUTTON`. Aus dieser Klasse wird mit Hilfe einer Ersatzfensterfunktion eine Subklasse erzeugt. Diese Ersatzfunktion implementiert ein spezielles Verhalten, wenn die Nachricht `WM_LBUTTONDOWN` empfangen wird. Sie zerstört das Fenster, wodurch die Anwendung beendet wird. Das korrekte Beenden der Anwendung wird durch den Empfang der Nachricht `WM_DESTROY` gewährleistet. Die Nachricht `WM_QUIT` wird durch einen Aufruf von `PostQuitMessage` hinterlegt.

Listing 9.1: *Die Subklasse der Klasse `BUTTON`*

```
#include <windows.h>

WNDPROC OldWndProc;

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            DestroyWindow(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
}
```

```

        default:
            return CallWindowProc(OldWndProc,
                                   hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE d2,
                  LPSTR d3, int d4)
{
    MSG msg;
    HWND hwnd;

    hwnd = CreateWindow("BUTTON", "Hello, World!",
                        WS_VISIBLE | BS_CENTER, 100,100,100,80,
                        NULL, NULL, hInstance, NULL);

    OldWndProc =
        (WNDPROC)SetWindowLong(hwnd, GWL_WNDPROC, (LONG)WndProc);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Sehen Sie sich den Mechanismus der neuen Fensterfunktion `WndProc` einmal genauer an. Dieser richtet einen Verweis auf die alte Fensterfunktion ein, so daß die Standardbearbeitung der Nachrichten dort geschieht. Die alte Prozedur wird über die Win32-Funktion `CallWindowProc` aufgerufen.

Das nächste Beispiel erstellt eine Subklasse mit Hilfe der Funktion `SetClassLong`.

```

#include <windows.h>

WNDPROC OldWndProc;

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            MessageBeep(MB_OK); // Standard-Sound
        default:
            return CallWindowProc(OldWndProc,
                                   hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance,
                  HINSTANCE d2, LPSTR d3, int d4)
{
    HWND hwnd;

    hwnd = CreateWindow("BUTTON", "",
                        0, 0, 0, 0, 0,
                        NULL, NULL, hInstance, NULL);

    OldWndProc =
        (WNDPROC)SetClassLong(hwnd, GCL_WNDPROC, (LONG)WndProc);
    DestroyWindow(hwnd);
    MessageBox(NULL, "Hello, World!", "", MB_OK);
}

```

Listing 9.2:
Die Subklasse
der Klasse
BUTTON

Dieses Beispiel erstellt ein Schaltflächen-Steuerelement, das jedoch nicht angezeigt wird. Diese Schaltfläche muß eingerichtet werden, damit über ihren Handle das Verhalten der Klasse verändert werden kann. Direkt nach dem Aufruf von `SetClassLong` wird das Schaltflächen-Steuerelement zerstört.

Das Ergebnis des Aufrufs von `SetClassLong` bleibt jedoch bestehen. Das nachfolgend angezeigte Meldungsfenster enthält eine Schaltfläche mit der Bezeichnung OK. Das Verhalten dieser Schaltfläche (wird die Schaltfläche betätigt, ertönt ein Signal) ist in der neuen Fensterfunktion definiert. Wenn das Programm andere Dialoge oder Meldungsfenster darstellt, weisen alle neu erstellten Schaltflächen das veränderte Verhalten auf.

9.5.3 Globale Subklassen

In der 16-Bit-Version von Windows wurde häufig ein dem soeben vorgestellten Verfahren ähnlicher Subklassen-Mechanismus verwendet, um das systemweite Verhalten bestimmter Fenstertypen, wie z.B. Dialogfeld-Steuerelementen, zu verändern. (Die 3D-Steuerelement-Bibliothek *CTL3D.DLL* wurde beispielsweise auf diese Weise implementiert.) Wurde aus einer Fensterklasse eine Subklasse erzeugt, betraf dies alle neu erstellten Fenster dieser Klasse, unabhängig davon, welche Anwendung die Fenster generiert hatten. Leider ist diese Vorgehensweise unter Win32 nicht mehr möglich. Lediglich Fenster derselben Anwendung sind von den genannten Änderungen betroffen.

Wie können Entwickler nun das globale Verhalten bestimmter Fenstertypen beeinflussen? Dazu müssen Sie eine DLL verwenden und gewährleisten, daß diese in den Adreßraum jeder Anwendung geladen wird.

Unter Windows NT wird dazu ein Eintrag in der Windows-Registrierung generiert. Der folgende Registrierungseintrag muß modifiziert werden:

```
\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\
Windows\APPINIT_DLLS
```

DLLs, die unter diesem Registrierungsschlüssel aufgeführt sind, werden in den Adreßraum aller neu erstellten Prozesse geladen. Möchten Sie dem Schlüssel DLLs hinzufügen, trennen Sie die Pfadangaben bitte mit jeweils einem Leerzeichen.

Listing 9.4 zeigt eine DLL, die, wie in dem Beispiel des Listings 9.3, eine Subklasse aus der Klasse `BUTTON` erzeugt. Fügen Sie den vollständigen Pfad dieser DLL dem zuvor aufgeführten Registrierungsschlüssel hinzu, wird immer dann ein Signalton erzeugt, wenn eine Schaltfläche betätigt wird.

```

#include <windows.h>

WNDPROC OldWndProc;

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            MessageBeep(MB_OK); // Standard-Sound
        default:
            return CallWindowProc(OldWndProc,
                                  hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

BOOL WINAPI DllMain (HANDLE hModule, DWORD dwReason,
                    LPVOID lpReserved)
{
    HWND hwnd;

    switch(dwReason)
    {
        case DLL_PROCESS_ATTACH:
            hwnd = CreateWindow("BUTTON", "",
                                0, 0, 0, 0, 0,
                                NULL, NULL, hModule, NULL);

            OldWndProc = (WNDPROC)SetClassLong(hwnd, GCL_WNDPROC,
                                                (LONG)WndProc);
            DestroyWindow(hwnd);
    }
    return TRUE;
}

```

*Listing 9.3:
Erzeugen einer
Subklasse in
einer DLL*

Sie kompilieren diese DLL mit der Kommandozeilenanweisung

```
CL /LD BEEPBTN.C USER32.LIB
```

Der Schalter `/LD` instruiert den Compiler, eine DLL anstelle einer ausführbaren Datei zu erzeugen.

Fügen Sie der Registrierung nur eine überprüfte DLL hinzu. Eine fehlerhafte DLL kann Ihr System zum Absturz bringen oder den nächsten Systemstart vereiteln. Sollte dies geschehen, booten Sie Ihren Rechner unter MS-DOS, und benennen die DLL-Datei um, so daß diese während des nächsten Systemstarts nicht geladen wird. Befindet sich Ihre DLL auf einer NTFS-Partition, ist diese Vorgehensweise nicht möglich.



Das Hinzufügen des Pfads der DLL zum Registrierungsschlüssel `APPINIT_DLLS` ist die einfachste, aber nicht einzige Technik, um eine DLL in den Adreßraum einer anderen Anwendung zu implementieren. Dieses Verfahren weist außerdem einige Nachteile auf. Windows 95/

98 unterstützt diese Methode beispielsweise nicht. (Weitere Informationen finden Sie in der Microsoft-Developer-Bibliothek. Unsere Versuche ergaben, daß dieses Verfahren nicht angewendet werden kann. Microsoft bestätigte, daß der genannte Registrierungseintrag unter Windows 95/98 nicht unterstützt wird.)

Ein weiterer Nachteil dieser Technik besteht darin, daß eine DLL in den Adreßraum jeder GUI-Anwendung geladen wird, an die *USER32.DLL* gebunden ist. Der kleinste Fehler in Ihrer DLL wirkt sich auf die Stabilität des gesamten Systems aus.

Glücklicherweise sind andere Techniken verfügbar, die Ihnen das Implementieren Ihrer DLL in den Adreßraum eines anderen Prozesses ermöglichen.

Die erste dieser Techniken erfordert den Aufruf einer Windows-Hook-Funktion. Mit *SetWindowsHookEx* installieren Sie eine Hook-Funktion in dem Adreßraum einer anderen Anwendung. Auf diese Weise können Sie einer Fensterklasse, die im Besitz einer anderen Anwendung ist, eine neue Fensterfunktion hinzufügen.

Die zweite Technik verwendet die Funktion *CreateRemoteThread*, mit deren Hilfe ein Thread erzeugt wird, der in dem Kontext eines anderen Prozesses ausgeführt wird.

9.5.4 Superklassen

Eine Superklasse ist eine neue Klasse, die auf einer bestehenden Klasse basiert. Eine Anwendung verwendet die Funktion *GetClassInfo*, um die Struktur *WNDCLASS* der bestehenden Klasse zu ermitteln. Die Struktur beschreibt die Klasse. Nachdem die Struktur entsprechend modifiziert wurde, kann sie in einem Aufruf der Funktion *RegisterClass* verwendet werden, um die neue Klasse zu registrieren.

Das Beispiel in Listing 9.5 demonstriert das Erstellen einer Superklasse. In diesem Beispiel wird eine neue Fensterklasse mit der Bezeichnung *BEEPBUTTON* erstellt, deren Verhalten auf dem der Standardklasse *BUTTON* basiert. Die neue Klasse wird dazu verwendet, eine einfache Nachricht darzustellen. Kompilieren Sie das Programm mit der Kommandozeilenanweisung

```
CL SUPERCLS.C USER32.LIB
```

```
Listing 9.4: #include <windows.h>
Die Superklasse WNDPROC OldWndProc;
der Klasse LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
BUTTON WPARAM wParam, LPARAM lParam)
{
```

```
switch(uMsg)
{
    case WM_LBUTTONDOWN:
        MessageBeep(MB_OK); // Standard-Sound
    default:
        return CallWindowProc(OldWndProc,
                               hwnd, uMsg, wParam, lParam);
}
return 0;
}
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE d2,
                  LPSTR d3, int d4)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    GetClassInfo(hInstance, "BUTTON", &wndClass);
    wndClass.hInstance = hInstance;
    wndClass.lpszClassName = "BEEPBUTTON";
    OldWndProc = wndClass.lpfnWndProc;
    wndClass.lpfnWndProc = WndProc;
    RegisterClass(&wndClass);
    hwnd = CreateWindow("BEEPBUTTON", "Hello, World!",
                       WS_VISIBLE | BS_CENTER, 100,100,100,80,
                       NULL, NULL, hInstance, NULL);
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (msg.message == WM_LBUTTONUP)
        {
            DestroyWindow(hwnd);
            PostQuitMessage(0);
        }
        DispatchMessage(&msg);
    }
    return msg.wParam;
}
```

Sie haben den Unterschied zwischen Subklassen und Superklassen kennengelernt und wissen nun, wie diese implementiert werden. Doch wie unterscheiden sich diese beiden Klassen hinsichtlich ihrer Verwendung? Wann setzen Sie Subklassen ein, und wann verwenden Sie Superklassen?

Der Unterschied ist recht einfach. Subklassen modifizieren das Verhalten einer bereits bestehenden Klasse, während Superklassen eine neue Klasse erstellen, die auf einer bestehenden Klasse basiert. Verwenden Sie somit Subklassen, ändern Sie implizit das Verhalten jedes Features Ihrer Anwendung, das sich auf die Klasse bezieht, aus der Sie eine Subklasse generiert haben. Superklassen hingegen betreffen lediglich die Fenster, die explizit auf der neuen Klasse basieren. Fenster, die auf der originalen Klasse basieren, sind nicht betroffen.

9.6 Dialogfelder

Eine Anwendung verwendet zusätzlich zu dem Hauptfenster mit der Titel- und der Menüleiste sowie dem von der Anwendung definierten Inhalt des Fensters häufig ein Dialogfeld, um Informationen mit dem Anwender auszutauschen. Das Hauptfenster der Anwendung wird gewöhnlich während der gesamten Ausführung des Programms angezeigt, während Dialogfelder lediglich für den kurzen Zeitraum des Datenaustauschs geöffnet werden. Dies ist jedoch nicht der wesentliche Unterschied zwischen einem Hauptfenster und einem Dialogfeld. Einige Anwendungen nutzen sogar einen Dialog als Hauptfenster. In anderen Anwendungen wird ein Dialog bisweilen beinahe für die gesamte Dauer der Programmausführung dargestellt.

Ein Dialogfeld enthält gewöhnlich mehrere Dialogfeld-Steuerelemente, die in diesem Fall Child-Fenster bilden, über die der Anwender und die Anwendung Daten austauschen. Verschiedene Win32-Funktionen dienen dem Entwurf, der Darstellung sowie der Verwaltung der Inhalte eines Dialogs. Anwendungsentwickler müssen nicht die Steuerelemente eines Dialogs zeichnen oder die Anwenderschnittstellenereignisse verwalten. Statt dessen können sie sich darauf konzentrieren, den Datenaustausch zwischen den Dialogfeld-Steuerelementen und der Anwendung zu steuern.

Dialoge sind vielseitige Elemente unter Windows. Um ihre effiziente Anwendung zu erleichtern, stellt Windows zwei Dialogfeldtypen zur Verfügung: Ungebundene und gebundene Dialoge.

9.6.1 Modale und nicht-modale Dialoge

Modale Dialoge

Ruft eine Anwendung einen modalen Dialog auf, ist der Zugriff auf das Fenster, das den Dialog besitzt, gesperrt. Die Anwendung kann somit in dieser Zeitspanne nicht genutzt werden. Der Anwender muß zunächst den modalen Dialog beenden, bevor die Ausführung der Anwendung fortgesetzt werden kann.

DialogBox() Modale Dialoge werden gewöhnlich mit Hilfe der Funktion `DialogBox` erstellt und aktiviert. Diese Funktion erzeugt das Dialogfenster aus einer Dialogvorlagen-Ressource und stellt den Dialog anschließend gebunden dar. Die Anwendung, die `DialogBox` aufruft, übergibt der Funktion die Adresse einer Rückruffunktion. `DialogBox` wird erst dann beendet, wenn der Dialog durch einen Aufruf von `EndDialog` geschlos-

sen wird. Dieser Aufruf geschieht über die angegebene Rückruffunktion (indem beispielsweise auf ein Anwenderschnittstellenereignis, wie einem Klick auf die Schaltfläche OK, reagiert wird).

Obwohl ein modaler Dialog ohne Besitzer erstellt werden kann, ist diese Vorgehensweise nicht empfehlenswert. Wird solch ein Dialogfeld verwendet, müssen verschiedene Aspekte berücksichtigt werden. Da das Hauptfenster einer Anwendung nicht gesperrt ist, müssen einige Vorkehrungen getroffen werden, um zu gewährleisten, daß die zu bearbeitenden Nachrichten weiterhin an dieses Fenster gesendet oder für dieses hinterlegt werden. Windows zerstört einen Dialog nicht, der versteckt ist oder keinen Besitzer hat, wenn andere Fenster der Anwendung zerstört werden.

Nicht-modale Dialoge

Im Gegensatz zu modalen Dialogen wird die Ausführung einer Anwendung während der Darstellung eines nicht-modalen Dialogs nicht unterbrochen, indem das Fenster, das den Dialog besitzt, gesperrt wird. Nicht-modale Dialoge werden über dem besitzenden Fenster angezeigt, wenn dieses fokussiert wird. Sie bieten eine effektive Möglichkeit, Informationen, die für den Anwender relevant sind, kontinuierlich darstellen zu lassen.

Nicht-modale Dialoge werden gewöhnlich mit der Funktion `CreateDialog` erstellt. Da kein Äquivalent der Funktion `DialogBox` für nicht-modale Dialoge besteht, muß die Anwendung die Nachrichten für den nicht-modalen Dialog entgegennehmen und weiterleiten. Die meisten Anwendungen verwenden dazu ihre Hauptnachrichtenschleife. Damit der Dialog wie erwartet auf Tastaturereignisse reagiert, so daß der Anwender die einzelnen Steuerelemente über Tastaturkürzel anwählen kann, muß die Anwendung die Funktion `IsDialogMessage` aufrufen.

CreateDialog()

Ein nicht-modaler Dialog gibt keinen Rückgabewert an den Besitzer zurück. Der Besitzer und der nicht-modale Dialog können jedoch über die Funktion `SendMessage` miteinander kommunizieren.

Die Dialogfeldprozedur für einen nicht-modalen Dialog muß nicht die Funktion `EndDialog` aufrufen. Der Dialog wird gewöhnlich durch einen Aufruf von `DestroyWindow` zerstört. Diese Funktion kann als Reaktion auf ein von der Dialogfeldprozedur ausgelöstes Anwenderschnittstellenereignis aufgerufen werden.

Anwendungen sind, bevor sie beendet werden, selbst für die Zerstörung aller ungebundenen Dialoge verantwortlich.

9.6.2 Meldungsfenster

Meldungsfenster sind spezielle Dialoge, die eine benutzerdefinierte Nachricht ausgeben und einen Titel sowie vordefinierte Schaltflächen und Symbole anzeigen. Sie werden dazu verwendet, dem Anwender kurze, informative Nachrichten anzuzeigen und eine begrenzte Auswahl an Schaltflächen zur Verfügung zu stellen. Meldungsfenster melden beispielsweise einen aufgetretenen Fehler und ermitteln von dem Anwender, ob die zu dem Fehler führende Operation erneut ausgeführt oder abgebrochen werden soll.

MessageBox() Meldungsfenster werden mit der `MessageBox`-Funktion erstellt und angezeigt. Die Anwendung, die diese Funktion aufruft, bestimmt die darzustellende Zeichenfolge sowie einige Flags, die den Typ und den Aufbau des Meldungsfensters bestimmen.

Meldungsfenster werden gewöhnlich modal dargestellt. Eine Anwendung kann jedoch zwei weitere Verhaltensmodi bestimmen:

- den Task-gebundenen Modus und
- den System-gebundenen Modus.

Verwenden Sie ein Task-gebundenes Meldungsfenster, wenn Sie die Interaktion mit allen Top-Level-Fenstern einer Anwendung unterbinden möchten. Die Interaktion mit dem Fenster, das das Meldungsfenster besitzt, ist natürlich weiterhin möglich.

Ein System-gebundenes Meldungsfenster sollte lediglich für Situationen eingesetzt werden, die die unbedingte Aufmerksamkeit des Anwenders erfordern, wie z.B. schwerwiegende Fehler. System-gebundene Meldungsfenster sperren so lange die Interaktion mit allen anderen Anwendungen, bis der Anwender auf die Nachricht reagiert.



System-gebundene Meldungsfenster sollten mit großer Sorgfalt eingesetzt werden. Nichts ist ärgerlicher als eine fehlprogrammierte Anwendung, die solch ein Meldungsfenster wiederholt in einer Schleife anzeigen läßt. In diesem Fall ist ein Zugriff auf das gesamte System unmöglich.

9.6.3 Dialogvorlagen

Wenngleich ein Dialog von Grund auf erstellt werden kann, verwenden die meisten Anwendungen eine Dialogvorlagen-Ressource, um den Typ und Aufbau der Steuerelemente eines Dialogs zu bestimmen.

Dialogvorlagen können über spezielle Anweisungen in der Ressourcendatei manuell oder über einen visuellen Ressourcendatei-Editor, wie dem Ressource-Editor des Visual Studio, erstellt werden (siehe Kapitel 10).

Die Dialogvorlage definiert den Stil, die Position sowie die Größe des Dialogs und führt dessen Steuerelemente auf. Der Stil, die Position und der Aufbau der Steuerelemente werden ebenfalls in der Dialogvorlage gespeichert. Die verschiedenen Dialogfeldfunktionen zeichnen den gesamten Dialog, basierend auf der Dialogvorlage. Diese Funktionen berücksichtigen nicht Steuerelemente, die von dem Besitzer gezeichnet werden.

9.6.4 Die Dialogfeldprozedur

Der Begriff Dialogfeldprozedur ist eine andere Bezeichnung für die Fensterfunktion des Dialogfelds. Zwischen einer Dialogfeldprozedur und einer Fensterfunktion bestehen nur geringe Unterschiede. Erwähnenswert ist lediglich der Umstand, daß `DefDlgProc` und nicht `DefWindowProc` die Standard-Dialogfeldprozedur zur Bearbeitung von Nachrichten ist.

Eine Dialogfeldprozedur reagiert gewöhnlich auf `WM_INITDIALOG`- und `WM_COMMAND`-Nachrichten. Andere Nachrichten werden nur selten bearbeitet. Erhält die Dialogfeldprozedur die Nachricht `WM_INITDIALOG`, initialisiert sie die Steuerelemente des Dialogs. Windows sendet keine `WM_CREATE`-Nachricht zur Dialogfeldprozedur. Statt dessen wird die Nachricht `WM_INITDIALOG` gesendet. Dies geschieht jedoch erst, nachdem alle Steuerelemente erstellt wurden und bevor der Dialog angezeigt wird. Auf diese Weise ist es der Dialogfeldprozedur möglich, die Steuerelemente korrekt zu initialisieren, bevor der Anwender diese sieht.

Die überwiegende Anzahl der Steuerelemente sendet `WM_COMMAND`-Nachrichten zu dem besitzenden Fenster (also zum Dialog). Damit die Funktion ausgeführt werden kann, die von einem Steuerelement repräsentiert wird, muß die Dialogfeldprozedur auf `WM_COMMAND`-Nachrichten reagieren, indem sie zunächst das Steuerelement identifiziert und anschließend die entsprechende Aktion ausführt.

9.7 Standarddialoge

Win32 implementiert einige häufig verwendete Dialoge, die eine einheitliche Benutzerschnittstelle garantieren und den Programmierer von der Notwendigkeit befreien, eigene Implementierungen für jede Anwendung vornehmen zu müssen.

Diese Standarddialoge sind jedem Windows-Anwender vertraut. Sie werden zum Öffnen und Speichern von Dateien, zum Selektieren einer Farbe oder einer Schriftart, zum Einrichten des Druckers und zum Drucken sowie zum Selektieren einer Seitengröße und zum Suchen und Ersetzen von Text verwendet.

Es gibt zwei Möglichkeiten, Standarddialoge einzusetzen.

- ➔ Anwendungen können Standarddialoge verwenden, ohne Modifizierungen daran vorzunehmen. Dazu ruft die Anwendung eine der Standarddialog-Funktionen auf, die in der Win32-API enthalten sind.
- ➔ Eine weitere Möglichkeit besteht darin, einen Standarddialog den Bedürfnissen der Anwendung anzupassen, indem eine spezielle Hook-Funktion implementiert und dieser eine benutzerdefinierte Dialogvorlage übergeben wird.

Windows 95/98 verwendet Standarddialoge, die sich von denen unterscheiden, die Windows 3.1 und Windows NT anbieten. Die meisten dieser Differenzen betreffen den Aufbau und nicht die grundsätzliche Verwendung der Dialoge. Besondere Unterschiede werden in den folgenden Abschnitten erwähnt.



Der Aufbau aller Standarddialoge wurde für Windows 95/98 verändert. Anwendungen, die eigene Dialogvorlagen zur Verfügung stellen, müssen diesen Umstand berücksichtigen, um einen visuellen Aufbau zu präsentieren, der dem Betriebssystem angepaßt ist.

Tritt in einem Standarddialog ein Fehler auf, wird gewöhnlich die Funktion `CommDlgExtendedError` verwendet, um zusätzliche Informationen über die Ursache des Problems zu ermitteln.

Dialog	Beschreibung

Datei öffnen



Mit Hilfe dieses Dialogs durchsucht der Anwender das Dateisystem und selektiert eine Datei zum Öffnen.

Der Dialog DATEI ÖFFNEN wird nach einem Aufruf der Funktion `GetOpenFileName` angezeigt. Der einzige Parameter dieser Funktion ist ein Zeiger auf eine `OPENFILENAME`-Struktur. Die Member-Variablen dieser Struktur initialisieren den Dialog und optional die Adresse einer Hook-Funktion sowie den Namen einer benutzerdefinierten Dialogvorlage, die zum Anpassen des Dialogs verwendet werden kann. Wird der Dialog verlassen, kann die Anwendung die Auswahl des Anwenders aus dieser Struktur ermitteln.

Datei speichern

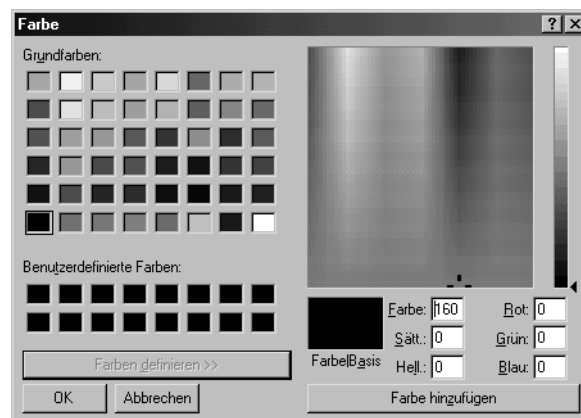


Mit Hilfe dieses Dialogs durchsucht der Anwender das Dateisystem und selektiert eine Datei zum Öffnen.

Tabelle 9.5:
Die Standard-
dialoge

Dialog**Beschreibung**

Der Dialog **SPEICHERN UNTER** wird nach einem Aufruf von `GetSaveFileName` angezeigt. Dieser Funktion wird ebenfalls ein Zeiger auf eine `OPENFILENAME`-Struktur als einziger Parameter übergeben.

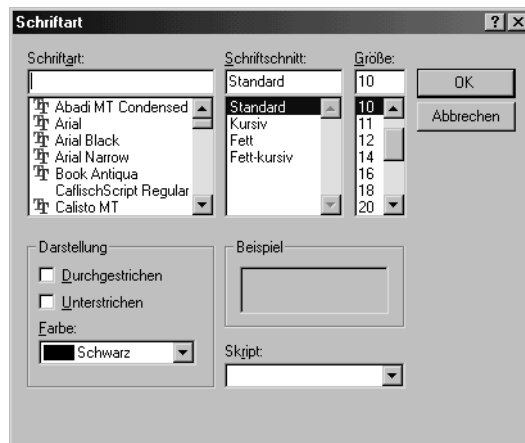
Farbe

Der Dialog **FARBE** wird verwendet, wenn der Anwender eine Farbe selektieren soll. Die Auswahl geschieht aus der Systempalette. Das Bestimmen einer benutzerdefinierten Farbe ist ebenfalls möglich.

Der Dialog **FARBE** wird nach einem Aufruf der Funktion `ChooseColor` angezeigt. Anwendungen können die Initialisierungswerte dieses Dialogs über einen Zeiger auf die Struktur `CHOOSECOLOR` steuern, die der Funktion als einziger Parameter übergeben wird. Das Anpassen des Dialogs geschieht ebenfalls über diese Struktur. Dazu muß in der Struktur eine Hook-Funktion und der Name einer Dialogfeldvorlage vermerkt werden. Wird der Dialog verlassen, ist die Farbauswahl in der Member-Variablen `rgb-Result` in der `CHOOSECOLOR`-Struktur gespeichert.

Dialog**Beschreibung**

Schriftart

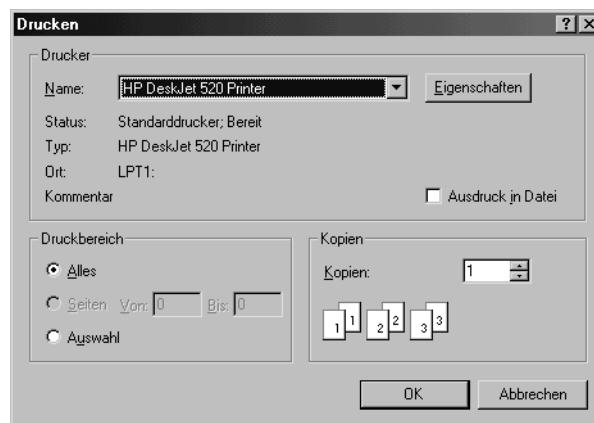


Mit Hilfe des Dialogs SCHRIFTART selektiert der Anwender eine Schrift, einen Schriftstil, eine Schriftgröße, besondere Effekte, die Schriftfarbe und, für Windows 95/98, die Schriftgruppe.

Der Dialog SCHRIFTART wird über die Struktur CHOOSE_FONT initialisiert. Diese Struktur kann ebenfalls dazu verwendet werden, eine Hook-Funktion und den Namen einer benutzerdefinierten Dialogvorlage anzugeben. Die Member-Variable der Struktur, die mit `lpLogFont` bezeichnet ist, verweist auf die Struktur LOGFONT. Diese initialisiert den Dialog und nimmt die Informationen über die selektierte Schrift auf, wenn der Dialog geschlossen wird. LOGFONT kann einem Aufruf der GDI-Funktion `CreateFontIndirect` übergeben werden, um den Font zur Verwendung einzurichten.

Dialog	Beschreibung

Drucken



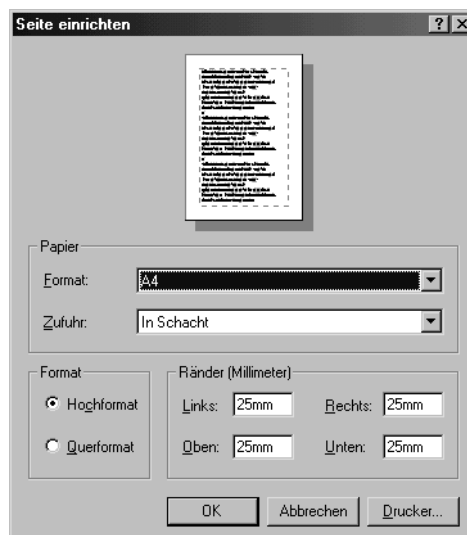
In der Windows-3.1-Version des Dialogs DRUCKEN selektierte der Anwender Druckparameter und startete den Ausdruck. Der Drucker wurde über den Dialog DRUCKER EINRICHTEN konfiguriert.

Unter Windows 95/98 sind diese Dialoge anders aufgebaut. Der Dialog DRUCKEN (Abbildung 9.6) kombiniert die Funktionalität der Windows-3.1-Dialoge DRUCKEN und DRUCKER EINRICHTEN. Die Auswahl der Papierzufuhr und -größe, die bisher in dem Dialog DRUCKER EINRICHTEN geschah, ist nun in einem neuen Dialog möglich, der mit SEITE EINRICHTEN bezeichnet ist.

Eine Anwendung muß zunächst den Inhalt der Struktur PRINTDLG konfigurieren, bevor der Dialog DRUCKEN verwendet werden kann. Der Dialog wird schließlich nach einem Aufruf der Funktion `PrintDlg` angezeigt, der ein Zeiger auf die Struktur übergeben wird.

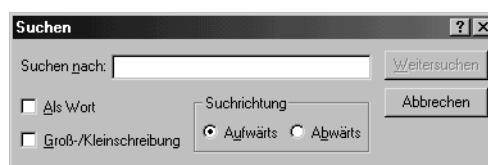
Dialog	Beschreibung

Seite einrichten



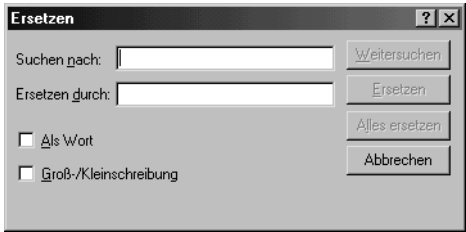
Der Dialog SEITE EINRICHTEN wird angezeigt, wenn die Anwendung die Funktion `PageSetupDlg` aufruft. Der einzige Parameter dieser Funktion ist ein Zeiger auf die Struktur `PAGESETUPDLG`. Über diese Struktur kontrolliert die Anwendung die Felder des Dialogs und modifiziert diese, sofern erforderlich. Verläßt der Anwender den Dialog, ist seine Auswahl in der Struktur gespeichert.

Suchen



Nicht-modaler Dialog zur Textsuche. Die Anwendung, die den SUCHEN-Dialog erstellt, ist für die Bereitstellung der Nachrichtenschleife sowie für die Weiterleitung der Nachrichten über die Funktion `IsDialogMessage` verantwortlich.

Der Dialog SUCHEN, wird nach einem Aufruf der Funktion `FindText` angezeigt. Die Funktion gibt einen Dialog-Handle zurück, der in der Nachrichtenschleife der Anwendung in einem Aufruf der Funktion `IsDialogMessage` verwendet werden kann. Der Dialog wird über die Struktur `FINDREPLACE` initialisiert, die alle Werte aufnimmt, die der Anwender in dem Dialog angibt.

Dialog	Beschreibung
Ersetzen	Der Dialog kommuniziert über verschiedene Nachrichten mit anderen Fenstern. Anwendungen sollte die Nachrichtenzeichenfolge »FINDMSGSTRING« mit einem Aufruf der Funktion RegisterWindowMessage registrieren, bevor FindText aufgerufen wird. Der Dialog SUCHEN sendet diese Nachricht immer dann zur Anwendung, nachdem der Anwender einen neuen Wert eingegeben hat.  Nicht-modaler Dialog mit dem Textstellen ersetzt werden können. Die Anwendung, die den ERSETZEN-Dialog erstellt, ist für die Bereitstellung der Nachrichtenschleife sowie für die Weiterleitung der Nachrichten über die Funktion IsDialogMessage verantwortlich. Empfängt die Anwendung eine Nachricht von einem der Dialoge SUCHEN oder ERSETZEN, kann sie die Flags der Struktur FINDREPLACE überprüfen, um zu ermitteln, welche Aktion der Anwender selektiert hat.



Die Windows-95/98-Version der Standarddateialoge weist ein neues Feature auf. Soll der Dialog angepaßt werden, ist eine Kopie der gesamten Dialogfeldvorlage nicht mehr erforderlich, um Modifikationen vorzunehmen. Statt dessen ist es möglich, eine Dialogfeldvorlage zu generieren, die lediglich die Steuerelemente enthält, die Sie dem Standarddialog hinzufügen möchten. Das optionale Feld, das mit der ID `stc32` bezeichnet ist, zeigt an, wo die Standardkomponenten des Dialogs positioniert werden sollen.

Ein Beispiel für Standarddialoge

Das Beispiel erstellt alle Standarddialoge und zeigt diese nacheinander an. Es erfüllt keine besondere Funktion, sondern demonstriert Ihnen lediglich, wie diese Dialoge verwendet werden. Das Beispiel wird mit folgender Kommandozeilenanweisung kompiliert:

```
CL COMMDLGS.C COMDLG32.LIB USER32.LIB
```

```

#include <windows.h>

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    OPENFILENAME ofn;
    CHOOSECOLOR cc;
    CHOOSEFONT cf;
    PRINTDLG pd;
    PAGESETUPDLG psd;
    FINDREPLACE fr;
    COLORREF crCustColors[16];
    LOGFONT lf;
    char szFindWhat[80];
    char szReplaceWith[80];
    HWND hdlgFt, hdlgFr;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE+1);
        wndClass.lpszClassName = "COMMDLGS";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("COMMDLGS",
                       "Common Dialogs Demonstration",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                       NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    memset(&ofn, 0, sizeof(ofn));
    ofn.lStructSize = sizeof(OPENFILENAME);
    GetOpenFileName(&ofn);
    memset(&ofn, 0, sizeof(ofn));
    ofn.lStructSize = sizeof(OPENFILENAME);
    GetSaveFileName(&ofn);
    memset(&cc, 0, sizeof(cc));
    memset(crCustColors, 0, sizeof(crCustColors));
    cc.lStructSize = sizeof(cc);
    cc.lpCustColors = crCustColors;
    ChooseColor(&cc);

```

Listing 9.5:
Standarddialoge

```

memset(&cf, 0, sizeof(cf));
memset(&lf, 0, sizeof(lf));
cf.lStructSize = sizeof(cf);
cf.lpLogFont = &lf;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
ChooseFont(&cf);
memset(&pd, 0, sizeof(pd));
pd.lStructSize = sizeof(pd);
PrintDlg(&pd);
memset(&psd, 0, sizeof(psd));
psd.lStructSize = sizeof(psd);
PageSetupDlg(&psd);
memset(&fr, 0, sizeof(fr));
memset(szFindWhat, 0, sizeof(szFindWhat));
memset(szReplaceWith, 0, sizeof(szReplaceWith));
fr.lStructSize = sizeof(fr);
fr.hwndOwner = hwnd;
fr.lpstrFindWhat = szFindWhat;
fr.lpstrReplaceWith = szReplaceWith;
fr.wFindWhatLen = sizeof(szFindWhat);
fr.wReplaceWithLen = sizeof(szReplaceWith);
hdlgFt = FindText(&fr);
hdlgFr = ReplaceText(&fr);

while (GetMessage(&msg, NULL, 0, 0))
    if(!IsDialogMessage(hdlgFt, &msg))
        if(!IsDialogMessage(hdlgFr, &msg))
            DispatchMessage(&msg);
return msg.wParam;
}

```

OLE-Standarddialoge

Das Betriebssystem stellt mit der OLE-2-Implementierung Standarddialoge mit den folgenden Funktionen zur Verfügung: Einfügen von Objekten, Spezielles Einfügen, Ändern der Quelle, Bearbeiten von Bindungen, Aktualisieren von Bindungen, Objekteigenschaften, Konvertierung und Ändern von Symbolen. Anwendungen rufen diese Dialoge gewöhnlich nicht direkt auf, sondern verwenden die Klassen der MFC (besonders die Mantelklassen dieser Dialoge), um die OLE-Funktionalität zu implementieren.

9.8 Steuerelemente

Ein Steuerelement ist ein spezielles Fenster, das dem Anwender gewöhnlich die Ausführung einfacher Funktionen ermöglicht und die darauf bezogenen Nachrichten an das besitzende Fenster sendet. Eine Schaltfläche hat beispielsweise eine einfache Funktion: Der Anwender kann die Schaltfläche betätigen. Geschieht dies, sendet die Schaltfläche eine `WM_COMMAND`-Nachricht an das besitzende Fenster (gemeinhin ein Dialog).

Windows bietet verschiedene integrierte Steuerelementklassen für die meisten der vorwiegend verwendeten Steuerelemente an. Ein Beispieldialog mit einigen dieser Steuerelemente ist in Einige Standardsteuerelemente dargestellt.

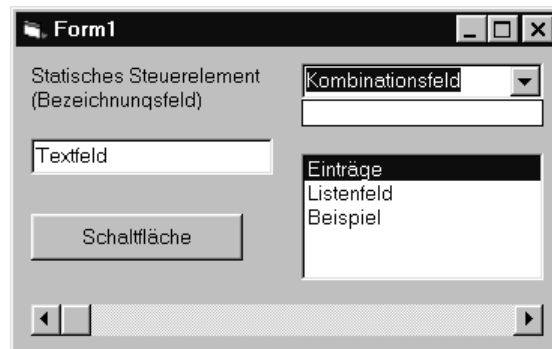


Abbildung 9.2:
Einige Standardsteuerelemente

Windows 95/98 enthält einige neue Steuerelementklassen, die als Windows-95/98-Standardsteuerelemente bezeichnet werden. Diese Bezeichnung kann zu Mißverständnissen führen, da die neuen Steuerelemente auch unter Windows NT 3.51 und WIN32S, Version 1.3, erhältlich sind.

Anwendungen können ebenfalls eigene Steuerelemente erstellen. Diese werden von den Standardsteuerelementklassen abgeleitet oder ohne Vorlage erzeugt.

Die Steuerelementklasse und der Stil des Steuerelements (der das Verhalten einer Klasse bestimmt) werden gewöhnlich in der Ressourcedatei der Anwendung definiert. Anwendungen, die eigene Steuerelemente erstellen, selektieren die entsprechende Klasse und bestimmen den Stil mit einem Parameter, der `CreateWindow` übergeben wird.

Statische Textfelder

Statische Textfelder sind wahrscheinlich die einfachsten Steuerelemente. Sie stellen lediglich Text dar, wie z.B. die Bezeichnungen anderer Steuerelemente. Statische Textfelder reagieren nicht auf Anwenderschnittstelleneignisse und senden keine Nachrichten an das besitzende Fenster.

Schaltflächen

Schaltflächen reagieren auf Mausklicks. Es gibt verschiedene Schaltflächentypen.

- ➔ Eine gewöhnliche Schaltfläche sendet eine `WM_COMMAND`-Nachricht an das besitzende Fenster, wenn sie betätigt wird.
- ➔ Ein Kontrollkästchen kann einen von zwei möglichen Zuständen annehmen, aktiviert oder deaktiviert. Eine Variante des Kontrollkästchens verwendet einen dritten Status, der mit abgeblendet bezeichnet wird.
- ➔ Ein Optionsfeld ist ein Steuerelement, das gewöhnlich in Gruppen eingesetzt wird. Optionsfelder werden zur Auswahl einer Option aus mehreren Auswahlmöglichkeiten verwendet.

Varianten dieser Steuerelementstile definieren sekundäre Verhaltensaspekte.

Eingabefelder

Ein Eingabefeld ist ein rechteckiger Bereich, in dem der Anwender unformatierten Text eingeben kann. Dieser Text kann aus wenigen Zeichen bestehen, wie z.B. dem Namen einer Datei, oder eine vollständige Textdatei sein. Der Client-Bereich des Windows-Editors ist beispielsweise ein großes Textfeld. Anwendungen kommunizieren gewöhnlich über einige Nachrichten mit dem Textfeld. Die Nachrichten werden dazu verwendet, Text aus dem Textfeld auszulesen oder dort auszugeben.

Listenfelder

Ein Listenfeld enthält mehrere in Zeilen angeordnete Werte. Der Anwender selektiert mit der Maus den gewünschten Wert in der Liste. Enthält das Listenfeld so viele Werte, daß diese nicht mehr gleichzeitig angezeigt werden können, wird eine vertikale Bildlaufleiste an dem Listenfeld angeordnet.

Kombinationsfelder

Ein Kombinationsfeld kombiniert die Funktionalität eines Listenfeldes mit der eines einzeiligen Eingabefelds. Der Anwender kann einen Wert in den Textfeldbereich des Kombinationsfelds eingeben. Alternativ dazu betätigt er die neben dem Eingabefeld angeordnete Schaltfläche, die mit einem nach unten weisenden Pfeil beschriftet ist, um sich den Inhalt des Listenfelds anzeigen zu lassen. Dort kann der Anwender einen Eintrag selektieren.

Bildlaufleisten

Eine Bildlaufleiste besteht aus einem rechteckigen Bereich, an dessen Ende zwei Pfeile angeordnet sind. Innerhalb des rechteckigen Bereichs befindet sich ein Schieberegler. Es gibt vertikale und horizontale Bildlaufleisten. Bildlaufleisten werden dazu verwendet, die Position des sichtbaren Abschnitts innerhalb eines größeren Bereichs anzuzeigen. Anwendungen verwenden Bildlaufleisten außerdem, um die Funktionalität eines Schieberegler-Steuerelements zu implementieren. Eines der neuen Windows-95/98-Steuerelemente ist ein Schieberegler-Steuerelement, so daß Bildlaufleisten nicht länger für diesen Zweck eingesetzt werden müssen.

Windows 95/98-Standardsteuerelemente

Windows 95/98 verwendet einige neue Standardsteuerelemente. Neu seit Windows 98 ist die Unterstützung der Internet Explorer 4.0-Steuerelemente (IE-Werkzeugleiste, IP-Adresse etc.)

In den Klammern stehen die Namen der zugehörigen MFC-Klassen.

Animation	Dient der Darstellung kleiner Animationen während zeitintensiver Operationen. (CAnimateCtrl).
Datums-/Zeitauswahl	Zur Einstellung von Datum und Zeit. (CDateTimeCtrl).
Tastenkürzel	Nimmt Tastenkombinationen vom Anwender entgegen, die von der Anwendung zur Konfiguration einer Zugriffstaste über die WM_SETHOTKEY-Nachricht verwendet werden kann. (CHotkeyCtrl)
Monatskalender	Zur Auswahl eines Datums. (CMonthCalCtrl)
Fortschrittsanzeige	Gibt Aufschluß über den Fortschritt eines zeitintensiven Prozesses. Fortschrittsleisten nehmen keine Eingaben vom Anwender entgegen. Sie werden lediglich zu informativen Zwecken eingesetzt. (CProgressCtrl)
Werkzeugleiste	In der Größe veränderbare Werkzeugleiste, die andere Kindfenster aufnehmen kann. (CReBarCtrl)

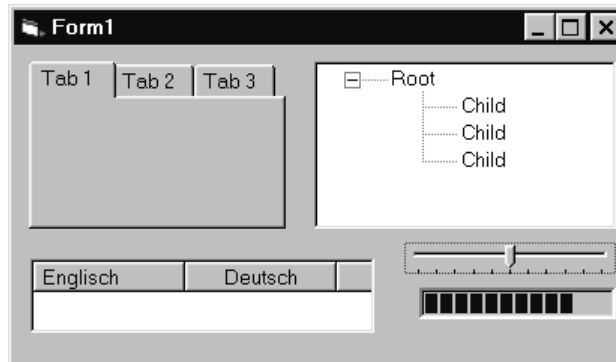


Symbolleiste	Leiste zur Aufnahme von Symbolschaltflächen. (CToolBarCtrl)
Statusleiste	Stellt am unteren Rand des Fensters (gewöhnlich das Hauptfenster der Anwendung) eine Statusleiste dar. (CStatusBarCtrl)
Bilderliste	Zur Verwaltung von Symbolen und Bitmaps. (CImageList)
QuickInfo	Erzeugt ein QuickInfo-Fenster mit erklärendem Text. (CToolTipCtrl)
Schieberegler	Die Funktion dieses Steuerelements ist dem eines Lautstärkereglers ähnlich, den viele Stereo-Systeme verwenden. Der Anwender kann den Schieberegler mit der Maus an die gewünschte Position über das Steuerelement bewegen. Schieberegler sind besonders für Multimedia-Anwendungen als Lautstärkeregler oder für die Videosteuerung geeignet. (CSliderCtrl)
Drehregler	Erhöhen oder verringern den Wert eines zugeordneten Steuerelements – meist ein Eingabefeld. (CSpinButtonCtrl)
Kombinationsfeld	Erweitertes Kombinationsfeld, das Symbole unterstützt. (CComboBoxEx)
Spaltenüberschrift	Stellt Überschriften zur Verfügung, die beispielsweise in der Listenansicht verwendet werden. (CHeaderCtrl)
IP-Adresse	Eingabefeld für IP-Adressen. (CIPAddressCtrl)
Listenelement	Erweitert die Funktionalität eines Listenfeldes, indem die Einträge in verschiedenen Ansichten dargestellt werden können. Ein gewöhnliches Listen-Steuerelement führt die Einträge mit deren Symbolen und Beschriftungen auf. Das Steuerelement kann diese Einträge als Symbole oder als Listeneinträge anzeigen. (CListCtrl)

RTF-Eingabefeld	Erweitert die Funktionalität des Windows-3.1-Eingabefelds, indem Microsoft-RTF-Dateien (Rich-Text-Format) darin bearbeitet werden können. Rich-Text-Steuerelemente verfügen über die Funktionalität einer einfachen Textverarbeitung. (CRichEditCtrl)
Register	Dient der Implementierung mehrseitiger Dialoge, die auch als Registerdialoge oder Eigenschaftenseiten bekannt sind. Ein Registerkarten-Steuerelement stellt eine Anwenderschnittstelle zur Verfügung, in der ein Anwender die gewünschte Dialogseite (Eigenschaftenseite) mit einem Klick auf das entsprechende Register öffnet. Das Register ist wie mehrere übereinander angeordnete Seiten aufgebaut. Ein Klick auf den Registerreiter einer Seite ordnet diese über allen anderen Seiten an. (CTabCtrl)
Strukturelement	Führt eine Liste mit Einträgen in einer Baumhierarchie auf. Struktur-Steuerelemente sind für die Anzeige untergliederter Listen geeignet, wie z.B. Verzeichnislisten. Diese Steuerelemente stellen einen effizienten Mechanismus für die Darstellung einer großen Anzahl verschiedener Einträge zur Verfügung. Der Mechanismus verfügt über die Möglichkeit, die einem Eintrag untergeordneten Einträge ein- und auszublenden. (CTreeCtrl)

Alle Windows-95/98-Standardsteuerelemente werden ebenfalls unter Windows NT ab Version 3.51 unterstützt. Abbildung 9.3 stellt einige Windows-95/98-Standardsteuerelemente in einem Dialog dar.

Abbildung 9.3:
Einige Win-
dows-95/98-
Standardsteuer-
elemente



9.9 Zusammenfassung

Ein Fenster ist ein rechteckiger Bereich auf dem Bildschirm, über den Anwendungen und Anwender miteinander kommunizieren. Anwendungen zeichnen in das Fenster, um Informationen für den Anwender darzustellen. Anwendungen empfangen über einen Handle Nachrichten über Anwenderschnittstellenereignisse.

Fenster werden hierarchisch angeordnet. Zuoberst befindet sich das Desktop-Fenster. Top-Level-Fenster sind Fenster, deren übergeordnetes Fenster das Desktop-Fenster ist oder denen kein übergeordnetes Fenster zugewiesen ist. Das Parent-Fenster eines Child-Fensters ist ein Top-Level-Fenster oder ein anderes Child-Fenster. Fenster, die demselben Fenster untergeordnet sind, befinden sich auf der gleichen Gliederungsstufe. Die Reihenfolge, in der gleichgestellte Fenster angezeigt werden, wird als Z-Reihenfolge bezeichnet. Eine besondere Fenstergruppe enthält Top-Level-Fenster, die mit dem `Topmost`-Attribut ausgezeichnet sind. Diese Fenster werden immer über allen anderen Fenstern derselben Z-Reihenfolge angezeigt.

Ein Top-Level-Fenster kann im Besitz eines Fensters sein, daß nicht das übergeordnete Fenster ist.

Fenster, die gewöhnlich mit dem Anwender interagieren, sind überlappte Fenster (normale Anwendungsfenster), Pop-up-Fenster (Dialoge) und Steuerelemente.

Fenster Nachrichten werden in einer Fensterfunktion bearbeitet. Eine Fensterfunktion und andere Fensterattribute beziehen sich auf die Fensterklasse, von der sich das Fenster ableitet. Anwendungen können eigene Fensterklassen, Subklassen und Superklassen aus bestehenden

Fensterklassen erzeugen. Subklassen modifizieren das Verhalten einer bestehenden Fensterklasse, während Superklassen neue Fensterklassen sind, die auf einer bestehenden Klasse basieren.

Die Win32-API stellt einige Funktionen zur Verfügung, die der Erstellung, Anzeige und Verwaltung von Dialogen dienen. Windows unterscheidet zwischen modalen und nicht-modalen Dialogen. Während ein modaler Dialog angezeigt wird, ist der Zugriff auf das besitzende Fenster nicht möglich. Das Fenster wird erst dann wieder freigegeben, wenn der Anwender den Dialog schließt. Nicht-modale Dialoge werden angezeigt, ohne den Zugriff auf das besitzende Fenster zu sperren. Anwendungen müssen für nicht-modale Dialoge Nachrichtenschleifen zur Verfügung stellen und Dialognachrichten über die Funktion `IsDialogMessage` weiterleiten.

Unter Windows stehen Ihnen einige Standarddialoge für allgemeine Aufgaben zur Verfügung. Dazu zählen Dialoge zum Öffnen und Speichern einer Datei, zum Drucken und Einrichten der Seite, zur Auswahl von Farben und Schriften und zur Suche nach sowie zum Ersetzen von Text. Einige Standarddialoge dienen der Implementierung von OLE-Funktionen.

Steuerelemente sind Schaltflächen, statischer Text, Textfelder, Listfelder, Kombinationsfelder und Bildlaufleiste. Anwendungen können neue Steuerelementtypen implementieren. Windows 95/98 definiert einige neue Standardsteuerelemente: Listenansichten, Strukturansichten, Registerkarten-Steuerelemente, Zugriffstasten-Steuerelemente, Schieberegler, Fortschrittsleisten, Auf-Ab-Schaltflächen und Rich-Text-Steuerelemente.

Steuerelemente werden gewöhnlich über Dialogfeldvorlagen in der Ressourcdatei der Anwendung definiert. Steuerelemente kommunizieren mit der Anwendung, indem sie Nachrichten (`WM_COMMAND`) an das besitzende Fenster senden (das Dialogfeld).

Ressourcen

Kapitel 10

Ressourcendateien sind Windows-Programmierern wohl bekannt. Anwendungen definieren mit Hilfe dieser Ressourcendateien die sichtbaren Elemente ihrer Anwenderschnittstelle: Menüs, Dialoge, Zeichenfolgen, Bitmaps und weitere Ressourcetypen.

Ressourcendateien werden in einer für den Anwender lesbaren Form erstellt und mit dem Resource-Compiler kompiliert. Das kompilierte Ergebnis wird gewöhnlich an die Anwendung gebunden, um eine binäre Datei zu erzeugen, die den ausführbaren Programmcode und die Resource-Informationen enthält.

Das Verwenden von Ressourcendateien ist keine Pflicht. Es wäre jedoch unbedacht, komplexen Programmcode zu schreiben, um die Anwenderschnittstellenelemente einer Anwendung zu implementieren, da Ressourcendateien genau diesem Zweck dienen. Ressourcendateien erleichtern außerdem die Lokalisierung der Anwenderschnittstellenelemente einer Anwendung, die in mehrsprachigen Umgebungen eingesetzt wird.

Früher verwendeten Programmierer einen Text-Editor, um eine Ressourcendatei zu bearbeiten. In der heutigen Zeit werden statt dessen grafische Resource-Editoren verwendet, wie z.B. der integrierte Resource-Editor des Visual Studio. Natürlich kann eine Ressourcendatei weiterhin mit einem Text-Editor bearbeitet werden. Die Syntax und Struktur einer Ressourcendatei sind einfach aufgebaut. Eine Erläuterung der Ressourcendateien wird Ihnen helfen, die Features aktueller Editoren zu verstehen.



Gewöhnlich wird der Begriff Ressourcendatei für Dateien verwendet, die von dem Ressource-Compiler generiert wurden. Damit sind Dateien mit der Endung `.res` gemeint. In den folgenden Abschnitten wird der Ausdruck ausschließlich für Dateien gebraucht, die Ressourcenskripte enthalten. Die Namen dieser Dateien enden mit `.rc`.

10.1 Elemente einer Ressourcendatei

Eine Ressourcendatei oder ein Ressourcenskript kann sehr viele Ressourcenskriptanweisungen und Präprozessordirektiven enthalten. Betrachten Sie dazu auch die Beispielressourcendatei in Listing 10.1.

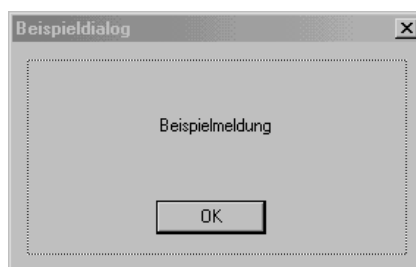
Listing 10.1: *Ressourcenskript-Beispiel*

```
#include <windows.h>

DlgBox DIALOG 0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Beispieldialog"
BEGIN
    DEFPUSHBUTTON "OK",IDOK,65,66,50,14
    CTEXT "Beispielmeldung",IDC_STATIC,66,30,52,8,
    SS_CENTERIMAGE
END
```

Dieses einfache Skript definiert einen Dialog mit einer Schaltfläche und einem statischen Textfeld (Abbildung 10.1). Die Anwendung, die diese Ressource verwendet, greift auf den Dialog über dessen Namen (`DlgBox`) zu. Einige in der Datei `windows.h` definierte Konstanten (z.B. `IDOK`, `DS_MODALFRAME` oder `WS_GROUP`) sind nicht Bestandteil der Ressourcenskriptsprache. Die C/C++-Anweisung `#include` wird verwendet, um Dateien anzugeben, die Makro-Definitionen enthalten.

Abbildung 10.1:
Beispieldialog



Eine Ressourcendatei-Anweisung besteht gewöhnlich aus einem Bezeichner, der eine Zeichenfolge (DlgBox in dem vorherigen Beispiel) oder ein numerischer Wert sein kann, dem die Anweisung selbst folgt. Die Anweisung kann aus einer (der Anweisung folgen ein oder mehrere Parameter) oder mehreren Zeilen (der Anweisung folgt ein Block aus Skriptanweisungen) bestehen.

10.1.1 Bearbeitung von Ressourcendateien

Die Syntax und Semantik des Präprozessors ist ähnlich der Syntax und Semantik des C/C++-Präprozessors. Der Präprozessor des Resource-Compilers, kurz RC-Präprozessor genannt, kann die folgenden Anweisungen ausführen:

- ➡ Makro-Definitionen: `#define` und `#undef`
- ➡ Bedingte Kompilierung: `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`
- ➡ Header-Dateien: `#include`
- ➡ Compilerzeitfehler: `#error`

Die Bedeutung dieser Anweisungen sollte auch dem unerfahrenen C-Programmierer bekannt sein.

Der Präprozessor versteht (und entfernt) außerdem C- und C++-Kommentare, also Kommentare, die zwischen `/*` und `*/` eingeschlossen sind, sowie Kommentarzeilen, die mit `//` beginnen.

Der Präprozessor kennt ebenfalls den Operator `#` sowie den Operator `##`.

Eine Header-Datei kann sowohl von C-Source- als auch von Ressourcenskriptdateien verwendet werden. Während des Kompilierens einer Ressource definiert der Resource-Compiler das Symbol `RC_INVOKED`. Dieses Symbol wird in der Header-Datei zum Schutz gegen Compiler-Fehler verwendet. Enthält die Header-Datei beispielsweise eine Klassendeklaration, können Sie diese möglicherweise wie folgt schützen:

```
#ifndef RC_INVOKED
class myclass
{
    ...
};
#endif // RC_INVOKED
```

Ein weiteres nützliches Symbol ist `APSTUDIO_INVOKED`. Dieses Symbol wird definiert, wenn eine Ressourcendatei in den integrierten Resource-Editor des Developer Studio geladen wird. (Das Bearbeiten von Ressourcen unter Visual C++ 1.0 geschah mit Hilfe des separaten Programms Application Studio. Daher der Name dieser Konstante.)

Ressourcenskripte können ebenfalls konstante Ausdrücke enthalten. Nachfolgend die entsprechenden Operatoren: Addition (+), Subtraktion (-), Multiplikation (*), Division (/), unäres NOT (~), binäres AND (&) sowie binäres OR (|).

10.1.2 Einzeilige Anweisungen

Einzeilige Anweisungen definieren Bitmaps, Mauszeiger, Schriftarten, Symbole, Zeichenfolgentabellen und die Sprache der Ressource.

Die Anweisung `BITMAP` spezifiziert eine Bitmap-Datei (die mit einem Bitmap-Editor erstellt wurde), die zur Definition einer Bitmap-Ressource verwendet werden soll. Hier ein Beispiel:

```
MyBitmap BITMAP mybitmap.bmp
```

Die Anweisung `CURSOR` gibt eine Datei an, die die Figur eines Mauszeigers definiert. Die Cursor-Datei ist eine binäre Datei, die mit Hilfe eines separaten Editors erzeugt wird. Nachfolgend finden Sie ein Beispiel für diese Anweisung aufgeführt:

```
MyCursor CURSOR mycursor.cur
```

Die Anweisung `FONT` spezifiziert eine Schriftart-Ressource:

```
MyFont FONT myfont.fnt
```

Die Anweisung `ICON` bestimmt eine binäre Symboldatei (die mit einem Symbol-Editor erstellt wurde), die eine Symbol-Ressource definiert:

```
MyIcon ICON myicon.ico
```

Die Anweisung `LANGUAGE` spezifiziert die Sprache für alle folgenden Ressourcen bis zum Ende des Ressourcenskripts oder bis zur nächsten `LANGUAGE`-Anweisung. Diese Anweisung wird außerdem zur Bestimmung der Sprache einer einzelnen Ressource in einer mehrzeiligen Ressource-Anweisung verwendet, sofern `LANGUAGE` vor dem Schlüsselwort `BEGIN` angeordnet ist. Der Anweisung `LANGUAGE` folgen Bezeichner für die Sprache sowie für die untergeordnete Sprache. Verwenden Sie die in der Datei `winnls.h` definierten Konstanten für diese Bezeichner.

Die `MESSAGETABLE`-Anweisung bezeichnet eine Nachrichtentabelle, die vorwiegend unter Windows NT verwendet wird. Eine Nachrichtentabelle wird mit dem Nachrichten-Compiler `mc.exe` erstellt.

Die Anweisungen `BITMAP`, `CURSOR`, `FONT` und `SYMBOL` akzeptieren einen Attributparameter. Der Attributparameter bestimmt die Lade- und Speichereigenschaften einer Ressource. Die 32-Bit-Version von Windows verwendet lediglich einen Parameter: Das Attribut `DISCARDABLE` gibt an, daß eine Ressource aus dem Speicher entfernt werden kann, wenn diese nicht mehr benötigt wird:

```
TempBmp BITMAP DISCARDABLE "c:\\bitmaps\\tempbmp.bmp"
```

10.1.3 Mehrzeilige Ressource-Anweisungen

Mehrzeilige Ressource-Anweisungen definieren Dialoge, Zeichenfolgentabellen, Tastaturkürzeltabellen, Menüs und Versionsinformationen. Die Anweisungen beginnen mit einem Bezeichner, der Anweisung und optionalen Parametern, denen ein zwischen den Schlüsselworten BEGIN und END aufgeführter Befehlsblock folgt:

```
identifizier STATEMENT [optional-parameters]
[optional instructions]
BEGIN
    [instructions]
END
```

Optionale Anweisungen können die Befehle CHARACTERISTICS (bestimmen einen einzelnen 32-Bit-Wert, der von den Ressource-Dateiverwaltungshilfsmitteln verwendet wird), LANGUAGE und VERSION (bestimmt eine 32-Bit-Versionsnummer, die von den Ressource-Verwaltungshilfsmitteln verwendet wird) enthalten. Andere optionale Anweisungen sind spezifische Mehrzeilenbefehle, wie z.B. CAPTION. Diese Anweisung definiert den Titel eines Dialogfelds.

Aufgrund der relativen Komplexität, beschreiben die folgenden Abschnitte mehrzeilige Anweisungen detailliert.

Tastaturkürzel

Tastaturkürzel sind Tasten oder Tastenkombinationen, deren Betätigung zur Ausführung einer bestimmten Aufgabe führt. Wenn Sie beispielsweise die Tastenkombination Strg + C betätigen, um ein Objekt in die Zwischenablage zu kopieren, verwenden Sie ein Tastaturkürzel.

Eingeschlossen zwischen den Schlüsselworten BEGIN und END, enthält eine Tastaturkürzel-Anweisung beliebig viele Tastaturereignisse, gefolgt von dem Bezeichner des entsprechenden Tastaturkürzels, wie in dem folgenden Beispiel aufgeführt:

```
MyAcc ACCELERATOR
BEGIN
    "C", ID_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
    "V", ID_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT
    "X", ID_EDIT_CUT, VIRTKEY, CONTROL, NOINVERT
END
```

Dieses Beispiel bedingt eine Definition der symbolischen Konstanten ID_EDIT_COPY, ID_EDIT_PASTE und ID_EDIT_CUT in einer Header-Datei. Diese Konstanten verweisen auf numerische Bezeichner.

Tastaturkürzel werden interpretiert, wenn eine Anwendung die Funktion TranslateAccelerator aufruft, nachdem eine Nachricht über GetMessage (oder PeekMessage) ausgelesen wurde. TranslateAccelerator über-

setzt die Nachricht `WM_KEYDOWN` (oder `WM_SYSKEYDOWN`) in die Nachricht `WM_COMMAND` (oder `WM_SYSCOMMAND`). Die Bezeichner, die den Tastaturkürzel-Schlüsseln in einer Tastaturkürzel-Anweisung folgen, bilden die Anweisungsbezeichner in den `WM_COMMAND`-Nachrichten.

Dialoge

Zusammen mit Menü-Anweisungen definieren Dialoganweisungen die überwiegend sichtbaren Elemente einer gewöhnlichen Anwendung. Eine Dialoganweisung definiert das Layout eines Dialogfeldes.

Eine einfache Dialoganweisung ist in Listing 10.1 aufgeführt. Die Anweisung besteht aus mehreren Zeilen. Die erste Zeile enthält einen Bezeichner, das Schlüsselwort `DIALOG`, und vier numerische Parameter, die die Position der linken oberen Ecke sowie die Größe des Dialogs bestimmen.

Alle Informationen über die Größe und Position in einer Dialoganweisung werden in Dialog-Einheiten angegeben. Dialogeinheiten leiten sich von der Größe der Schriftart ab, die für den Dialog bestimmt wurde. Die Basiseinheiten eines Dialogs repräsentieren die durchschnittliche Höhe und Breite eines Zeichens der selektierten Schriftart. Vier horizontale Dialogeinheiten sind eine horizontale Basiseinheit. Acht vertikale Dialogeinheiten sind eine vertikale Basiseinheit.

Eine Anwendung kann für Dialoge, die die Systemschriftart verwenden, die Größe der Dialog-Basiseinheiten in Pixeln ermitteln. Dazu muß die Funktion `GetDialogBaseUnits` aufgerufen werden. Für Dialoge, die andere Schriftarten verwenden, muß möglicherweise eine explizite Berechnung der durchschnittlichen Größe eines Zeichens vorgenommen werden, um die Basiseinheit zu ermitteln.

Nachdem die Basiseinheiten des Dialogs bekannt sind, können diese mit den folgenden Formeln in Dialogeinheiten und Pixel umgerechnet werden:

```
pixelX = (dialogunitX * baseunitX) / 4  
pixelY = (dialogunitY * baseunitY) / 8  
dialogunitX = (pixelX * 4) / baseunitX  
dialogunitY = (pixelY * 8) / baseunitY
```

Der Zeile, die das Schlüsselwort `DIALOG` enthält, folgt eine Dialoganweisung, die aus mehreren optionalen Befehlen bestehen kann. Dazu zählen allgemein verwendete oder spezifische Dialog-Anweisungen.

Der optionalen Anweisung `CAPTION` folgt eine Zeichenfolge, die den Titel des Dialogs angibt. Die Voreinstellung ist ein Dialog ohne Titel.

Die `STYLE`-Anweisung bestimmt den Stil des Dialogs. Stilwerte sind gewöhnlich in der Header-Datei `windows.h` vordefiniert. Mehrere Werte

können mit Hilfe des logischen ODER-Operators (|) miteinander kombiniert werden. Der Standardstil eines Dialogs, der keine `STYLE`-Anweisung voraussetzt, lautet `WS_POPUP | WS_BORDER | WS_SYSMENU`.

`EXSTYLE` ist kongruent mit `STYLE`. Diese Anweisung spezifiziert erweiterte Stile.

Die Anweisung `CLASS` kann zur Bestimmung einer speziellen Fensterklasse für einen Dialog verwendet werden. Diese Anweisung sollte bedacht eingesetzt werden, da sie das Verhalten des Dialogs neu definiert.

Mit `FONT` bestimmen Sie die Schriftart, die in dem Dialog verwendet werden soll. Die Voreinstellung ist die Systemschriftart.

Die Anweisung `MENU` bezeichnet die Menü-Ressource, die das Menü des Dialogs bildet. Wird diese Anweisung nicht verwendet, erhält der Dialog keine Menüleiste.

Zwischen den Schlüsselworten `BEGIN` und `END` befinden sich einige Steuerelementanweisungen, die die Steuerelemente des Dialogs spezifizieren. Es gibt verschiedene Typen von Steuerelementanweisungen. Jede Steuerelementanweisung führt den Steuerelementtyp, den Steuerelementtext, einen Steuerelementbezeichner (Text oder Integer), die Position des Steuerelements, den Steuerelementstil und erweiterte Stilparameter auf:

```
CONTROL-STATEMENT control-text, identifier, x, y, width, height [, style [,
extended-style]]
```

- ➡ Ein Textfeld-Steuerelement wird mit der Anweisung `EDITTEXT` definiert.
- ➡ `LTEXT`, `CTEXT`, `RTEXT` oder `ICON` definieren ein statisches Steuerelement. Die ersten drei der genannten Steuerelementanweisungen definieren ein links ausgerichtetes, zentriertes oder rechts ausgerichtetes statisches Steuerelement. Die letzte Anweisung spezifiziert ein statisches Steuerelement mit dem Stil `SS_ICON`.
- ➡ Ein Schaltflächen-Steuerelement wird mit einem der folgenden Schlüsselworte definiert: `AUTO3STATE`, `AUTOCHECKBOX`, `AUTORADIOBUTTON`, `CHECKBOX`, `DEFPUSHBUTTON`, `GROUPBOX`, `PUSHBOX`, `PUSHBUTTON`, `RADIOBUTTON`, `STATE3`, `USERBUTTON`.
- ➡ `COMBOBOX` definiert ein Kombinationsfeld.
- ➡ Die Anweisung `LISTBOX` spezifiziert ein Listenfeld.
- ➡ `SCROLLBAR` definiert eine Bildlaufleiste.

Die Anweisung `CONTROL` kann dazu verwendet werden, ein allgemeines Steuerelement zu erstellen. Die Syntax dieser Anweisung unterscheidet sich ein wenig von der Syntax anderer Steuerelementanweisungen:

```
CONTROL control-text, identifier, class-name, x, y, width, height [, extended-style]
```

Der Parameter `class-name` spezifiziert die Fensterklasse des Steuerelements, die eine der Windows-Steuerelementklassen sein kann. Die `CONTROL`-Anweisung kann somit als alternative Syntax für alle anderen Steuerelementanweisungen verwendet werden.

Eine Variante der `DIALOG`-Anweisung ist `DIALOGEX`. Sie erweitert die Syntax der `DIALOG`-Anweisung wie folgt:

- Die Anweisung ermöglicht Ihnen, Hilfe-Bezeichner für den Dialog und die darin enthaltenen Steuerelemente zu definieren.
- Sie können Schriftart- und Kursiv-Einstellungen in dem Abschnitt `FONT` vornehmen.
- Spezifische Steuerelementdaten können den Steuerelementanweisungen hinzugefügt werden (zwischen `BEGIN` und `END`).
- Die Anweisung erlaubt die Schlüsselworte `BEDIT`, `HEDIT` und `IEDIT` für Stift-Steuerelemente.

Menüs

Menü-Anweisungen in dem Ressourcenskript dienen der Definition von Menüleisten oder Pop-up-Menüs. Diese Anweisungen enthalten eine oder mehrere Menü-Definitionsanweisung(en) innerhalb der Schlüsselworte `BEGIN` und `END`. Dazu ein Beispiel:

```
MyMenu MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N", ID_FILE_NEW
        MENUITEM SEPARATOR
        MENUITEM "E&xit", ID_FILE_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "Cu&t\tCtrl+X", ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C", ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V", ID_EDIT_PASTE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About", ID_HELP_ABOUT
    END
END
```

Die Bezeichner (beispielsweise `ID_FILE_NEW`) dieses Beispiels sind in einer der Header-Dateien definiert und verweisen auf numerische Werte.

Eine Menü-Definitionsanweisung kann einen Menüeintrag oder ein Untermenü spezifizieren. Verwenden Sie `MENUITEM`, um einen Menüeintrag zu definieren. Der Anweisung folgt entweder der Text des Eintrags und der Menübezeichner oder das Schlüsselwort `SEPARATOR`. Dieses Schlüsselwort bestimmt einen Separator, also eine vertikale Linie für Menüleisten oder eine horizontale Linie für Pop-up-Menüs (Untermenüs).

Dem Bezeichner eines Menüeintrags kann eine Optionsliste folgen. Optionen werden durch Kommata oder Leerzeilen getrennt angegeben und umfassen die folgenden Schlüsselworte: `CHECKED`, `GRAYED`, `HELP`, `INACTIVE`, `MENUBARBREAK`, `MENUBREAK`.

Möchten Sie ein Untermenü definieren, verwenden Sie bitte die Anweisung `POPUP`. Dieser Anweisung folgt zwischen den Schlüsselworten `BEGIN` und `END` der Titel des Untermenüs und einige Menüeinträge. Ein Untermenü kann weitere Untermenüs enthalten.

Zeichenfolgentabellen

Eine Zeichenfolgentabellen-Ressource definiert eine beliebige Anzahl verschiedener Zeichenfolgen. Die Anwendung kann mit symbolischen Bezeichnern auf diese Zeichenfolgen verweisen. Der Vorteil einer Zeichenfolgentabelle besteht darin, daß alle Textkomponenten einer Anwendung innerhalb einer Ressourcendatei einfach in eine andere Sprache übersetzt werden können.

Eine Zeichenfolgentabelle wird mit dem Schlüsselwort `STRINGTABLE` definiert, dem optionale Anweisungen sowie eine oder mehrere Zeichenfolgendefinitionen folgen können. Diese werden zwischen `BEGIN` und `END` aufgeführt:

```
STRINGTABLE
BEGIN
    IDS_HELLO "Hallo"
    IDS_GOODBYE "Auf Wiedersehen"
END
```

`IDS_HELLO` und `IDS_GOODBYE` sind symbolische Bezeichner, die in einer Header-Datei definiert sein müssen.

Das Verwenden einer Zeichenfolge aus einer Zeichenfolgentabelle ist sehr unkompliziert. Eine Anwendung kann eine Zeichenfolgenressource mit Hilfe der Funktion `LoadString` einlesen.

Für MFC-Anwendungen gestaltet sich der Einsatz von Zeichenfolgentabellen wesentlich einfacher. Viele MFC-Funktionen, die einen Zeichenfolgenparameter akzeptieren, können ebenfalls einen numerischen Parameter entgegennehmen, der eine Zeichenfolgenressource

in der Ressourcendatei der Anwendung repräsentiert. Eine MFC-Anwendung kann beispielsweise ein Nachrichtenfeld mit `AfxMessageBox` wie folgt darstellen:

```
AfxMessageBox(IDS_ERROR);
```

`IDS_ERROR` ist ein symbolischer Verweis auf einen numerischen Bezeichner.

Die Klasse `CString` bietet eine besondere Unterstützung für Zeichenfolgenressourcen. Die Member-Funktion `CString::LoadString` initialisiert ein `CString`-Objekt mit einem Wert, der aus einer Zeichenfolgentabelle der Ressourcendatei ermittelt wird.

Symbolleisten

Symbolleisten-Ressourcen werden in MFC-Anwendungen verwendet. Eine Symbolleiste wird in einer Ressourcendatei mit der Anweisung `TOOLBAR` definiert. Diese Anweisung führt die Bezeichner der Symbolleistenschaltflächen auf. Die Größe der Schaltflächen wird ebenfalls bestimmt, wie in dem folgenden Beispiel dargestellt:

```
IDR_MAINFRAME TOOLBAR DISCARDABLE 16, 15
BEGIN
    BUTTON        ID_FILE_NEW
    BUTTON        ID_FILE_OPEN
    BUTTON        ID_FILE_SAVE
    SEPARATOR
    BUTTON        ID_EDIT_CUT
    BUTTON        ID_EDIT_COPY
    BUTTON        ID_EDIT_PASTE
    SEPARATOR
    BUTTON        ID_FILE_PRINT
    BUTTON        ID_APP_ABOUT
END
```

Für jede Symbolleisten-Ressource sollte eine entsprechende Bitmap-Ressource bestehen, die die Bitmaps der Schaltflächen enthält. Die Schaltflächen-Bitmaps sollten horizontal in der Reihenfolge angeordnet werden, in der ihre Bezeichner in der Symbolleisten-Ressource aufgeführt sind (ausgenommen davon sind Separatoren). Die Bitmap-Ressource sollte denselben Bezeichner wie die Symbolleisten-Ressource verwenden:

```
IDR_MAINFRAME BITMAP MOVEABLE PURE "res\\Toolbar.bmp"
```

Sie greifen auf Symbolleisten-Ressourcen über die MFC-Funktion `CToolBar::LoadToolBar` zu.

Versionsinformationen

Die Versionsinformations-Ressource bezeichnet die Version einer binären Datei (gewöhnlich eine ausführbare Datei oder eine Bibliothek). Versionsinformationen werden von Bibliotheksfunktionen zur Dateiinstallation verwendet.

Die Versionsinformations-Ressource-Anweisung enthält mehrere Versionsanweisungen, die die Versionsnummer der Datei und des Produkts bestimmen und zusätzliche Versionsinformationen definieren, wie z.B. die Sprache und das Ziel-Betriebssystem. Versionsinformationen können mit den Funktionen `GetFileVersionInfo`, `GetFileVersionInfoSize` und `VerQueryValue` ermittelt werden.

10.1.4 Benutzerdefinierte Ressourcen und Datenressourcen

Die Syntax einer Ressourcendatei erlaubt die Erstellung benutzerdefinierter Ressourcetypen. Eine anwenderdefinierte Ressource-Anweisung kann ein- oder mehrzeilig sein. Einzeilige Anweisungen bezeichnen benutzerdefinierte Ressourcen, die in separaten Dateien gespeichert sind. Die Syntax einer einzeiligen Anweisung lautet wie folgt:

```
name type [load-memory-options] filename
```

Mehrzeilige benutzerdefinierte Ressourcenanweisungen dienen der Definition einer benutzerdefinierten Ressource in eine Ressourcendatei. Nachfolgend finden Sie die Syntax einer mehrzeiligen benutzerdefinierten Ressourcenanweisung aufgeführt:

```
name type [load-memory-options]
BEGIN
    raw-data
END
```

Der Datenblock kann dezimale, hexadezimale oder oktale Integer-Werte respektive Zeichenfolgen in Anführungszeichen enthalten. Zeichenfolgen mit Rohdaten müssen explizit mit einem Nullwert terminiert werden. Einzelne Dateneinträge werden durch Kommata getrennt angegeben.

Rohdaten können ebenfalls in einer `RCDATA`-Anweisung spezifiziert werden. Die Syntax dieser Anweisung gleicht der einer mehrzeiligen benutzerdefinierten Ressource-Anweisung. Für eine Rohdaten-Anweisung wird jedoch das Schlüsselwort `RCDATA` anstelle von `TYPE` verwendet. Die Anweisung kann außerdem die Option `CHARACTERISTICS`, `LANGUAGE` und `VERSION` enthalten.

10.2 Kompilieren und Verwenden von Ressourcenskripten

Bevor ein Ressourcenskript von einer Anwendung verwendet werden kann, muß es kompiliert werden. Obwohl nicht unbedingt erforderlich, wird die kompilierte Ressourcendatei häufig an andere Komponenten der Anwendung gebunden. Die Ressource wird somit ein Teil der ausführbaren Anwendungsdatei (.EXE).

10.2.1 Ausführen des Resource-Compilers

Eine Ressourcendatei kann mit Hilfe des Resource-Compilers *rc.exe* über die Kommandozeile kompiliert werden. Für die meisten Dateien kann diese Anweisung ohne Parameter verwendet werden. Um beispielsweise die in Listing 10.1 aufgeführte Ressourcendatei zu kompilieren, geben Sie

```
rc dlg.rc  
ein.
```

In der 16-Bit-Version von Windows, wurde der Compiler ebenfalls dazu verwendet, der ausführbaren Datei Ressourcen hinzuzufügen. Unter Win32 wird dazu der 32-Bit-Linker verwendet.

10.2.2 Ausführen des Linkers

Der Visual-C++-Linker *LINK.EXE* bindet eine Ressourcendatei und andere Komponenten an die ausführbare Datei. Gewöhnlich geben Sie den Namen der kompilierten Ressourcendatei in der Kommandozeile des C/C++-Compilers wie jede andere Objektdatei oder Bibliothek an:

```
CL DLG.CPP DLG.RES USER32.LIB
```

Das Programm *cvtres.exe* muß in dem Verzeichnis des Linkers *link.exe* enthalten sein, damit dieser korrekt ausgeführt wird. *cvtres.exe* konvertiert kompilierte Ressourcendateien in das COF-Format (Common Object File Format), das von dem Linker bearbeitet werden kann.

10.2.3 Ressourcen-DLLs

Ressourcen müssen nicht an die ausführbare Datei Ihrer Anwendung gebunden werden. Sie können ebenfalls an eine separate DLL gebunden werden. Diese Vorgehensweise hat den Vorteil, daß eine Modifizierung der Ressourcendatei nicht ein erneutes Kompilieren der ge-

samten Anwendung erfordert. Lediglich die DLL muß ersetzt werden. Sie können Ihre Anwendung auch mit verschiedenen DLLs ausliefern, um unterschiedliche Sprachen zu unterstützen. MFC-Anwendungen erleichtern diese Absicht mit einem Aufruf von `AfxSetResourceHandle`. Rufen Sie diese Funktion mit der Zugriffsnummer der Ressource-DLL auf, lädt MFC alle Ressourcen aus dieser DLL und nicht aus der ausführbaren Datei Ihrer Anwendung.

10.3 Lokalisation von Ressourcen

Visual Studio unterstützt mehrere Sprachen in einer Ressourcendatei. Möchten Sie die Sprache einer Ressource bestimmen, so markieren Sie den gewünschten Ressource-Bezeichner in der Ressourcen-Ansicht und wählen aus dem Menü ANSICHT den Befehl EIGENSCHAFTEN aus.



Abbildung 10.2:
Lokalisierung
von Ressourcen

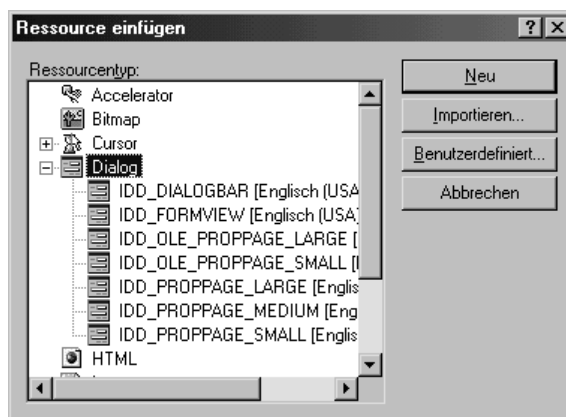
Beachten Sie bitte, daß die Ressourcendatei verschiedene lokalisierte Versionen einer Ressource enthalten kann, die sich den selben Bezeichner teilen. Um eine Kopie der Ressource in einer anderen Sprache einzufügen, klicken Sie bitte in der Ressourcen-Ansicht mit der rechten Maustaste auf den Ressource-Bezeichner, und rufen Sie in dem anschließend erscheinenden Kontextmenü den Befehl KOPIE EINFÜGEN auf.

Das Kompilieren lokalisierter Ressourcen wird über Präprozessordefinitionen in der Ressourcendatei gesteuert. Sie können lokalisierte und sprachspezifisch eingerichtete Ressourcen nutzen, indem Sie den Konfigurationseinstellungen der Ressource im Dialog PROJEKT-EINSTELLUNGEN die entsprechenden Präprozessordefinitionen hinzufügen.

10.4 Ressourcenvorlagen

Visual Studio unterstützt Ressourcenvorlagen. Ressourcenvorlagen sind vordefinierte Ressourcen, die als Vorlage für neue Ressourcen verwendet werden können. Wählen Sie beispielsweise aus dem Menü EINFÜGEN den Befehl RESSOURCE aus, und klicken Sie in dem anschließend dargestellten Dialog auf das Pluszeichen vor DIALOG, so können Sie eine der unter dem Eintrag dargestellten Dialogvorlagen auswählen.

Abbildung 10.3:
Ressourcen-
vorlagen



Sie müssen zunächst eine Ressourcenvorlagendatei erstellen, bevor Sie Ihre eigenen Ressourcenvorlagen generieren. Wählen Sie dazu aus dem Menü DATEI den Eintrag NEU. Achten Sie darauf, daß das Kontrollkästchen DEM PROJEKT HINZUFÜGEN nicht aktiviert ist. Nachdem die gewünschten Ressourcen der Vorlage hinzugefügt wurden, speichern Sie die Vorlage in dem Verzeichnis MSDEV98\TEMPLATE, indem Sie aus dem Menü DATEI den Eintrag SPEICHERN UNTER auswählen. Die neue Vorlage wird aufgeführt, wenn Sie das nächste Mal versuchen, eine Ressource einzufügen.

10.5 Zusammenfassung

Ressourcendateien definieren den visuellen Aufbau Ihrer Anwendung. Ressourcen enthalten Dialoge, Menüs, Bitmaps, Symbole, Mauszeiger, Zeichenfolgen und weitere Typen.

Obwohl Ressourcen gewöhnlich mit grafischen Editoren erzeugt werden, ist auch eine direkte Bearbeitung möglich (und bisweilen erforderlich). Ressourcendateien (Dateien mit der Endung .rc) sind für den Anwender lesbare ASCII-Textdateien.

Ressourcendateien enthalten Präprozessordirektiven sowie einzeilige und mehrzeilige Ressourcenanweisungen. Präprozessordirektiven sind ihren Pendants, den C/C++-Anweisungen, sehr ähnlich. Einzeilige Ressourcendatei-Anweisungen bezeichnen die Ressourcen, die in separaten Binärdateien gespeichert sind (und mit speziellen Editoren bearbeitet werden): Bitmaps, Symbole, Mauszeiger, Schriftarten und allgemeine Ressourcen. Einzeilige Anweisungen werden ebenfalls dazu verwendet, Nachrichtentabellen-Ressourcen (Windows NT) und die Sprache der folgenden Ressourcenanweisungen zu definieren.

Mehrzeilige Anweisungen definieren Menüs, Dialoge, Zeichenfolgen, Tastaturkürzeltabellen, Versionsinformationen und allgemeine Ressourcen.

Ressourcendateien werden mit dem Ressource-Compiler `RC.EXE` kompiliert. Die daraus resultierende Ressourcendatei (gewöhnlich eine Datei mit der Endung `.RES`) kann an andere Komponenten der Anwendung gebunden werden. Eine kompilierte Ressourcendatei kann außerdem in der `cl`-Kommandozeile angegeben werden.

Zeichnen und Gerätekontexte

Kapitel

11

Das Zeichnen auf dem Bildschirm, Drucker oder einem anderen Ausgabegerät ist einer der wichtigsten Aspekte einer Windows-Anwendung. Während Windows-Anwendungen ausgeführt werden, zeichnen diese kontinuierlich den Inhalt ihrer Fenster als Reaktion auf die Aktionen des Anwenders oder andere Ereignisse.

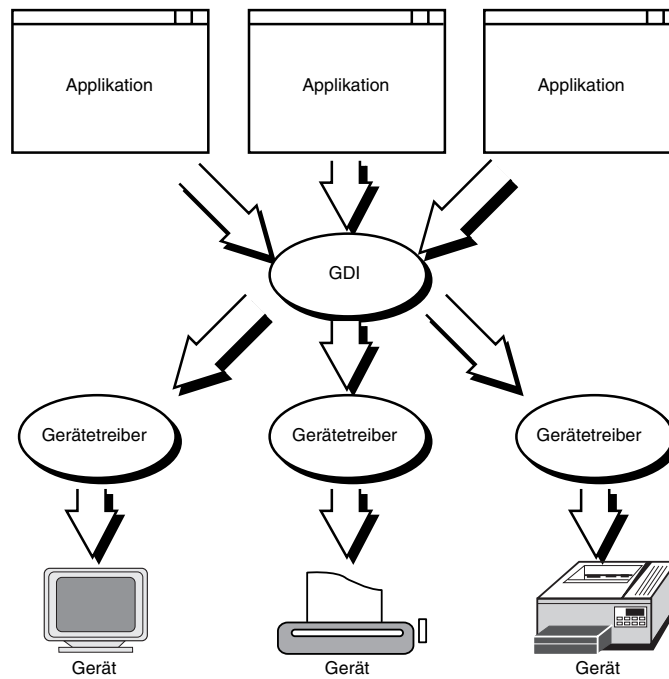
Anwendungen, die auf Hardware-Geräten zeichnen, verwenden dazu verschiedene geräteunabhängige Systemfunktionen. Würden sie das nicht tun, müßten sie wie ihre MS-DOS-Pendants, Geräte-Inkompatibilitäten berücksichtigen und wären auf Gerätetreiber für unterschiedliche Videokarten, Drucker oder andere Grafik-Hardware angewiesen. Geräteunabhängigkeit ist somit einer der großen Vorteile eines grafischen Betriebssystems wie Windows.

11.1 Das GDI, Gerätetreiber und Ausgabegeräte

Anwendungen zeichnen auf ein Ausgabegerät, indem sie Funktionen des GDI (Graphics Device Interface) aufrufen. Die GDI-Bibliothek *GDI.DLL*, die diese Funktionen enthält, ruft wiederum gerätespezifische Funktionen oder Gerätetreiber auf. Die Gerätetreiber führen Operationen auf der physikalischen Hardware aus. Gerätetreiber sind entweder ein Bestandteil von Windows oder, für weniger allgemein verwendete Hardware, Add-Ons von Drittanbietern. Die übergreifende Beziehung zwischen grafischen Anwendungen, dem GDI, Gerätetreiber-Software und Hardware-Geräten ist schematisch in Abbildung 11.1 dargestellt.

*GDI und Geräte-
treiber*

Abbildung 11.1:
Interaktion zwischen
Anwendungen, GDI,
Gerätetreibern und
Ausgabegeräten



Die überwiegende Anzahl der Zeichenfunktionen verwendet den Handle eines Gerätekontextes als Parameter. Zusätzlich zur Bezeichnung des Geräts, auf dem gezeichnet werden soll, spezifiziert der Gerätekontext einige Eigenschaften:

- ➔ Konvertieren logischer Koordinaten in die physikalischen Koordinaten des Geräts
- ➔ Verwenden von Zeichenobjekten, wie z. B. Schriftarten, Stifte oder Pinsel, um die geforderte Funktion auszuführen
- ➔ Anwenden von Zeichenfunktionen auf sichtbare Bereiche

11.2 Gerätekontexte

Ein Gerätekontext bestimmt die Eigenschaften eines Hardware-Geräts. Systemzeichenfunktionen verwenden diese Informationen, um geräteunabhängige Zeichenaufrufe in mehrere gerätespezifische Verrichtungen zu konvertieren, die mit Hilfe eines Low-Level-Treiberprogramms ausgeführt werden.

Bevor ein Gerätekontext benutzt werden kann, muß er erstellt werden. Die vorwiegend verwendete Funktion zur Erzeugung eines Gerätekontextes ist mit `CreateDC` bezeichnet. Anwendungen rufen diese Funktion auf und übergeben ihr das Gerät, für das ein Gerätekontext erzeugt werden soll, die Treiber-Software, die physikalische Schnittstelle, an der das Gerät angeschlossen ist und gerätespezifische Initialisierungsdaten.

Gerätekontexte erzeugen

Möchte eine Anwendung auf den Bildschirm zeichnen, muß sie nicht selbst einen Gerätekontext mit `CreateDC` erstellen. Statt dessen kann die Anwendung den Handle eines Gerätekontextes ermitteln, der den Client-Bereich des Fensters repräsentiert. Dies geschieht mit der Funktion `GetDC` oder mit `GetWindowDC` für das gesamte Fenster (einschließlich des Nicht-Client-Bereichs).

Eine typische GDI-Zeichenfunktion ist `Rectangle`. Eine Anwendung kann diese Funktion aufrufen, um ein Rechteck zu zeichnen:

Ausgabe in Gerätekontexte

```
Rectangle(hDC, 0, 0, 200, 100);
```

Dieser Aufruf zeichnet ein Rechteck auf das Gerät mit dem Handle `hDC`. Die obere linke Ecke befindet sich an den logischen Koordinaten [0,0], während die Koordinaten [200,100] die rechte untere Ecke des Rechtecks bilden. Bevor das Rechteck auf dem Bildschirm angezeigt wird, sind natürlich einige komplexe Berechnungen und Funktionsaufrufe erforderlich, die für Sie unsichtbar bleiben. Wie ermittelt das GDI beispielsweise die physikalischen Koordinaten, die den logischen Koordinaten entsprechen? Woher bezieht es die Farbe des Rechtecks und dessen Inhalts? Woher weiß es, welcher Linien- und Füllstil verwendet werden soll? Diese Informationen sind in dem Gerätekontext gespeichert. Das Konvertieren der Koordinaten wird durch den Umwandlungsmodus und Transformationsfunktionen bestimmt. Der Aufbau und die Farbe von gezeichneten Objekten sind Funktionen der GDI-Objekte, die in dem Gerätekontext selektiert wurden. Sie erfahren später in diesem Kapitel mehr darüber.

11.2.1 Gerätekontextarten

Windows unterscheidet zwischen allgemeinen und privaten Anzeige-Gerätekontexten. Allgemeine Gerätekontexte repräsentieren eine Resource, die sich verschiedene Anwendungen teilen. Private Gerätekontexte werden für Fenster mit einer Fensterklasse erstellt, der der Stil `CS_OWNDC` zugewiesen ist. Private Gerätekontexte werden gelöscht, wenn das entsprechende Fenster zerstört wird.

11.2.2 Speicher-, Metadatei- und Informationsgerätekontexte

Gerätekontexte präsentieren gewöhnlich physikalische Geräte, wie den Bildschirm, Drucker, Plotter oder ein Fax-Modem. Windows verfügt außerdem über einige spezielle Gerätekontexte. Der bereits genannte Speicher-Gerätekontext stellt eine Bitmap dar. Anwendungen verwenden diesen Gerätekontext, um in eine Bitmap zu zeichnen.

Speicher-Gerätekontexte

Zusätzlich zur Erzeugung von Bitmaps (wie z.B. mit dem Bitmap-Editor PAINT) haben Speicher-Gerätekontexte einen weiteren Nutzen, der sich in grafikintensiven Anwendungen offenbart. Das Zeichnen in einen Speicher-Gerätekontext und das Übertragen des Inhalts, nachdem die Zeichnung vollständig ist, vermindert das Bildschirmflackern. Der sachgemäße Einsatz mehrerer Speicher-Gerätekontexte kann zur Erzeugung flüssiger Animationen verwendet werden. Verschiedene Funktionen, die in diesem Kapitel beschrieben werden, kopieren Bitmap-Daten aus einem Gerätekontext in einen anderen.

CreateCompatibleDC Sie erstellen einen Speicher-Gerätekontext mit einem Aufruf der Funktion `CreateCompatibleDC`. Diese Funktion erstellt einen Speicher-Gerätekontext, der kompatibel zu einem bestimmten physikalischen Gerät ist.

Metadatei-Gerätekontexte

Ein weiterer Gerätekontext ist der Metadatei-Gerätekontext. Eine Metadatei ist eine geräteunabhängige Aufzeichnung von GDI-Verrichtungen. Win32 kennt zwei Metadatei-Typen: Standard- und erweiterte Metadateien. Standardmetadateien sind kompatibel zu Windows 3.1, stellen jedoch keine vollständige Geräteunabhängigkeit zur Verfügung. Neue Anwendungen sollten daher erweiterte Metadateien verwenden.

CreateMetaFile Ein Metadatei-Gerätekontext wird mit einem Aufruf der Funktion `CreateMetaFile` oder, sollen erweiterte Metadateien erzeugt werden, mit `CreateEnhMetaFile` erstellt. Hat eine Anwendung das Zeichnen in einen Metadatei-Gerätekontext beendet, schließt sie die Datei mit `CloseMetaFile` (`CloseEnhMetaFile`). Dieser Aufruf gibt eine Metadatei-Zugriffsnummer zurück, die in Aufrufen von `PlayMetaFile` (`PlayEnhMetaFile`) oder verschiedenen Funktionen zur Bearbeitung von Metadateien verwendet werden kann. Die Zugriffsnummer kann ebenfalls für Metadateien ermittelt werden, die bereits gespeichert wurden. Rufen Sie dazu `GetMetaFile` (`GetEnhMetaFile`) auf.

Nur wenige Anwendungen bearbeiten Metadateien direkt. Die meisten Anwendungen verwenden Metadateien implizit über OLE. Das geräteunabhängige Metadateiformat wird von OLE zur grafischen Darstel-

lung von eingebetteten oder gebundenen Objekten benutzt. Anwendungen, die eingebettete Objekte anzeigen, müssen daher nicht die OLE-Server-Anwendung aufrufen (die sogar möglicherweise nicht auf dem System installiert ist), wenn ein OLE-Objekt gezeichnet werden soll. Statt dessen führen Sie die aufgezeichnete Metadatei aus.

Informationsgerätekontexte

Informationskontexte ermitteln Informationen über spezifische Geräte.

Ein Informationskontext wird mit `CreateIC` erstellt. Das Erzeugen eines Informationskontextes ist einfacher als die Erstellung eines Gerätekontextes, weshalb er überwiegend zum Ermitteln der Informationen über ein Gerät verwendet wird. Ein Informationskontext wird mit `DeleteDC` gelöscht, nachdem er verwendet wurde. *CreateIC*

11.3 Koordinaten

Anwendungen bestimmen gewöhnlich mit logischen Koordinaten die Position und Größe auszugebender Objekte. Bevor ein Objekt an einer physikalischen Position auf dem Bildschirm oder Drucker ausgegeben werden kann, müssen einige Berechnungen vorgenommen werden, um diese physikalische Position auf dem Gerät zu ermitteln.

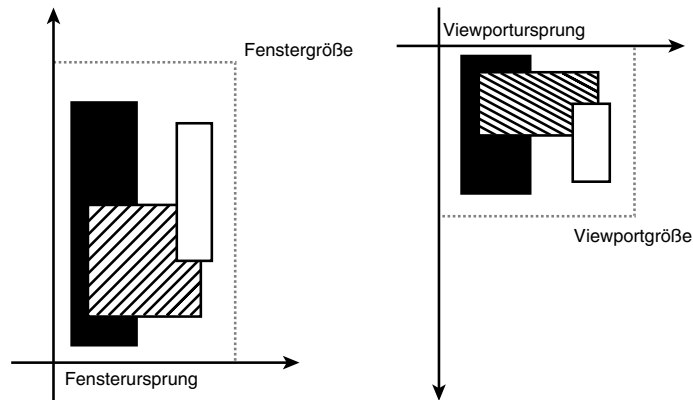
11.3.1 Logische Koordinaten und Gerätekoordinaten

Die Umwandlung von logischen in physikalische Koordinaten kann sich als sehr aufwendig erweisen. Sie geschieht, indem die Eigenschaften des Fensters sowie des Viewports eingerichtet werden. Das Fenster repräsentiert in diesem Zusammenhang den logischen Koordinatenraum. Der Viewport bildet den physikalischen Koordinatenraum des Geräts.

Sowohl dem Fenster als auch dem Viewport müssen zwei Wertepaare zur Verfügung gestellt werden. Ein Paar nimmt die horizontalen und vertikalen Anfangskoordinaten auf, während das andere Paar die horizontalen und vertikalen Koordinaten der Ausdehnung enthält.

Abbildung 11.2 zeigt, wie die logischen Koordinaten eines Rechtecks in gerätespezifische physikalische Koordinaten umgewandelt werden. Die Abbildung demonstriert, daß die absolute Größe der logischen und physikalischen Ausdehnung keine Auswirkungen hat. Wichtig ist die relative Größe, also die Anzahl der logischen Einheiten, die in physikalische Einheiten oder umgekehrt konvertiert werden.

Abbildung 11.2:
Das logische
und physika-
lische Koordi-
natensystem



Die Ausgangskordinaten der meisten Geräte befinden sich in der linken oberen Ecke. Die vertikale Koordinate nimmt von diesen Ausgangskordinaten nach unten zu. Die Ausgangskordinaten der meisten logischen Koordinatensysteme befinden sich in der unteren linken Ecke. Die vertikale Koordinate nimmt nach oben zu.

Der Ursprung und die Ausdehnung logischer und physikalischer Koordinatensysteme kann mit Hilfe der folgenden vier Funktionen eingerichtet werden:

- `SetViewportExtEx,`
- `SetViewportOrgEx,`
- `SetWindowExtEx` und
- `SetWindowOrgEx.`

(Die alten Funktionen `SetViewportExt`, `SetViewportOrg`, `SetWindowExt` und `SetWindowOrg` werden nicht mehr unter Win32 unterstützt.)

Nachfolgend finden Sie die GDI-Konvertierung von logischen zu physikalischen Koordinaten und umgekehrt aufgeführt:

$$\begin{aligned} Dx &= (Lx - xWO) * xVE/xWE + xVO \\ Dy &= (Ly - yWO) * yVE/yWE + yVO \\ Lx &= (Dx - xVO) * xWE/xVE + xWO \\ Ly &= (Dy - yVO) * yWE/yVE + yWO \end{aligned}$$

Die Bedeutung dieser Berechnungen ist selbsterklärend. Dx ist beispielsweise die horizontale Gerätekoordinate und yWe ist die vertikale Ausdehnung des Fensters. Abbildung 11.3 erläutert die Berechnungen anhand einer Grafik.

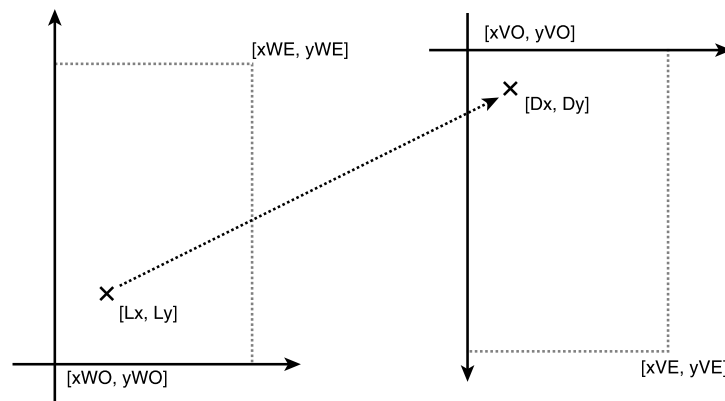


Abbildung 11.3:
Umwandeln
logischer in
physikalische
Koordinaten

Obwohl Windows 95/98 und Windows NT 32-Bit-Koordinatenwerte in den GDI-Funktionsaufrufen verwenden, werden die Koordinaten lediglich unter Windows NT intern als 32-Bit-Werte geführt. Windows 95/98 verwendet 16-Bit-Werte. Die oberen 16 Bit werden ignoriert.



Um Änderungen von einer Konvertierung zur anderen zu erleichtern, bietet Windows einige Hilfsfunktionen an. Dazu zählen:

- ➡ OffsetViewportOrg,
- ➡ OffsetWindowOrg,
- ➡ ScaleViewportExt und
- ➡ ScaleWindowExt.

Beachten Sie bitte, daß eine Anwendung die horizontale oder vertikale Ausrichtung eines Fensters oder Viewports ändern kann, indem sie einen negativen Ausdehnungswert bestimmt.

Anwendungen nutzen die Funktionen `LPtoDP` und `DPtoLP`, um explizit mehrere physikalische Koordinaten in logische Koordinaten und umgekehrt zu konvertieren. *LPtoDP*

11.3.2 Eingeschränkte Abbildungsmodi

Die bisherigen Erläuterungen betreffen lediglich den uneingeschränkten Abbildungsmodus.

Das GDI unterstützt verschiedene Abbildungsmodi, z.B. den uneingeschränkten Abbildungsmodus `MM_ANISOTROPIC`. Hinzu kommen noch eine Reihe von eingeschränkten Abbildungsmodi.

Tabelle 11.1:
Abbildungsmodi

Modus	Beschreibung
MM_TEXT	Der Ursprung des logischen Koordinatensystems ist die linke obere Ecke. Vertikale Koordinaten nehmen nach unten zu. In diesem Modus sind keine Umwandlungen erforderlich, da eine logische Einheit einem Pixel entspricht.
MM_LOENGLISH	Der Ursprung ist in der unteren linken Ecke. Die vertikalen Koordinaten nehmen nach oben zu. Eine logische Einheit entspricht einem Hundertstel eines Inch (0.01").
MM_HIENGLISH	Der Ursprung ist in der unteren linken Ecke. Die vertikalen Koordinaten nehmen nach oben zu. Eine logische Einheit entspricht einem Tausendstel eines Inch (0.001").
MM_LOMETRIC	Der Ursprung ist in der unteren linken Ecke. Die vertikalen Koordinaten nehmen nach oben zu. Eine logische Einheit entspricht einem Zehntel eines Millimeters (0.1 mm).
MM_HIMETRIC	Der Ursprung ist in der unteren linken Ecke. Die vertikalen Koordinaten nehmen nach oben zu. Eine logische Einheit entspricht einem Hundertstel eines Millimeters (0.01 mm).
MM_TWIPS	Der Ursprung ist in der unteren linken Ecke. Die vertikalen Koordinaten nehmen nach oben zu. Eine logische Einheit entspricht einem Zwanzigstel eines Punkts (1/1440").
MM_ISOTROPIC	Die einzige Einschränkung besteht darin, daß logische horizontale und vertikale Einheiten die gleiche Länge aufweisen. Anwendungen können den Ursprung des logischen und physikalischen Koordinatensystems und die horizontale Ausdehnung selbst bestimmen. Das GDI berechnet die vertikale Ausdehnung anhand der horizontalen Ausdehnung.
MM_ANISOTROPIC	Uneingeschränkter Abbildungsmodus. Beide Achsen des Koordinatensystems können unabhängig voneinander skaliert werden.

In den sechs eingeschränkten Abbildungsmodi kann die Anwendung die Ursprungskoordinaten des Viewport und des Fensters verändern. Versuche, die Ausdehnung des Viewport oder Fensters zu modifizieren (mit `SetViewportExtEx` oder `SetWindowExtEx`), werden jedoch ignoriert.

11.3.3 Koordinatentransformation

So flexibel die Koordinatenumwandlung unter Windows auch ist, Windows NT erweitert diese Fähigkeit mit dem Konzept der Koordinatentransformation. Dieses Verfahren ermöglicht Anwendungen, eine beliebige lineare Transformation als Umwandlung eines logischen Koordinatenraums in einen physikalischen Koordinatenraum zu verwenden.

Zur Erläuterung der Koordinatentransformation ist ein kurzer Exkurs in die Geometrie notwendig.

Lineare Transformationen lassen sich einer der folgenden Kategorien zuordnen: Verschieben, Skalieren, Rotieren, Zuschneiden und Spiegeln.

Verschieben

Bedeutet, daß Konstanten sowohl den horizontalen als auch den vertikalen Koordinaten eines Objekts hinzugefügt werden. (siehe Abbildung 11.4)

$$x^1 = x + Dx$$

$$y^1 = y + Dy$$

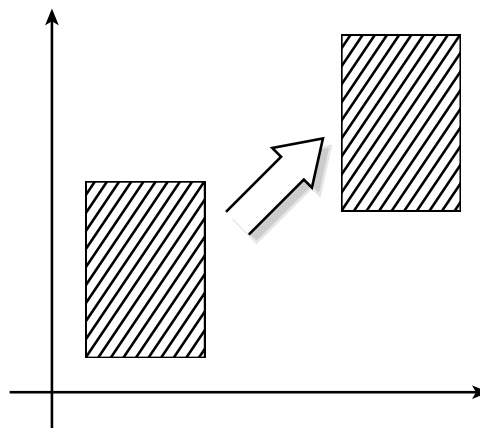


Abbildung 11.4:
Verschieben

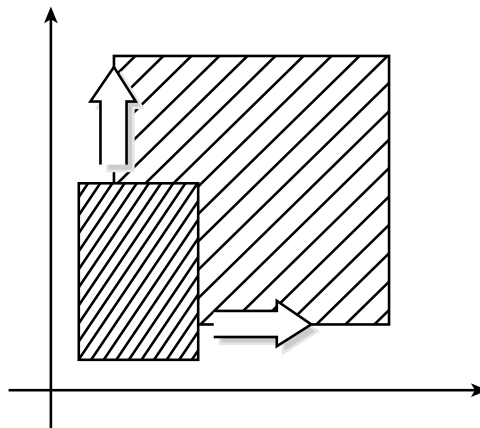
Skalieren

Das Vergrößern oder Verkleinern der horizontalen oder vertikalen Ausdehnung eines Objekts. (siehe Abbildung 11.5)

$$x^1 = xS_x$$

$$y^1 = yS_y$$

Abbildung 11.5:
Skalieren



Rotieren

Während des Rotierens werden die Punkte eines Objekts um die Ausgangsposition gedreht. Ist der Winkel der Rotation bekannt, kann die Rotation wie folgt berechnet werden. (siehe Abbildung 11.6)

$$x^1 = x \cos \alpha - y \sin \alpha$$

$$y^1 = x \sin \alpha + y \cos \alpha$$

Zuschneiden

Verfahren, das ein Rechteck in ein Parallelogramm umwandelt. Dazu werden zwei der horizontalen Punkte verschoben. Die folgende Formel führt die entsprechende Berechnung aus. (siehe Abbildung 11.7)

$$x^1 = x + S_x y$$



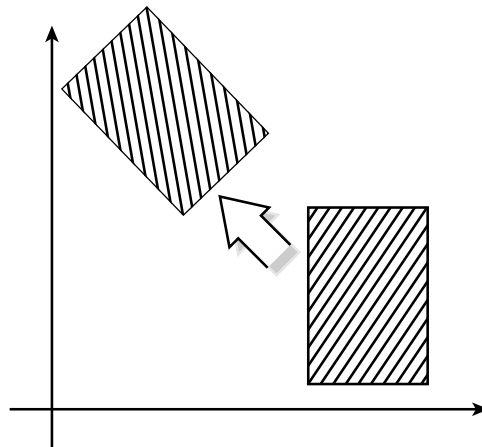


Abbildung 11.6:
Rotation

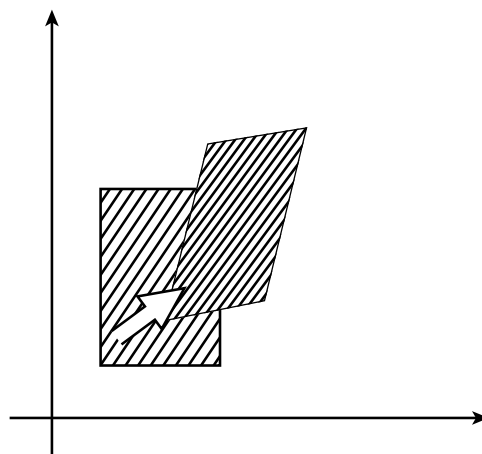


Abbildung 11.7:
Zuschneiden

Spiegeln

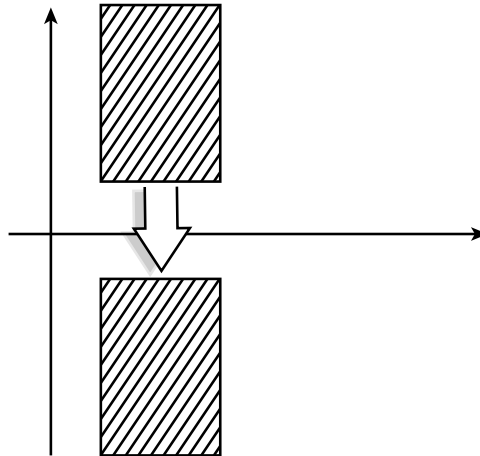
Ein Objekt wird entweder an der horizontalen oder vertikalen Achse gespiegelt. Abbildung 11.8 zeigt eine Spiegelung an der horizontalen Achse. Das Spiegeln geschieht mit Hilfe der folgenden Formel:

$$y^1 = -y$$

Eine Spiegelung an der vertikalen Achse wird wie folgt berechnet:

$$x^1 = -x$$

Abbildung 11.8:
Spiegelung an
der horizon-
talen Achse



All diese Transformationen können ebenfalls in 3×3 -Matrixen berechnet werden. Die Matrix einer Übersetzung ist nachfolgend dargestellt:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

Die Matrix für die Skalierung:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die Matrix einer Rotation, berechnet mit trigonometrischen Funktionen des Rotationswinkels:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die Matrix einer Zuschneidung:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & S_x & 0 \\ S_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Eine Spiegelung an der horizontalen Achse wird in einer Matrix wie folgt berechnet:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Schließlich eine Spiegelung an der vertikalen Achse:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Lineare Transformationen können miteinander kombiniert werden. Das Ergebnis einer Kombination von zwei linearen Transformationen ist eine dritte lineare Transformation. Für eine Matrix formuliert, kann die resultierende Transformation als das Produkt der Matrizen bezeichnet werden, die die originale Transformation repräsentieren.

Lineare Transformationen sind nicht austauschbar. Die Reihenfolge, in der sie ausgeführt werden, ist somit wichtig.



Obwohl jede lineare Transformation mit einer der hier vorgestellten grundlegenden Transformationen berechnet werden kann, ist eine allgemeine lineare Transformation keine einfache Verschiebung, Skalierung, Rotation, Zuschneidung oder Spiegelung. Eine allgemeine lineare Transformation kann wie folgt berechnet werden:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & 0 \\ M_{21} & M_{22} & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

Dies ist der Matrixtyp einer Anwendung, der der Funktion `SetWorldTransform` übergeben werden muß. Der zweite Parameter dieser Funktion ist ein Zeiger auf die `XFORM`-Struktur, die wie folgt aufgebaut ist:

```
typedef struct _XFORM
{
    FLOAT eM11;
    FLOAT eM12;
    FLOAT eM21;
    FLOAT eM22;
    FLOAT eDx;
    FLOAT eDy;
} XFORM;
```

Lernen Sie nun die Funktion `CombineTransform` kennen. Diese Funktion multipliziert zwei Transformationsmatrizen, die über die `XFORM`-Struktur definiert sind.

Nachdem eine Transformation für einen Gerätekontext eingerichtet wurde, wandelt diese logische Koordinaten in Seitenraumkoordinaten um. Seitenraumkoordinaten sind ein weiterer Aspekt der Transformation, die von dem Abbildungsmodus spezifiziert wurde.

Obwohl Anwendungen die Funktion `DPtoLP` verwenden können, um anhand der physikalischen Koordinaten die Transformationskoordinaten zu ermitteln, ist eine explizite Berechnung der Transformationsmatrix, die mit der invertierten Matrix korrespondiert, bisweilen nützlich. Zur Ermittlung der invertierten Matrix sollte zunächst die Begrenzung der Transformationsmatrix berechnet werden:

$$D = \begin{vmatrix} M_{11} & M_{12} & 0 \\ M_{21} & M_{22} & 0 \\ D_x & D_y & 1 \end{vmatrix} = \begin{vmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{vmatrix} = M_{11} M_{22} - M_{12} M_{21}$$

Ist dieser Wert 0, existiert die invertierte Matrix nicht. Dies geschieht, wenn die Transformation Mängel aufweist und viele Punkte im Transformationsraum auf einen Punkt im Seitenraum verweisen. Verweist beispielsweise der Transformationsraum auf eine Linie im Seitenraum, entspricht ein Punkt im Seitenraum nicht länger einem einzelnen Punkt im Transformationsraum. Die invertierte Transformation ist daher nicht möglich.

Nachdem die Begrenzung ermittelt wurde, kann die invertierte Matrix einfach berechnet werden:

$$A^{-1} = \frac{1}{D} \begin{bmatrix} \begin{vmatrix} M_{22} & 0 \\ D_y & 1 \end{vmatrix} & - \begin{vmatrix} M_{12} & 0 \\ D_y & 1 \end{vmatrix} & \begin{vmatrix} M_{12} & 0 \\ M_{22} & 0 \end{vmatrix} \\ - \begin{vmatrix} M_{21} & 0 \\ D_x & 1 \end{vmatrix} & \begin{vmatrix} M_{11} & 0 \\ D_x & 1 \end{vmatrix} & - \begin{vmatrix} M_{11} & 0 \\ M_{21} & 0 \end{vmatrix} \\ \begin{vmatrix} M_{21} & M_{22} \\ D_x & D_y \end{vmatrix} & - \begin{vmatrix} M_{11} & M_{12} \\ D_x & D_y \end{vmatrix} & \begin{vmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{vmatrix} \end{bmatrix} =$$

$$= \begin{bmatrix} M_{22}/D & -M_{12}/D & 0 \\ -M_{21}/D & M_{11}/D & 0 \\ (M_{21}D_y - M_{22}D_x)/D & (M_{12}D_x - M_{11}D_y)/D & 1 \end{bmatrix}$$

Die Funktion in Listing 11.1 erstellt eine invertierte Transformation. Existiert die invertierte Transformation nicht, gibt die Funktion die originale Transformation zurück. Der Rückgabewert der Funktion ist `FALSE`, wenn ein Fehler auftrat. Wie andere `XFORM`-Funktionen akzeptiert auch `InvertTransform` denselben Zeiger für die Eingabe- und Ausgabe-`XFORM`-Struktur.

```

BOOL InvertTransform(LPXFORM lpxformResult, CONST XFORM *lpxform)
{
    XFORM xformTmp;
    FLOAT D;
    D = lpxform->eM11*lpxform->eM22
        - lpxform->eM12*lpxform->eM21;
    if (D == 0.0)
    {
        lpxformResult->eM11 = 1.0;
        lpxformResult->eM12 = 0.0;
        lpxformResult->eM21 = 0.0;
        lpxformResult->eM22 = 1.0;
        lpxformResult->eDx = 0.0;
        lpxformResult->eDy = 0.0;
        return FALSE;
    }
    xformTmp.eM11 = lpxform->eM22 / D;
    xformTmp.eM12 = -lpxform->eM12 / D;
    xformTmp.eM21 = -lpxform->eM21 / D;
    xformTmp.eM22 = lpxform->eM11 / D;
    xformTmp.eDx = (lpxform->eM21*lpxform->eDy -
        lpxform->eM22*lpxform->eDx) / D;
    xformTmp.eDy = (lpxform->eM12*lpxform->eDx -
        lpxform->eM11*lpxform->eDy) / D;
    *lpxformResult = xformTmp;
    return TRUE;
}

```

Listing 11.1:
Invertieren einer
Transformation

Die Funktion `SetWorldTransform` kann erst dann ausgeführt werden, nachdem der Grafikmodus mit `SetGraphicMode` auf `GM_ADVANCED` gesetzt wurde. Um den Grafikmodus auf `GM_COMPATIBLE` zurückzusetzen, muß die originale Matrix wieder eingerichtet werden.

11.4 Zeichenobjekte

Koordinatentransformationen definieren, wo auf dem Ausgabegerät gezeichnet werden soll. Welche Figuren gezeichnet werden sollen, wird durch den Einsatz der GDI-Objekte bestimmt.

Das GDI bietet verschiedene Zeichenobjekte an: Stifte, Pinsel, Schriftarten, Paletten und Bitmaps. Anwendungen, die diese Objekte verwenden, müssen die folgenden Schritte ausführen:

- GDI-Objekte verwenden*
1. Erstellen des GDI-Objekts
 2. Selektieren des GDI-Objekts im Gerätekontext
 3. Aufruf der GDI-Ausgabefunktionen
 4. Auswahl des GDI-Objekts rückgängig machen
 5. Objekt zerstören

Der folgende Programmcode demonstriert diese Schritte. Dort wird ein Stift-Objekt zum Zeichnen eines Rechtecks in einen Gerätekontext verwendet. Der Kontext wird durch den Handle `hDC` bezeichnet:

```
HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
HPEN hOldPen = (HPEN)SelectObject(hDC, hPen);
Rectangle(hDC, 0, 0, 100, 100);
SelectObject(hOldPen);
DeleteObject(hPen);
```

GDI-Objekte werden mit einer der verschiedenen Funktionen erstellt, die in den folgenden Abschnitten vorgestellt werden. Nachdem ein GDI-Objekt erzeugt wurde, geschieht der Zugriff darauf über einen Handle.

SelectObject Das Objekt wird mit der Funktion `SelectObject` in den Gerätekontext selektiert. (Paletten werden mit `SelectPalette` selektiert.) Diese Funktion gibt einen Handle zurück, der auf das zuvor selektierte Objekt verweist (Stift, Pinsel, Schriftart oder Bitmap). Nachdem der Zeichenvorgang beendet ist, kann dieser Handle dazu verwendet werden, den ursprünglichen Zustand des Gerätekontextes wiederherzustellen. Objekte, die nicht mehr benötigt werden, sollten mit der Funktion `DeleteObject` zerstört werden.

Ein GDI-Objekt muß nicht neu erstellt werden. Anwendungen können auch vordefinierte Systemobjekte mit einem Aufruf der Funktion `GetStockObject` verwenden. `GetStockObject` ermittelt den Handle eines Stifts, Pinsels, einer Schriftart und der Systempalette. Wenngleich `DeleteObject` für ein vordefiniertes Objekt nicht aufgerufen werden muß, sollten Sie diese Funktion dennoch verwenden.

11.4.1 Stifte

Stifte werden dazu verwendet, Linien, Kurven und die Konturen anderer Figuren zu zeichnen. Ein Stift wird mit `CreatePen` erstellt. Während des Aufrufs dieser Funktion bestimmt die Anwendung die Breite, Farbe und den Stil des Stifts.

CreatePen

Die Stiftfarbe wird als RGB-Wert übergeben. Ist ein entsprechender Eintrag in der logischen Palette enthalten, ersetzt Windows gewöhnlich die nächstliegende Palettenfarbe durch diesen Eintrag. Wenn die Breite des Stifts jedoch größer als eins ist und der Stil `PS_INSIDEFRAME` verwendet wird, greift Windows auf eine zusammengesetzte Dither-Farbe zurück.

Gestrichelte oder punktierte Stiftstile werden nicht für Stifte unterstützt, deren Breite größer als eins ist. Unter Windows NT können solche Stifte mit `ExtCreatePen` erstellt werden. Diese Funktion ist auch unter Windows 95/98 erhältlich. Ihre Funktionalität ist jedoch eingeschränkt.

`ExtCreatePen` bietet außerdem eine bessere Kontrolle über die gezeichneten Figuren.

Eine weitere Funktion, die zur Erstellung eines Stifts verwendet wird, ist `CreatePenIndirect`. Diese Funktion nimmt einen Zeiger auf die `LOGPEN`-Struktur als Parameter entgegen. Die `LOGPEN`-Struktur definiert die Breite, Farbe und den Stil des Stifts.

Das Zeichnen mit einem Stift richtet sich nach dem Vordergrund-Mixmodus. Dieser Modus wird mit der Funktion `SetROP2` aktiviert. Verschiedene Einstellungen definieren mehrere logische Operationen, die die Stiftfarbe und Pixel-Farbe betreffen. Der aktuelle Mixmodus kann mit Hilfe der Funktion `GetROP2` ermittelt werden.

11.4.2 Pinsel

Ein Pinsel wird zum Ausfüllen des Inhalts der Zeichenfiguren verwendet. Dazu muß die Farbe und das Füllmuster des Pinsels bestimmt werden.

Ein Pinsel wird mit einem Aufruf der Funktion `CreateBrushIndirect` erstellt. Dieser Funktion wird ein Zeiger auf die `LOGBRUSH`-Struktur übergeben, in der die Farbe, das Muster und der Stil des Pinsels definiert sind.

CreateBrush-Indirect

Das Füllmuster kann auf einer Bitmap basieren. Wird der Pinselstil auf den Wert `BS_DIBPATTERN` oder `BS_DIBPATTERNPT` gesetzt, bestimmt der Member `lStyle` in der `LOGBRUSH`-Struktur die Zugriffsnummer dieser Bitmap.



Windows 95/98 unterstützt 8×8-Pixel-Bitmaps. Wird eine größere Bitmap angegeben, verwendet das Betriebssystem lediglich einen Abschnitt der Bitmap.

Ein Pinsel kann außerdem schraffiert werden. Dazu bestimmt das Element `lbStyle` der `LOGBRUSH`-Struktur das Schraffurmuster.

Das Element `lbColor` spezifiziert die Vordergrundfarbe eines schraffierten Pinsels. Die Hintergrundfarbe und der Hintergrundmodus werden mit den Funktionen `SetBkColor` und `SetBkMode` gesetzt.

Ein auf Füllmuster und schraffierte Pinsel bezogenes Problem ist die Ausgangsposition des Pinsels. Um eine flüssige Darstellung zu gewährleisten, muß die Position des Pinsels ausgerichtet werden, wenn Abschnitte einer Figur zu verschiedenen Zeitpunkten gezeichnet werden. Unter Windows 95/98 wird dazu die Funktion `UnrealizeObject` aufgerufen, bevor ein Pinsel in einen Gerätekontext selektiert wird. Diese Vorgehensweise ist unter Windows NT nicht erforderlich, da dort eine automatische Ausrichtung erfolgt.

Anwendungen können die Ausgangsposition eines Pinsels mit der Funktion `SetBrushOrgEx` bestimmen. Diese Position ist ein Koordinatenpaar, das die relative Entfernung des Pinselmusters von der oberen linken Ecke des Dokumentfensters beschreibt.

Weitere Funktionen unterstützen das Erstellen und Verwenden von Pinseln. Feste Pinsel, Füllmusterspindel und schraffierte Pinsel können mit `CreateSolidBrush`, `CreatePatternBrush` und `CreateHatchBrush` erzeugt werden. Pinsel, die auf geräteunabhängigen Bitmaps basieren, können mit `CreateDIBPatternBrushPt` erstellt werden.

Das Zeichnen der Innenfläche eines Objekts richtet sich nach dem Vordergrund-Mixmodus, der mit einem Aufruf von `SetROP2` gesetzt wird.

11.4.3 Schriftarten

CreateFont Bevor eine Anwendung Text ausgeben kann, muß sie eine logische Schriftart selektieren. Logische Schriftarten werden mit der Funktion `CreateFont` erstellt.

Anwender, die an Anwendungen gewöhnt sind, die eine explizite Auswahl einer Schriftart durch die Angabe des Namens, der Attribute und Größe ermöglichen, könnte die Verwendung von `CreateFont` zunächst verwirren. Obwohl die Auswahl einer Schriftart über deren Namen möglich ist, bietet `CreateFont` sehr viele zusätzliche Parameter.

Diese Methode der Erzeugung einer logischen Schriftart ist ein weiteres Feature, über das Windows vollständige Geräteunabhängigkeit implementiert. Die Anwendung ist nicht auf eine bestimmte Schriftart angewiesen (die möglicherweise nicht für alle Ausgabegeräte oder Computer erhältlich wäre), sondern kann Schriftarten anhand ihrer Eigenschaften selektieren. Fordert eine Anwendung eine Schriftart mit `CreateFont` an, stellt Windows aus einer Menge der verfügbaren Schriftarten den Font zur Verfügung, dessen Eigenschaften weitgehend denen der gewünschten Schriftart entsprechen.

Natürlich kann `CreateFont` der Name und die Größe einer Schriftart übergeben werden. Windows versucht anschließend, den gewünschten Font zu selektieren, sofern dieser im System vorhanden ist.

Anwendungen können außerdem `CreateFontIndirect` aufrufen, um eine Schriftart zu erhalten. Dieser Funktion wird als Parameter ein Zeiger auf eine `LOGFONT`-Struktur übergeben. Sie wird vorwiegend in Verbindung mit dem Dialog `SCHRIFTART` verwendet, der die Auswahl des Anwenders in einer `LOGFONT`-Struktur ablegt.

*CreateFont-
Indirect*

Die Funktion `EnumFontFamilies` führt alle Schriftfamilien oder die Schriftarten einer Schriftfamilie auf.

Der Anwendungsprogrammierer wird von vielen anderen Funktionen unterstützt, die sich auf die Schriftarten beziehen. Funktionen wie `GetCharABCWidths` dienen der Ermittlung der Breite eines Zeichens. Die Funktionen `GetTabbedExtent` und `GetTextExtentPint32` berechnen die Breite und Höhe einer Zeichenfolge.

Anwendungen können Schriftarten mit `AddFontResource`, `CreateScalableFontResource` und `RemoveFontResource` installieren und deinstallieren.

11.4.4 Paletten

Paletten wären nicht notwendig, wenn alle Ausgabegeräte eine Farbtiefe von 24-Bit-RGB-Werten darstellen könnten. Leider bieten die meisten Bildschirme der unteren Preisklasse lediglich einen Kompromiß aus Farbtiefe und Bildschirmauflösung. Viele PCs arbeiten gegenwärtig mit einer Auflösung von 800×600, 1024×768 oder 1280×1024 Pixeln und 256 Farben.

Welche Paletten ein Gerät unterstützt, kann mit der Funktion `GetDeviceCaps` ermittelt werden. Nach dem Aufruf dieser Funktion wird das Flag `RC_PALETTE` in dem Wert `RASTERCAPS` geprüft. Die Farbpalette definiert die Farben, die gegenwärtig von der Anwendung verwendet werden können.

GetDeviceCaps

Die Systempalette

Die Systempalette führt alle Farben auf, die derzeit von dem Gerät dargestellt werden können. Anwendungen können die Systempalette nicht direkt modifizieren, aber deren Inhalte mit Hilfe der Funktion `GetSystemPaletteEntries` einsehen. Die Systempalette enthält eine Anzahl (gewöhnlich 2 bis 20) statischer Farben, die nicht verändert werden können. Anwendungen begrenzen die Anzahl der statischen Farben mit `SetSystemPaletteUse`.

Die Standardpalette

Die Standardpalette verfügt gemeinhin über 20 Farbeinträge. Dieser Wert kann jedoch für jedes Gerät unterschiedlich sein. Fordert die Anwendung eine Farbe an, die nicht in der Palette enthalten ist, selektiert Windows eine Farbe aus der Palette, die weitgehend mit der gewünschten Farbe übereinstimmt. Für Pinsel wird die Farbe in diesem Fall aus mehreren Farben zusammengesetzt (Dither-Farbe). Diese Vorgehensweise ist für farbsensitive Anwendungen nicht immer ausreichend.

Logische Paletten

Anwendungen können daher eine logische Palette spezifizieren, die die Standardpalette ersetzt. Eine logische Palette kann mehrere Farben enthalten (die Anzahl der möglichen Farben ist in dem Wert `SIZEPALETTE` gespeichert und kann mit `GetDeviceCaps` ermittelt werden). Sie wird mit `CreatePalette` erstellt, und ihre Farben können später mit einem Aufruf der Funktion `SetPaletteEntries` modifiziert werden. Mit Hilfe der Funktion `SelectPalette` selektieren Sie eine Palette in einem Gerätekontext. Sie löschen eine Palette, die nicht mehr benötigt wird, mit `DeleteObject`.

Bevor Sie eine Palette verwenden können, muß diese mit `RealizePalette` realisiert werden. Abhängig von dem Anzeigegerät und dem Einsatz der Palette als Vordergrund- oder Hintergrundpalette, realisiert Windows eine Farbpalette unterschiedlich. Eine Palette kann als Vordergrundpalette eingesetzt werden, wenn das Fenster, das die Palette verwenden soll, das aktive Fenster oder ein vom aktiven Fenster abgeleitetes Fenster ist. Innerhalb des Systems kann immer nur eine Vordergrundpalette verwendet werden. Eine Vordergrundpalette kann alle Farben der Systempalette überschreiben, die nicht statisch sind. Dazu werden alle nichtstatischen Einträge als nicht benötigt markiert, bevor die Vordergrundpalette realisiert wird.

Wenn eine Palette realisiert wird, füllt Windows die nicht benötigten Einträge der Systempalette mit den Farben der logischen Palette. Sind keine weiteren der nichtbenötigten Einträge verfügbar, legt Windows die verbleibenden Farben in der logischen Palette ab. Dazu verwendet das Betriebssystem die Farben, die weitgehend den Farben der physikalischen Palette entsprechen. Natürlich kann Windows diese Farben auch aus mehreren Farben zusammensetzen. Das Betriebssystem realisiert zunächst die Vordergrundpalette und anschließend die restlichen Hintergrundpaletten im FIFO-Verfahren.

Beachten Sie bitte, daß jede Änderung der Systempalette globale Auswirkungen hat. Diese Änderungen betreffen die gesamte Darstellungsoberfläche und nicht nur das Fenster der Anwendung. Änderungen an der Systempalette können dazu führen, daß Anwendungen ihre Fensterinhalte neu zeichnen. Deshalb ist es empfehlenswert, eine Palette als Hintergrundpalette einzurichten. Auf diese Weise werden Änderungen an der Palette vermieden, wenn das Fenster, das die Palette verwendet, den Fokus erhält oder verliert.

Hintergrundpalette

Windows definiert einige Nachrichten, die sich auf Paletten beziehen. Ein Top-Level-Fenster empfängt die Nachricht `WM_PALETTECHANGED`, wenn das Betriebssystem die Palette verändert. Bevor ein Top-Level-Fenster zum aktiven Fenster wird, erhält es die Nachricht `WM_QUERYNEWPALETTE`. Die Anwendung realisiert daraufhin die Palette. Dazu werden die Funktionen `SelectPalette`, `UnrealizeObject` und `RealizePalette` aufgerufen.

Palettennachrichten

Ein interessantes Feature der Paletten ist die Palettenanimation. Diese Technik ändert die Einträge der logischen Palette in regelmäßigen Abständen, um den Eindruck einer Animation zu erwecken. Anwendungen verwenden dazu die Funktion `AnimatePalette`.

Palettenanimation

Um zu gewährleisten, daß eine in der Palette enthaltene Farbe selektiert wurde (besonders wichtig für die Palettenanimation), sollten Anwendungen die Makros `PALETTEINDEX` oder `PALETTEINDEX` verwenden.

Eine Anwendung, die eine einfache Palettenanimation implementiert, ist in Listing 11.2 aufgeführt. Diese Anwendung kann mit der Anweisung

```
CL ANIMATE.C GDI32.LIB USER32.LIB
```

über die Kommandozeile kompiliert werden. Die Anwendung wird nur dann korrekt ausgeführt, wenn Ihre Grafik-Hardware für 256 Farben konfiguriert wurde.

Listing 11.2:
*Paletten-
animation*

```
#include <windows.h>

struct
{
    WORD palVersion;
    WORD palNumEntries;
    PALETTEENTRY palPalEntry[12];
} palPalette =
{
    0x300,
    12,
    {
        {0xFF, 0x00, 0x00, PC_RESERVED},
        {0xC0, 0x40, 0x00, PC_RESERVED},
        {0x80, 0x80, 0x00, PC_RESERVED},
        {0x40, 0xC0, 0x00, PC_RESERVED},
        {0x00, 0xFF, 0x00, PC_RESERVED},
        {0x00, 0xC0, 0x40, PC_RESERVED},
        {0x00, 0x80, 0x80, PC_RESERVED},
        {0x00, 0x40, 0xC0, PC_RESERVED},
        {0x00, 0x00, 0xFF, PC_RESERVED},
        {0x40, 0x00, 0xC0, PC_RESERVED},
        {0x80, 0x00, 0x80, PC_RESERVED},
        {0xC0, 0x00, 0x40, PC_RESERVED}
    }
};

void Animate(HWND hwnd, HPALETTE hPalette)
{
    HDC hDC;
    PALETTEENTRY pe[12];
    HPALETTE hOldPal;
    static int nIndex;
    int i;

    for (i = 0; i < 12; i++)
        pe[i] = palPalette.palPalEntry[(i + nIndex) % 12];
    hDC = GetDC(hwnd);
    hOldPal = SelectPalette(hDC, hPalette, FALSE);
    RealizePalette(hDC);
    AnimatePalette(hPalette, 0, 12, pe);
    nIndex = (++nIndex) % 12;
    SelectPalette(hDC, hOldPal, FALSE);
    ReleaseDC(hwnd, hDC);
}

void DrawCircle(HWND hwnd, HPALETTE hPalette)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT rect;
    HPALETTE hOldPal;
    int i;

    hDC = BeginPaint(hwnd, &paintStruct);
    if (hDC != NULL)
    {
        hOldPal = SelectPalette(hDC, hPalette, FALSE);
        RealizePalette(hDC);
        GetClientRect(hwnd, &rect);
        DPTOLP(hDC, (LPPPOINT)&rect, 2);
        for (i = 0; i < 12; i++)
        {
            HBRUSH hbr;
```

```

        HBRUSH hbrOld;
        hbr = CreateSolidBrush(PALETTEINDEX(i));
        hbrOld = (HBRUSH)SelectObject(hDC, hbr);
        Rectangle(hDC,
            MulDiv(i, rect.right, 24),
            MulDiv(i, rect.bottom, 24),
            rect.right - MulDiv(i, rect.right, 24),
            rect.bottom - MulDiv(i, rect.bottom, 24)
        );
        SelectObject(hDC, hbrOld);
        DeleteObject(hbr);
    }
    SelectPalette(hDC, hOldPal, FALSE);
    EndPaint(hwnd, &paintStruct);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
    static HPALETTE hPalette;

    switch(uMsg)
    {
        case WM_CREATE:
            hPalette = CreatePalette((LPLOGPALETTE)&palPalette);
            break;
        case WM_PAINT:
            DrawCircle(hwnd, hPalette);
            break;
        case WM_TIMER:
            Animate(hwnd, hPalette);
            break;
        case WM_DESTROY:
            DeleteObject(hPalette);
            hPalette = NULL;
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HELLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("HELLO", "HELLO",
        WS_OVERLAPPEDWINDOW,

```

```

        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
        NULL, NULL, hInstance, NULL);
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
SetTimer(hwnd, 1, 100, NULL);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
KillTimer(hwnd, 1);
return msg.wParam;
}

```

Diese Anwendung zeichnet zwölf Rechtecke. Jedes Rechteck erhält eine andere Farbe, die aus der logischen Palette ausgewählt wird. Die Anwendung richtet einen Zeitgeber ein. Wird die Nachricht `WM_TIMER` empfangen, ruft die Anwendung die Funktion `AnimatePalette` auf. Die daraus resultierende Animation erweckt den Eindruck einer Bewegung durch einen Tunnel.

11.4.5 Bitmap-Objekte

Bitmaps sind ebenfalls GDI-Objekte. Anwendungen zeichnen gewöhnlich in eine Bitmap oder übertragen deren Inhalt zu einem Ausgabegerät.

Eine Bitmap ist ein rechteckiger Pixelbereich. Jedem Pixel kann eine andere Farbe zugewiesen werden, die durch einen oder mehrere Pixel repräsentiert wird. Die Anzahl dieser Bits ist von der Farbtiefe der Bitmap abhängig. Eine Bitmap mit einer Farbtiefe von 8 Bit kann bis zu 256 Farben darstellen. Eine Echtfarben-Bitmap kann bei einer Farbtiefe von 24 Bit aus 16.777.216 Farben bestehen.

CreateBitmap Ein leeres GDI-Bitmap-Objekt wird mit `CreateBitmap` erstellt. Obwohl mit dieser Funktion Farb-Bitmaps erzeugt werden können, sollten Sie `CreateBitmap` lediglich für monochrome Bitmaps verwenden. Rufen Sie `CreateCompatibleBitmap` auf, wenn Sie eine farbige Bitmap benötigen.

DIBs Bitmap-Objekte sind geräteabhängig. Einige Funktionen erlauben der Anwendung jedoch, in geräteunabhängige Bitmaps (DIB) zu zeichnen. (Windows-Bitmap-Dateien sind geräteunabhängig.)

Eine Anwendung muß eine Bitmap zunächst in einem Gerätekontext selektieren, um darin zeichnen zu können.

LoadBitmap Um eine Bitmap aus einer Ressourcendatei zu laden, verwenden Sie die Funktion `LoadBitmap`. Diese Funktion erstellt ein Bitmap-Objekt und initialisiert es mit der Bitmap der Ressourcendatei (zweiter Funktionsparameter).

11.5 Clipping

Das Clipping ist eine sehr wichtige Technik in der Multitasking-Umgebung. Dank dieser Technik zeichnen Anwendungen nicht versehentlich außerhalb des Client-Bereichs ihrer Fenster. Außerdem steuert das Clipping Situationen, in denen Bereiche der Anwendungsfenster von anderen Fenstern verdeckt oder nicht angezeigt werden.

Doch nicht nur das Betriebssystem, sondern auch Anwendungen können auf die Clipping-Funktionen zugreifen. Sie können einen Clipping-Bereich für einen Gerätekontext bestimmen und die Grafikausgabe somit auf diesen Bereich begrenzen.

Ein Clipping-Bereich ist gewöhnlich, aber nicht immer, rechteckig. Tabelle 11.1 führt die verschiedenen Bereichstypen sowie die entsprechenden Funktionen auf, die zur Erzeugung der Bereiche erforderlich sind.

Symbolische Bezeichner	Beschreibung
Elliptischer Bereich	CreateEllipticRgn, CreateEllipticRgnIndirect
Polygon-Bereich	CreatePolygonRgn, CreatePolyPolygonRgn
Rechteckiger Bereich	CreateRectRgn, CreateRectRgnIndirect
Abgerundeter, rechteckiger Bereich	CreateRoundRectRgn

Tabelle 11.2:
Clipping-
Bereiche

Einige Ausgabegeräte können lediglich mit einem rechteckigen Clipping-Bereich arbeiten.



Mit einem Aufruf von `SelectObject` oder `SelectClipRgn` selektieren Anwendungen einen Clipping-Bereich in einem Gerätekontext. Diese beiden Funktionen führen zu demselben Ergebnis. Eine weitere Funktion ermöglicht die Kombination eines neuen Bereichs mit einem bereits bestehenden Clipping-Bereich, der mit `CombineRgn` erzeugt wurde. Diese Funktion trägt die Bezeichnung `SelectClipRgnExt`.

Das Clipping geschieht ebenfalls mit Hilfe von Clipping-Pfaden. Diese Pfade definieren komplexe Clipping-Figuren, die mit gewöhnlichen Clipping-Bereichen nicht erstellt werden können. Ein Clipping-Pfad wird mit den Funktionen `BeginPath` und `EndPath` erstellt und anschließend mit `SelectClipPath` selektiert.

Clipping-Pfade können zur Gestaltung interessanter Spezialeffekte verwendet werden. Ein Beispiel hierfür ist in Listing 11.3 aufgeführt. Diese Anwendung, die in Abbildung 11.9 dargestellt ist, verwendet eine Zeichenfolge zur Erstellung eines Clipping-Pfads. Kompilieren Sie dieses Programm über die Kommandozeile:

```
CL CLIPPATH.C GDI32.LIB USER32.LIB
```

Abbildung 11.9:
Clipping-Pfade



Listing 11.3:
Das Verwenden
von Clipping-
Pfaden

```
#include <windows.h>
#include <math.h>

void DrawHello(HWND hwnd)
{
    PAINTSTRUCT paintStruct;
    RECT rect;
    HFONT hFont;
    SIZE sizeText;
    POINT ptText;
    HDC hDC;
    double a, d, r;

    hDC = BeginPaint(hwnd, &paintStruct);
    if (hDC != NULL)
    {
        GetClientRect(hwnd, &rect);
        DPTOLP(hDC, (LPPPOINT)&rect, 2);
        hFont = CreateFont((rect.bottom - rect.top) / 2,
                           (rect.right - rect.left) / 13, 0, 0,
                           FW_HEAVY, FALSE, FALSE, FALSE,
                           ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                           CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                           DEFAULT_PITCH | FF_DONTCARE, "Arial");
        SelectObject(hDC, hFont);
        GetTextExtentPoint32(hDC, "Hello, World!", 13, &sizeText);
```

```

    ptText.x = (rect.left + rect.right - sizeText.cx) / 2;
    ptText.y = (rect.top + rect.bottom - sizeText.cy) / 2;
    SetBkMode(hDC, TRANSPARENT);
    BeginPath(hDC);
    TextOut(hDC, ptText.x, ptText.y, "Hello, World!", 13);
    EndPath(hDC);
    SelectClipPath(hDC, RGN_COPY);
    d = sqrt((double)sizeText.cx * sizeText.cx +
             sizeText.cy * sizeText.cy);
    for (r = 0; r <= 90; r+= 1)
    {
        a = r / 180 * 3.14159265359;
        MoveToEx(hDC, ptText.x, ptText.y, NULL);
        LineTo(hDC, ptText.x + (int)(d * cos(a)),
               ptText.y + (int)(d * sin(a)));
    }
    EndPaint(hwnd, &paintStruct);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_PAINT:
            DrawHello(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HELLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("HELLO", "HELLO",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                       NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Dieses Programm gibt den Text »Hello, World!« in großen Zeichen und der Schriftart Arial aus. Die Größe wird proportional zur Größe des Client-Bereichs berechnet. Dieser Text bildet den Clipping-Pfad. Anschließend werden einige Linien von der linken oberen Ecke des Text-rechtecks gezeichnet. Infolge des Clippings werden lediglich die Linien innerhalb des Textes angezeigt.

11.6 Zeichenfunktionen

Sie haben nun den Gerätekontext als »Zeichenfläche« kennengelernt, auf der GDI-Funktionen grafische Ausgaben zeichnen. Sie wissen, daß das GDI bestimmte Werkzeuge verwendet, wie z.B. Stifte, Pinsel oder Schriftarten, um Grafiken zu erstellen. Dieser Abschnitt erläutert die eigentlichen Zeichenoperationen, die das GDI verwendet.

Die GDI-Ausgabe erfolgt gewöhnlich in mehreren Schritten, die in Abbildung 11.10 dargestellt sind. Dazu zählen das Ermitteln des Handles des Gerätekontextes, das Einrichten des Gerätekontextes zum Zeichnen, das Ausführen der Zeichenoperationen, das Wiederherstellen des ursprünglichen Zustands des Gerätekontextes sowie die Freigabe des Gerätekontextes. Einige Anwendungen führen diese Schritte möglicherweise in einer anderen Reihenfolge aus, verzichten auf irrelevante Schritte oder verwenden andere Initialisierungen der Zeichenfunktionen, um besonderen Anforderungen gerecht zu werden.

11.6.1 Linien

MoveToEx Die einfachste Windows-Zeichenfunktion zieht eine Linie. Eine Linie wird mit einem Aufruf der Funktion *MoveToEx* gezeichnet, der ein Aufruf der Funktion *LineTo* folgt.

➡ *MoveToEx* aktualisiert die gegenwärtige Position, die sich in dem Koordinatenraum des Gerätekontextes befindet, der von den Zeichenfunktionen verwendet wird.

➡ *LineTo* zeichnet die Linie von dieser Position bis zu den Koordinaten, die in den Parametern übergeben werden. Die Linie wird mit dem aktuellen, in dem Gerätekontext selektierten Stift gezeichnet.

Auf Rastergeräten wird eine Linie mit einem DDA-Algorithmus (Digital Differential Analyzer) gezeichnet. Dieser Algorithmus ermittelt, welcher Pixel der Zeichenoberfläche gesetzt werden soll. Anwendungen, die den Einsatz eines DDA-Algorithmus erfordern, der nicht dem Standard entspricht, können die Funktion *LineDDA* verwenden.

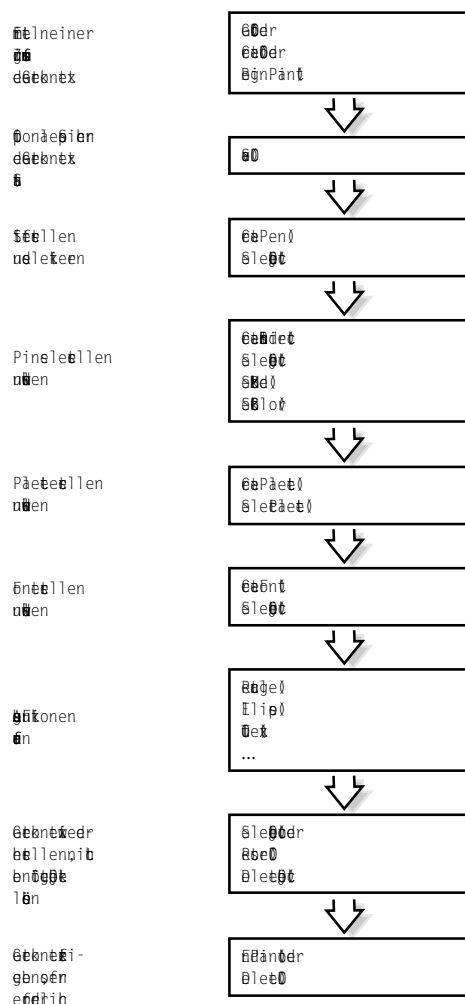


Abbildung 11.10:
Typische
Schritte der GDI-
Ausgabe

Polylinien

Eine Polylinie – eine Linie, die aus mehreren Linienabschnitten besteht – wird durch ein Array definiert, das die Punkte der Linie aufnimmt. Ein Zeiger auf dieses Array wird der Funktion `Polyline` übergeben. `Polyline` benötigt und aktualisiert nicht die gegenwärtige Position. Statt dessen zeichnet die Funktion `PolylineTo` ab der aktuellen Position und aktualisiert diese, so daß sie den letzten Punkt in der Polylinie kennzeichnet.

Die `PolyPolyline`-Funktion kann dazu verwendet werden, mehrere Polylinien mit einem Funktionsaufruf zu zeichnen.

11.6.2 Kurven

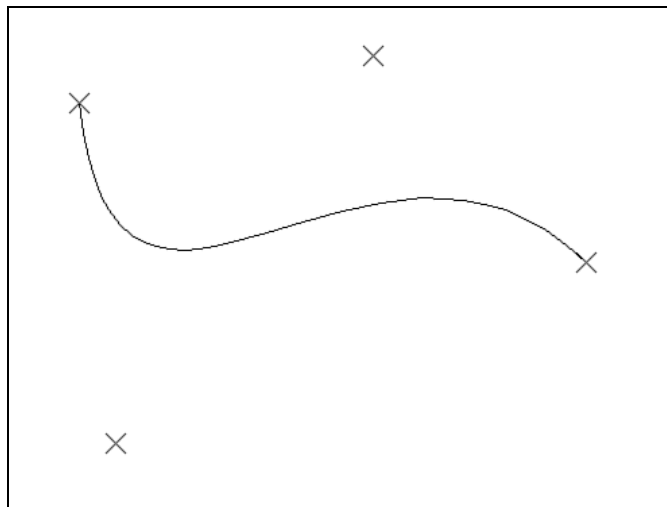
Arc Arc ist die einfachste Funktion zum Zeichnen einer Kurve. Eine mit dieser Funktion gezeichnete Kurve ist ein Abschnitt einer Ellipse. Der Kreisbogen wird mit dem gegenwärtig selektierten Stift gezeichnet. **ArcTo** gleicht der Funktion Arc, aktualisiert jedoch zusätzlich die aktuelle Position.

Bézier-Kurven

Win32-Anwendungen können ebenfalls Bézier-Kurven zeichnen. Bézier-Kurven sind eine kubische Interpolation zwischen zwei Endpunkten, die durch zwei Kontrollpunkte definiert werden. Abbildung 11.11 führt ein Beispiel auf, das eine Bézier-Kurve darstellt.

PolyBezier Die Funktion PolyBezier zeichnet eine oder mehrere Bézier-Kurve(n). Ein Parameter dieser Funktion ist ein Array aus Punkten. Die Punkte definieren die Kurve(n). Der Endpunkt einer Kurve dient als Startpunkt der nächsten Kurve. Die Anzahl der Punkte dieses Arrays muß somit ein Vielfaches des Werts 3, addiert mit dem Wert 1 (der erste Startpunkt) sein (4, 7, 10 usw.).

Abbildung 11.11:
Eine Bézier-
Kurve



Die **PolyBezierTo**-Funktion entspricht der **PolyBezier**-Funktion, aktualisiert jedoch außerdem die gegenwärtige Position.

Win32 verfügt über die Möglichkeit, Linien und Kurven miteinander zu kombinieren. Die Kontur eines Tortendiagramms kann beispielsweise mit Hilfe der Funktion **AngleArc** gezeichnet werden. Komplexe Kombinationen zeichnet die Funktion **PolyDraw**.

11.6.3 Ausgefüllte Figuren

GDI-Zeichenfunktionen können ebenfalls dazu verwendet werden, ausgefüllte Figuren zu erstellen. Die Kontur ausgefüllter Figuren wird, ähnlich wie Linien und Kurven, mit dem aktuellen Stift gezeichnet. Die Innenfläche dieser Figuren wird mit dem derzeit selektierten Pinsel ausgefüllt.

Eine einfache GDI-Figur ist ein Rechteck, das mit einem Aufruf der Funktion `Rectangle` erstellt wird. Varianten dieser Funktion sind `RoundRect` (Rechteck mit abgerundeten Ecken), `FillRect` (zeichnet die Innenfläche eines Rechtecks), `FrameRect` (zeichnet den Rahmen eines Rechtecks) und `InvertRect` (invertiert den Rechteckbereich auf dem Bildschirm).

Rectangle
FillRect
FrameRect
InvertRect

Andere Figuren können mit einer der folgenden Funktionen erstellt werden: `Ellipse`, `Chord`, `Pie` und `Polygon`. Mehrere Polygone können mit einem Aufruf der Funktion `PolyPolygon` gezeichnet werden.

Ellipse
Chord
Pie
Polygon

11.6.4 Bereiche

Bereiche wurden bereits während der Erläuterung des Clippings erwähnt. Das GDI kennt jedoch weitere Anwendungsmöglichkeiten für Bereiche.

Bereiche (aufgeführt in Tabelle 11.1) können ausgefüllt (`FillRgn`, `PaintRgn`), mit einem Rahmen versehen (`FrameRgn`) oder invertiert (`InvertRgn`) werden.

FillRgn
PaintRgn
FrameRgn
InvertRgn

Bereiche können außerdem miteinander kombiniert werden. Verwenden Sie dazu die Funktion `CombineRgn`. Möchten Sie prüfen, ob zwei Bereiche identisch sind, rufen Sie `EqualRgn` auf. Ein Bereich wird über ein bestimmtes Offset mit `OffsetRgn` angezeigt.

Das einen Bereich begrenzende Rechteck erhalten Sie mit einem Aufruf von `GetRgnBox`. Um zu ermitteln, ob ein bestimmter Punkt oder ein Rechteck in einem Bereich vorhanden ist, rufen Sie `PtInRegion` respektive `RectInRegion` auf.

11.6.5 Bitmaps

Bitmap-Objekte wurden bereits weiter oben erörtert. Windows bietet verschiedene Funktionen an, die Bitmap-Objekte kopieren und bearbeiten.

Einzelne Pixel einer Bitmap können mit `SetPixel` gesetzt werden. Die `GetPixel`-Funktion gibt die Farbe des angegebenen Pixels zurück.

SetPixel

ExtFloodFill Ein Bereich in einer Bitmap, der durch die Pixel einer bestimmten Farbe begrenzt ist, kann mit Hilfe der Funktion `ExtFloodFill` ausgefüllt werden.

BitBlt Die Funktion `BitBlt` dient der Bearbeitung von Bitmaps. Diese Funktion kopiert die Bitmap eines Gerätekontextes in einen anderen Gerätekontext. Sie wird außerdem dazu verwendet, Bereiche einer Bitmap aus einem Speichergerätekontext auf den Bildschirm oder umgekehrt zu kopieren. Eine weitere Anwendungsmöglichkeit besteht darin, eine Bitmap an eine andere Position innerhalb desselben Gerätekontextes zu kopieren.

`BitBlt` gibt einen Fehler zurück, wenn die Quell- und Zielgerätekontexte nicht kompatibel miteinander sind. Um einen Speichergerätekontext zu erstellen, der mit dem Bildschirm kompatibel ist, sollten Sie die `CreateCompatibleDC`-Funktion verwenden.

Obwohl `BitBlt` logische Koordinaten nutzt und die erforderliche Skalierung während des Kopierens der Bitmaps vornimmt, wird die Funktion inkorrekt ausgeführt, wenn eine Rotation oder Zuschneidung vorliegt.

`BitBlt` kann nicht nur Bitmaps kopieren, sondern ebenfalls die Pixel der Quell- und Ziel-Bitmap miteinander kombinieren. Die Funktion verwendet dazu verschiedene Pixel-Operationen.

Eine Variante der `BitBlt`-Funktion ist `MaskBlt`. Diese Funktion verwendet während der Kombination zweier Bitmaps eine dritte Bitmap als Maske.

Die Funktion `PatBlt` zeichnet die Ziel-Bitmap mit dem aktuellen Pinsel.

StretchBlt `StretchBlt` kopiert die Quell-Bitmap in die Ziel-Bitmap. Die Quell-Bitmap wird währenddessen vergrößert oder verkleinert, so daß sie das gesamte Ziel-Rechteck einnimmt. Das Einpassen wird mit Hilfe der Funktion `SetStretchBltMode` gesteuert.

PlgBlt Die `PlgBlt`-Funktion kopiert die Quell-Bitmap in ein Ziel-Parallelogramm. Das Parallelogramm wird durch ein Array definiert, das drei Punkte enthält. Diese Punkte repräsentieren drei der Eckpunkte. Der vierte Eckpunkt wird mit der Formel

$$D = B + C - A$$

berechnet.

Geräteunabhängige Bitmaps – DIBs

Die bisher beschriebenen Bitmaps waren einem Gerätekontext zugewiesen und somit geräteabhängig. Windows kann jedoch ebenfalls geräteunabhängige Bitmaps verwalten, die im Speicher oder auf der

Festplatte abgelegt werden. Eine DIB (Device Independent Bitmap – geräteunabhängige Bitmap) wird über die `BITMAPINFO`-Struktur definiert. Anwendungen können eine DIB mit `CreateDIBitmap` erstellen. Die Bits einer geräteunabhängigen Bitmap werden mit der Funktion `SetDIBits` gesetzt. Das Bearbeiten der Farbtabelle einer DIB geschieht mit `SetDIBColorTable`. Die Funktion `SetDIBitsToDevice` kopiert eine DIB auf ein Gerät. `StretchDIBits` kopiert Bits von einem Gerät in eine geräteunabhängige Bitmap.

11.6.6 Pfade

Pfade wurden bereits während der Erläuterung des Clippings erwähnt. Pfade repräsentieren komplexe Figuren, die mit Hilfe mehrerer GDI-Ausgabefunktionen erstellt werden, wie z.B. `Rectangle`, `Ellipse`, `TextOut`, `LineTo`, `PolyBezier` und `Polygon`.

Sie erstellen einen Pfad, indem Sie zunächst die `BeginPath`-Funktion aufrufen, die gewünschten Zeichenoperationen ausführen und schließlich `EndPath` aufrufen. Das Funktionspaar `BeginPath` und `EndPath` wird auch als Pfadgruppe bezeichnet.

Nach einem Aufruf von `EndPath` wird der Pfad in dem Gerätekontext selektiert. Anwendungen können daraufhin eine der folgenden Aufgaben ausführen:

- ➡ Zeichnen der Konturen oder/und Ausfüllen des Pfads (`StrokePath`, `FillPath`, `StrokeAndFillPath`)
- ➡ Verwenden des Pfads für das Clipping (`SelectClipPath`)
- ➡ Konvertieren des Pfads in einen Bereich (`PathToRegion`)
- ➡ Bearbeiten des Pfads (`GetPath`, `FlattenPath`, `WidenPath`)

11.6.7 Textausgabe

Die überwiegend verwendete GDI-Textausgabefunktion ist `TextOut`. Diese Funktion gibt Text in der aktuellen Schriftart an den angegebenen Koordinaten aus. Die Funktion `TabbedTextOut` ist eine Variante von `TextOut`, die Text an den angegebenen Tabstop-Positionen darstellen kann. Ein Aufruf der Funktion `PolyTextOut` führt zur Ausgabe mehrerer Zeichenfolgen. Die `ExtTextOut`-Funktion stellt den Text in einem Rechteck dar, das ausgefüllt oder ausgeschnitten werden kann.

Die Funktionen `DrawText` und `DrawTextExt` geben formatierten Text in einem Rechteck aus. Der Text wird mit bestimmten Attributen formatiert, die über die Funktionen `SetTextColor`, `SetTextAlign`, `SetBkColor`, `SetBkMode`, `SetTextCharacterExtra` und `SetTextJustification` gesetzt

werden. Eine Anwendung kann die Größe eines Textblocks mit `GetTabbedTextExtent` oder `GetTextExtentPoint32` ermitteln, bevor dieser ausgegeben wird.

```
GetClientRect(hwnd, &clientRect);
DPtoLP(hDC, (LPPPOINT)&clientRect, 2);
DrawText(hDC, "Hello, World!", -1, &clientRect,
        DT_CENTER | DT_VCENTER | DT_SINGLELINE);
```

11.7 Hinweise zum Drucken

Das GDI ist auch für die Ausgabe einer Hardcopy auf Druckern, Plottern und anderen Ausgabegeräten verantwortlich. Die meisten Anwendungen müssen keine detaillierten Informationen darüber besitzen, wie der Druckvorgang ausgeführt wird. Die Ausgabe auf einem Ausgabegerät unterscheidet sich nicht von der Ausgabe auf den Bildschirm. Führen Sie dazu die Standardfunktionsaufrufe des GDIs auf dem Drucker-Gerätekontext aus. Bisweilen müssen die physikalischen Eigenschaften der Ausgabeseite und die Einschränkungen des Geräts (ein Plotter unterstützt beispielsweise keine Bitmap-Operationen) berücksichtigt werden. WYSIWYG-Anwendungen können häufig den Programmcode zum Drucken verwenden, der auch die Ausgabe auf den Bildschirm durchführt. Dazu sind nur geringfügige Änderungen an dem Programm notwendig.

Das Drucken geschieht mit Hilfe verschiedener Windows-Komponenten.

- ➔ Die wichtigste dieser Komponenten ist der Druck-Spooler, der den Druckvorgang verwaltet.
- ➔ Der Druckprozessor konvertiert die Druckjobs des Spoolers in Aufrufe an den Gerätetreiber.
- ➔ Der Gerätetreiber generiert daraufhin die Ausgabe, die von dem Drucker bearbeitet wird.
- ➔ Der Port-Monitor übermittelt die Geräteanweisungen über eine bestimmte Schnittstelle oder Netzwerkverbindung zu dem physikalischen Gerät.

Einige Win32-Funktionen stellen Druckjobs in die Druckerwarteschlange, ermitteln Informationen über Druckjobs und Drucker und steuern den Druckvorgang.

Windows-3.1-Anwendungen haben häufig Drucker-Escape-Codes verwendet, um bestimmte Aufgaben auszuführen. Diese wurden durch neue Win32-Funktionen ersetzt. Sie sollten die Escape-Funktionen nicht mehr in Ihrer Anwendung nutzen, um den Drucker zu steuern.

11.8 Zusammenfassung

Das Windows-GDI stellt geräteunabhängige Funktionen zur Verfügung, die Anwendungen zu grafischen Ausgaben auf allen zu Windows kompatiblen Ausgabegeräten verwenden können. Das GDI erzeugt Ausgaben auf dem Bildschirm, auf Druckern, Plottern, Fax-Modems und anderen speziellen grafischen Geräten.

Jede grafische Ausgabe wird an Gerätekontexte weitergeleitet. Ein Gerätekontext ist eine Beschreibung des Ausgabegeräts inklusive dessen Eigenschaften und Parameter und agiert als eine Schnittstelle zwischen geräteunabhängigen GDI-Routinen und der Gerätetreiber-Software. Der Gerätekontext ist somit die »Zeichenfläche«, auf der GDI-Zeichenfunktionen ausgeführt werden.

Das GDI gibt Grafiken mit Hilfe einiger Werkzeuge aus:

- ➔ Stifte zeichnen Linien oder Konturen von Figuren.
- ➔ Pinsel füllen die Innenflächen der Figuren.
- ➔ Schriftarten werden zur Darstellung von Text verwendet.
- ➔ Bitmaps sind rechteckige Pixelbereiche, in die über Speicher-Gerätekontexte gezeichnet werden kann. Bitmaps können in einem Gerätekontext bearbeitet oder von einem Gerätekontext in einen anderen kopiert werden.
- ➔ Paletten sind logische Farbsammlungen, die das GDI möglichst originalgetreu darzustellen versucht, indem es die Farbeinstellungen des anzeigenden Geräts konfiguriert.
- ➔ Bereiche sind reguläre oder nichtreguläre Figuren, die beispielsweise für das Clipping verwendet werden können.

Anwendungen, die mit Clipping arbeiten, müssen die Ausgabe nicht auf den sichtbaren Bereich ihrer Fenster begrenzen. Clipping-Funktionen können ebenfalls grafische Effekte erzeugen.

Die Zeichenwerkzeuge Clipping definieren, wie das GDI Zeichenoperationen ausführt. Verschiedene grafische Funktionen werden zum Zeichnen von Linien, Kurven, ausgefüllten Figuren, Text und zur Bearbeitung von Bitmaps verwendet.

Das GDI verfügt über weitere Funktionen, die den Ausdruck, den Spooler und den Drucker verwalten. Diese Features sind jedoch erst dann sinnvoll, wenn eine Anwendung den Druckvorgang explizit steuern muß. Die meisten WYSIWYG-Anwendungen modifizieren lediglich geringfügig den Programmcode für die Ausgabe auf dem Bildschirm, um zu drucken.

Threads und Prozesse

Kapitel

12

Wie jede sich entwickelnde Umgebung präsentiert auch Windows ein Konglomerat aus Altem und Neuem. Dies wird besonders in einem Vergleich der verschiedenen Win32-Plattformen im Bereich Multitasking deutlich.

- ➡ **Das Alte:** die kooperative Multitasking-Umgebung der 16-Bit-Version von Windows. Die Fehler und Einschränkungen dieses Betriebssystems blieben auch mit Win32s bestehen. Wenngleich diese Erweiterung eine beinahe vollständige Implementierung der Win32-Programmierschnittstelle bildete, konnte sie das zugrundeliegende Betriebssystem nicht verändern oder dessen Einschränkungen aufheben.
- ➡ **Das Neue:** das Multithread-Betriebssystem Windows NT. Als ein Betriebssystem, das von Grund auf neu entwickelt wurde, bietet Windows NT eine äußerst stabile Multitasking-Umgebung für sehr zuverlässige Anwendungen (wie z.B. die Server eines Unternehmens).
- ➡ **Der Mix:** Windows 95/98. Das Ziel der Entwickler bestand darin, sowohl die neuen Möglichkeiten zu implementieren, als auch ein System zu schaffen, das kompatibel zum alten 16-Bit-Windows sein sollte. Das Ergebnis ist eine bemerkenswerte Kombination: Windows 95/98 verfügt über eine sehr stabile Multitasking-Umgebung und bietet gleichzeitig eine Kompatibilität zu 16-Bit-Anwendungen. Hinsichtlich dieser stabilen Multitasking-Umgebung werden einige Anwender überrascht sein, wenn Windows 95/98 bisweilen aufgrund einer inkorrekten 16-Bit-Anwendung schneller »abstürzt« als Windows 3.1. (Windows 95/98 bewältigt derartige Situationen jedoch sehr viel besser als die Vorgängerversion.)

12.1 Multitasking in der Win32-Umgebung

Das Multitasking wird für jede der drei Windows-Umgebungen separat erläutert, da sehr große Unterschiede bestehen. Richten Sie Ihre Aufmerksamkeit jedoch zunächst auf die grundlegenden Konzepte des Multitasking unter Windows.

12.1.1 Multitasking-Konzepte

Multitasking ist die Fähigkeit eines Betriebssystems, mehrere Anwendungen gleichzeitig zu laden und auszuführen. Ein Multitasking-Betriebssystem ist stabil und zuverlässig, wenn es die Anwendungen erfolgreich voneinander trennt und diese glauben läßt, allein den Computer und dessen Ressourcen nutzen zu können. Ein gutes Multitasking-Betriebssystem schirmt die Anwendungen außerdem von den Fehlern anderer Anwendungen ab. Führt eine Anwendung beispielsweise eine Überprüfung der Array-Begrenzungen inkorrekt aus, sollte das Multitasking-Betriebssystem darauf achten, daß diese Anwendung nicht den Speicher einer anderen Anwendung überschreibt.

Hardware-Unterstützung ist erforderlich

Multitasking-Betriebssysteme sind in einem hohen Maße auf die System-Hardware angewiesen, um die genannten Fähigkeiten implementieren zu können. Ohne die Unterstützung einer Speicherverwaltungseinheit, die bei einem Zugriff auf eine unzulässige Speicheradresse einen Interrupt auslöst, würde das Betriebssystem beispielsweise nichts von solch einem Zugriff erfahren und müßte jede einzelne Anweisung des Anwendungscodes überprüfen. Diese Vorgehensweise wäre sehr zeitaufwendig und außerdem nicht realisierbar.

Ein weiterer wichtiger Aspekt des Multitasking ist die PROZESSZUTEILUNG. Da die meisten Prozessoren lediglich einen Anweisungsstrom gleichzeitig ausführen können, wäre Multitasking nicht möglich, gäbe es nicht eine besondere Technik des Umschaltens zwischen Kontexten. Ein KONTEXTSCHALTER, der von einem Ereignis ausgelöst wird (wie z.B. von dem Interrupt eines Zeitgebers oder von einem Aufruf einer bestimmten Funktion), speichert den Kontext des Prozessors (Befehlszeiger, Stack-Zeiger, Registerinhalte) für das derzeit ausgeführte Programm und lädt den Kontext eines anderen Programms.

Ein anderer Aspekt des Multitasking ist das Vermögen des Betriebssystems, Zugriff auf verschiedene Systemressourcen zu gewähren (z.B. das Dateisystem oder den Bildschirm), Abstürze zu vermeiden und Me-

chanismen zur Verfügung zu stellen, über die konkurrierende Anwendungen miteinander kommunizieren und ihre Ausführung synchronisieren können.

In welchem Maß Multitasking zur Verfügung gestellt wird, ist von dem verwendeten Betriebssystem abhängig. Herkömmliche Mainframe-Betriebssysteme verfügen seit langer Zeit über ein stabiles Multitasking. Multitasking auf Desktop-Computern ist jedoch eine relativ neue Besonderheit, da diese Rechner nur sukzessive die Leistung erhielten, um mehrere Aufgaben gleichzeitig ausführen zu können. (Viele Programmierer waren darüber erstaunt, daß sogar MS-DOS eine grundlegende Unterstützung des Multitasking zur Verfügung stellte, das für TSR-Anwendungen notwendig ist. TSR ist die Abkürzung für Terminate and Stay Resident.)

Die Unterschiede zwischen dem Multitasking der verschiedenen Win32-Umgebungen bestehen vorwiegend in dem verwendeten Zuteilungsmechanismus.

- ➡ In einer KOOPERATIVEN MULTITASKING-UMGEBUNG (die auch häufig als nicht präemptiv bezeichnet wird) ist das Betriebssystem explizit darauf angewiesen, daß die Anwendungen die Steuerung des Prozessors abgeben, indem sie unterschiedliche Betriebssystemfunktionen aufrufen. Das Umschalten zwischen Kontexten geschieht somit an exakt definierten Punkten während der Programmausführung.
- ➡ In einer PRÄEMPTIVEN MULTITASKING-UMGEBUNG kann das Betriebssystem die Ausführung einer Anwendung zu jeder Zeit unterbrechen. Dies geschieht gewöhnlich, wenn das Betriebssystem auf ein Hardware-Ereignis reagieren muß, z. B. auf einen Interrupt des Zeitgebers. Die Programmausführung kann somit nicht nur an einem vordefinierten, sondern an jedem Punkt unterbrochen werden. Das System wird dadurch natürlich äußerst komplex. Besonders in präemptiven Multitasking-Umgebungen sind die Möglichkeiten des *Wiedereintritts* sehr unterschiedlich. Ein Programm wird möglicherweise unterbrochen, während es eine Systemfunktion ausführt. Während dieser Unterbrechung kann eine andere Anwendung dieselbe Funktion aufrufen oder die Ausführung dieser Funktion fortsetzen, bevor der Aufruf des ersten Programms beendet ist.

Ein Stichwort, das häufig im Zusammenhang mit Multitasking und Windows genannt wird, ist THREADS. Threads können wie folgt beschrieben werden: Während Multitasking die Möglichkeit bietet, mehrere Programme gleichzeitig ausführen zu lassen, erlauben Threads mehrere Pfade der Ausführung innerhalb desselben Pro-

Threads

gramms. Dieser Mechanismus offeriert dem Programmierer ein leistungsfähiges Werkzeug. Der Preis: Probleme, die bisher die Entwickler von Betriebssystemen betrafen, wie z.B. der Wiederanlauf und die Synchronisation der Prozesse, müssen nun ebenfalls von dem Anwendungsentwickler berücksichtigt werden.

12.1.2 Kooperatives Multitasking unter Windows 3.x

Obwohl Visual C++ Windows 3.1 nicht mehr unterstützt, auch nicht die Entwicklung von Win32s-Anwendungen, ist das kooperative Windows-3.1-Modell in Windows 95/98 und in einem geringen Maße in Windows NT enthalten. Ein detailliertes Verständnis des kooperativen Multitasking kann Ihnen helfen, stabile Anwendungen zu programmieren. Eine inkorrekt programmierte 32-Bit-Anwendung kann nicht das gesamte Betriebssystem zum Absturz bringen, jedoch ein ungewohntes Verhalten aufweisen oder nicht mehr reagieren, wenn die Regeln des kooperativen Multitasking nicht eingehalten werden.

Unter Windows 3.1 müssen Anwendungen regelmäßig eine der folgenden Funktionen aufrufen, um die Kontrolle an das Betriebssystem abzugeben: `GetMessage`, `PeekMessage` (ohne das Flag `PM_NOYIELD`) und `Yield`.

- ➔ `Yield` übergibt die Kontrolle dem Betriebssystem und ermöglicht somit die Ausführung anderer Aufgaben. Diese Funktion ist beendet, wenn das Betriebssystem die Kontrolle an das aufrufende Programm zurückgibt.
- ➔ `GetMessage` und `PeekMessage` geben nicht nur die Steuerung des Prozessors ab, sondern prüfen außerdem, ob bestimmte Nachrichten in der Nachrichtenwarteschlange einer Anwendung enthalten sind. Diese Funktionen befinden sich im Kern jeder Nachrichtenschleife einer Anwendung. Sie müssen aufgerufen werden, damit die Anwendung weiterhin auf alle Ereignisse reagiert.

Das kooperative Multitasking mag ein Relikt vergangener Tage für 32-Bit-Entwickler sein, die Notwendigkeit zur Überprüfung und Bearbeitung verschiedener Nachrichten besteht jedoch auch künftig.

12.1.3 Präemptives Multitasking unter Windows NT

Zwischen Windows 3.1 und Windows NT bestehen bedeutende Unterschiede. Dies wird besonders an dem stabilen Multitasking von Windows NT deutlich. Abgestürzte Anwendungen, eine nicht mehr reagierende Tastatur sowie vergebliche Versuche, das System wieder instand zu setzen, gehören unter Windows NT der Vergangenheit an.

Dieses Programm reagiert immer und bietet in jeder Situation eine Möglichkeit, ein abgestürztes Programm zu verlassen.

Windows NT bietet präemptives Multitasking für konkurrierende 32-Bit-Prozesse. 16-Bit-Prozesse werden gesondert behandelt. Solche Prozesse werden von Windows NT wie ein einzelner Prozeß bearbeitet (ein WOW-Prozeß = Windows On Windows). Windows NT führt 16-Bit-Prozesse seit der Version 3.5 nicht mehr in einem separaten Speicherbereich aus, sondern startet statt dessen einen gesonderten WOW-Prozeß. 16-Bit-Anwendungen, die sich einen WOW-Prozeß teilen, müssen die Regeln des kooperativen Multitaskings einhalten. Stürzt eine 16-Bit-Anwendung ab, reagieren auch alle anderen 16-Bit-Prozesse nicht mehr, die denselben WOW-Prozeß verwenden. Andere Prozesse, auch andere WOW-Prozesse, sind davon nicht betroffen.

Können Sie nun, da es präemptives Multitasking unter Windows NT gibt, auf Ihr Wissen über nichtkooperativen Programmcode verzichten? Nein. Nachfolgend finden Sie den Grund für diese Antwort.

Wenngleich Windows NT einer nichtkooperativen 32-Bit-Anwendung die Kontrolle entziehen kann, um andere Anwendungen auszuführen, besitzt das Betriebssystem keine Möglichkeit, Nachrichten zu bearbeiten, die an eine nichtkooperative Anwendung gerichtet sind. Eine Anwendung, die nicht imstande ist, ihre Nachrichtenwarteschlange auszulesen, reagiert daher nicht mehr und erscheint fehlerhaft. Der Anwender kann in diesem Fall nicht mit der Anwendung interagieren. Ein Klick auf das Fenster der Anwendung ordnet diese nicht vor allen anderen Anwendungsfenstern an. Außerdem zeichnet die Anwendung Bereiche ihres Fensters nicht mehr neu, wenn diese von anderen Fenstern überdeckt wurden.

Kooperation unter Windows NT

Um diesen Zustand zu vermeiden, sollte eine Anwendung jede Anstrengung unternehmen, um die Nachrichtenwarteschlange auszulesen und die darin enthaltenen Nachrichten auch dann weiterzuleiten, wenn zeitintensive Aufgaben ausgeführt werden. Doch selbst wenn dies nicht möglich ist, bedroht eine abgestürzte Anwendung nicht das gesamte System, sondern dient lediglich als ein Beispiel für eine äußerst »anwenderunfreundliche« Anwendung.

Unter Windows NT können zeitintensive Prozesse sehr einfach implementiert werden. Windows NT ist im Gegensatz zu seinem 16-Bit-Vorgänger ein Multithread-Betriebssystem.

Ein Windows-NT-Programm kann neue Threads sehr einfach erzeugen. Soll beispielsweise eine äußerst lange Berechnung vorgenommen werden, kann diese Aufgabe an einen sekundären Thread delegiert werden, während der primäre Thread weiterhin die Nachrichten bear-

beitet. Ein sekundärer Thread kann sogar Benutzerschnittstellenfunktionen ausführen. Ein Beispiel hierfür ist eine Anwendung, deren primärer Thread Nachrichten bearbeitet, die an das Hauptfenster der Anwendung gerichtet sind, während die Nachrichten für einen Dialog dieser Anwendung von einem sekundären Thread bearbeitet werden. (Threads, die Fenster besitzen und Nachrichten bearbeiten, werden in der MFC-Terminologie als Benutzerschnittstellen-Threads bezeichnet, während alle anderen Threads Worker-Threads genannt werden.)

12.1.4 Windows 95/98: Ein Mix unterschiedlicher Multitasking-Tricks

Windows 95/98 kombiniert die besten Features der Betriebssysteme Windows 3.1 und Windows NT. Windows 95/98 bietet ein Windows-NT-ähnliches Multitasking und Multithreading. Windows 95/98 ist möglicherweise progressiver, da der Code dieses Betriebssystems umfassender für die Intel-Prozessorfamilie optimiert wurde, als der portierbare Programmcode von Windows NT. Seitdem jedoch Windows NT 4.0 erhältlich ist, stellt dieses Betriebssystem eine große Konkurrenz für Windows 95/98 dar. Windows 95/98 wiederum bietet eine erhebliche Kompatibilität zu DOS- und 16-Bit-Windows-Anwendungen. All diese Fähigkeiten werden von einem Betriebssystem zur Verfügung gestellt, das nur geringfügig mehr Ressourcen beansprucht als sein Vorgänger.

Diese Kompatibilität wurde teilweise dadurch erreicht, daß sehr viel Programmcode von Windows 3.1 in Windows 95/98 übernommen wurde. Windows 95/98 ist dennoch ein 32-Bit-Betriebssystem. Ein Nebeneffekt dieses Umstands ist der, daß einige Parameter, die unter Windows NT als 32-Bit-Werte übergeben werden, unter Windows 95/98 auf 16 Bit beschränkt sind (besonders erwähnenswert sind grafische Koordinaten). Auch das Multitasking unter Windows 95/98 ist davon betroffen.

Einige Abschnitte des übernommenen Windows-3.1-Programmcodes wurden nicht unter Berücksichtigung des Wiedereintritts entwickelt. Da 16-Bit-Anwendungen kooperatives Multitasking verwenden, besteht keine Möglichkeit, die Programmausführung während eines Systemaufrufs zu unterbrechen. Das Entwickeln von Mechanismen, die wiederholte Aufrufe von Systemfunktionen ermöglichen, während ein vorheriger Aufruf vorübergehend unterbrochen wird, war somit nicht notwendig.

Da Windows-95/98-Prozesse zu jedem Zeitpunkt unterbrochen werden können, hatte Microsoft zwei Möglichkeiten zur Auswahl. Die erste Möglichkeit bestand darin, die Systemaufrufe von Windows 3.1

vollständig umzuschreiben. Abgesehen von dem hohen Arbeitsaufwand hätte diese Absicht zu einem Verlust der Vorteile des importierten Windows-3.1-Programmcodes geführt. Die Abwärtskompatibilität wäre verlorengegangen. Das Modifizieren der Funktionen hätte ein neues Betriebssystem hervorgebracht, was bereits einmal geschehen ist, wie Windows NT beweist.

Die andere, einfachere Lösung schützt das System, während 16-Bit-Anwendungen ausgeführt werden. Das bedeutet, daß während eine Anwendung 16-Bit-Code unter Windows 95/98 verwendet, alle anderen Anwendungen davon abgehalten werden, ebenfalls 16-Bit-Programmcodes auszuführen.

Diese Vorgehensweise führt dazu, daß 16-Bit-Anwendungen immer im 16-Bit-Modus ausgeführt werden. In der Zeit, in der eine 16-Bit-Anwendung die Steuerung des Prozessors übernimmt, kann somit keine andere Anwendung 16-Bit-Code ausführen.

Eine nichtkooperative 16-Bit-Anwendung (eine Anwendung, die nicht die Kontrolle an das Betriebssystem zurückgegeben hat, so daß das Betriebssystem diese Kontrolle nicht an eine 32-Bit-Anwendung weitergeben kann) kann daher, wie unter Windows 3.1, das gesamte System abstürzen lassen.

Windows 95/98 bewältigt solche Situationen jedoch sehr gut. Das Betriebssystem verfügt beispielsweise über die Möglichkeit, den Prozeß, der zum Absturz führte, zu beenden und die von diesem Prozeß vorgenommenen Einstellungen rückgängig zu machen, ohne die Stabilität und Reservierung von Ressourcen zu gefährden, was bisweilen unter Windows 3.1 geschieht.

12.2 Programmierung mit Prozessen und Threads

Die Win32-API enthält viele Funktionen für den Zugriff auf die Multitasking- und Multithreading-Features der 32-Bit-Version von Windows. Diese Funktionen ersetzen in einigen Fällen herkömmliche Unix-, C- oder MS-DOS-Bibliotheksfunktionen. Andere Funktionen repräsentieren eine neue Funktionalität. Eine weitere Funktionsgruppe (z.B. die `yield`-Funktionen) ist dem Windows-3.1-Programmierer bekannt.

In diesem Kapitel lernen Sie einige Programmiertechniken zum Multitasking kennen.

12.2.1 Kooperatives Multitasking: Abgabe der Kontrolle in der Nachrichtenschleife

Da sehr viel Programmcode von Windows 3.1 in Windows 95/98 übernommen wurde, sind sich die Techniken für das Multitasking und die Bearbeitung von Nachrichten in der Windows-Programmierung sehr ähnlich. Listing 12.1 zeigt eine einfache Windows-Nachrichtenschleife, die einen Aufruf der Funktion `GetMessage` enthält. In Windows 3.1 gewährleistet diese Schleife, daß die Nachrichtenwarteschlange der Anwendung bearbeitet wird und andere Anwendungen die Kontrolle über das System erhalten. Unter Windows 95/98 oder Windows NT entfällt die zuletzt genannte Aufgabe.

Listing 12.1:
Eine einfache
Nachrichten-
schleife

```
int WINAPI WinMain(...)
{
    MSG msg;
    ...
    // Hier erfolgt die Initialisierung der Anwendung
    ...

    // HauptNachrichtenschleife
    while (GetMessage(&msg, NULL, 0, 0)) // Dieser Aufruf
                                        // übergibt die
                                        // Kontrolle!
    {
        // Die Nachrichtenweiterleitung erfolgt hier
        ...
    }
}
```

12.2.2 Bearbeiten von Nachrichten während der Ausführung langer Prozesse

Wenngleich in einer 32-Bit-Umgebung die kooperative Übergabe der Kontrolle nicht erforderlich ist, sollte dennoch die Bearbeitung der Nachrichten fortgesetzt werden. Eine Anwendung, die die Steuerung des Prozessors nicht abgibt, stellt inzwischen kein Problem mehr unter Windows 95/98 dar. Der Programmierer sollte dennoch darauf achten, daß seine Anwendung auf alle Ereignisse korrekt reagiert. Dies ist besonders dann wichtig, wenn die Anwendung eine zeitintensive Aufgabe ausführen muß, z.B. eine komplexe Berechnung oder ein Ausdruck.

Das Beispielprogramm in Listing 12.2 (Ressourcendatei) und Listing 12.3 (Quelldatei) demonstrieren eine einfache Technik. Auch dieses Programm kann über die Kommandozeile mit der folgenden Anweisung kompiliert werden:

```
rc loop.rc
CL LOOP.C LOOP.RES USER32.LIB
```

Alternativ dazu können Sie ein Visual-C++-Projekt erstellen und diesem die Dateien *LOOP.C* und *LOOP.RC* hinzufügen, um das Projekt aus dem Visual Studio heraus zu kompilieren.

```
#include "windows.h"

DlgBox DIALOG 20, 20, 90, 64
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "LOOP"
BEGIN
    DEFPUSHBUTTON "CANCEL" IDCANCEL, 29, 44, 32, 14, WS_GROUP
    CTEXT "Iterating" -1, 0, 8, 90, 8
    CTEXT "0" 1000, 0, 23, 90, 8
END
```

Listing 12.2:
Bearbeitung der
Nachrichtens-
schleife
(LOOP.RC)

```
#include <windows.h>

HINSTANCE hInstance;
BOOL bDoAbort;
HWND hwnd;

BOOL CALLBACK DlgProc(HWND hwndDlg, UINT uMsg,
                      WPARAM wParam, LPARAM lParam)
{
    if (uMsg == WM_COMMAND && LOWORD(wParam) == IDCANCEL)
    {
        EnableWindow(hwnd, TRUE);
        DestroyWindow(hwndDlg);
        return (bDoAbort = TRUE);
    }
    return FALSE;
}

void DoIterate(HWND hwndDlg)
{
    MSG msg;
    int i;
    char buf[18];

    i = 0;
    while (!bDoAbort)
    {
        SetWindowText(GetDlgItem(hwndDlg, 1000),
                      _itoa(i++, buf, 10));
        if (i % 100 == 0)
            while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
                DispatchMessage(&msg);
    }
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    HWND hwndDlg;

    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            hwndDlg =
                CreateDialog(hInstance, "DlgBox", hwnd, DlgProc);
            ShowWindow(hwndDlg, SW_NORMAL);
            UpdateWindow(hwndDlg);
            EnableWindow(hwnd, FALSE);
            bDoAbort = FALSE;
            break;
    }
}
```

Listing 12.3:
Bearbeiten der
Nachrichtens-
schleife (LOOP.C)


```

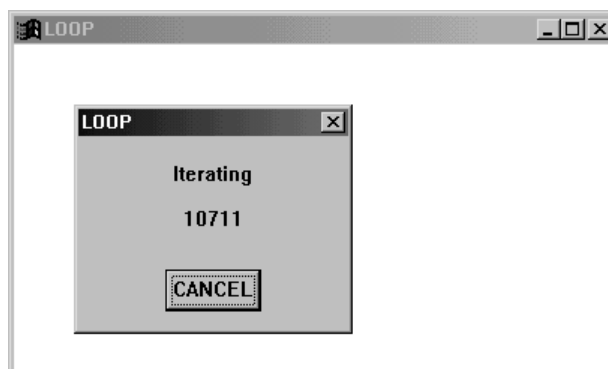
        DoIterate(hwndDlg);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hThisInstance,
                  HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    WNDCLASS wndClass;

    hInstance = hThisInstance;
    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "LOOP";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("LOOP", "LOOP",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                       NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Abbildung 12.1:
Kooperierende
Anwendung



In diesem Programm wird eine Schleife mit einem zeitintensiven Prozeß gestartet, wenn der Anwender in den Client-Bereich des Hauptfensters der Anwendung klickt. Die Bearbeitung in der `DoIterate`-Funktion ist nicht besonders komplex. Diese Funktion erhöht lediglich den Wert der Variablen `i` und zeigt so lange das Ergebnis an, bis der Anwender die Schleife unterbricht.

Bevor die Iteration gestartet wird, ruft die Anwendung einen nicht-modalen Dialog auf. Dieser ermöglicht die Interaktion mit dem Hauptfenster durch einen Aufruf der Funktion `EnableWindow`. Diese Vorgehensweise hat denselben Effekt wie die Verwendung eines modalen Dialogs. Wir müssen jedoch nicht `DialogBox` aufrufen und behalten somit die Kontrolle, während der Dialog angezeigt wird.

Innerhalb der Iterationsschleife wird die Funktion `PeekMessage` wiederholt aufgerufen. Auf diese Weise wird gewährleistet, daß die Anwendung die Steuerung abgibt und der Dialog, über den die Iteration abgebrochen werden kann, auf die Anwenderschnittstellenereignisse reagiert.

GetMessage versus PeekMessage

Worin aber besteht der Unterschied zwischen `GetMessage` und `PeekMessage`? Wann verwenden Sie welche Funktion? Hier die Antwort:

- ➡ Wenn Sie `PeekMessage` aufrufen, teilen Sie dem Betriebssystem mit, daß Sie die Nachrichtenwarteschlange auslesen und gleichzeitig die Kontrolle über den Prozessor zurückbekommen möchten, um Ihr Programm fortsetzen zu können.
- ➡ `GetMessage` wiederum informiert das Betriebssystem, daß bis zur nächsten Nachricht keine Aufgaben ausgeführt werden müssen. Mit dieser Funktion gewährleisten Sie somit, daß andere Prozesse die CPU optimal nutzen können und Ihr Programm nicht unnötige Prozessorzeit mit einem Aufruf von `PeekMessage` verschwendet.

Ihr Motto sollte lauten: Nur weil das Betriebssystem präemptiv ist, sollten meine Anwendungen nicht aufhören, kooperativ zu sein.

Der Aufruf von `PeekMessage` sollte lediglich dann verwendet werden, wenn die Anwendung im Hintergrund arbeitet. Rufen Sie `PeekMessage` anstelle von `GetMessage` auf, wird nicht nur Prozessorzeit verschwendet. Dieser Aufruf führt außerdem dazu, daß Windows in dieser Zeit keine Optimierung des virtuellen Speichers und kein Power-Management für batteriebetriebene Systeme vornehmen kann. `PeekMessage` sollte deshalb niemals in einer allgemeinen Nachrichtenschleife eingesetzt werden.



12.2.3 Verwenden eines sekundären Threads

Die soeben vorgestellte Technik kann in Programmen für Win32-Plattformen verwendet werden. Sie ist jedoch umständlich zu realisieren. Für zeitintensive Berechnungen sollte ein sekundärer Thread verwendet werden, in dem diese ohne Unterbrechungen mit Aufrufen von `PeekMessage` durchgeführt werden können. Das Beispiel in Listing 12.4 wird mit der vorherigen Ressourcendatei und Anweisung kompiliert.

Listing 12.4:
Bearbeitung in
einem sekundären Thread
(LOOP.C)

```
#include <windows.h>

HINSTANCE hInstance;
volatile BOOL bDoAbort;
HWND hwnd;

BOOL CALLBACK DlgProc(HWND hwndDlg, UINT uMsg,
                      WPARAM wParam, LPARAM lParam)
{
    if (uMsg == WM_COMMAND && LOWORD(wParam) == IDCANCEL)
    {
        EnableWindow(hwnd, TRUE);
        DestroyWindow(hwndDlg);
        return (bDoAbort = TRUE);
    }
    return FALSE;
}

DWORD WINAPI DoIterate(LPVOID hwndDlg)
{
    int i;
    char buf[18];

    i = 0;
    while (!bDoAbort)
    {
        SetWindowText(GetDlgItem((HWND)hwndDlg, 1000),
                      _itoa(i++, buf, 10));
    }
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    HWND hwndDlg;
    DWORD dwThreadId;

    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            hwndDlg =
                CreateDialog(hInstance, "DlgBox", hwnd, DlgProc);
            ShowWindow(hwndDlg, SW_NORMAL);
            UpdateWindow(hwndDlg);
            EnableWindow(hwnd, FALSE);
            bDoAbort = FALSE;
            CreateThread(NULL, 0, DoIterate, (LPDWORD)hwndDlg, 0,
                          &dwThreadId);

            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
}
```

```

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hThisInstance,
                  HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    WNDCLASS wndClass;

    hInstance = hThisInstance;
    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "LOOP";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("LOOP", "LOOP",
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                      NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Der wesentliche Unterschied zwischen den beiden Versionen steht in *LOOP.C*. Die dort innerhalb der Funktion *DoIterate* verwendete Iterationsschleife ruft nicht wie bisher *PeekMessage* und *DispatchMessage* auf. Diese Vorgehensweise ist auch nicht notwendig. Die *DoIterate*-Funktion wird nun aus einem sekundären Thread heraus aufgerufen, der mit *CreateThread* in der Funktion *WndProc* erzeugt wird.

Der primäre Thread der Anwendung setzt die Ausführung fort, nachdem der sekundäre Thread erstellt wurde und dieser Nachrichten an die primäre Nachrichtenschleife in *WinMain* zurückgibt. Die Nachrichtenschleife leitet auch Nachrichten an den Dialog weiter.

Einsatz von volatile-Variablen

Von besonderem Interesse ist die veränderte Deklaration der globalen Variable *bDoAbort*. Über diese Variable wird der sekundäre Thread benachrichtigt, daß er die Ausführung unterbrechen soll. Der Wert der Variablen wird in dem primären Thread gesetzt, wenn der Anwender den Dialog schließt. Der optimierende Compiler wird über diesen Vorgang nicht informiert, so daß die Struktur

```
while (!bDoAbort)
{
    ...
}
```

eventuell in einer Weise optimiert wird, die nicht dem Wert entspricht, den `bDoAbort` aus dem Speicher gelesen hat. Doch warum sollte dies geschehen? Nichts innerhalb der `while`-Schleife modifiziert den Wert der Variablen, so daß der Compiler diesen in einem Register aufbewahren kann. Verändert ein anderer Thread den Wert im Speicher, würde der aktuelle Thread diese Veränderung nicht beachten.

Die Lösung des Problems ist das C-Schlüsselwort `volatile`. Das Deklarieren einer Variablen als `volatile` teilt dem Compiler mit, daß der Wert einer solchen Variablen, unabhängig von den Optimierungsrichtlinien des Compilers, immer dann im Speicher abgelegt werden soll, wenn er modifiziert wird. Er wird außerdem jedesmal erneut aus dem Speicher ausgelesen, wenn ein Zugriff darauf stattfindet. Wir gewährleisten daher, daß, wenn der primäre Thread `bDoAbort` auf einen neuen Wert gesetzt wird, der sekundäre Thread diese Veränderung nachvollziehen kann.

12.2.4 Thread-Objekte

Thread erzeugen Unser zweites `LOOP.C`-Beispiel enthält einen Aufruf der Funktion `CreateThread`. Daraufhin wird ein sekundärer Thread erzeugt. Der von dieser Funktion zurückgegebene Wert, der in dem Beispiel unberücksichtigt bleibt, ist der Handle des neuen `THREAD-OBJEKTS`.

Das Thread-Objekt enthält die Eigenschaften des Threads, einschließlich der Sicherheitsattribute, Priorität und anderen Informationen. Funktionen zur Bearbeitung von Threads greifen auf die Thread-Objekte über deren Handle zu, die beispielsweise von `CreateThread` zurückgegeben werden.

Thread beenden Der sekundäre Thread unseres Beispiels verwendet einen einfachen Mechanismus zum Beenden. Sobald `CreateThread` ausgeführt wurde, wird der Thread automatisch beendet, da die Thread-Funktion `ExitThread` implizit aufruft.



Das Thread-Objekt bleibt auch dann bestehen, wenn ein Thread beendet wurde. Dieser Zustand ändert sich erst dann, wenn alle Handles des Threads (auch das mit `CreateThread` ermittelte Handle) mit einem Aufruf der Funktion `CloseHandle` geschlossen werden.

Der zum Beenden des Threads erforderliche Code (der von der Thread-Funktion zurückgegebene Wert oder der `ExitThread` übergebene Wert) wird mit Hilfe der Funktion `GetExitCodeThread` ermittelt.

Die Priorität eines Threads wird mit `GetThreadPriority` ermittelt und mit `SetThreadPriority` gesetzt.

Ein Thread kann im `WARTEZUSTAND` gestartet werden, indem `CREATE_SUSPENDED` als eines der Erstellungsflags des Threads in dem Aufruf von `CreateThread` übergeben wird. Die Ausführung eines wartenden Threads wird mit `ResumeThread` fortgesetzt.

12.2.5 Erstellen und Verwalten von Prozessen

MS-DOS-Programmierer verwendeten lange Zeit die `exec`-Funktionsfamilie, um neue Prozesse zu erzeugen. Windows-Entwickler benutzten `WinExec`, während Unix-Anwender mit `fork` arbeiteten. Unter Win32 wurde diese Funktionalität in der `CreateProcess`-Funktion zusammengefaßt.

Diese Funktion startet die angegebene Anwendung. Sie gibt den Handle eines `PROZESS-OBJEKTS` zurück, das später dazu verwendet werden kann, auf den neu erstellten Prozeß zuzugreifen. Das Prozeß-Objekt enthält einige Eigenschaften des neuen Prozesses, wie z. B. Sicherheitsattribute oder Thread-Informationen.

Der Prozeß wird mit einem Aufruf der Funktion `ExitProcess` beendet. Ein Prozeß wird ebenfalls beendet, wenn sein primärer Thread geschlossen wird.

12.2.6 Synchronisierungsobjekte

Die Variable `bDoAbort` in dem zuvor aufgeführten Multithreading-Beispiel bietet eine einfache Möglichkeit der Synchronisierung von zwei oder mehreren unabhängig ausgeführten Threads. Für unsere Zwecke war diese globale, mit dem Schlüsselwort `volatile` deklarierte Variable ausreichend. Komplexe Situationen erfordern jedoch adäquate Lösungen.

Eine dieser Situationen tritt ein, wenn ein Thread darauf wartet, daß ein anderer Thread eine bestimmte Aufgabe beendet. Wäre eine Variable, auf die beide Threads zugreifen können, der einzige verfügbare Synchronisierungsmechanismus, müßte der wartende Thread eine Schleife ausführen, die wiederholt den Wert dieser Variablen prüft. Geschieht dieser Schleifendurchlauf sehr häufig, geht sehr viel Bearbei-

tungskapazität verloren. Die Zeitintervalle können vergrößert werden, indem eine Verzögerung in die Überprüfung eingefügt wird, wie in dem folgenden Beispiel dargestellt:

```
while (!bStatus) Sleep(1000);
```

Leider ist auch diese Vorgehensweise nicht immer angemessen. Wir können nicht zehntel oder hundertstel Millisekunden warten, bevor die nächste Aktion ausgeführt wird.

Die Win32-API stellt einige Funktionen zur Verfügung, die verwendet werden können, um darauf zu warten, daß ein bestimmtes Objekt oder mehrere Objekte ein SIGNAL geben. Diese Funktionen beziehen sich auf verschiedene Objekttypen. Dazu zählen Synchronisierungsobjekte und andere Objekte, die in einen Signalstatus und Nichtsignalstatus gesetzt werden können.

Synchronisierungsobjekte sind

- ➔ Semaphoren,
- ➔ Ereignisse und
- ➔ Mutexe (Abkürzung für MUTual EXclusion = gegenseitiger Ausschluß).

Semaphore

Semaphor-Objekte begrenzen die Anzahl der konkurrierenden Zugriffe auf eine Ressource. Die maximale Anzahl wird während der Erstellung eines Semaphor-Objekts mit der `CreateSemaphore`-Funktion angegeben. Immer dann, wenn ein Thread erzeugt wird, der auf ein Signal des Semaphor-Objekts wartet, wird der Zähler des Objekts um den Wert 1 verringert. Der Zähler kann mit `ReleaseSemaphore` zurückgesetzt werden.

Ereignisse

Der Status eines Ereignis-Objekts kann explizit auf signalisierend oder nichtsignalisierend gesetzt werden. Während der Erstellung eines Ereignisses mit `CreateEvent` wird dessen anfänglicher Status und Typ bestimmt. Ein manuell eingerichteter nichtsignalisierender Status kann mit der Funktion `ResetEvent` zugewiesen werden. Ein automatisch konfiguriertes Ereignis wird immer dann auf den nichtsignalisierenden Status gesetzt, wenn ein neuer Thread erzeugt wird. Der signalisierende Status des Ereignisses kann mit `SetEvent` gesetzt werden.

Mutexe

Ein Mutex-Objekt befindet sich im nichtsignalisierenden Zustand, wenn sein Besitzer ein Thread ist. Ein Thread wird Besitzer eines Mutex-Objekts, wenn er dessen Zugriffsnummer in einer Wartefunktion angibt. Das Mutex-Objekt wird mit `ReleaseMutex` freigegeben.

Threads warten auf ein einzelnes Objekt, indem sie eine der Funktionen `WaitForSingleObject` oder `WaitForSingleObjectEx` einsetzen. Mit Hilfe der Funktionen `WaitForMultipleObjects`, `WaitForMultipleObjectsEx` oder `MsgWaitForMultipleObjects` warten Threads auf mehrere Objekte.

Synchronisierungsobjekte können ebenfalls für die Synchronisierung von Interprozessen verwendet werden. Semaphoren, Ereignisse und Mutexe erhalten eine Bezeichnung während ihrer Erzeugung mit der entsprechenden Funktion. Ein anderer Prozeß kann anschließend eine Zugriffsnummer auf diese Objekte mit `OpenSemaphore`, `OpenEvent` und `OpenMutex` öffnen.

Kritische Abschnitte

KRITISCHE ABSCHNITTE sind eine Variante der Mutex-Objekte. Objekte, über die auf kritische Abschnitte zugegriffen wird, können lediglich von einem Thread desselben Prozesses verwendet werden. Diese Objekte stellen jedoch einen effizienteren Mechanismus zum gegenseitigen Ausschluß zur Verfügung. Sie schützen die kritischen Abschnitte des Programmcodes. Ein Thread wird mit einem Aufruf von `EnterCriticalSection` Besitzer eines kritischen Abschnitts. Der Besitz wird mit `LeaveCriticalSection` freigegeben. Ist der kritische Abschnitt während des Aufrufs von `EnterCriticalSection` bereits im Besitz eines anderen Threads, wartet die Funktion, bis der kritische Abschnitt freigegeben wird.

Ein weiterer effektiver Mechanismus ist der SYNCHRONISIERTE VARIABLENZUGRIFF. Mit den Funktionen `InterlockedIncrement` oder `InterlockedDecrement` kann ein Thread den Wert einer Variablen vergrößern oder verringern und das Ergebnis prüfen, ohne von einem anderen Thread unterbrochen zu werden (der möglicherweise ebenfalls diese Variable inkrementieren oder dekrementieren möchte, bevor der erste Thread das Ergebnis überprüfen kann). Die Funktionen können außerdem für die Interprozeß-Synchronisierung verwendet werden, wenn sich die Variable im globalen Speicher befindet.

Threads können nicht nur auf Synchronisierungsobjekte, sondern ebenfalls auf bestimmte andere Objekte warten. Der Status eines Prozeß-Objekts wird auf signalisierend gesetzt, wenn der Prozeß beendet wird. Für ein Thread-Objekt gilt derselbe Sachverhalt. Ein mit `Find-`

`FirstChangeNotification` erstelltes Objekt, über das auf eine veränderte Benachrichtigung zugegriffen wird, nimmt den signalisierenden Status an, nachdem Änderungen in dem Dateisystem vorgenommen wurden. Ein Objekt, über das auf die Konsoleneingabe zugegriffen wird, erhält den signalisierenden Status, wenn eine ungelesene Eingabe in dem Puffer der Konsole enthalten ist.

12.2.7 Programmieren mit Synchronisierungsobjekten

Die Techniken zur Einbindung von Synchronisierungsmechanismen und zur Nutzung mehrerer Threads können nicht nur in Programmen verwendet werden, die die grafische Schnittstelle verwenden. Auch andere Anwendungen, wie z.B. Konsolenanwendungen, nutzen diese Verfahren. Das C++-Beispiel in Listing 12.5 ist solch eine Konsolenanwendung, die Sie mit

CL MUXE.CPP

kompilieren.

Listing 12.5:
C++-Beispiel für
ein Mutex-Objekt

```
#include <iostream.h>
#include <windows.h>

void main()
{
    HANDLE hMutex;

    hMutex = CreateMutex(NULL, FALSE, "MYMUTEX");
    cout << "Attempting to gain control of MYMUTEX object...";
    cout.flush();
    WaitForSingleObject(hMutex, INFINITE);
    cout << '\n' << "MYMUTEX control obtained." << '\n';
    cout << "Press ENTER to release the MYMUTEX object: ";
    cout.flush();
    cin.get();
    ReleaseMutex(hMutex);
}
```

Dieses kleine Programm erstellt ein Mutex-Objekt und versucht, Besitzer dieses Objekts zu werden. Wird lediglich eine Instanz dieser Anwendung ausgeführt (im DOS-Fenster von Windows 95/98 oder Windows NT), geschieht zunächst nichts.

Erst wenn Sie ein zweites DOS-Fenster öffnen und das Programm dort ebenfalls ausführen lassen, sehen Sie, daß die erste Instanz des Programms die Kontrolle über das Mutex-Objekt erhält. Die zweite Anwendung wird während des Versuchs, ebenfalls die Kontrolle zu erhalten, in den Wartemodus gesetzt. Dieser Modus wird so lange aufrechterhalten, bis die erste Anwendung das Objekt freigibt. Dies geschieht mit einem Aufruf der Funktion `ReleaseMutex`. Nach der Freigabe wird die von der zweiten Instanz aufgerufene Funktion `WaitForSingleObject` beendet, so daß die Anwendung die Kontrolle über das

Objekt erhält. Die Anzahl der Prozesse, die über diesen Mechanismus kooperieren können, ist nicht begrenzt. Sie können so viele Instanzen in verschiedenen DOS-Fenstern aufrufen, wie Sie möchten und Ihr Arbeitsspeicher zuläßt.

Die beiden Instanzen dieses Programms greifen über den Namen des Objekts darauf zu. Ein Name bezeichnet somit ein globales Objekt. Daran erkennen Sie, wie bezeichnete Synchronisierungsobjekte zur Synchronisierung von Threads und Prozessen, zur Überwachung des Zugriffs auf begrenzte Ressourcen oder zur Bereitstellung einfacher Kommunikationsmechanismen zwischen Prozessen eingesetzt werden.

12.3 Zusammenfassung

Multitasking repräsentiert die Fähigkeit eines Betriebssystems, mehrere Prozesse gleichzeitig ausführen zu lassen. Dies geschieht mit einem Kontextschalter, der zwischen den Anwendungen umschaltet.

- In einem kooperativen Multitasking-System müssen Anwendungen explizit die Steuerung des Prozessors an das Betriebssystem abgeben. Das Betriebssystem kann ein nichtkooperatives Programm nicht unterbrechen.
- In einem präemptiven Multitasking-System unterbricht das Betriebssystem Anwendungen, die auf nichtsynchrone Ereignissen basieren, wie z. B. Interrupts der Zeitgeber-Hardware. Ein derartiges Betriebssystem ist sehr komplex und muß Situationen, wie z. B. Wiederanläufe, bewältigen.

Windows 3.1 und Win32s sind Beispiele für ein kooperatives Multitasking-System. Windows NT und Windows 95/98 sind präemptive Multitasking-Systeme. Windows 95/98 übernahm jedoch einige Einschränkungen von Windows 3.1. Der Grund hierfür besteht darin, daß einige interne 16-Bit-Funktionen von Windows 3.1 in Windows 95/98 implementiert wurden.

Sowohl Windows 95/98 als auch Windows NT sind Multithreading-Betriebssysteme. Threads sind parallele Ausführungspfade innerhalb eines Prozesses.

Obwohl Windows-95/98- und Windows-NT-Programme nicht wie bisher die Kontrolle an das Betriebssystem abgeben müssen, ist eine Bearbeitung von Nachrichten weiterhin erforderlich, auch während der Ausführung längerer Prozesse. Auf diese Weise wird gewährleistet, daß Anwendungen kontinuierlich auf die Benutzerschnittstellenereignisse reagieren.

Für die Synchronisierung der Ausführung von Threads bestehen verschiedene Methoden. Die Win32-API stellt einen Zugriff auf bestimmte Synchronisierungsobjekte zur Verfügung, wie z.B. Semaphoren, Mute-xe und Ereignisse.

DLLs – Dynamische Bibliotheken

Kapitel 13

Dynamische Linkbibliotheken sind vermutlich das hervorstechendste Mittel zur Speichereinsparung unter Windows. DLLs werden nur bei Bedarf in den Speicher geladen, insgesamt aber nur einmal, auch wenn sie gleichzeitig von mehreren Anwendungen ausgeführt werden.

Unter Win32, wo jede Anwendung ihren eigenen Adreßraum hat, bedeutet dies, daß alle Anwendungen, die die DLL benutzen, sie in ihren speziellen Adreßraum laden, die DLL aber jeweils auf den gleichen Ort im physikalischen Speicher abgebildet wird.

13.1 Arten von Bibliotheken

Bibliotheken, Sammlungen von Funktionen, Klassen, Ressourcen, können gemeinhin in drei verschiedenen Varianten vorliegen:

Bibliothek	Extension	Beschreibung
Objektmodul	.obj	Kompilierte Version einer Quelltextdatei. Das Objektmodul wird vom Linker vollständig in die EXE-Datei eingebunden.
statische	.lib	Kompilierter Code einer Sammlung von Funktionen, Klassen, die aus einer oder mehreren Quelltextdateien stammen können (Bearbeitung mittels dem Programm <i>LIB.EXE</i>) Statische Bibliotheken verfügen über eine Art »Inhaltsverzeichnis«, das es dem Linker ermöglicht, nur die wirklich von einer Anwendung benötigten Funktionen in die EXE-Datei einzubinden.

Tabelle 13.1:
Arten von Bibliotheken

Bibliothek	Extension	Beschreibung
dynamische	.dll, .drv, etc.	Kompilierte Version einer oder mehrerer Quelltextdateien. Eine DLL wird jeweils nur einmal in den Arbeitsspeicher geladen – auch wenn mehrere Anwendungen gleichzeitig auf den Code der DLL zugreifen. DLLs können statisch (bei Programmbeginn) oder dynamisch (während der Programmausführung) geladen werden. (Letztere Option wird in Visual C++ jetzt sogar durch Linkeroptionen unterstützt)

13.2 Programmieren mit DLLs

Bei der Implementierung eigener DLLs gilt es folgende Punkte zu beachten:

Auf der Seite der DLL:

- Eine DLL enthält keine main- oder WinMain-Funktion, dafür aber eine spezielle Ein- und Austrittsfunktion: `DllMain`.
- Funktionen, die die DLL für den Aufruf durch externe Anwendungen freigeben soll, müssen exportiert werden.

Auf der Seite der aufrufenden Anwendung

- Die DLL muß zusammen mit der Anwendung geladen werden.
- Die Anwendung muß DLL-Funktionen explizit importieren.

Die DLL-Eintrittsfunktion

Eine DLL besitzt eine spezielle Eintrittsfunktion `DllMain`, die im übrigen auch als Austrittsfunktion fungiert.

Über die Eintrittsfunktion der DLL können Sie Initialisierungen vornehmen und beim Freigeben der DLL Aufräumarbeiten erledigen (beispielsweise dynamischen Speicher löschen). Damit Sie erkennen können, ob die `DllMain`-Funktion als Eintritts- oder Austrittsfunktion aufgerufen wurde, wird ihr vom Betriebssystem ein spezieller Parameter übergeben (`DWORD ul_reason_for_call`). Anhand dieses Parameters, der einen der Werte

- ➡ DLL_PROCESS_ATTACH,
- ➡ DLL_PROCESS_DETACH,
- ➡ DLL_THREAD_ATTACH oder
- ➡ DLL_THREAD_DETACH

annehmen kann, läßt sich feststellen, ob die Funktion als Eintritts- oder Austrittsfunktion aufgerufen wurde, und ob es sich bei dem Aufrufer um einen Prozeß oder einen Thread handelt.

Wenn Sie den MFC-Anwendungs-Assistenten (dll) verwenden, arbeiten Sie statt mit der `DllMain`-Funktion mit Konstruktor und Destruktor eines `CWinApp`-Objekts. Die Eintrittsfunktion ist in der internen Implementierung der MFC versteckt (analog zur Eintrittsfunktion von .exe-Anwendungen).



Funktionen, Klassen exportieren

Alle Funktionen, Klassen einer DLL, die von anderen Modulen (DLLs oder EXE-Dateien) verwendet werden sollen, müssen exportiert werden. Dazu stellen Sie der Definition und Deklaration der zu exportierenden Funktion (Klasse) einfach das Schlüsselwort `__declspec(dllexport)` voran.

Als Alternative dazu können Sie die zu exportierenden Funktionen auch im `EXPORTS`-Bereich der Moduldefinitionsdatei (Extension .def) der DLL ausführen.



DLL-Funktionen werden zudem häufig als extern »C« deklariert. Dies hat im Grunde nichts mit dem Export zu tun. Es unterbindet lediglich die interne Namenserverweiterung von C++-Compilern (notwendig wegen der Funktionenüberladung), so daß die Funktionen auch von C-Anwendungen aufgerufen werden können.

DLL laden

Eine Anwendung, die Funktionen einer DLL aufrufen will, muß zuerst die DLL laden.

Dies kann in Form einer Importbibliothek (.lib) geschehen. Diese wird automatisch bei der Erstellung der DLL miterzeugt und kann als »Ersatz« für die DLL in das Projekt der Anwendung eingebunden werden. Beim Erstellen der Anwendung kann der Linker dann alle relevanten Informationen, die er zum Aufruf der DLL-Funktionen benötigt, der Importbibliothek entnehmen. Bei Verwendung einer Importbibliothek wird die DLL beim Programmstart der Anwendung geladen.

Importbibliotheken



Als Alternative dazu können Sie die zu exportierenden Funktionen auch im IMPORTS-Bereich der Moduldefinitionsdatei (Extension .def) der DLL ausführen.

Dynamische Einbindung

Die Einbindung der DLL kann aber auch dynamisch durch einen Aufruf der API-Funktion `LoadLibrary` (oder `AfxLoadLibrary` für erweiterte MFC-Anwendungen) geschehen. Als Argument wird `LoadLibrary` der Pfad zur .dll-Datei übergeben. Konnte die DLL geladen werden, liefert die Funktion einen Instanz-Handle auf die DLL zurück. Über diesen Handle können dann Funktionen der DLL aufgerufen werden. Auch die Funktion `FreeLibrary` zum Freigeben der DLL benötigt den Handle.

Beim dynamischen Einbinden der DLL mit Hilfe der Funktion `LoadLibrary` werden die Funktionen nicht importiert. Statt dessen verwendet man die API-Funktion `GetProcAddress(HINSTANCE derDLL, LPCSTR func-Name)`, um sich einen Zeiger auf die gewünschte Funktion zurückliefern zu lassen.



Visual C++ 6.0 erlaubt auch das verzögerte Laden von DLLs über Linkereinstellungen (/DELAYLOAD, /DELAY), so daß man sich die Verwendung von `LoadLibrary` in vielen Fällen sparen kann.

Funktionen, Klassen importieren

Die aufrufende Anwendung muß die DLL-Funktionen, die sie aufrufen will, vorab importieren. Dazu stellen Sie der Definition der zu importierenden Funktion einfach das Schlüsselwort `__declspec(dllimport)` voran.

Die Header-Datei der DLL mit den zu exportierenden Funktionen (Klassen) benötigt man im Grunde zweimal:

einmal für das DLL-Projekt und

noch einmal für die Anwendungen, die auf die Funktionen der DLL zugreifen wollen.

Makro für Export und Import

Um nicht extra zwei Header-Datei aufsetzen zu müssen, in denen die Funktionen einmal mit `__declspec(dllimport)` und einmal mit `__declspec(dllexport)` deklariert werden, kann man sich auch eines Makros bedienen. Der Win32-DLL-Assistent verwendet beispielsweise das folgende Makro:

```
#ifndef DLL2_EXPORTS
#define DLL2_API __declspec(dllexport)
#else
```

```
#define DLL2_API __declspec(dllimport)
#endif
```

Der Schalter `DLL2_EXPORTS` wird entsprechend nur im DLL-Projekt definiert.

Ein Beispielprojekt

Am einfachsten ist es, DLL und Testanwendung im Visual Studio in einem gemeinsamen Arbeitsbereich zu erstellen.

1. Legen Sie einen neuen Arbeitsbereich an.
2. Legen Sie in dem Arbeitsbereich ein Win32 Dynamic-Link Library-Projekt und ein Win32-Anwendungs-Projekt an.
3. Wechseln Sie zurück zum DLL-Projekt (Befehl PROJEKT/AKTIVES PROJEKT FESTLEGEN) und setzen Sie den Quelltext der DLL auf.

Die folgende Beispiel-DLL verfügt nur über eine einzige Funktion namens `DLL_func`, die einen `MessageBeep` auslöst und von der DLL exportiert wird.

```
// DLL.h - Die Header-Datei
#ifndef DLL_EXPORTS
#define DLL_API __declspec(dllexport)
#else
#define DLL_API __declspec(dllimport)
#endif
DLL_API int DLL_func(void);

// DLL.cpp - die Quelltextdatei
#include "stdafx.h"
#include "DLL.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved

)
{
    return TRUE;
}

DLL_API int DLL_func(void)
{
    MessageBeep(MB_OK);
    return 0;
}
```

4. Lassen Sie die DLL erstellen. Der Linker legt im Zuge der Erstellung automatisch eine Importbibliothek zu der DLL an.
5. Kopieren Sie die DLL in ein Verzeichnis, wo Sie von der Testanwendung aufgerufen werden kann (Windows-Verzeichnis oder Verzeichnis der EXE-Datei).
6. Wechseln Sie nun zur Testanwendung.

7. Setzen Sie den Code auf, der die Funktion aus der DLL aufrufen soll. Vergessen Sie dabei auch nicht, die Header-Datei der DLL aufzunehmen.

```
// TestApp.cpp - Quelltextdatei der Testanwendung
#include <windows.h>
#include "StdAfx.h"
#include "Dll.h"

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            DLL_func();
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HELLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }

    hwnd = CreateWindow("HELLO", "HELLO",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                        NULL, NULL, hInstance, NULL);
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}
```

8. Nehmen Sie die Importbibliothek der DLL in das Projekt auf (Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN).
9. Erstellen Sie die Testanwendung und führen Sie sie aus.

Sie können jetzt auch die DLL direkt ausführen lassen.

1. Kehren Sie zurück zum DLL-Projekt.
2. Rufen Sie den Befehl PROJEKT/EINSTELLUNGEN auf und geben Sie im Feld AUSFÜHRBARES PROGRAMM FÜR DEBUG-SITZUNG auf der Seite DEBUG den Pfad und Namen der Testanwendung an.

13.3 Zusammenfassung

Dynamische Linkbibliotheken werden zur Schonung des Arbeitsspeichers eingesetzt, da sie stets nur einmal in den Speicher geladen werden, auch wenn sie gleichzeitig von mehreren Anwendungen ausgeführt werden.

Unter Win32, wo jede Anwendung ihren eigenen Adreßraum hat, laden alle Anwendungen, die die DLL benutzen, sie in ihren speziellen Adreßraum, die DLL wird aber jeweils auf den gleichen Ort im physikalischen Speicher abgebildet.

Eine DLL verfügt über eine Ein- und Austrittsfunktion namens `DllMain`. In dieser kann der Programmierer beispielsweise Code zur Anforderung und Freigabe von Ressourcen für die Funktionen/Klassen der DLL aufsetzen.

Funktionen (Klassen), die von der DLL exportiert werden, müssen mit `__declspec(dllexport)` deklariert und definiert werden.

DLL-Funktionen (DLL-Klassen), die von einer Anwendung importiert werden, müssen mit `__declspec(dllimport)` deklariert werden.

Meist werden DLLs mit Hilfe einer für die DLL erstellten Importbibliothek geladen.

DLLs können aber auch mit Hilfe der Funktion `LoadLibrary` dynamisch geladen werden. (In Visual C++ stehen auch spezielle Linkeroptionen zum dynamischen Laden zur Verfügung.)

Speicherverwaltung

Kapitel

14

Mit den Anfängen der 32-Bit-Version von Windows wurde die Speicherverwaltung zu einem interessanten Aspekt. Die große Anzahl der Segmente, Selektoren und all die Utensilien zur Speicherverwaltung im 16-Bit-Modus der untergliederten Intel-Prozessorarchitektur bestehen nicht mehr. Die Speicherverwaltung wurde derart formalisiert, daß den meisten Anwendungen die Schlüsselworte `malloc` oder `new` genügen. Wäre dieses Buch eine Einführung, würde ich an dieser Stelle das Kapitel wahrscheinlich beenden und mit dem nächsten Thema fortfahren.

Allerdings besitzt auch die Win32-Speicherverwaltung ihre Feinheiten. Programmierer müssen sich diese lediglich nicht mehr aneignen, um einfache Aufgaben auszuführen.

14.1 Prozesse und der Speicher

Win32 verfügt über eine hochentwickelte Speicherverwaltung, deren wesentliche Merkmale das Ausführen einer Anwendung in einem separaten Adreßraum und das Erweitern des reservierten Speichers mit Hilfe von Auslagerungsdateien sind. Diese Merkmale sind ein Bestandteil der virtuellen Win32-Speicherverwaltung.

14.1.1 Separate Adreßräume

Programmierer, die mit der 16-Bit-Version von Windows vertraut sind, müssen sich erst daran gewöhnen, daß eine Adresse nicht länger einen definierten Punkt im physikalischen Speicher repräsentiert. Während ein Prozeß an der Adresse `0x10000000` Daten vorfindet, verwahrt ein

anderer Prozeß dort einen Abschnitt seines Programmcodes. Für einen anderen Prozeß ist diese Adresse möglicherweise unzulässig. Wie ist dies möglich?

Logische Adressen Die Adressen, die Win32 verwendet, werden häufig als LOGISCHE ADRESSEN bezeichnet. Jeder Win32-Prozeß verfügt über einen eigenen 32-Bit-Adreßbereich (mit einigen spezifischen Einschränkungen des Betriebssystems, wie Sie später feststellen werden). Greift ein Win32-Prozeß auf die Daten einer logischen Adresse zu, interveniert die Speicherverwaltungs-Hardware des Computers und wandelt diese Adresse in eine PHYSIKALISCHE ADRESSE um (dazu später mehr). Dieselbe logische Adresse kann (und wird gewöhnlich) für unterschiedliche Prozesse in verschiedene physikalische Adressen konvertiert.

Dieser Mechanismus hat verschiedene Konsequenzen. Die meisten dieser Auswirkungen sind vorteilhaft, einige führen jedoch dazu, daß ein Teil der Programmieraufgaben nur schwer umgesetzt werden kann.

Vorteile Der große Vorteil separater logischer Adreßräume besteht darin, daß Prozesse nicht wie bisher versehentlich den Programmcode oder die Daten anderer Prozesse überschreiben können. Unzulässige Zeiger können zum Absturz des entsprechenden Prozesses führen, aber nicht die Daten in dem Adreßraum anderer Prozesse oder des Betriebssystems zerstören



Vorsicht! Die Adressen des Win95/98-Betriebssystems sind nicht in gleicher Weise geschützt wie die Adressen von WinNT.

Nachteile Die Tatsache, daß sich Prozesse nicht mehr denselben logischen Adreßraum teilen, erschwert jedoch gleichzeitig die Entwicklung interagierender Prozesse. So ist es beispielsweise nicht möglich, die Adresse eines Objekts im Speicher wie bisher an einen anderen Prozeß zu senden, so daß dieser die Adresse verwenden kann. Eine derartige Adresse hat lediglich für die versendende Anwendung eine Bedeutung. Für die empfangende Anwendung ist diese Adresse ein zufälliger, unbedeutender Speicherbereich.

Die Win32-API bietet interagierenden Anwendungen glücklicherweise einige Mechanismen, die diesen Umstand berücksichtigen. Einer dieser Mechanismen ist die Verwendung eines GEMEINSAMEN SPEICHERS. Ein gemeinsamer Speicher ist ein physikalischer Speicherbereich, der in dem logischen Adreßraum verschiedener Prozesse liegt. Anwendungen können miteinander kooperieren, indem Sie in diesen gemeinsamen Speicher Daten ablegen oder daraus auslesen.

Unter der vereinfachten Speicherverwaltung von Win32, teilen sich alle Win32-Anwendungen denselben Adreßraum.



14.1.2 Adreßräume

Wie bereits erwähnt wurde, ist die Verwendung von 32-Bit-Adressen innerhalb des logischen Adreßraums eines Prozesses eingeschränkt. Einige Adreßbereiche sind beispielsweise für das Betriebssystem reserviert. Außerdem differieren die Einschränkungen für die unterschiedlichen Win32-Umgebungen.

Das Verwenden von 32-Bit-Adressen bedingt einen Adreßraum mit einer Größe von vier Gbyte ($2^{32} = 4.294.967.296$). Windows reserviert die oberen zwei Gbyte für eigene Zwecke, während die unteren zwei Gbyte von Anwendungen verwendet werden können.

4 Gbyte Speicher

Windows 95 reserviert außerdem die unteren vier Mbyte des Adreßraums. Dieser Bereich, der in der Microsoft-Dokumentation häufig als KOMPATIBILITÄTSBEREICH bezeichnet wird, dient der Kompatibilität zu 16-Bit-DOS- und Windows-Anwendungen.

Sie wissen bereits, daß Win32-Anwendungen in separaten Adreßräumen ausgeführt werden. Davon betroffen sind die nichtreservierten Bereiche des logischen Adreßraums. Für die reservierten Bereiche gelten andere Sachverhalte.

Unter Windows 95 sind alle reservierten Bereiche gemeinsamer Speicher. Legt eine Anwendung somit ein Objekt an einer bestimmten Position im reservierten Speicher ab (untere 4 Mbyte oder obere 2 Gbyte), können alle anderen Anwendungen ebenfalls auf dieses Objekt zugreifen. Anwendungen sollten sich diesen Umstand nicht zunutze machen, da andernfalls keine Kompatibilität mehr zu Windows NT besteht. Anwendungen können jedoch auf einfachere Möglichkeiten zurückgreifen, um explizit einen gemeinsamen Speicherbereich einzurichten. Dieser Mechanismus ist sowohl für Windows 95 als auch für Windows NT geeignet.

Windows 95 unterteilt die oberen 2 Gbyte in zwei Bereiche. Der Bereich zwischen 2 und 3 Gbyte ist gemeinsamer Speicher, der Speicherzuordnungsdateien sowie einige 16-Bit-Komponenten enthält. Der RESERVIERTER SYSTEMBEREICH zwischen 3 und 4 Gbyte nimmt den Programmcode des Betriebssystems auf. Nichtprivilegierte Anwendungen können nicht auf diesen Bereich zugreifen.

14.1.3 Virtueller Speicher

*Von logischen
zu virtuellen
Adressen* Bisher wurde eine Frage nicht angesprochen: Wie werden logische Adressen in physikalische Adressen umgewandelt? Die meisten Computer verfügen nicht über genügend Speicher, um einem 4 Gbyte großen Adreßraum gerecht zu werden.

Die Antwort lautet: Nicht alle logischen Adressen einer Anwendung werden im physikalischen Speicher abgelegt.

Auslagerungsdateien

Bereits Windows 3.1 verwendete eine AUSLAGERUNGSDATEI. Der Auslagerungsmechanismus erweitert den vom System verwendeten Speicher, indem nicht benötigte Datenblöcke auf der Festplatte gespeichert und erst dann wieder geladen werden, wenn die Notwendigkeit dazu besteht. Obwohl Auslagerungsdateien sehr viel langsamer als das RAM arbeiten, ermöglichen sie dem System, mehrere Anwendungen gleichzeitig oder solche Anwendungen auszuführen, die intensiven Gebrauch von Ressourcen machen.

Auslagerungsdateien können effizient eingesetzt werden, da die Anwendungen nur solche Datenblöcke auslagern, die selten benötigt werden. Arbeiten Sie beispielsweise mit einer Textverarbeitung, um zwei Dokumente gleichzeitig zu editieren, kann es geschehen, daß Sie eines dieser Dokumente besonders intensiv bearbeiten, während das andere Dokument für längere Zeit unberührt bleibt. Das Betriebssystem kann nun den physikalischen Speicher freigeben, den dieses Dokument belegt, indem es das Dokument auf der Festplatte auslagert. Der physikalische Speicher steht daraufhin anderen Anwendungen zur Verfügung. Wechseln Sie nach einiger Zeit zu dem ausgelagerten Dokument, werden Sie einige Festplattenaktivitäten und eine geringe Zeitverzögerung bemerken, bevor das Dokument angezeigt wird. Das Betriebssystem hat in dieser Zeit die relevanten Abschnitte der Auslagerungsdatei in den Speicher eingelesen, so daß andere nicht benötigte Datenblöcke möglicherweise wieder ausgelagert wurden.

Die Seitentabelle

Abbildung 13.1 zeigt, wie das Betriebssystem und die Hardware des Computers logische Adressen umwandeln. Eine Tabelle, die als SEITENTABELLE bezeichnet wird, enthält Informationen über alle Blöcke oder SEITEN des Speichers. Diese Tabelle dient der Umwandlung von Adreßraumblöcken einer Anwendung in physikalische Speicherblöcke oder in Abschnitte der Auslagerungsdatei.

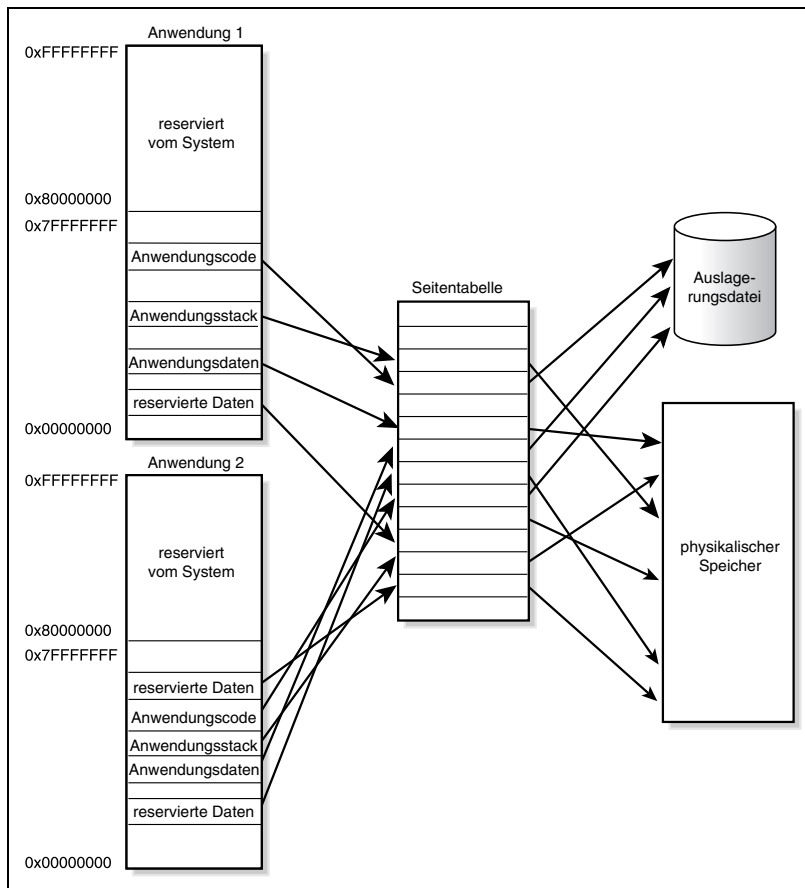


Abbildung 14.1:
Umwandeln
logischer Adres-
sen in physikali-
schen Speicher

Nachdem eine logische Adresse physikalischem Speicher zugewiesen wurde, können Daten in den Speicher geschrieben oder daraus ausgelesen werden. Da die Zuweisung von der Prozessor-Hardware unterstützt wird, ist zusätzlicher Programmcode zur Ermittlung von Speicheradressen nicht erforderlich.

Verweist eine logische Adresse auf einen Block in der Auslagerungsdatei des Systems, werden verschiedene Ereignisse ausgelöst. Der Versuch, auf solch eine unzulässige Adresse zuzugreifen, führt dazu, daß das Betriebssystem den geforderten Datenblock aus der Auslagerungsdatei in den Speicher lädt. Eventuell müssen dazu einige andere Datenblöcke ausgelagert werden. Nachdem sich die Daten im physikalischen Speicher befinden und die Seitentabelle aktualisiert wurde, wird die Steuerung an die Anwendung zurückgegeben. Der Zugriff auf die gewünschte Speicherposition kann nun geschehen. Die Anwendung merkt von diesen Vorgängen kaum etwas. Das einzige Zeichen, das

darauf hinweist, daß sich der geforderte Datenblock nicht im Speicher befand, ist die Zeitverzögerung, die sich infolge des Einlesens der Daten von der Festplatte ergeben hat.

Die Tatsache, daß logische Adressen auf physikalische Speicherbereiche, Blöcke der Auslagerungsdatei oder auf keines dieser Elemente verweisen können, impliziert interessante Möglichkeiten. Ein Mechanismus, der den Inhalt einer Datei (der Auslagerungsdatei) logischen Adressen zuordnen kann, beinhaltet das Potential für nützliche Features. Die Win32-API stellt Funktionen zur Verfügung, die Anwendungen nutzen können, um explizit den virtuellen Speicher verwalten und über Speicherzuordnungsdateien auf Festplattendaten zugreifen zu können. Diese und andere Mechanismen zur Speicherverwaltung sind in den folgenden Abschnitten beschrieben.

14.2 Von 16- zu 32-Bit

Da sehr viele Windows-Programmierer Erfahrungen in der Programmierung der 16-Bit-Version von Windows besitzen, beginnt die Erläuterung der 32-Bit-Speicherverwaltung mit den Unterschieden zwischen 32-Bit- und 16-Bit-Programmen. Einige Unterschiede, wie z.B. die Größe der Integer-Werte, das Fehlen der Typdeklarationen `near` und `far` sowie Unterschiede in der Adressenberechnung, betreffen die Programmierpraktiken. Einige der nachfolgend aufgeführten Differenzen bilden Richtlinien für das Portieren einer 16-Bit-Anwendung in die 32-Bit-Umgebung.

Integer-Größe

int belegt 4 Byte Einen der wesentlichen Unterschiede zwischen der 16-Bit- und 32-Bit-Umgebung demonstriert das folgende Beispiel. Dieses Programm wird über die Kommandozeile mit der Anweisung

```
CL INTSIZE.CPP
```

kompiliert.

```
#include <iostream.h>
void main(void)
{
    cout << "sizeof(int) = " << sizeof(int);
}
```

Wenn Sie dieses Programm starten, erhalten Sie die folgende Ausgabe:

```
sizeof(int) = 4
```

Unix-Programmierer werden möglicherweise erleichtert sein, dieses Ergebnis zu sehen. Die Probleme, die das Portieren eines Unix-Programms bisher bereitete, das Integer und Zeiger derselben Größe verwendet (32 Bit), bestehen nicht mehr. 16-Bit-Windows-Programmierer hingegen müssen ihre alten Programme nach 16-Bit-Integer-Werten durchsuchen.

Etwas, das nicht verändert wurde, ist die Größe der von Windows definierten Typen. Die Typen `WORD` und `DWORD` sind somit weiterhin 16 und 32 Bit breit. Das Verwenden dieser Typen gewährleistet, daß nach dem Speichern einer Datei deren Inhalt sowohl von der 16-Bit- als auch von der 32-Bit-Version einer Anwendung eingelesen werden kann. Verwendet eine Anwendung statt dessen den Typ `int`, um Daten auf der Festplatte zu speichern, ist der Inhalt der entsprechenden Datei von dem verwendeten Betriebssystem abhängig.

Typmodifizierer und Makros

Eine weitere Folge der 32-Bit-Adressierung besteht darin, daß Sie nicht wie bisher Typmodifizierer verwenden müssen, um zwischen den Zeigern `near` und `far` zu unterscheiden oder große Datenmengen zu bestimmen. Bedeutet dies, daß bereits bestehende Programme modifiziert und alle Verweise auf die Schlüsselworte `_near`, `_far` oder `_huge` entfernt werden müssen? Glücklicherweise nicht. Der 32-Bit-Compiler ignoriert diese Schlüsselworte einfach, um die Abwärtskompatibilität zu gewährleisten.

*near und far sind
obsolet*

Alle Typen, die in der Header-Datei `windows.h` definiert sind, wie z. B. `LPSTR` für einen `far`-Zeiger auf Zeichen oder `LPVOID` für einen `far`-Zeiger auf einen `void`-Typ, sind weiterhin verwendbar. In der 32-Bit-Umgebung bilden diese Typen das Äquivalent zu ihren Korrelaten. `LPSTR` ist gleich `PSTR` und `LPVOID` ist gleich `PVOID`. Um die Abwärtskompatibilität zu bewahren, sollten die korrekten Typen verwendet werden. Ein weiterer Grund, der für diese Vorgehensweise spricht, besteht darin, daß die Schnittstelle zu den meisten Windows-Funktionen diese Typen (`near` oder `far`) benutzt.

Adreßberechnungen

Natürlich müssen Ihre Anwendungen modifiziert werden, wenn diese spezifische Adreßberechnungen zu der untergliederten Intel-Architektur durchführen. (Solche Berechnungen würden außerdem gegen die plattformunabhängige Philosophie der Win32-API verstoßen. Das Kompilieren Ihrer Anwendungen unter Windows NT, MIPS, Alpha oder anderen Plattformen wäre sehr schwierig.)

LOWORD Ihre besondere Aufmerksamkeit sollte dem Makro `LOWORD` gelten. Der unter Windows 3.1 mit `GlobalAlloc` reservierte Speicher wurde an einer Segmentbegrenzung ausgerichtet, deren Offset auf 0 gesetzt wurde. Einige Programmierer nutzten diesen Umstand, um Adressen einzurichten, indem sie einfach das »untere Wort« der Zeigervariablen mit dem Makro `LOWORD` modifizierten. Die Annahme, daß ein reservierter Speicherblock an einer Segmentbegrenzung beginnt, ist unter Windows 95 nicht länger gültig. Die fragwürdige Verwendung von `LOWORD` ist daher nicht mehr möglich.

Bibliotheksfunktionen

In der 16-Bit-Umgebung gab es häufig zwei Versionen bestimmter Funktionen. Eine Version für nahe Adressen und eine Version für entfernte Adressen. Oft mußten beide Versionen benutzt werden. Einige Medium-Speichermodell-Programme verwendeten beispielsweise wiederholt `_fstrncpy`, um Zeichen in oder aus einem entfernten Speicherbereich zu kopieren. In der 32-Bit-Umgebung werden solche Funktionen nicht mehr benötigt.

windowsx.h Die Header-Datei `windowsx.h` definiert diese nicht mehr benötigten Funktionen und führt deren Korrelate auf. Wenn Sie diese Header-Datei in Ihre Anwendung aufnehmen, müssen Sie nicht den gesamten Quellcode Ihres Programms durchsuchen und die entsprechenden Funktionsverweise entfernen oder verändern.

Speichermodelle

Nur noch ein Speichermodell Seitdem es den PC gibt, haben Programmierer gelernt, die Vielzahl der Compiler-Schalter und Optionen zu steuern, die der Kontrolle der Adressen dienen. Schlanke, kleine, kompakte, mittlere, große, riesige und benutzerdefinierte Speichermodelle, Adressenumwandlung, 64-Kbyte-Programmcodem und Datensegmente sind in der 32-Bit-Version von Windows nicht mehr vorhanden. Es gibt lediglich ein Speichermodell, in dem Adressen und Programmcodem in einem 32-Bit-Speicherbereich abgelegt werden.

Selektor-Funktionen

Die Windows-3.1-API enthält einige Funktionen (z.B. `AllocSelector` oder `FreeSelector`), die der Anwendung die direkte Manipulation des physikalischen Speichers ermöglichen. Diese Funktionen sind nicht in der Win32-API vorhanden. 32-Bit-Anwendungen sollten nicht versuchen, den physikalischen Speicher zu bearbeiten. Diese Aufgabe sollte den Gerätetreibern überlassen werden.

14.3 Einfache Speicherverwaltung

Zu Beginn dieses Kapitels wurde bereits erwähnt, daß die Speicherreservierung in der 32-Bit-Umgebung sehr einfach ist. Speicher muß nicht wie bisher separat reserviert und gesperrt werden. Unterschiede zwischen globalen und lokalen Heaps bestehen nicht mehr. Statt dessen präsentiert die 32-Bit-Umgebung neue Funktionen.

14.3.1 Speicherreservierung mit `malloc` und `new`

Da die Anwendungen im Real-Mode des Prozessors physikalische Adressen verwendeten, um auf die Objekte im Speicher zuzugreifen, konnte das Betriebssystem keine Speicherverwaltungsfunktionen ausführen. Die Anwendungen mußten einen komplizierten Mechanismus einsetzen, um die Kontrolle über diese Objekte abzugeben (`GlobalAlloc/GlobalFree`, `GlobalLock/GlobalUnlock`). Wurde ein Objekt nicht benutzt, verwahrte die Anwendung lediglich einen vom System definierten Handle darauf. Die eigentliche Adresse wurde erst dann ermittelt, wenn das Objekt ausgelesen, beschrieben oder freigegeben werden sollte. Das Betriebssystem konnte diese Objekte daher frei verschieben. Da `malloc` aber nicht nur Speicher reservierte, sondern diesen auch sperrte, verhinderte diese Funktion die Speicherverwaltung durch das Betriebssystem und konnte zu einer unerwünschten Fragmentierung des verfügbaren Speichers führen.

Windows 3.1 verwendet Intel-Prozessoren im PROTECTED-MODE. In diesem Modus erhalten Anwendungen keinen Zugriff auf physikalische Adressen. Das Betriebssystem kann einen Speicherblock verschieben, ohne daß dadurch die Adresse ungültig wird, die die Anwendung mit `GlobalLock` oder `LocalLock` ermittelt hat. Das Verwenden von `malloc` wurde somit nicht nur sicher, sondern war sogar die empfohlene Vorgehensweise. Verschiedene Implementierungen dieser Funktion (wie z.B. in Microsoft C/C++, Version 7), dienten der Lösung eines anderen Problems. Aufgrund einer systemweiten Begrenzung der Selektoren (8.192), war die Anzahl der Aufrufe von Funktionen zur Speicherreservierung ohne die vorherige Freigabe von Speicher eingeschränkt. Die neue `malloc`-Implementierung stellte ein Subreservierungsschema zur Verfügung, das der Reservierung einer großen Zahl kleiner Speicherblöcke diente.

Die 32-Bit-Umgebung erleichtert die Speicherreservierung, indem die Unterschiede zwischen globalen und lokalen Heaps aufgehoben werden. (Natürlich ist es weiterhin möglich, wenn auch nicht empfehlenswert, Speicher mit `GlobalAlloc` zu reservieren und mit `LocalFree` freizugeben.)

In einer Win32-Anwendung kann Speicher somit mit `malloc` (oder `new`) reserviert und mit `free` (oder `delete`) freigegeben werden. Das Betriebssystem ist für alle weiteren Aspekte der Speicherverwaltung zuständig. Diese Vorgehensweise ist für die meisten Anwendungen ausreichend.

14.3.2 Das Problem der abweichenden Zeiger

Die Arbeit mit einem linearen 32-Bit-Adreßraum hat eine unerwartete Konsequenz. In einer 16-Bit-Umgebung reserviert ein Aufruf der Funktion `GlobalAlloc` einen neuen SELEKTOR. Im Protected-Mode der untergliederten Intel-Architektur definieren Selektoren Speicherblöcke. Die Länge des Blocks wird als ein Abschnitt des Selektors angegeben. Versuche, auf Speicherbereiche außerhalb der reservierten Begrenzung des Selektors zuzugreifen, resultieren in einer Schutzverletzung.

In der 32-Bit-Umgebung teilen sich automatische und statische Objekte, dynamisch reservierter globaler und lokaler Speicher, der Stack und alle Elemente, die sich auf dieselbe Anwendung beziehen, einen flachen 32-Bit-Adreßraum. Abweichende Zeiger werden von der Anwendung voraussichtlich nicht bemerkt. Die Möglichkeit einer Verfälschung des Speichers durch solche Zeiger ist sehr groß. Der Programmierer muß daher gewährleisten, daß Zeiger auf Adressen innerhalb der vorgesehenen Begrenzung verweisen.

Betrachten Sie bitte einmal den folgenden Programmcode-Abschnitt:

```
HGLOBAL hBuf1, hBuf2;
LPSTR lpszBuf1, lpszBuf2;
hBuf1 = GlobalAlloc(GPTR, 1024);
hBuf2 = GlobalAlloc(GPTR, 1024);
lpszBuf1 = GlobalLock(hBuf1);
lpszBuf2 = GlobalLock(hBuf2);
lpszBuf1[2000] = 'X'; /* Error! */
```

Dieser Programmabschnitt versucht, außerhalb der Begrenzungen des ersten mit `GlobalAlloc` reservierten Puffers zu schreiben. In der 16-Bit-Umgebung würde dieser Versuch zu einer Schutzverletzung führen. In der 32-Bit-Umgebung ist die Speicherposition, auf die `lpszBuf1[2000]` verweist, möglicherweise zulässig und repräsentiert eine Position innerhalb des zweiten Puffers. Der Versuch, in diesen Speicherbereich zu schreiben, wäre erfolgreich und würde den Inhalt des zweiten Puffers verfälschen.

Glücklicherweise ist es praktisch unmöglich, daß eine Anwendung den Adreßraum einer anderen Anwendung mit abweichenden Zeigern verfälscht. Dies erhöht die Stabilität des Betriebssystems.

14.3.3 Aufteilen des Speichers zwischen den Anwendungen

Da jede 32-Bit-Anwendung über einen eigenen virtuellen Adreßraum verfügt, können sich die Anwendungen nicht wie bisher durch den Austausch von Zeigern über Windows-Nachrichten den Speicher teilen. Das Flag `GMEM_DDESHARE`, das in Aufrufen von `GlobalAlloc` verwendet wird, besitzt nun keine Funktion mehr. Die Übergabe des Handles eines 32-Bit-Speicherblocks an eine andere Anwendung ist bedeutungslos. Der Handle verweist lediglich auf einen zufälligen Speicherbereich des virtuellen Adreßraums der Zielanwendung.

Möchten zwei Anwendungen über einen gemeinsamen Speicherbereich miteinander kommunizieren, können sie dazu die DDEML-Bibliothek oder Speicherzuordnungsdateien verwenden, die später in diesem Kapitel beschrieben werden.

14.4 Virtueller Speicher und erweiterte Speicherverwaltung

In der Win32-Entwicklungsumgebung verfügen die Anwendungen über eine verbesserte Kontrolle darüber, wie Speicher reserviert und verwendet wird. In dieser Umgebung werden erweiterte Speicherverwaltungsfunktionen zur Verfügung gestellt. Abbildung 14.2 zeigt die unterschiedlichen Ebenen der Speicherverwaltungsfunktionen in der Win32-API.

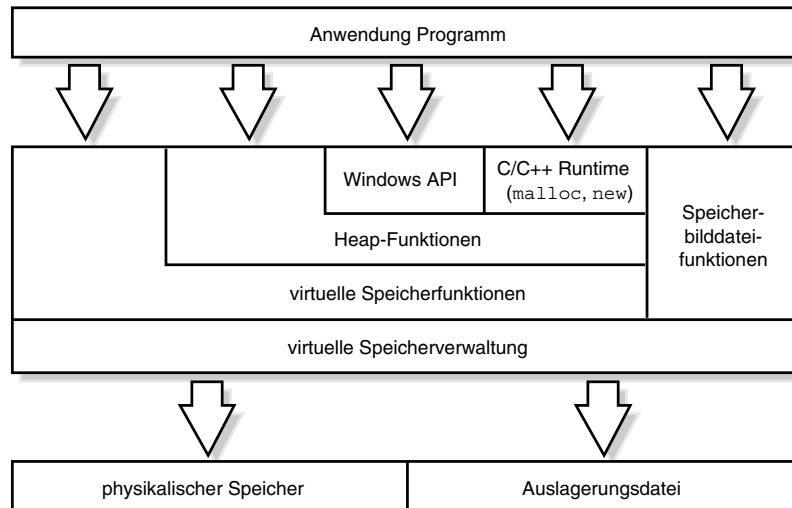
14.4.1 Die virtuelle Speicherverwaltung von Win32

Abbildung 14.1 erweckt möglicherweise den Eindruck, daß virtuelle Speicherseiten in jedem Fall einem physikalischen Speicher, einer Seitendatei oder einer Auslagerungsdatei zugeordnet sind. Dem ist jedoch nicht so. Die Win32-Speicherverwaltung unterscheidet zwischen reservierten und zugewiesenen Seiten. Eine im virtuellen Speicher ZUGEWIESENE SEITE ist einem physikalischen Speicherbereich oder einer Seitendatei zugeordnet. Einer RESERVierten SEITE hingegen ist noch kein physikalischer Speicher zugeordnet.

Reservierte und zugewiesene Seiten

Warum sollten Sie Adressen reservieren wollen, ohne den entsprechenden physikalischen Speicherbereich zu reservieren? Ein Grund hierfür könnte darin bestehen, daß Sie nicht genau wissen, wieviel Speicher eine bestimmte Operation erfordert.

Abbildung 14.2:
Speicherverwal-
tungsfunkti-
onen in der 32-
Bit-Umgebung



Dieser Mechanismus ermöglicht Ihnen, mehrere aufeinanderfolgende Adreßbereiche im virtuellen Speicher Ihres Prozesses zu reservieren, ohne diesen Bereichen physikalische Ressourcen zuzuweisen. Dies geschieht erst dann, wenn die Ressourcen benötigt werden. Wenn der Versuch unternommen wird, auf eine nicht zugewiesene Seite zuzugreifen, generiert das Betriebssystem einen Ausnahmefehler, den Ihre Anwendung über eine Fehlerbehandlungsroutine abfangen kann. Ihr Programm kann daraufhin das Betriebssystem anweisen, der Seite physikalischen Speicher zuzuordnen. Die Bearbeitung wird anschließend fortgesetzt. Auf diese Weise führt Windows 95 einige der Speicherverwaltungsfunktionen aus, wie z.B. die Stack-Reservierung oder die Bearbeitung der Seitentabelle.

Ein weiteres Beispiel für diese Vorgehensweise sind FLACHE MATRIZEN, die aus zweidimensionalen Arrays bestehen. Die meisten Elemente dieser Arrays enthalten den Wert Null. Flache Matrizen werden häufig in technischen Anwendungen verwendet. Natürlich kann Speicher für die gesamte Matrix reserviert werden, die irreversible Zuweisung geschieht jedoch erst dann, wenn eine Seite Elemente enthält, deren Wert ungleich Null ist. Auf diese Weise werden weniger physikalische Ressourcen beansprucht.

14.4.2 Virtuelle Speicherfunktionen

Virtuellen Speicher einrichten und freigeben Eine Anwendung kann Speicher mit `VirtualAlloc` reservieren (und später auch zuordnen lassen). Die Anwendung gibt dazu die logische Adresse und Größe des zu reservierenden Speicherblocks an. Zusätzli-

che Parameter bestimmen den Typ der Reservierung (zugewiesen oder reserviert) sowie des Zugriffs auf die Schutz-Flags. Das folgende Beispiel reserviert 1 Mbyte Speicher ab Adresse 0x10000000, in den geschrieben und der ausgelesen werden kann:

```
VirtualAlloc(0x10000000, 0x00100000, MEM_RESERVE,
            PAGE_READWRITE);
```

Die Anwendung kann die Speicherseiten später physikalischem Speicher mit einem wiederholten Aufruf der Funktion `VirtualAlloc` zuweisen (statt `MEM_RESERVE` wird das Argument `MEM_COMMIT` übergeben).

Zugewiesener oder reservierter Speicher wird mit `VirtualFree` freigegeben.

`VirtualAlloc` kann ebenfalls zum Einrichten von ÜBERWACHUNGSSEITEN verwendet werden. Überwachungsseiten dienen als einmaliger Alarmmechanismus. Dieser Mechanismus erzeugt einen Ausnahmefehler, wenn die Anwendung versucht, auf die Überwachungsseite zuzugreifen. Überwachungsseiten bilden einen Schutz gegen abweichende Zeiger, die beispielsweise auf Speicherbereiche außerhalb einer Array-Begrenzung verweisen.

`VirtualLock` sperrt einen Speicherblock im physikalischen Speicher (RAM). Das Betriebssystem kann solch einen Block nicht auslagern. Verwenden Sie diese Funktion, um zu gewährleisten, daß ein Zugriff auf kritische Daten ohne Speicher- und Einlesevorgänge auf der Festplatte erfolgt. `VirtualLock` sollte nicht häufig eingesetzt werden, da die Funktion die Performance des Betriebssystems durch die Einschränkung der Speicherverwaltung vermindert. Die mit `VirtualLock` gesperrten Speicherbereiche können mit `VirtualUnlock` freigegeben werden. *Speicher sperren*

Eine Anwendung kann die Schutz-Flags zugewiesener Speicherseiten mit Hilfe der Funktion `VirtualProtect` verändern. `VirtualProtectEx` verändert die Schutz-Flags eines Speicherblocks, der einem anderen Prozeß zugewiesen ist. *VirtualProtect*

`VirtualQuery` ermittelt Informationen über Speicherseiten. `VirtualQueryEx` ermittelt Informationen über die Speicherseiten eines anderen Prozesses. *VirtualQuery*

Listing 14.2 führt den Programmcode einer Anwendung auf, die die Anwendung der virtuellen Speicherfunktionen demonstriert. Kompilieren Sie dieses Programm mit

```
CL -GX SPARSE.CPP
```


Listing 14.1:
Verwalten flacher Matrizen mit virtuellen Speicherfunktionen

```
#include <iostream.h>
#include <windows.h>

#define PAGESIZE 0x1000

void main(void)
{
    double (*pdMatrix)[10000];
    double d;
    LPVOID lpvResult;
    int x, y, i, n;

    pdMatrix = (double (*)[10000])VirtualAlloc(NULL,
        100000000 * sizeof(double),
        MEM_RESERVE, PAGE_NOACCESS);

    if (pdMatrix == NULL)
    {
        cout << "Speicher konnte nicht reserviert werden.\n";
        exit(1);
    }

    n = 0;
    for (i = 0; i < 10; i++)
    {
        x = rand() % 10000;
        y = rand() % 10000;
        d = (double)rand();
        cout << "MATRIX[" << x << ', ' << y << "] = " << d << '\n';
        try
        {
            pdMatrix[x][y] = d;
        }
        catch (...)
        {
            if (d != 0.0)
            {
                n++;
                lpvResult = VirtualAlloc((LPVOID)&pdMatrix[x][y],
                    PAGESIZE, MEM_COMMIT, PAGE_READWRITE);
                if (lpvResult == NULL)
                {
                    cout << "Speicher kann nicht zugewiesen "
                        "werden.\n";
                    exit(1);
                }
                pdMatrix[x][y] = d;
            }
        }
    }

    cout << "Matrix angelegt, " << n << " Seiten verwendet.\n";
    cout << "Zugewiesene Bytes: " << n * PAGESIZE << '\n';

    for(;;)
    {
        cout << "    Zeilenindex eingeben: ";
        cout.flush();
        cin >> x;
        cout << "    Spaltenindex eingeben: ";
        cout.flush();
        cin >> y;

        try
        {
            d = pdMatrix[x][y];
        }
    }
}
```

```

    catch (...)
    {
        cout << "Ausnahmebehandlung aufgerufen.\n";
        d = 0.0;
    }
    cout << "MATRIX[" << x << ', ' << y << "] = " << d << '\n';
}

```

Dieses Programm erzeugt eine Matrix von 10.000×10.000 `double`-Werten. Es reserviert jedoch nicht die gesamten 800.000.000 Byte, sondern lediglich den benötigten Speicher. Diese Vorgehensweise ist besonders für Matrizen geeignet, die wenige Elemente enthalten, deren Wert ungleich Null ist. In dem Beispiel werden lediglich 10 der 100.000.000 Elemente Zufallswerten zugewiesen.

Die Anwendung reserviert für die Matrix 800.000.000 Byte, die jedoch keinem physikalischen Speicher zugewiesen werden. Anschließend bestimmt sie für zehn Elemente der Matrix Zufallswerte. Ist eines dieser Elemente in einer virtuellen Speicherseite enthalten, die noch keinem physikalischen Speicher zugewiesen wurde, tritt ein Ausnahmefehler auf. Der Fehler wird mit einer C++-Fehlerbehandlungsroutine abgefangen. Die Behandlungsroutine prüft, ob der zuzuweisende Wert ungleich Null ist. In diesem Fall wird die Seite physikalischem Speicher zugeordnet und die Wertzuweisung wiederholt.

In diesem einfachen Beispiel gehen wir davon aus, daß der abgefangene Ausnahmefehler immer ein Win32-Fehler ist, der eine Speicherzugriffsverletzung anzeigt. Innerhalb komplexer Programme ist diese Annahme nicht in jedem Fall richtig. Solche Anwendungen erfordern eine komplizierte Fehlerbehandlungsroutine, um Ausnahmefehler zuverlässig zu erkennen.



In dem letzten Abschnitt des Programms wird der Anwender dazu aufgefordert, einen Zeilen- und Spaltenindex einzugeben. Die Anwendung versucht anschließend, den Wert des entsprechenden Matrixelements zu ermitteln. Befindet sich das Element in einer nichtzugewiesenen Seite, tritt erneut ein Ausnahmefehler auf. Diesmal wird der Fehler jedoch so interpretiert, daß das ausgewählte Element den Wert 0 enthält.

Die Anwenderschnittstellenschleife des Beispiels enthält keine Option zum Beenden des Programms. Betätigen Sie dazu die Tastenkombination **Strg** + **C**.

Die Ausgabe des Programms könnte wie folgt aussehen:

```

MATRIX[41,8467] = 6334
MATRIX[6500,9169] = 15724
MATRIX[1478,9358] = 26962

```

```

MATRIX[4464,5705] = 28145
MATRIX[3281,6827] = 9961
MATRIX[491,2995] = 11942
MATRIX[4827,5436] = 32391
MATRIX[4604,3902] = 153
MATRIX[292,2382] = 17421
MATRIX[8716,9718] = 19895
Matrix angelegt, 10 Seiten verwendet.
Zugewiesene Bytes: 40960
  Zeilenindex eingeben: 41
Spaltenindex eingeben: 8467
MATRIX[41,8467] = 6334
  Zeilenindex eingeben: 41
Spaltenindex eingeben: 8400
MATRIX[41,8400] = 0
  Zeilenindex eingeben:

```

14.4.3 Heap-Funktionen

HeapCreate Zusätzlich zu dem Standard-Heap können Prozesse weitere Heaps mit der Funktion `HeapCreate` erzeugen. Heap-Verwaltungsfunktionen reservieren Speicherblöcke in einem neu erstellten privaten Heap, oder geben diese Blöcke frei.

Ein Beispiel für die Verwendung dieser Funktionen ist die Erstellung eines neuen Heaps während des Programmstarts. Für diesen Heap wird eine Größe bestimmt, die den Speicheranforderungen der Anwendung gerecht wird. Ein mißlungener Versuch der Erstellung des Heaps kann dazu führen, daß der entsprechende Prozeß beendet wird. Wird `HeapCreate` jedoch erfolgreich ausgeführt, kann der Prozeß den erforderlichen Speicher nutzen.

HeapAlloc Nachdem ein Heap mit `HeapCreate` erzeugt wurde, können die verschiedenen Prozesse Speicher darin reservieren. Dies geschieht mit `HeapAlloc`. `HeapRealloc` verändert die Größe des zuvor reservierten Speicherblocks, und `HeapFree` gibt die Speicherblöcke wieder frei. Die Größe eines reservierten Blocks kann mit `HeapSize` ermittelt werden.

Beachten Sie bitte, daß sich der mit `HeapAlloc` reservierte Speicher nicht von dem Speicher unterscheidet, der mit den Standardfunktionen `GlobalAlloc`, `GlobalLock` oder `malloc` reserviert wurde.

Heap-Verwaltungsfunktionen können außerdem für den Standard-Heap eines Prozesses verwendet werden. Der Handle des Standard-Heaps wird mit `GetProcessHeap` ermittelt. Die Funktion `GetProcessHeaps` gibt eine Liste aller Heap-Handles zurück, die der Prozeß besitzt.

HeapDestroy `HeapDestroy` zerstört einen Heap. Diese Funktion sollte nicht auf dem mit `GetProcessHeap` ermittelten Standard-Heap eines Prozesses ausgeführt werden. (Das Zerstören des Standard-Heaps führt zu einer Zerstörung des Anwendungsstacks, der globalen und automatischen Variablen usw.)

Die Funktion `HeapCompact` versucht, den angegebenen Heap zu komprimieren, indem beieinanderliegende freie Speicherblöcke mit freien, nicht zugewiesenen Speicherblöcken verbunden werden. Beachten Sie bitte, daß Objekte, die in dem Heap mit `HeapAlloc` reserviert wurden, nicht verschoben werden können. Der Heap kann daher sehr einfach fragmentieren. `HeapCompact` kann solche Heaps nicht defragmentieren.

14.4.4 Weitere und veraltete Funktionen

Zusätzlich zu den bereits beschriebenen API-Funktionen bestehen weitere Funktionen, auf die der Win32-Programmierer zurückgreifen kann. Einige Funktionen, die unter Windows 3.1 erhältlich waren, wurden entfernt oder sind veraltet.

Funktionen zur Speichermanipulation sind `CopyMemory`, `FillMemory`, `MoveMemory` und `ZeroMemory`. Diese Funktionen bilden die Korrelate zu den C-Laufzeitfunktionen `memcpy`, `memmove` und `memset`.

Einige Windows-API-Funktionen prüfen, ob ein gegebener Zeiger eine bestimmte Art des Zugriffs auf eine Adresse oder einen Adreßbereich unterstützt. Diese Funktionen sind mit `IsBadCodePtr`, `IsBadStringPtr`, `IsBadReadPtr` und `IsBadWritePtr` bezeichnet. Für die beiden zuletzt genannten Funktionen bestehen Huge-Versionen (`IsBadHugeReadPtr`, `IsBadHugeWritePtr`), die die Abwärtskompatibilität zu Windows 3.1 gewährleisten.

Informationen über den freien Speicher können mit `GlobalMemoryStatus` ermittelt werden. Diese Funktion ersetzt die veraltete Funktion `GetFreeSpace`.

Andere veraltete Funktionen manipulieren Selektoren (z.B. `AllocSelector`, `ChangeSelector`, `FreeSelector`), den Stack des Prozessors (`SwitchStackBack`, `SwitchStackTo`), Segmente (`LockSegment`, `UnlockSegment`) oder den MS-DOS-Speicher (`GlobalDOSAlloc`, `GlobalDOSFree`).

14.4.5 Speicherzuordnungsdateien und gemeinsamer Speicher

Anwendungen können nicht wie bisher mit dem Flag `GMEM_DDESHARE` über den globalen Speicher miteinander kommunizieren. Statt dessen müssen sie Speicherzuordnungsdateien verwenden, um einen gemeinsamen Speicher verwenden zu können. Was aber sind Speicherzuordnungsdateien?

Seitendateien Der Mechanismus des virtuellen Speichers ermöglicht dem Betriebssystem, einen nicht existierenden Speicher einer Datei zuzuordnen, die Seitendatei genannt wird. Der virtuelle Speicher ist somit der Inhalt der Seitendatei. Der Zugriff auf die Seitendatei geschieht über Zeiger, so als wäre die Datei ein Speicherobjekt. Der Mechanismus weist somit den Inhalt der Seitendatei bestimmten Speicheradressen zu. Diese Vorgehensweise ist nicht nur für die Seitendatei, sondern auch für die sogenannten Speicherzuordnungsdateien möglich.

Speicherzuordnungsdateien Sie erzeugen eine Speicherzuordnungsdatei mit `CreateFileMapping`. Sie können außerdem die Funktion `OpenFileMapping` verwenden, um eine bereits bestehende Speicherzuordnungsdatei zu öffnen. Die Funktion `MapViewOfFile` weist einem virtuellen Speicherblock den Abschnitt einer Zuordnungsdatei zu.

Eine Speicherzuordnungsdatei kann von den Anwendungen als ein gemeinsamer Speicher verwendet werden. Öffnen beispielsweise zwei Anwendungen dieselbe Speicherzuordnungsdatei, erzeugen sie auf diese Weise einen gemeinsamen Speicherblock.

Natürlich wäre es unangebracht, eine Speicherzuordnungsdatei für einen kleinen Speicherbereich zu erstellen, den sich einige Anwendungen teilen sollen. Es ist daher nicht erforderlich, explizit eine Speicherzuordnungsdatei zu öffnen und zu verwenden, um einen gemeinsamen Speicher zu erzeugen. Anwendungen können der Funktion `CreateFileMapping` den besonderen Handle `0xFFFFFFFF` übergeben, um einen Bereich der Seitendatei des Systems zu nutzen.

Die nachfolgenden Listings demonstrieren den Einsatz gemeinsamer Speicherobjekte für die Intertask-Kommunikation. Sie implementieren einen einfachen Mechanismus, in dem ein Programm, der Client, eine einfache Nachricht für eine andere Anwendung im gemeinsamen Speicher hinterläßt. Das andere Programm, der Server, nimmt die Nachricht entgegen und gibt diese aus. Die Programme wurden für die Windows-95- und Windows-NT-Kommandozeile geschrieben. Öffnen Sie zwei DOS-Fenster, starten Sie das Server-Programm in dem ersten und das Client-Programm in dem zweiten Fenster, und sehen Sie, wie der Client die Nachricht an den Server sendet, der wiederum die Nachricht ausgibt.

Listing 14.2:
Intertask-
Kommunikation
über den
gemeinsamen
Speicher (Server)

```
#include <iostream.h>
#include <windows.h>
void main(void)
{
    HANDLE hmmf;
    LPSTR lpMsg;
    hmmf = CreateFileMapping((HANDLE)0xFFFFFFFF, NULL,
                           PAGE_READWRITE, 0, 0x1000, "MMFDEMO");

    if (hmmf == NULL)
    {
```

```

        cout << "Failed to allocated shared memory.\n";
        exit(1);
    }
    lpMsg = (LPSTR)MapViewOfFile(hmmf, FILE_MAP_WRITE, 0, 0, 0);
    if (lpMsg == NULL)
    {
        cout << "Failed to map shared memory.\n";
        exit(1);
    }
    lpMsg[0] = '\0';
    while (lpMsg[0] == '\0') Sleep(1000);
    cout << "Message received: " << lpMsg << '\n';
    UnmapViewOfFile(lpMsg);
}

#include <iostream.h>
#include <windows.h>
void main(void)
{
    HANDLE hmmf;
    LPSTR lpMsg;
    hmmf = CreateFileMapping((HANDLE)0xFFFFFFFF, NULL,
                           PAGE_READWRITE, 0, 0x1000, "MMFDEMO");
    if (hmmf == NULL)
    {
        cout << "Failed to allocated shared memory.\n";
        exit(1);
    }
    lpMsg = (LPSTR)MapViewOfFile(hmmf, FILE_MAP_WRITE, 0, 0, 0);
    if (lpMsg == NULL)
    {
        cout << "Failed to map shared memory.\n";
        exit(1);
    }
    strcpy(lpMsg, "This is my message.");
    cout << "Message sent: " << lpMsg << '\n';
    UnmapViewOfFile(lpMsg);
}

```

*Listing 14.3:
Intertask-Kom-
munikation über
den gemeinsa-
men Speicher
(Client)*

Die beiden Programme sind beinahe identisch. Sie erstellen zunächst eine Dateizuordnung mit der Bezeichnung MMFDEMO aus der Seitendatei des Systems. Nachdem die Zuordnung erstellt wurde, setzt der Server das erste Byte der Zuordnung auf Null und führt eine Warteschleife aus. In dieser Warteschleife wird jede Sekunde geprüft, ob das erste Byte der Zuordnung ungleich Null ist. Der Client hingegen hinterlegt eine Nachricht an dieser Position und wird daraufhin beendet. Bemerkt der Server, daß Daten vorhanden sind, gibt er das Ergebnis aus und wird ebenfalls beendet.

Beide Programme werden über die Kommandozeile mit den Anweisungen

CL MMFSRVR.CPP

und

CL MMFCLNT.CPP

kompiliert.

14.4.6 Gemeinsamer Speicher und Basis-Zeiger

Ein gemeinsames Speicherzuordnungsdateiobjekt muß nicht zwangsläufig in jedem Prozeß dieselbe Adresse erhalten. Gemeinsamen Speicherobjekten werden unter Windows 95, nicht jedoch unter Windows NT, identische Positionen im Adreßraum der Prozesse zugewiesen. Dies kann zu Problemen führen, wenn Anwendungen Zeiger in die gemeinsamen Daten einbinden möchten. Eine Lösung dieses Problems besteht darin, Basiszeiger zu verwenden und diese relativ zu dem Beginn des zugeordneten Bereichs zu setzen.

Basiszeiger sind spezifische Microsoft-Erweiterungen der C/C++-Sprache. Ein Basiszeiger wird mit dem Schlüsselwort `-based` deklariert, wie nachfolgend dargestellt:

```
void *vpBase;  
void -based(vpBase) *vpData;
```

Die Referenzen, die mit den Basiszeigern vorgenommen werden, verweisen immer auf Adressen von Daten, die relativ zu der angegebenen Basis angegeben werden. Basiszeiger sind auch dann sehr nützlich, wenn Daten gespeichert werden sollen, die Zeiger auf die Festplatte enthalten.

14.5 Threads und Speicherverwaltung

Für das Multithreading unter der 32-Bit-Version für Windows müssen einige Aspekte hinsichtlich der Speicherverwaltung berücksichtigt werden. Da Threads bisweilen konkurrierend auf dasselbe Objekt im Speicher zugreifen, kann es geschehen, daß die Bearbeitung einer Variablen durch einen Thread von einem anderen Thread unterbrochen wird. Wir benötigen daher einen Synchronisierungsmechanismus, um derartige Situationen zu vermeiden. Außerdem ist es möglich, daß Threads über eine private anstelle einer gemeinsamen Kopie eines Datenobjekts verfügen möchten.

14.5.1 Synchronisierter Variablenzugriff

Das erste der genannten Probleme wird vorwiegend mit dem `SYNCHRONISIERTEN VARIABLENZUGRIFF` gelöst. Dieser Mechanismus ermöglicht einem Thread, den Wert einer Integer-Variablen zu verändern und das Ergebnis zu überprüfen, ohne von einem anderen Thread unterbrochen zu werden.

Wenn Sie eine Variable innerhalb eines Threads in der gewohnten Weise inkrementieren oder dekrementieren, kann ein anderer Thread den Wert dieser Variablen verändern, bevor der erste Thread diesen überprüfen kann. Die Funktionen `InterlockedIncrement` und `InterlockedDecrement` bieten daher eine Möglichkeit, einen 32-Bit-Wert zu inkrementieren oder zu dekrementieren und anschließend das Ergebnis zu prüfen. Eine dritte Funktion, die mit `InterlockedExchange` bezeichnet ist, verändert den Wert einer Variablen und ermittelt den alten Wert, ohne von einem anderen Thread unterbrochen zu werden.

14.5.2 Lokaler Thread-Speicher

Während automatische Variablen für die Funktion, in der sie deklariert wurden, immer lokal sind, gilt dieser Umstand nicht für globale und statische Objekte. Greift Ihr Programmcode überwiegend auf solche Objekte zu, wird es sehr schwierig sein, Ihre Anwendung Thread-sicher zu machen.

Glücklicherweise bietet die Win32-API eine Möglichkeit, LOKALEN THREAD-SPEICHER zu reservieren. Die Funktion `TlsAlloc` reserviert einen TLS-Index (THREAD-LOCAL STORAGE), der vom Typ `DWORD` ist. Threads können diesen Speicher verwenden, um beispielsweise einen Zeiger auf einen privaten Speicherblock abzulegen. Sie verwenden dazu die Funktionen `TlsSetValue` und `TlsGetValue`. `TlsFree` gibt den TLS-Index frei.

Wenn Ihnen diese Vorgehensweise zu kompliziert erscheint, können Sie einen einfacheren Mechanismus verwenden, der von dem Visual-C++-Compiler angeboten wird. Objekte können mit dem Typmodifizierer `thread` als lokaler Thread deklariert werden, wie nachfolgend dargestellt:

```
-declspec(thread) int i;
```

Das Verwenden von `-declspec(thread)` gestaltet sich in DLLs als äußerst kompliziert, da die Erweiterung der globalen Speicherreservierung einer DLL zur Laufzeit für die in der DLL enthaltenen Thread-Objekte nicht problemlos möglich ist. Verwenden Sie in DLLs statt dessen die TLS-APIs, wie z.B. `TlsAlloc`.

14.6 Zugriff auf den physikalischen Speicher und die E/A-Schnittstellen

Programmierer der 16-Bit-Version von Windows waren damit vertraut, direkt auf den physikalischen Speicher oder die Eingabe-/Ausgabe-Schnittstellen zuzugreifen. So war beispielsweise eine 16-Bit-Anwendung möglich, die über die dem Speicher zugeordneten Ein- und Ausgabeschnittstellen auf die Hardware-Geräte zugreifen konnte.

Für 32-Bit-Betriebssysteme ist diese Vorgehensweise nicht mehr möglich. Win32 ist ein plattformunabhängiges Betriebssystem. Alle plattformabhängigen Elemente sind vollständig inkompatibel mit dem Betriebssystem. Dazu zählen alle Arten des Zugriffs auf die physikalische Hardware, wie z.B. Schnittstellen, physikalische Speicheradressen usw.

Wie aber schreiben Sie Anwendungen, die direkt mit der Hardware kommunizieren können? Dazu benötigen Sie eines der verschiedenen DDKs (Device Driver Kits). Über ein DDK kann eine Treiber-Bibliothek erzeugt werden, die den gesamten Low-Level-Zugriff auf das Gerät enthält und Ihre High-Level-Anwendung von allen Plattformabhängigkeiten befreit.

14.7 Zusammenfassung

Die Speicherverwaltung von Win32 weist deutliche Unterschiede zur Speicherverwaltung der 16-Bit-Version von Windows auf. Entwickler müssen sich nicht wie bisher mit der untergliederten Intel-Architektur beschäftigen. Neue Möglichkeiten bedeuten jedoch ebenfalls neue Verpflichtungen für den Programmierer.

Win32-Anwendungen werden in separaten Adreßräumen ausgeführt. Der Zeiger einer Anwendung ist für eine andere Anwendung unbedeutend. Alle Anwendungen können über 32-Bit-Adressen auf 4 Gbyte Adreßraum zugreifen (obwohl die unterschiedlichen Win32-Implementierungen bestimmte Bereiche dieses Adreßraums für eigene Zwecke reservieren).

Ein Win32-Betriebssystem verwendet die virtuelle Speicherverwaltung, um eine logische Adresse in dem Adreßraum einer Anwendung einer physikalischen Adresse im Speicher oder einem Datenblock in der Auslagerungs- oder Seitendatei des Systems zuzuweisen. Anwendun-

gen können die Möglichkeiten der virtuellen Speicherverwaltung explizit nutzen, um Speicherzuordnungsdateien zu erstellen oder um große virtuelle Speicherblöcke zu reservieren.

Speicherzuordnungsdateien bieten einen effizienten Mechanismus zur Intertask-Kommunikation. Durch den Zugriff auf dasselbe Speicherzuordnungsdateiobjekt können zwei oder mehrere Anwendungen diese Datei als gemeinsamen Speicher nutzen.

Spezielle Features berücksichtigen die Probleme der Speicherverwaltung in Threads. Threads können über den synchronisierten Variablenzugriff verschiedene Aufgaben für gemeinsame Objekte ausführen. Über den lokalen Thread-Speicher können Threads private Objekte im Speicher reservieren.

Einige der alten Windows- und DOS-Speicherverwaltungsfunktionen sind nicht mehr erhältlich. Anwendungen können infolge der Plattformunabhängigkeit von Win32 nicht wie bisher direkt auf den physikalischen Speicher zugreifen. Ist ein direkter Zugriff auf die Hardware erforderlich, muß möglicherweise das entsprechende DDK verwendet werden.

Dateiverwaltung

Kapitel

15

Die Win32-API bietet einige Funktionen und Konzepte für den Zugriff und die Verwaltung von Dateien. Außerdem stehen Ihnen Funktionen zur Low-Level-Ein- und Ausgabe sowie zum Ein- und Ausgabestrom zur Verfügung, die in der C- und C++-Laufzeitbibliothek enthalten sind. Dieses Kapitel beschreibt alle Möglichkeiten der Dateiverwaltung, die 32-Bit-Anwendungen nutzen können.

Abbildung 15.1 erklärt die Beziehung zwischen der Low-Level-Ein- und Ausgabe von DOS/Unix, dem Ein- und Ausgabestrom von C/C++ und den Ein- und Ausgabe-Dateifunktionen von Win32.

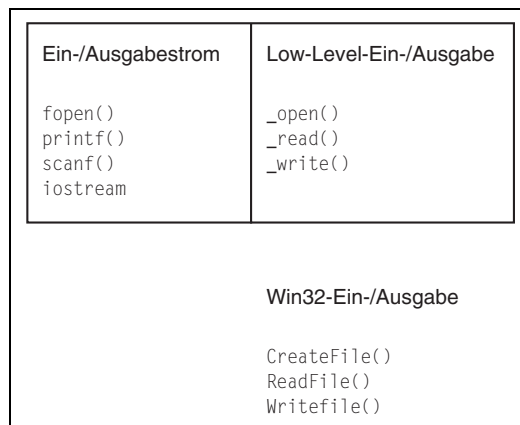


Abbildung 15.1:
Ein- und Ausgabefunktionen

Win32-Anwendungen sollten die Ein- und Ausgabe-Dateifunktionen von Win32 verwenden, die einen vollständigen Zugriff auf die Sicherheits-Features und auf andere Attribute ermöglichen. Diese Funktionen gestatten außerdem asynchrone und überlappte Ein- und Ausgabeoperationen.

15.1 Übersicht über das Dateisystem

Eine gewöhnliche Datei ist eine Datensammlung, die auf einem beständigen Medium, wie z.B. einer Diskette, gespeichert wird. Dateien sind in DATEISYSTEMEN organisiert. Dateisysteme implementieren ein bestimmtes Schema zum Speichern von Dateien auf einem physikalischen Medium und zur Darstellung verschiedener Dateiattribute, wie Dateinamen, Genehmigungen und Besitzinformationen.

Informationen über das Dateisystem werden mit der Funktion `GetVolumeInformation` ermittelt. Daten über den Typ des Speichergeräts erhalten Sie mit einem Aufruf von `GetDriveType`.

15.1.1 Unterstützte Dateisysteme

Windows NT kennt drei Dateisysteme (das OS/2-High-Performance-Dateisystem HPFS wird nicht mehr unterstützt).

- ➔ Das FAT-Dateisystem (File Allocation Table) ist kompatibel zu früheren Versionen von MS-DOS.
- ➔ NTFS (New Technology File System) ist das »native« Dateisystem von Windows NT.
- ➔ VFAT, die Erweiterung des FAT-Dateisystems, das im Protected-Mode ausgeführt wird, unterstützt lange Dateinamen und ist zu früheren Versionen von MS-DOS kompatibel.

Windows 95 kann NTFS-Laufwerke nicht bearbeiten. Seit Windows 98 unterstützt das Betriebssystem jedoch das FAT32-Dateisystem, eine erweiterte, stabile und flexible Variante des alten FAT-Systems.

Aus der Sicht einer Anwendung besteht der wesentliche Unterschied zwischen den Dateisystemen in der Unterstützung bestimmter Attribute. NTFS-Laufwerke bieten beispielsweise das Konzept des Dateibesitzes und Sicherheitsattribute, die ein FAT-Dateisystem nicht zur Verfügung stellt.

15.1.2 CD-ROMs

ISO-9660-CD-ROM-Laufwerke präsentieren sich den Anwendungen als gewöhnliche Laufwerke. Sowohl Windows NT als auch Windows 95 unterstützen lange Dateinamen.

15.1.3 Netzwerklaufwerke

Windows unterstützt gemeinsame Dateien in einem Netzwerk. Netzwerk-Dateisysteme können über lokale Laufwerksbuchstaben und Netzwerkweiterleitung angesprochen werden. Anwendungen können alternativ dazu über UNC-Namen (Universal Naming Convention) auf Dateien in einem Netzwerk zugreifen. Ein Beispiel für einen UNC-Namen ist `\\server\vol1\myfile.txt`. Einige Netzwerke unterstützen lange Dateinamen.

15.1.4 Datei- und Laufwerkskomprimierung

Windows NT verfügt seit Version 3.51 über die Dateikomprimierung auf NTFS-Laufwerken. Windows 95/98 hingegen unterstützt die DriveSpace-Komprimierung von FAT-Laufwerken. Diese beiden Komprimierungsmechanismen sind leider nicht kompatibel miteinander. Windows NT kann lediglich auf nichtkomprimierte FAT-Laufwerke zugreifen.

15.2 Win32-Dateiobjekte

Unter der 32-Bit-Version von Windows wird eine geöffnete Datei wie ein Objekt des Betriebssystems behandelt. Der Zugriff darauf geschieht über einen Win32-Handle. Diese sollte nicht mit den »Dateihandles« von DOS und Unix verwechselt werden. Diese vom Betriebssystem zugewiesenen Integer-Werte repräsentieren geöffnete Dateien.

Da eine Datei ein Kernel-Objekt ist, sind zusätzlich zu den Dateisystemfunktionen weitere Operationen möglich, die auf den Handles dieser Objekte basieren. So kann beispielsweise die Funktion `WaitForSingleObject` auf einem Datei-Handle ausgeführt werden, der für die Konsolen-Ein- und Ausgabe geöffnet wurde.

15.2.1 Erstellen und Öffnen von Dateien

Ein Dateiobjekt wird mit `CreateFile` erstellt. Diese Funktion kann sowohl zum Erstellen einer neuen als auch zum Öffnen einer bereits bestehenden Datei verwendet werden. Hinsichtlich der genannten Funktionalität scheint die Bezeichnung der Prozedur nicht richtig zu sein. Denken Sie jedoch daran, daß ein neues DATEIOBJEKT erzeugt wird, das entweder eine neue oder eine bestehende Datei auf dem Speichermedium repräsentiert.

CreateFile

Die Parameter der Funktion bestimmen den Zugriffsmodus (lesen oder schreiben), den gemeinsamen Dateimodus, Sicherheitsattribute, Erstellungs-Flags, Dateiattribute und eine optionale Datei, die als Attributvorlage dient.

Möchten Sie beispielsweise die Datei `C:\README.TXT` zum Lesen öffnen, können Sie den folgenden Aufruf der Funktion `CreateFile` vornehmen:

```
hReadme = CreateFile("C:\\README.TXT", GENERIC_READ, 0, NULL,  
                    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
```

- Der erste Parameter ist der Dateiname. Anwendungen können ebenfalls den UNC-Namen verwenden. Die Länge des Dateinamens ist durch den Wert in der Konstante `MAX_PATH` begrenzt. Unter Windows NT kann diese Begrenzung ignoriert werden, wenn Sie dem Pfad die Zeichen `"\\?\"` voranstellen und die Wide-Version von `CreateFile` mit der Bezeichnung `CreateFileW` aufrufen. Das Präfix `"\\?\"` weist das Betriebssystem an, den Pfadnamen nicht zu analysieren.
- Der vierte Parameter vom Typ `LPSECURITY_ATTRIBUTES` ist besonders interessant. Über diesen Parameter ermitteln Anwendungen die Sicherheitsattribute des neuen Dateiobjekts und können die Attribute für neu erstellte Dateien bestimmen. Diese Funktionalität ist lediglich unter Windows NT für ein NTFS-Laufwerk erhältlich. Ein nützliches Element der `SECURITY_ATTRIBUTES`-Struktur mit der Bezeichnung `bInheritHandle` kontrolliert, ob sich der Handle eines Objekts von den Child-Prozessen ableitet.



Win32-Anwendungen sollten zum Öffnen von Dateien nicht die Funktion `OpenFile` verwenden. Diese Funktion wird lediglich aus Gründen der Kompatibilität zur 16-Bit-Version von Windows zur Verfügung gestellt.

Eine geöffnete Datei wird mit der Funktion `CloseHandle` geschlossen.

15.2.2 Einfache Ein- und Ausgabe

Die Ein- und Ausgabe geschieht mit Hilfe der Funktionen `ReadFile` und `WriteFile`. Das einfache Beispiel in Listing 15.1 wird mit

```
CL FILECOPY.CPP
```

kompiliert. Es kopiert den Inhalt einer Datei in eine andere Datei. Es verwendet dazu die Funktionen `CreateFile`, `ReadFile` und `WriteFile`.

```

#include <windows.h>
#include <iostream.h>

void main(int argc, char *argv[])
{
    HANDLE hSrc, hDest;
    DWORD dwRead, dwWritten;
    char pBuffer[1024];

    if (argc != 3)
    {
        cout << "Aufruf: FILECOPY Quelldatei Zieldatei\n";
        exit(1);
    }

    hSrc = CreateFile(argv[1], GENERIC_READ, 0, NULL,
                     OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
    if (hSrc == INVALID_HANDLE_VALUE)
    {
        cout << "Oeffnen nicht moeglich: " << argv[1] << '\n';
        exit(1);
    }

    hDest = CreateFile(argv[2], GENERIC_WRITE, 0, NULL,
                       CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
    if (hDest == INVALID_HANDLE_VALUE)
    {
        cout << "Erzeugen nicht moeglich: " << argv[2] << '\n';
        CloseHandle(hSrc);
        exit(1);
    }

    do
    {
        ReadFile(hSrc, pBuffer, sizeof(pBuffer), &dwRead, NULL);
        if (dwRead != 0)
            WriteFile(hDest, pBuffer, dwRead, &dwWritten, NULL);
    } while (dwRead != 0);
    CloseHandle(hSrc);
    CloseHandle(hDest);
}

```

*Listing 15.1:
Kopieren einer
Datei mit Hilfe
der Win32-Datei-
funktionen*

Win32 stellt für Dateien mit wahlfreiem Zugriff die Funktion `SetFilePointer` zur Verfügung, um den Dateizeiger vor dem Lese- oder Schreibvorgang zu setzen. Der Dateizeiger ist ein 64-Bit-Wert, der die Position der nächsten Schreib-/Leseoperation innerhalb einer Datei bestimmt. Die Funktion wird fehlerhaft ausgeführt, wenn ihr der Handle eines Geräts übergeben wird, auf dem keine Suchoperationen ausgeführt werden können. Dazu zählen die Konsole oder eine Kommunikationsschnittstelle.

`SetFilePointer` wird außerdem dazu verwendet, den aktuellen Wert des Dateizeigers zu ermitteln. Der Aufruf

```
dwPos = SetFilePointer(hFile, 0, NULL, FILE_CURRENT);
```

bewegt den Dateizeiger nicht, gibt jedoch dessen gegenwärtigen Wert zurück.

15.2.3 Asynchrone Ein- und Ausgabeoperationen

Während der Programmierung interaktiver Anwendungen, die Datei-Ein- und Ausgabefunktionen ausführen, werden Sie wiederholt mit Reaktionsproblemen konfrontiert. Gewöhnliche Dateisystemaufrufe sind BLOCKIERENDE AUFRUFE. Ein Aufruf der Funktion `scanf` ist beispielsweise erst dann beendet, wenn genügend Zeichen in dem Eingabepuffer des Betriebssystems enthalten sind. Dies ist besonders ein Problem der schnellen, auf Festplatten basierenden Dateisysteme. Wird die Eingabeoperation auf einer Kommunikationsschnittstelle ausgeführt, verringert dies zusätzlich die Performance.

Die 32-Bit-Version von Windows bietet einige Lösungen für dieses Problem. So können Sie beispielsweise mehrere Threads verwenden. Ein Thread könnte die Eingabefunktion ausführen und blockiert bleiben, ohne die Reaktion der Anwenderschnittstelle des Programms zu beeinflussen, die von einem anderen Thread verwaltet wird. Ein einfaches Kommunikationsprogramm, das Multithreading verwendet, ist in Listing 15.2 aufgeführt. Kompilieren Sie dieses Programm mit der Kommandozeilenanweisung

CL COMMTHRD.C

(Die Verwendung der Konsole und Kommunikationsschnittstellen wird später in diesem Kapitel erörtert.)

*Listing 15.2:
Ein einfaches
Kommunikationsprogramm,
das mehrere
Threads verwendet*

```
#include <windows.h>
volatile char cWrite;
DWORD WINAPI ReadComm(LPVOID hCommPort)
{
    HANDLE hConOut;
    DWORD dwCount;
    char c;
    hConOut = CreateFile("CONOUT$", GENERIC_WRITE,
                        FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL, 0);

    while (TRUE)
    {
        ReadFile((HANDLE)hCommPort, &c, 1, &dwCount, NULL);
        if (dwCount == 1)
            WriteFile(hConOut, &c, 1, &dwCount, NULL);
        if (cWrite)
        {
            if (cWrite == 24) break;
            c = cWrite;
            WriteFile(hCommPort, &c, 1, &dwCount, NULL);
            cWrite = '\0';
        }
    }
    CloseHandle(hConOut);
    return 0;
}

void main(void)
{
    HANDLE hConIn, hCommPort;
```

```

HANDLE hThread;
DWORD dwThread;
DWORD dwCount;
COMMTIMEOUTS ctmoCommPort;
DCB dcbCommPort;
char c;
hConIn = CreateFile("CONIN$", GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ, NULL, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, 0);
SetConsoleMode(hConIn, 0);
hCommPort = CreateFile("COM2", GENERIC_READ|GENERIC_WRITE,0,
    NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0);
memset(&ctmoCommPort, 0, sizeof(ctmoCommPort));
ctmoCommPort.ReadTotalTimeoutMultiplier = 1;
SetCommTimeouts(hCommPort, &ctmoCommPort);
dcbCommPort.DCBlength = sizeof(DCB);
GetCommState(hCommPort, &dcbCommPort);
SetCommState(hCommPort, &dcbCommPort);
hThread = CreateThread(NULL, 0, ReadComm, (LPDWORD)hCommPort,
    0, &dwThread);

while (TRUE)
{
    ReadFile(hConIn, &c, 1, &dwCount, NULL);
    cWrite = c;
    if (c == 24) break;
    while (cWrite);
}
if (WaitForSingleObject(hThread, 5000) == WAIT_TIMEOUT)
    TerminateThread(hThread, 1);
CloseHandle(hConIn);
CloseHandle(hCommPort);
}

```

Dieses Programm verwendet zur Kommunikation die Schnittstelle COM2. Damit Sie das Programm ausführen können, muß ein Modem an dieser Schnittstelle angeschlossen sein. Sollte Ihr Modem an einer anderen Schnittstelle angeschlossen sein, ändern Sie bitte die Bezeichnung der Schnittstelle in dem zweiten Aufruf der Funktion `CreateFile` und kompilieren die Anwendung erneut. Starten Sie das Programm, und geben Sie einige Modem-Anweisungen ein (z.B. `ATI1`). Das Modem sollte darauf mit den entsprechenden Meldungen reagieren.

Nachdem COM2 zum Lesen und Schreiben und die Konsole für die Eingabe geöffnet wurde, initialisiert die Anwendung die Schnittstelle. Ein Abschnitt der Initialisierung richtet einen Lese-Timeout mit einer Länge von einer Millisekunde ein. Überdies werden Kommunikationen über die DCB-Struktur initialisiert. (Ohne die Aufrufe von `GetCommState`/`SetCommState` kann die Schnittstelle unter Windows 95/98 nicht korrekt angesprochen werden.) Nach der Initialisierung der Schnittstelle erstellt das Programm einen sekundären Thread, der die Konsole zum Schreiben öffnet. Dieser Thread bearbeitet die Ein- und Ausgabe über die Schnittstelle in einer Schleife, während der erste Thread die gleiche Aufgabe für die Konsole ausführt. Ermittelt der primäre Thread die Eingabe eines Zeichens über die Konsole, legt er das Zeichen in einer globalen Variablen ab und wartet, bis der sekundäre Thread das Zei-

chen bearbeitet hat. Erhält der sekundäre Thread ein Zeichen von der Kommunikationsschnittstelle, übermittelt er dieses Zeichen an die Konsole.



Eine bessere Lösung bestünde darin, die globale Variable `cWrite` aus dem Programmcode zu entfernen und in dem primären Thread die Daten direkt an die Kommunikationsschnittstelle zu senden. Diese Vorgehensweise würde uns außerdem ermöglichen, auf den Lese-Timeout zu verzichten, wodurch der sekundäre Thread sehr viel effizienter arbeiten würde. Dies ist jedoch nicht möglich: Wird die Kommunikationsschnittstelle für eine synchrone Ein- und Ausgabe geöffnet und ein blockierender Aufruf in einem Thread vorgenommen (beispielsweise ein Aufruf der Funktion `ReadFile`, der erst dann beendet ist, wenn ein Zeichen eingelesen wurde), sind alle anderen Aufrufe für diese Schnittstelle ebenfalls blockiert. Daher ist es beispielsweise nicht möglich, einen Aufruf der blockierenden Funktion `ReadFile` in einem Thread Ihrer Anwendung vorzunehmen, während die Funktion `WriteFile` wiederholt in einem anderen Thread ausgeführt wird. (Diese Technik arbeitet unter Windows 95/98, nicht jedoch unter Windows NT.)

Die Schleifen werden mit der Tastenkombination `[Strg] + [X]` verlassen.

Überlappende Ein- und Ausgabe

Obwohl das Verwenden mehrerer Threads eine mögliche Option für 32-Bit-Anwendungen ist, stellt diese Vorgehensweise nicht die günstigste Lösung dar. 32-Bit-Anwendungen können ebenfalls die überlappende Ein- und Ausgabe verwenden. Diese Technik ermöglicht einer Anwendung, eine nicht blockierte Ein- und Ausgabeoperation zu initiieren. Verwendet eine Anwendung beispielsweise die Funktion `ReadFile` für die überlappte Eingabe, kehrt die Funktion auch dann zurück, wenn die Eingabeoperation noch nicht beendet ist. Nachdem die Eingabe abgeschlossen ist, ermittelt die Anwendung mit der Funktion `GetOverlappedResult` die Ergebnisse. Überlappende Ein- und Ausgabeoperationen können ebenfalls mit den Funktionen `ReadFileEx` und `WriteFileEx` vorgenommen werden.



Überlappende Ein- und Ausgabeoperationen können unter Windows 95/98 nicht für Dateien ausgeführt werden.

Die überlappte Eingabe kann außerdem in Kombination mit einem Synchronisierungsereignis verwendet werden. Prozesse erhalten über Synchronisierungsereignisse Nachrichten, wenn die Ein-/Ausgabeoperation beendet ist. Verwenden Sie Ereignisse zusammen mit der Funktion `WaitForMultipleObjects`, können Sie auf die Eingabe von mehreren Eingabegeräten gleichzeitig warten. Das zweite Kommunikationsprogramm in Listing 15.3 demonstriert diese Technik. Sie kompilieren die Anwendung über die Kommandozeile mit

CL COMMOVIO.C

```
#include <windows.h>
void main(void)
{
    HANDLE hConIn, hConOut, hCommPort;
    HANDLE hEvents[2];
    DWORD dwCount;
    DWORD dwWait;
    COMMTIMEOUTS ctmoCommPort;
    DCB dcbCommPort;
    OVERLAPPED ovr, ovw;
    INPUT_RECORD irBuffer;
    BOOL fInRead;
    char c;
    int i;
    hConIn = CreateFile("CONIN$", GENERIC_READ /*|GENERIC_WRITE*/,
                      FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
                      FILE_ATTRIBUTE_NORMAL, 0);
    SetConsoleMode(hConIn, 0);
    hConOut = CreateFile("CONOUT$", GENERIC_WRITE,
                      FILE_SHARE_READ, NULL, OPEN_EXISTING,
                      FILE_ATTRIBUTE_NORMAL, 0);
    hCommPort = CreateFile("COM2", GENERIC_READ|GENERIC_WRITE, 0,
                      NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL|
                      FILE_FLAG_OVERLAPPED, 0);
    memset(&ctmoCommPort, 0, sizeof(ctmoCommPort));
    SetCommTimeouts(hCommPort, &ctmoCommPort);
    dcbCommPort.DCBlength = sizeof(DCB);
    GetCommState(hCommPort, &dcbCommPort);
    SetCommState(hCommPort, &dcbCommPort);
    SetCommMask(hCommPort, EV_RXCHAR);
    memset(&ovr, 0, sizeof(ovr));
    memset(&ovw, 0, sizeof(ovw));
    ovr.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    hEvents[0] = ovr.hEvent;
    hEvents[1] = hConIn;
    fInRead = FALSE;
    while (1)
    {
        if (!fInRead)
            while (ReadFile(hCommPort, &c, 1, &dwCount, &ovr))
                if (dwCount == 1)
                    WriteFile(hConOut, &c, 1, &dwCount, NULL);
        fInRead = TRUE;
        dwWait =
            WaitForMultipleObjects(2, hEvents, FALSE, INFINITE);
        switch (dwWait)
        {
            case WAIT_OBJECT_0:
                if (GetOverlappedResult(hCommPort, &ovr, &dwCount,
```

*Listing 15.3:
Ein einfaches
Kommunikati-
onsprogramm,
das die über-
lappte Ein-/Aus-
gabe verwendet*

```

        FALSE))
        if (dwCount == 1)
            WriteFile(hConOut, &c, 1, &dwCount, NULL);
        fInRead = FALSE;
        break;
    case WAIT_OBJECT_0 + 1:
        ReadConsoleInput(hConIn, &irBuffer, 1, &dwCount);
        if (dwCount == 1 &&
            irBuffer.EventType == KEY_EVENT &&
            irBuffer.Event.KeyEvent.bKeyDown)
            for (i = 0;
                i < irBuffer.Event.KeyEvent.wRepeatCount; i++)
            {
                if (irBuffer.Event.KeyEvent.uChar.AsciiChar)
                {
                    WriteFile(hCommPort,
                        &irBuffer.Event.KeyEvent.uChar.AsciiChar,
                        1, &dwCount, &ovw);
                    if (irBuffer.Event.KeyEvent.uChar.AsciiChar
                        == 24) goto EndLoop;
                }
            }
        }
    }
}
EndLoop:
    CloseHandle(ovr.hEvent);
    CloseHandle(hConIn);
    CloseHandle(hConOut);
    CloseHandle(hCommPort);
}

```

Wie zuvor bereits erwähnt, sollte die Schnittstellenbezeichnung in dem zweiten Aufruf von `CreateFile` geändert und das Programm erneut kompiliert werden, wenn das Modem an einer anderen Schnittstelle als COM2 angeschlossen ist.

Das Programm öffnet zunächst die Konsole sowie die Kommunikationsschnittstelle und initialisiert die Schnittstelle. Während der Initialisierung wird die Funktion `SetCommMask` aufgerufen, wodurch die Ereignisbenachrichtigungen für die Schnittstelle gelesen werden können.

Die Kommunikationsschnittstelle wird mit dem `FILE_FLAG_OVERLAPPED`-Attribut geöffnet. Dies ermöglicht die überlappenden Ein- und Ausgabeoperationen. Innerhalb der Hauptschleife der Anwendung wird die Funktion `ReadFile` aufgerufen, der ein Zeiger auf eine `OVERLAPPED`-Struktur übergeben wird.

ReadFile `ReadFile` gibt sofort Daten zurück, wenn diese an der Schnittstelle liegen. Andernfalls zeigt `ReadFile` einen Fehler an. Mit `GetLastError` prüfen Sie, ob der Fehler `ERROR_IO_PENDING` aufgetreten ist. (Dieser Abschnitt wurde nicht in den Programmcode implementiert. Wir gehen davon aus, daß jeder von `ReadFile` angezeigte Fehler die Eingabe betrifft.)

Den Kern der Anwendung bildet der Aufruf von `WaitForMultipleObjects`. Die Funktion wartet auf zwei Objekte: ein Ereignisobjekt, das ein Abschnitt der `OVERLAPPED`-Struktur ist und zum Auslesen der Kommunikationsschnittstelle verwendet wird, sowie ein Konsolen-Eingabeobjekt. Für das zuletzt genannte Objekt muß keine überlappende Ein- und Ausgabe verwendet werden. Das Konsolenobjekt verfügt über einen eigenen Signalstatus, der anzeigt, daß Daten in dem Eingabepuffer der Konsole enthalten sind.

WaitForMultipleObjects

Der Rückgabewert von `WaitForMultipleObjects` zeigt an, daß Daten entweder an der Konsole oder an der Kommunikationsschnittstelle bereitstehen. Die folgende `switch`-Anweisung unterscheidet zwischen diesen beiden Möglichkeiten. Das Ermitteln des Konsolenereignisses erfordert einen etwas ungewöhnlichen Programmcode. Ein einfacher Aufruf von `ReadFile` genügt nicht, da dieser lediglich das Key-Down-Ereignis zurückgibt. Das Key-Up-Ereignis wird in dem Eingabepuffer der Konsole abgelegt und das Konsolenobjekt in den signalisierenden Status gesetzt. Ein anschließender Aufruf der Funktion `ReadFile` würde zu einem blockierenden Lesevorgang führen. Die Blockierung wäre erst dann aufgehoben, wenn nochmals eine Taste gedrückt und somit ein weiteres Key-Down-Ereignis ausgelöst würde. Es ist daher erforderlich, Low-Level-Konsolenfunktionen zum Ermitteln aller Konsolenereignisse zu verwenden, so daß nach einem erneuten Aufruf von `WaitForMultipleObjects` der Status des Konsolenobjekts nicht länger signalisierend ist.

15.3 Low-Level-Ein-/Ausgabe

Abbildung 15.1 verdeutlicht, daß die Bezeichnung Low-Level-Ein-/Ausgabe für die auf Dateibeschreibungen basierenden Ein- und Ausgabeoperationen falsch ist. Dieser Ausdruck ist ein Relikt aus den Zeiten von DOS und Unix. Obwohl Windows NT diese Funktionen aus Gründen der Kompatibilität mit diesen Betriebssystemen anbietet, sind sie bereits in `CreateFile`, `ReadFile` und `WriteFile` implementiert.

15.3.1 Dateibeschreibungen

Eine Dateibeschreibung ist ein Integer, der eine geöffnete Datei bezeichnet. Eine Dateibeschreibung wird ermittelt, wenn eine Anwendung die Funktionen `_open` oder `_creat` aufruft. In der Dokumentation zur Laufzeitbibliothek werden Dateibeschreibungen häufig als DateiHandles bezeichnet. Diese sollten nicht mit den Win32-Handles für Dateiobjekte verwechselt werden. Ein Handle, der von `CreateFile` zurückgegeben wurde, und eine mit `_open` ermittelte Dateibeschreibung sind nicht kompatibel.

15.3.2 Standarddateibeschreibung

Win32-Konsolenanwendungen können auf Ein- und Ausgabe-Standarddateibeschreibungen zugreifen. Diese sind in Tabelle 15.1 aufgeführt.

Tabelle 15.1:
Standarddatei-
beschreibungen

Dateibeschreibung	Stromname	Beschreibung
0	stdin	Standardeingabe
1	stdout	Standardausgabe
2	stderr	Standardfehler

Beachten Sie bitte, daß MS-DOS-Programmierer zwei weitere Datei-beschreibungen mit den Bezeichnungen `_stdprn` und `_stdaux` verwenden können. Diese Datei-beschreibungen sind nicht für Win32-Anwendungen erhältlich.

15.3.3 Low-Level-Ein-/Ausgabefunktionen

Eine Datei wird mit `_open` für die Low-Level-Ein- und Ausgabe geöffnet. Möchten Sie eine neue Datei für diese Ein- und Ausgabe erzeugen, rufen Sie bitte `creat` auf. Für diese Funktionen bestehen Unicode-Versionen, die Unicode-Dateinamen unter Windows NT akzeptieren und mit `_wopen` und `_wcreat` bezeichnet sind.

Das Lesen und Schreiben geschieht mit den Funktionen `_read` und `_write`. Die Suche innerhalb einer Datei wird mit `_lseek` ausgeführt. Die Funktion `_tell` ermittelt die aktuelle Position des Dateizeigers.

Der Inhalt eines Windows-Puffers wird mit einem Aufruf von `_commit` einer Datei zugewiesen. Die Datei wird mit `_close` geschlossen. Die Funktion `_eof` prüft, ob das Ende einer Datei erreicht wurde. Alle Low-Level-Ein- und Ausgabefunktionen verwenden die globale Variable `errno`, um Fehler anzuzeigen.

Die Bezeichnungen dieser Funktionen beginnen mit einem Unterstrich, der anzeigt, daß die Funktionen nicht in der ANSI-Standard-funktionsbibliothek enthalten sind. Für Anwendungen, die die alten Namen dieser Funktionen verwenden, stellt Microsoft die Bibliothek `oldnames.lib` zur Verfügung.

15.4 Ein-/Ausgabestrom

C/C++-Ein- und Ausgabestromfunktionen werden vorwiegend verwendet. Es gibt nicht viele Programmierer, die noch niemals `printf` oder einen `FILE`-Zeiger verwendet haben.

15.4.1 Der Ein- und Ausgabestrom unter C

C-Programme, die den Ein- und Ausgabestrom verwenden, nutzen die `FILE`-Struktur und die entsprechende Funktionsfamilie. Eine Datei wird mit `fopen` für den Ein-/Ausgabestrom geöffnet. Diese Funktion gibt einen Zeiger auf eine `FILE`-Struktur zurück, der für die nachfolgenden Operationen, wie z.B. in Aufrufen der Funktionen `fscanf`, `fprintf`, `fread`, `fwrite`, `fseek`, `ftell` oder `fclose`, benutzt werden kann. Die Visual-C++-Laufzeitbibliothek unterstützt alle Ein-/Ausgabestrom-Standardfunktionen sowie einige spezifische Microsoft-Funktionen.

Anwendungen, die Ein-/Ausgabestromfunktionen zusammen mit Low-Level-Ein-/Ausgabefunktionen verwenden, können mit `_fileno` eine Dateibeschreibung für einen gegebenen Strom ermitteln (der durch einen `FILE`-Zeiger bezeichnet ist). Die Funktion `_fdopen` öffnet einen Strom und weist diesem einer Dateibeschreibung zu, die eine zuvor geöffnete Datei bezeichnet.

Anwendungen können außerdem auf die Standardeingabe, Standardausgabe und den Standardfehler über die vordefinierten Ströme `stdin`, `stdout` und `stderr` zugreifen.

15.4.2 Ein- und Ausgabestrom unter C++ (die `iostream`-Klassen)

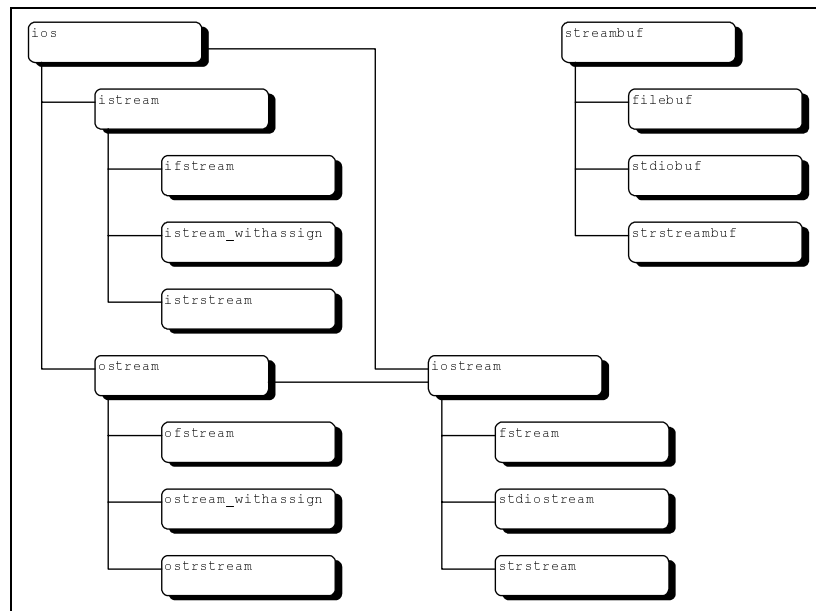
Die Visual-C++-Laufzeitbibliothek enthält eine vollständige Implementierung der C++-`iostream`-Klassen. Die `iostream`-Klassenbibliothek implementiert die in Abbildung 15.2 aufgeführten C++-Klassen.

Die Basisklasse der `iostream`-Klassen ist `ios`. Gewöhnlich leiten Anwendungen Klassen nicht direkt von `ios` ab. Sie verwenden statt dessen eine der abgeleiteten Klassen `istream` oder `ostream`.

Der Aufbau der `iostream`-Hierarchie wurde mittlerweile im ANSI-C++-Standard leicht abgewandelt. So sind die Klassen `istream_withassign` und `ostream_withassign` nicht mehr vorgesehen.



Abbildung 15.2:
Die C++-
iostream-Klassen



Varianten der istream-Klassen sind istrstream (für ein Array aus Zeichen im Speicher), ifstream (für Dateien) und istream_withassign (eine Variante von istream).

Varianten der ostream-Klassen sind ostrstream (Stromausgabe in ein Zeichenarray), ofstream (Ausgabe in eine Datei) und ostream_withassign (eine Variante von ostream).

Das vordefinierte Stromobjekt `cin`, das die Standardeingabe repräsentiert, ist vom Typ `istream_withassign` (nach ANSI-C++ vom Typ `istream`). Die vordefinierten Objekte `cout`, `cerr` und `clog`, die die Standardausgabe und Standardfehler repräsentieren, sind vom Typ `ostream_withassign` (nach ANSI-C++ vom Typ `ostream`).

Die Klasse `iostream` kombiniert die Funktionalität der Klassen `istream` und `ostream`. Abgeleitete Klassen sind `fstream` (Datei-Ein-/Ausgabe), `strstream` (Ein- und Ausgabestrom eines Zeichen-Arrays) und `stdiostream` (Standard-Ein- und Ausgabe).

Alle von `ios` abgeleiteten Objekte verwenden die Klassen `streambuf` oder die abgeleiteten Klassen `filebuf`, `stdiobuf` respektive `strstreambuf` zum Puffern der Ein- und Ausgabe.

15.5 Spezielle Geräte

Die Win32-Dateiverwaltungsroutinen können nicht nur Dateien verwalten, sondern auch viele weitere Gerätetypen. Dazu zählen die Konsole, Kommunikationsschnittstellen, bezeichnete Pipes und Mailslots. Funktionen, wie z.B. `ReadFile` oder `WriteFile`, können Socket-Handles akzeptieren, die abhängig von der WinSock-Implementierung mit `socket` oder `accept` erstellt werden.

15.5.1 Konsolen-Ein-/Ausgabe

Win32-Anwendungen führen mit `CreateFile`, `ReadFile` und `WriteFile` die Konsolen-Ein- und Ausgabe durch. Konsolen bieten eine Schnittstelle für zeichenbasierende Anwendungen.

Eine Anwendung ermittelt mit `GetStdHandle` den Datei-Handle einer Konsole, um die Ein- oder Ausgabe weiterzuleiten. Wurden die Standard-Handles weitergeleitet, gibt `GetStdHandle` die neuen Handles zurück. In diesem Fall können Anwendungen die Konsole explizit mit den speziellen Dateinamen `CONIN$` und `CONOUT$` öffnen.

Achten Sie darauf, den gemeinsamen Modus `FILE_SHARE_READ` oder `FILE_SHARE_WRITE` anzugeben, wenn Sie eine Konsole für die Ein- oder Ausgabe öffnen. Verwenden Sie außerdem den Erstellungsmodus `OPEN_EXISTING`:

```
CreateFile("CONIN$", GENERIC_READ, FILE_SHARE_READ, NULL,  
          OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
```

Möglicherweise möchten Sie nicht nur bestimmte Operationen für eine Konsole ausführen, die zum Lesen geöffnet wurde (wie z.B. das Leeren des Eingabepuffers oder das Setzen des Konsolenmodus), sondern die Konsole zum Lesen und Schreiben öffnen (`GENERIC_READ | GENERIC_WRITE`).

Die Konsole wird gewöhnlich für die zeilenorientierte Eingabe geöffnet. Mit `SetConsoleMode` ändern Sie den Ein- und Ausgabemodus. Um die Konsole beispielsweise in den reinen Eingabemodus zu setzen (jedes Zeichen wird sofort zurückgegeben, eine Kontrolle der Zeichenbearbeitung findet nicht statt), verwenden Sie den folgenden Aufruf:

```
SetConsoleMode(hConsole, 0);
```

Die `ReadFile`-Funktion liest die Tastatureingabe von der Konsole ein. Anwendungen sollten statt dessen jedoch `ReadConsole` verwenden, da diese Funktion im Gegensatz zu `ReadFile` sowohl ASCII- als auch Unicode-Zeichen unter Windows NT bearbeiten kann.

Mit `WriteFile` übertragen Sie Zeichen zur Konsole. `WriteConsole` weist dieselbe Funktionalität auf, kann jedoch zusätzlich eine Unicode-Ausgabe unter Windows NT vornehmen. Daher wird diese Funktion vorwiegend für die Konsolenausgabe verwendet.

Die letzten Abschnitte könnten den Eindruck bei Ihnen hinterlassen, daß die Konsole lediglich für Aufgaben verwendet werden kann, die mit den Standard-C/C++-Bibliotheksfunktionen einfacher ausgeführt werden können. Diese Annahme ist falsch. Eine Konsole dient nicht ausschließlich der Tastatureingabe und Zeichenausgabe. Sie bietet viele weitere Funktionen.

Eine Konsole bearbeitet ebenfalls die Mauseingabe und stellt einige Fensterverwaltungsfunktionen für das Konsolenfenster zur Verfügung. Die Low-Level-Konsoleneingabefunktionen `ReadConsoleInput` und `PeekConsoleInput` ermitteln Informationen über Tastatur-, Maus- und Fensterereignisse. Die Low-Level-Ausgabefunktion `WriteConsoleOutputAttribute` schreibt Text in die Konsole und bestimmt deren Hintergrundfarbe.

Grafische Windows-Anwendungen erhalten nicht automatisch Zugriff auf die Konsole. Diese Anwendungen können explizit eine Konsole mit `AllocConsole` erstellen. Ein Prozeß kann sich selbst mit einem Aufruf der Funktion `FreeConsole` von seiner Konsole lösen.

Der Titel und die Position des Konsolenfensters werden mit `SetConsoleTitle` und `SetConsoleWindowInfo` gesetzt.

15.5.2 Kommunikationsschnittstellen

Kommunikationsschnittstellen werden über `CreateFile`, `ReadFile` und `WriteFile` geöffnet und verwendet. Weitere Funktionen werden von den Anwendungen genutzt, um die Kommunikationsschnittstellen einzurichten und deren Verhalten zu kontrollieren.

Die Basiskonfiguration einer Kommunikationsschnittstelle ist in einer DCB-Struktur (Device Control Block) vermerkt. Die Member dieser Struktur bestimmen die Baud-Rate, Parität, Daten- und Stop-Bits, den Quittungsbetrieb und weitere Attribute der Schnittstelle. Die aktuellen Einstellungen werden mit `GetCommState` ermittelt und mit `SetCommState` gesetzt. Die Hilfsfunktion `BuildCommDCB` setzt die Eigenschaften der Struktur über eine Zeichenfolge im Format der MS-DOS-Anweisung MODE.

Der Lese- und Schreib-Timeout wird über die `COMMTIMEOUTS`-Struktur gesteuert. Die aktuellen Timeouts werden mit `GetCommTimeouts` ermittelt und mit `SetCommTimeouts` gesetzt. Die Hilfsfunktion `BuildCommDCBAndTimeouts` setzt die Eigenschaften der Strukturen DCB und `COMMTIMEOUTS` über Anweisungszeichenfolgen.

Die Größe der Ein- und Ausgabepuffer wird mit Hilfe der Funktion `SetupComm` kontrolliert.

`WaitCommEvent` wartet auf ein bestimmtes Ereignis der Kommunikationsschnittstelle.

Die Funktion `SetCommBreak` setzt die Kommunikationsschnittstelle in einen Haltemodus, der mit `ClearCommBreak` wieder gelöscht wird.

`ClearCommError` löscht eine Fehlerbedingung und zeigt den Status der Schnittstellengeräte an.

Die Funktion `PurgeComm` löscht jeden Ein-/Ausgabepuffer, der der Kommunikationsschnittstelle zugewiesen ist, und unterbricht die Ein- und Ausgabeoperationen.

`TransmitCommChar` übermittelt ein Zeichen an den Ausgabepuffer der Kommunikationsschnittstelle. Auf diese Weise kann beispielsweise die Tastenkombination `[Strg] + [C]` von einer Anwendung emuliert werden.

Die Kommunikationsschnittstelle kann außerdem für die überlappte Ein- und Ausgabe geöffnet werden. Eine Ereignismaske, die bestimmt, welche Ereignisse den Status eines Ereignisobjekts setzen (`OVERLAPPED-Struktur`), wird mit der Funktion `SetCommMask` eingerichtet. Die aktuelle Ereignismaske wird mit `GetCommMask` ermittelt.

Der Low-Level-Zugriff auf Schnittstellenfunktionen wird von `EscapeCommFunction` und `DeviceIoControl` zur Verfügung gestellt.

Weitere Informationen über die Kommunikationsschnittstelle und deren Status können mit `GetCommProperties` und `GetCommModemStatus` ermittelt werden.

Windows 95/98 bietet außerdem die Funktion `CommConfigDialog` an, die einen spezifischen Treiberkonfigurationsdialog für die angegebene Schnittstelle anzeigt.

15.6 Zusammenfassung

Win32-Anwendungen greifen über drei verschiedene Gruppen mehrerer Dateiverwaltungsfunktionen auf Dateien zu: Strom- und Low-Level-Ein-/Ausgabefunktionen, die in der C/C++-Laufzeitbibliothek enthalten sind, und Win32-Dateiverwaltungsfunktionen.

Eine Datei ist eine Datensammlung auf einem Speichergerät, wie z.B. einer Diskette. Dateien sind auf dem Gerät in Dateisystemen organisiert. Windows NT arbeitet mit den Dateisystemen DOS-FAT, Protec-

ted-Mode-FAT, NTFS und HPFS. Windows 95/98 hingegen verfügt lediglich über das FAT- und Protected-Mode-FAT-Dateisystem. Windows NT unterstützt die Dateikomprimierung auf NTFS-Laufwerken. Windows 95/98 komprimiert ganze Laufwerke mit DriveSpace. NTFS-Laufwerke unterstützen erweiterte Sicherheits-Features.

Dateien sind nicht nur auf Disketten, sondern auch auf CD-ROM- und Netzwerk-Laufwerken enthalten.

Die Win32-Dateiverwaltungsfunktionen behandeln eine geöffnete Datei wie ein Betriebssystemobjekt, auf das über eine Zugriffsnummer zugegriffen wird. Den Kern der Win32-Dateiverwaltung bilden die Funktionen `CreateFile`, `ReadFile` und `WriteFile`. Ein- und Ausgabeoperationen können über Datei-Handles synchron und asynchron ausgeführt werden. Für asynchrone Operationen ist eine überlappende Ein- und Ausgabe möglich. Diese Technik ermöglicht einer Anwendung, die Kontrolle auch dann direkt nach einem Ein-/Ausgabebefehl zurückzuerhalten, wenn die Ein-/Ausgabeoperation noch nicht beendet ist.

Die Win32-Dateiverwaltung führt ebenfalls die Standardfunktionen für die Eingabe, Ausgabe und Fehlerbehandlung aus. Die entsprechenden Handles werden mit `GetStdHandle` ermittelt.

Der Zugriff auf Dateien ist außerdem über Low-Level-Ein-/Ausgabefunktionen der Betriebssysteme DOS und Unix möglich. Den Namen dieser Funktionen steht ein Unterstrich voran (z.B. `_open`, `_read`), der anzeigt, daß die Funktionen nicht in der ANSI-Standardbibliothek enthalten sind. Die Bibliothek `oldnames.lib` kann an Anwendungen gebunden werden, wenn die Verwendung der alten Namen ohne Unterstriche erforderlich ist.

Zusätzlich zur Low-Level-Ein- und Ausgabe können Anwendungen den Ein- und Ausgabestrom verwenden. Dazu zählen C-Funktionen, wie z.B. `fopen`, `fprintf`, `fscanf` oder `fclose` und die C++-`iostream`-Klassen.

Sie greifen über die Win32-Dateiverwaltungsfunktionen ebenfalls auf spezielle Geräte zu. Die Konsole, Kommunikationsschnittstellen, bezeichnete Pipes und Mailslots sowie Sockets werden mit `socket` oder `accept` geöffnet. Weitere Funktionen dienen der detaillierten Steuerung der Konsole und Kommunikationsschnittstelle.

Die Windows- Zwischenablage

Kapitel 16

Früher war die Zwischenablage die einzige Möglichkeit für Anwendungen, Daten untereinander auszutauschen. Bevor OLE-Einbettung und Drag & Drop angeboten wurden, mußten die Anwender Zwischenablagefunktionen verwenden, wie z.B. Ausschneiden, Kopieren und Einfügen, um Daten von einer Anwendung zur anderen zu übertragen, oder um Daten innerhalb einer Anwendung zu verschieben.

Heute wird die Zwischenablage nur noch selten eingesetzt, da es OLE gibt. Dies bedeutet jedoch nicht, daß Anwendungen auf die Zwischenablage verzichten könnten. Verschiedene Zwischenablagekonzepte bestehen auch dann weiterhin, wenn Anwendungen erweiterte Methoden zum Datenaustausch verwenden.

Die Zwischenablage ist ein Win32-Element, in dem Anwendungen ihre Daten ablegen können. Auf diese Daten können alle Anwendungen zugreifen. In der Zwischenablage können verschiedene Datenformate gespeichert werden, die von dem Betriebssystem und von den Anwendungen unterstützt werden.

*Was ist die
Zwischenablage?*

16.1 Die Formate der Zwischenablage

Anwendungen legen Daten mit Hilfe der Funktion `SetClipboardData` in der Zwischenablage ab. Dieser Funktion wird zusätzlich zum Handle des Objekts ein Parameter übergeben, der das Format der Daten beschreibt. Anwendungen stellen Daten in unterschiedlichen Formaten zur Verfügung. Eine Textverarbeitung kann beispielsweise Daten in der Zwischenablage hinterlegen, die sowohl in einem privaten als auch in

einem einfachen Textformat vorliegen, das von anderen Anwendungen wie Notepad verwendet werden kann.

Anwendungen können Standard-, registrierte und private Formate in die Zwischenablage kopieren.

16.1.1 Standard-Zwischenablageformate

Windows kennt sehr viele Standard-Zwischenablageformate, die mit symbolischen Konstanten bezeichnet sind. Diese Formate sind in Tabelle 16.1 aufgeführt. Soll eine Anwendung den Handle eines bestimmten Typs zur Verfügung stellen, wenn `SetClipboardData` aufgerufen wird, ist der Zugriffstyp indiziert. Andernfalls verweist der Handle, dem `SetClipboardData` übergeben wurde, auf einen Speicherblock, der mit `GlobalAlloc` reserviert wurde.

Tabelle 16.1:
Standard-
Zwischenab-
lageformate

Format	Beschreibung
Textformate	
CF_OEMTEXT	Text, der Zeichen des OEM-Zeichensatzes enthält
CF_TEXT	Text, der Zeichen des ANSI-Zeichensatzes enthält
CF_UNICODETEXT	Text, der Unicode-Zeichen enthält
Bitmap-Formate	
CF_BITMAP	Geräteabhängige Bitmap (HBITMAP)
CF_DIB	Geräteunabhängige Bitmap (HBITMAPINFO)
CF_TIFF	TIFF-Format (Tagged Image File Format)
Metadatei-Formate	
CF_ENHMETAFILE	Erweiterte Metadatei (HENHMETAFILE)
CF_METAFILEPICT	Windows-Metadatei (METAFILEPICT)
Ersatzformate für private Formate	
CF_DSPBITMAP	Darstellung privater Daten als Bitmap
CF_DSPENHMETAFILE	Darstellung privater Daten in einer erweiterten Metadatei
CF_DSPMETAFILEPICT	Darstellung privater Daten in einer Metadatei
CF_DSPTTEXT	Darstellung privater Daten als Text
Audio-Formate	
CF_RIFF	RIFF-Format (Resource Interchange File Format)
CF_WAVE	Standard-Wave-Dateiformat für Audiodaten

Format	Beschreibung
Spezielle Formate	
CF_DIF	DIF-Format (Data Interchange Format) von Software-Arts
CF_OWNERDISPLAY	Daten, die von dem Besitzer der Zwischenablage dargestellt werden
CF_PALETTE	Farbpalette (HPALETTE)
CF_PENDATA	Microsoft-Pen-Erweiterungsdaten
CF_PRIVATEFIRST bis	Private Daten
CF_PRIVATELAST	
CF_SYLK	Microsoft-SYLK-Format (Symbolic Link)
Windows-95-Formate	
CF_GDIOBJFIRST bis	Von einer Anwendung definierte GDI-Objekte
CF_GDIOBJLAST	
CF_HDROP	Dateiliste (HDROP)
CF_LOCALE	Lokale Informationen für CF_TEXT-Daten

Windows kann unter bestimmten Umständen Daten in ein Format umwandeln, das nicht explizit von einer Anwendung zur Verfügung gestellt wird. Verfügt eine Anwendung beispielsweise über Daten im CF_TEXT-Format, kann Windows diese Daten auf Anforderung einer Anwendung in das CF_OEMTEXT-Format konvertieren. Windows konvertiert zwischen den Textformaten CF_TEXT, CF_OEMTEXT und (unter Windows NT) CF_UNICODETEXT, zwischen den Bitmap-Formaten CF_BITMAP und CF_DIB sowie zwischen den Metadatei-Formaten CF_ENHMETAFILE und CF_METAFILEPICT. Windows kann außerdem aus dem CF_DIB-Format ein CF_PALETTE-Format erzeugen.

Automatische Konvertierungen

16.1.2 Registrierte Formate

Anwendungen, die Daten in einem Format in der Zwischenablage ablegen müssen, das nicht den Standardformaten entspricht, können ein neues Zwischenablageformat mit `RegisterClipboardFormat` registrieren lassen. Eine Anwendung, die beispielsweise RTF-Text in der Zwischenablage hinterlegen möchte, führt den folgenden Aufruf aus:

```
cfRTF = RegisterClipboardFormat("Rich Text Format");
```

Rufen mehrere Anwendungen `RegisterClipboardFormat` mit demselben Formatnamen auf, wird das Format lediglich einmal registriert.

Windows registriert sehr viele Zwischenablageformate. Einige registrierte Formate beziehen sich beispielsweise auf OLE, andere auf den Windows-95/98-Kommandointerpreter. Die Bezeichnung eines registrierten Formats wird mit `GetClipboardFormatName` ermittelt.

16.1.3 Private Formate

Bisweilen kann eine Anwendung darauf verzichten, ein neues Zwischenablageformat zu registrieren. So z.B., wenn die Zwischenablage dazu verwendet wird, Daten innerhalb einer Anwendung zu transferieren und andere Anwendungen nicht auf diese Daten zugreifen. Für solche von den Anwendungen definierten privaten Formate kann eine Anwendung den Wertebereich von `CF_PRIVATEFIRST` bis `CF_PRIVATELAST` verwenden.

Damit Programme zur Einsicht der Zwischenablage die im privaten Format gespeicherten Daten anzeigen kann, muß der Besitzer der Zwischenablagedaten diese in einem der Formate `CF_DSPBITMAP`, `CF_DSPTEXT`, `CF_DSPMETAFILEPICT` oder `CF_DSPENHMETAFILE` zur Verfügung stellen. Diese Formate sind identisch mit `CF_BITMAP`, `CF_TEXT`, `CF_METAFILEPICT` und `CF_ENHMETAFILE`. Sie unterscheiden sich lediglich darin von ihren Korrelaten, daß sie ausschließlich zur Anzeige und nicht zum Einfügen verwendet werden.

16.2 Zwischenablageoperationen

Eine Anwendung muß zunächst einige Aufgaben ausführen, bevor sie die Zwischenablage nutzen kann. Dazu zählen

- das Einrichten der Zwischenablagedaten,
- die Übernahme des Besitzes an der Zwischenablage,
- das Transferieren der Daten und
- die Reaktion auf die Ereignisse, die sich auf die Zwischenablage beziehen.

Die Anwendung sollte außerdem spezifische Zwischenablagefunktionen in ihrer Anwenderschnittstelle zur Verfügung stellen (wie z.B. Anweisungen im Menü BEARBEITEN).

16.2.1 Transferieren von Daten in die Zwischenablage

Bevor Daten in die Zwischenablage übertragen werden können, muß die Anwendung zunächst das Datenobjekt zur Verfügung stellen und anschließend Besitzer der Zwischenablage werden.

Das Datenobjekt wird über einen Handle angesprochen. Der Handle kann auf einen Speicherblock verweisen, der mit `GlobalAlloc` und den Flags `GMEM_MOVEABLE` und `GMEM_DDESHARE` reserviert wurde. (Das Flag `GMEM_DDESHARE` zeigt nicht an, daß der Speicherblock gemeinsam von den Anwendungen genutzt werden kann.) Der Handle kann ebenfalls auf ein GDI-Objekt, wie z.B. eine Bitmap, verweisen. Beachten Sie bitte, daß die Anwendung mit dem Handle den Besitz an dem Objekt an die Zwischenablage übergibt. Die Anwendung sollte das Objekt daher nicht sperren oder löschen.

*Datenobjekt zur
Verfügung
stellen*

Der Besitz an der Zwischenablage geht mit einem Aufruf der Funktion `OpenClipboard` an die Anwendung über. Die Zwischenablage wird mit `EmptyClipboard` geleert. Alle zuvor an die Zwischenablage übertragenen Handles werden auf diese Weise freigegeben.

*Zwischenablage
besetzen*

Die Anwendung transferiert anschließend mit `SetClipboardData` die Daten zur Zwischenablage.

*Daten
übertragen*

Eine Anwendung kann `SetClipboardData` wiederholt aufrufen, wenn Daten in unterschiedlichen Formaten vorliegen. Um eine Grafik z.B. sowohl im Bitmap-Format als auch im Metafile-Format in die Zwischenablage zu übertragen, kann eine Anwendung `SetClipboardData` mit `CF_DIB` und `CF_ENHMETAFILE` verwenden.

Die Anwendung schließt die Zwischenablage mit Hilfe der Funktion `CloseClipboard`.

*Zwischenablage
schließen*

16.2.2 Verzögerte Übertragung

Die verzögerte Übertragung ist eine Technik, die der Erhöhung der Performance dient. Sie wird für Anwendungen verwendet, die häufig große Datenmengen in die Zwischenablage übertragen.

Eine Anwendung aktiviert die verzögerte Übertragung, indem sie der Funktion `SetClipboardData` als zweiten Parameter den Wert `NULL` übergibt. Das System informiert die Anwendung daraufhin, daß Daten in einem bestimmten Format übertragen werden können, wenn die Anwendung die Nachricht `WM_RENDERFORMAT` erhält. Die Anwendung reagiert auf diese Nachricht, indem sie `SetClipboardData` aufruft und die geforderten Daten in der Zwischenablage plaziert.

Nutzt eine Anwendung die verzögerte Übertragung, kann sie ebenfalls die Nachricht `WM_RENDERALLFORMATS` empfangen. Diese Nachricht wird an den Besitzer der Zwischenablage gesendet, bevor diese zerstört wird. Diese Vorgehensweise gewährleistet, daß die Daten in der Zwischenablage anderen Anwendungen erhalten bleiben.

Während der Bearbeitung einer `WM_RENDERFORMAT`- oder `WM_RENDERALLFORMATS`-Nachricht muß die Zwischenablage nicht von der Anwendung geöffnet werden, bevor sie `SetClipboardData` aufruft. Das nachfolgende Schließen ist ebenfalls nicht erforderlich.

16.2.3 Einfügen der Daten aus der Zwischenablage

Mit der Funktion `IsClipboardFormatAvailable` ermitteln Sie, ob Daten in einem besonderen Format in der Zwischenablage enthalten sind. Benötigen Sie eine Kopie dieser Daten, rufen Sie bitte `OpenClipboard` auf, gefolgt von einem Aufruf der Funktion `GetClipboardData`. Der mit `GetClipboardData` ermittelte Handle sollte nicht lange ungenutzt bleiben. Anwendungen sollten die dem Handle zugewiesenen Daten sofort kopieren und anschließend `CloseClipboard` aufrufen. Nach diesem Aufruf können andere Anwendungen die Zwischenablage leeren. Der ermittelte Handle erfüllt daraufhin keinen Zweck mehr.

Die Funktion `IsClipboardFormatAvailable` kann ebenfalls dazu verwendet werden, die Einträge des Anwendungsmenüs BEARBEITEN zu aktualisieren. Zeigt die Funktion beispielsweise an, daß in der Zwischenablage keine Daten in dem Format enthalten sind, das von der Anwendung bearbeitet werden kann, sollte die Anwendung den Menüeintrag EINFÜGEN sperren.

Informationen über das Format der Zwischenablagendaten werden mit den Funktionen `CountClipboardFormats` und `EnumClipboardFormats` ermittelt.

16.2.4 Steuerelemente und die Zwischenablage

Eingabefeld-Steuerelemente unterstützen die Zwischenablage (auch in Kombinationsfeldern). Eingabefelder reagieren auf verschiedene Nachrichten, indem Sie Zwischenablageoperationen ausführen.

<code>WM_COPY</code>	Erhält ein Eingabefeld die Nachricht <code>WM_COPY</code> , kopiert es die aktuelle Markierung im <code>CF_TEXT</code> -Format in die Zwischenablage.
<code>WM_CUT</code>	Die Nachricht <code>WM_CUT</code> führt dazu, daß das Eingabefeld die gegenwärtige Markierung im <code>CF_TEXT</code> -Format in die Zwischenablage kopiert. Die Markierung wird anschließend in dem Eingabefeld gelöscht.
<code>WM_PASTE</code>	Die Reaktion eines Eingabefelds auf die Nachricht <code>WM_PASTE</code> besteht darin, daß es die aktuelle Markierung durch den Inhalt der Zwischenablage (sofern dieser im <code>CF_TEXT</code> -Format vorliegt) ersetzt.

`WM_CLEAR` Eingabefeld-Steuerelemente bearbeiten außerdem die Nachricht `WM_CLEAR` (Löschen der aktuellen Selektion).

16.2.5 Zwischenablagenachrichten

Verschiedene Windows-Nachrichten betreffen die Zwischenablage.

- Anwendungen, die die verzögerte Übertragung verwenden, müssen die Nachrichten `WM_RENDERFORMAT` und `WM_RENDERALLFORMATS` bearbeiten.
- Die Nachricht `WM_DESTROYCLIPBOARD` wird an den Besitzer der Zwischenablage gesendet, wenn deren Inhalte zerstört werden. Eine Anwendung kann beispielsweise auf diese Nachricht reagieren, indem sie die Ressourcen freigibt, die sich auf die Zeichenelemente in der Zwischenablage beziehen.
- Einige Nachrichten werden an Anwendungen gesendet, die Daten im `CF_OWNERDISPLAY`-Format in der Zwischenablage plazieren. Dazu zählen `WM_ASKCBFORMATNAME`, `WM_DRAWCLIPBOARD`, `WM_HSCROLLCLIPBOARD`, `WM_VSCROLLCLIPBOARD` und `WM_PAINTCLIPBOARD`.

Weitere Windows-Nachrichten werden von Programmen zur Anzeige des Zwischenablageinhalts verwendet.

16.2.6 Zwischenablage-Viewer

Ein `ZWISCHENABLAGE-VIEWER` ist eine Anwendung, die den aktuellen Inhalt der Zwischenablage anzeigt. Ein Beispiel für einen Zwischenablage-Viewer ist das Programm `CLIPBRD.EXE`. Der `IDataObject-Viewer` (`DOBJVIEW.EXE`), der mit Visual C++ 6 ausgeliefert wird, kann ebenfalls zur Anzeige des Zwischenablageinhalts verwendet werden.

Ein Zwischenablage-Viewer dient lediglich der Ansicht des Inhalts der Zwischenablage. Er beeinflusst nicht die Zwischenablageoperationen.

Mehrere Zwischenablage-Viewer können gleichzeitig verwendet werden. Eine Anwendung fügt mit `SetClipboardViewer` ein Viewer-Fenster ein. Der Funktion wird der Handle des Fensters übergeben. Das Fenster empfängt anschließend die Nachrichten `WM_CHANGECHAIN` und `WM_DRAWCLIPBOARD`. Der Zwischenablage-Viewer kann sich selbst mit `ChangeClipboardChain` entfernen.

16.3 Eine einfache Implementierung

Das Programm in Listing 16.1 erläutert die zuvor beschriebenen Sachverhalte. Diese einfache Anwendung implementiert die vier wesentlichen Zwischenablagefunktionen: Ausschneiden, Kopieren, Einfügen und Entfernen. Die Ressourcendatei des Programms ist in Listing 16.2 aufgeführt. Die Header-Datei finden Sie in Listing 16.3. Kompilieren Sie die Anwendung in einem DOS-Fenster:

```
RC HELLOCF.RC
CL HELLOCF.C HELLOCF.RES USER32.LIB GDI32.LIB
```

*Listing 16.1:
Eine einfache
Zwischenablage-
anwendung*

```
#include <windows.h>
#include "hellocf.h"
HINSTANCE hInstance;
char *pszData;
void DrawHello(HWND hwnd)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT clientRect;
    if (pszData != NULL)
    {
        hDC = BeginPaint(hwnd, &paintStruct);
        if (hDC != NULL)
        {
            GetClientRect(hwnd, &clientRect);
            DPTOLP(hDC, (LPPOINT)&clientRect, 2);
            DrawText(hDC, pszData, -1, &clientRect,
                DT_CENTER | DT_VCENTER | DT_SINGLELINE);
            EndPaint(hwnd, &paintStruct);
        }
    }
}

void CopyData(HWND hwnd)
{
    HGLOBAL hData;
    LPVOID pData;
    OpenClipboard(hwnd);
    EmptyClipboard();
    hData = GlobalAlloc(GMEM_DDESHARE | GMEM_MOVEABLE,
        strlen(pszData) + 1);
    pData = GlobalLock(hData);
    strcpy((LPSTR)pData, pszData);
    GlobalUnlock(hData);
    SetClipboardData(CF_TEXT, hData);
    CloseClipboard();
}

void DeleteData(HWND hwnd)
{
    free(pszData);
    pszData = NULL;
    InvalidateRect(hwnd, NULL, TRUE);
}

void PasteData(HWND hwnd)
{
    HANDLE hData;
```

```

    LPVOID pData;
    if (!IsClipboardFormatAvailable(CF_TEXT)) return;
    OpenClipboard(hwnd);
    hData = GetClipboardData(CF_TEXT);
    pData = GlobalLock(hData);
    if (pszData) DeleteData(hwnd);
    pszData = malloc(strlen(pData) + 1);
    strcpy(pszData, (LPSTR)pData);
    GlobalUnlock(hData);
    CloseClipboard();
    InvalidateRect(hwnd, NULL, TRUE);
}

void SetMenus(HWND hwnd)
{
    EnableMenuItem(GetMenu(hwnd), ID_EDIT_CUT,
        pszData ? MF_ENABLED : MF_GRAYED);
    EnableMenuItem(GetMenu(hwnd), ID_EDIT_COPY,
        pszData ? MF_ENABLED : MF_GRAYED);
    EnableMenuItem(GetMenu(hwnd), ID_EDIT_PASTE,
        IsClipboardFormatAvailable(CF_TEXT)
        ? MF_ENABLED : MF_GRAYED);
    EnableMenuItem(GetMenu(hwnd), ID_EDIT_DELETE,
        pszData ? MF_ENABLED : MF_GRAYED);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                case ID_FILE_EXIT:
                    DestroyWindow(hwnd);
                    break;
                case ID_EDIT_CUT:
                    CopyData(hwnd);
                    DeleteData(hwnd);
                    break;
                case ID_EDIT_COPY:
                    CopyData(hwnd);
                    break;
                case ID_EDIT_PASTE:
                    PasteData(hwnd);
                    break;
                case ID_EDIT_DELETE:
                    DeleteData(hwnd);
                    break;
            }
            break;
        case WM_PAINT:
            DrawHello(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        case WM_INITMENUPOPUP:
            if (LOWORD(lParam) == 1)
            {
                SetMenus(hwnd);
                break;
            }
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
}

```

```

        return 0;
    }
    int WINAPI WinMain(HINSTANCE hThisInstance,
                      HINSTANCE hPrevInstance,
                      LPSTR d3, int nCmdShow)
    {
        HWND hwnd;
        MSG msg;
        WNDCLASS wndClass;
        HANDLE hAccTbl;
        pszData = malloc(14);
        strcpy(pszData, "Hello, World!");
        hInstance = hThisInstance;
        if (hPrevInstance == NULL)
        {
            memset(&wndClass, 0, sizeof(wndClass));
            wndClass.style = CS_HREDRAW | CS_VREDRAW;
            wndClass.lpfnWndProc = WndProc;
            wndClass.hInstance = hInstance;
            wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
            wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
            wndClass.lpszMenuName = "HelloMenu";
            wndClass.lpszClassName = "Hello";
            if (!RegisterClass(&wndClass)) return FALSE;
        }
        hwnd = CreateWindow("Hello", "Hello",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                           NULL, NULL, hInstance, NULL);
        ShowWindow(hwnd, nCmdShow);
        hAccTbl = LoadAccelerators(hInstance, "HelloMenu");
        UpdateWindow(hwnd);
        while (GetMessage(&msg, NULL, 0, 0))
        {
            if (!TranslateAccelerator(hwnd, hAccTbl, &msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
        return msg.wParam;
    }
}

```

Listing 16.2:
Die Ressourcen-
datei

```

#include "windows.h"
#include "hellocf.h"
HelloMenu MENU
BEGIN
    POPUP        "&File"
    BEGIN
        MENUITEM "E&xit",          ID_FILE_EXIT
    END
    POPUP        "&Edit"
    BEGIN
        MENUITEM "Cu&t\tCtrl+X",    ID_EDIT_CUT, GRAYED
        MENUITEM "&Copy\tCtrl+C",    ID_EDIT_COPY, GRAYED
        MENUITEM "&Paste\tCtrl+V",    ID_EDIT_PASTE, GRAYED
        MENUITEM "&Delete\tDel",      ID_EDIT_DELETE, GRAYED
    END
END
HelloMenu ACCELERATORS
BEGIN
    "X", ID_EDIT_CUT, VIRTKEY, CONTROL
    "C", ID_EDIT_COPY, VIRTKEY, CONTROL

```

```

        "V", ID_EDIT_PASTE, VIRTKEY, CONTROL
        VK_DELETE, ID_EDIT_DELETE, VIRTKEY
    END

#define ID_FILE_EXIT    1000
#define ID_EDIT_CUT     1001
#define ID_EDIT_COPY    1002
#define ID_EDIT_PASTE   1003
#define ID_EDIT_DELETE  1004

```

*Listing 16.3:
Die Header-Datei*

Verwenden Sie die Zwischenablagefunktionen des Programms, um zu sehen, wie es arbeitet. Kopieren Sie mit CUT oder COPY den angezeigten Text in die Zwischenablage. Sie haben außerdem die Möglichkeit, mit einer anderen Anwendung (beispielsweise mit dem Editor), einen Textblock zu erstellen, diesen in die Zwischenablage zu kopieren und anschließend in das Beispielprogramm einzufügen.

Das Programm verwendet ein einfaches Datenobjekt. Ein Zeiger weist auf die Zeichenfolge »Hello, World!«. Die Menüleiste der Anwendung enthält das Menü EDIT mit den Zwischenablagefunktionen CUT, COPY, PASTE und DELETE.

Zwischenablageoperationen werden ausgeführt, wenn der Anwender einen Eintrag aus diesem Menü auswählt. Die Funktion `CopyData` kopiert die aktuelle Zeichenfolge in die Zwischenablage, indem zunächst der Besitz an der Zwischenablage mit `EmptyClipboard` übernommen und anschließend die Funktion `SetClipboardData` aufgerufen wird. Die Funktion `PasteData` kopiert die Daten aus der Zwischenablage. Dazu werden die aktuellen Daten zunächst freigegeben und anschließend mit `GetClipboardData` ermittelt.

Die Zwischenablageoperationen

Die Funktion `SetMenus` aktualisiert den Zugriffstatus der Menüeinträge im Menü EDIT, der davon abhängig ist, ob Daten im CF_TEXT-Format in der Zwischenablage vorhanden sind. Sind derartige Daten nicht in der Zwischenablage enthalten, wird der Zugriff auf den Menüeintrag PASTE deaktiviert. Der Status der Menüeinträge CUT, COPY und DELETE wird ebenfalls aktualisiert, um anzuzeigen, ob die Anwendung über Daten verfügt, die in die Zwischenablage kopiert werden können.

*Aktualisierung
des Menüs*

Um zu gewährleisten, daß das Fenster der Anwendung korrekt während des Austauschs von Daten aktualisiert wird, rufen sowohl `DeleteData` als auch `PasteData` die Funktion `InvalidateRect` auf.

Beachten Sie bitte, daß der in `CopyData` reservierte Daten-Handle nicht von der Anwendung freigegeben wird. Nachdem dieser Handle der Zwischenablage übergeben wurde, ist die Anwendung nicht mehr für die Freigabe (wenn die Zwischenablage geleert wird) verantwortlich. Die Anwendung gibt ebenfalls nicht den mit `GetClipboardData` ermittelten Handle frei. Dieser Handle wird nach dem Auslesen der Zwischenablagendaten und dem Schließen der Zwischenablage ungültig.

16.4 Zusammenfassung

Die Windows-Zwischenablage ist der herkömmliche Mechanismus für den Datentransfer zwischen den Anwendungen. Die Zwischenablage ist ein Windows-Element, in dem Anwendungen Daten unterschiedlicher Formate ablegen können. Diese Daten können von anderen Anwendungen ausgelesen werden.

Windows definiert verschiedene Zwischenablage-Standardformate. Anwendungen können zusätzliche Zwischenablageformate registrieren oder private Formate verwenden.

Eine Anwendung überträgt Daten in die Zwischenablage, indem sie zunächst Besitzer der Zwischenablage wird. Dies geschieht mit einem Aufruf von `EmptyClipboard`. Die Anwendung überträgt die Daten sofort oder verzögert. Sie aktivieren die verzögerte Übertragung, indem Sie der `SetClipboardData`-Funktion den Wert `NULL` übergeben. Eine Anwendung, die die verzögerte Übertragung verwendet, muß die Nachrichten `WM_RENDERFORMAT` und `WM_RENDERALLFORMATS` bearbeiten.

Eingabefelder unterstützen die Zwischenablage. Sie reagieren auf die Nachrichten `WM_CUT`, `WM_COPY`, `WM_PASTE` und `WM_CLEAR`, indem Sie Daten im `CF_TEXT`-Format ausschneiden, kopieren, einfügen oder löschen.

Zwischenablage-Viewer sind Programme, die den aktuellen Inhalt der Zwischenablage anzeigen. Diese Programme dienen lediglich der Ansicht und haben keinen Einfluß auf die Zwischenablageoperationen.

Die Registrierung

Kapitel

17

Die Registrierung wurde mit OLE unter Windows 3.1 eingeführt. Programmierer versuchen häufig, dieses wesentliche Element des Betriebssystems zu ignorieren, das der Initialisierung und Konfiguration dient.

Was aber ist die Registrierung? Was sollten Sie als Win32-Entwickler über die Registrierung wissen, und wie greifen Sie darauf zu? Die Antworten auf diese Fragen finden Sie in den folgenden Abschnitten.

17.1 Die Struktur der Registrierung

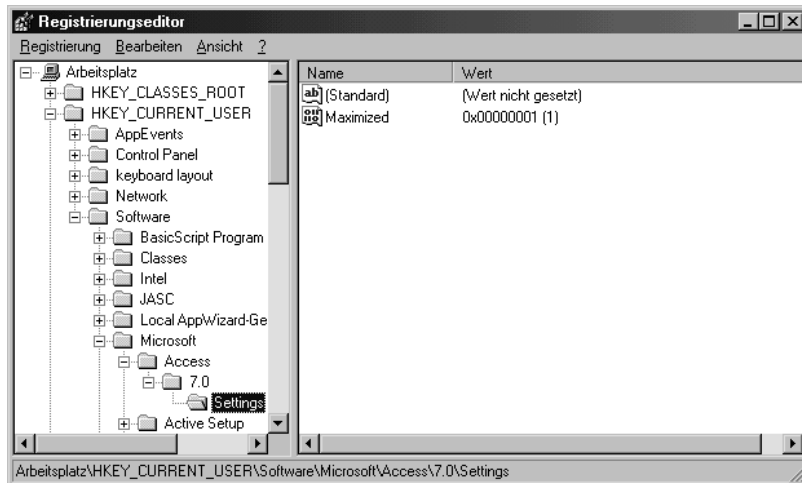
Die Registrierung ist ein hierarchisch organisierter Informationsspeicher. Jeder Eintrag dieser Informationsstruktur wird als SCHLÜSSEL bezeichnet. Ein Schlüssel kann mehrere Unterschlüssel und Dateneinträge enthalten, die als WERTE bezeichnet werden. Auf diese Weise speichert die Registrierung Informationen über das System, dessen Konfiguration, Hardware-Geräte und Software-Anwendungen. Die Registrierung ersetzt außerdem die bekannten INI-Dateien, indem sie spezifische Anwendungseinstellungen aufnimmt.

*Schlüssel und
Werte*

Jeder Registrierungsschlüssel trägt einen Namen. Schlüsselnamen bestehen aus druckbaren ASCII-Zeichen, Leerzeichen und Wildcards (* oder ?), dürfen jedoch keine Backslashes (\) enthalten. Schlüsselnamen, die mit einem Punkt (.) beginnen, sind reserviert. Die Groß- und Kleinschreibung wird während der Auswertung von Schlüsselnamen nicht berücksichtigt.

17.1.1 Registrierungswerte

Abbildung 17.1:
Einsicht in die
Registrierung
mit RegEdit.exe



Ein Wert wird in der Registrierung über seinen Namen identifiziert. Diese Namen können aus denselben Zeichen bestehen, die für die Schlüsselnamen gelten. Der Wert selbst kann eine Zeichenfolge oder ein positiver 32-Bit-Wert sein. Auch binäre Daten können einem Wert zugewiesen werden.

Die Windows-95/98-Registrierung und die Windows-NT-Registrierung unterscheiden sich voneinander. Unter Windows 95/98 bekommt jeder Registrierungsschlüssel einen Standardwert zugewiesen. Der Standardwert eines Schlüssels trägt keine Bezeichnung (in dem Registrierungseditor wird er durch das in Klammern gesetzte Wort STANDARD repräsentiert). Standardwerte sind auch in der Windows-NT-Registrierung vorhanden. Dort müssen sie jedoch explizit erzeugt werden, während unter Windows 95/98 jeder Schlüssel automatisch über einen Standardwert verfügt.

Ein weiterer Unterschied besteht darin, daß Windows NT verschiedene Zeichenfolgentypen in der Registrierung unterstützt, während Windows 95/98 lediglich einen Zeichenfolgentyp zuläßt. Nachfolgend sind einige Ausgaben aufgeführt, die das später in diesem Kapitel vorgestellte Programm zum Lesen der Registrierungseinträge generiert:

```
Enter key: HKEY_CURRENT_USER\Environment\include
Expandable string: f:\msvc20\include;f:\msvc20\mfc\include
```

Ermitteln Sie denselben Wert mit der Windows-95/98-Registrierung, erscheint dieser als ein binärer Wert. Dies ist jedoch eine Unzulänglichkeit des Registrierungseditors und nicht der Registrierung selbst.

Tabelle 17.1 enthält eine Liste aller Werttypen, die in der Registrierung von Windows 95/98 und Windows NT verwendet werden können.

Symbolischer Bezeichner	Beschreibung
REG_BINARY	Binärdaten
REG_DWORD	Double-Word im Maschinenformat (Low-Endian für Intel)
REG_DWORD_LITTLE_ENDIAN	Double-Word im Little-Endian-Format
REG_DWORD_BIG_ENDIAN	Double-Word im Big-Endian-Format
REG_EXPAND_SZ	Zeichenfolge mit nicht erweiterten Umgebungsvariablen
REG_LINK	Symbolische Unicode-Verbindung
REG_MULTI_SZ	Mehrere Zeichenfolgen, die mit zwei Null-Zeichen enden
REG_NONE	Nichtdefinierter Typ
REG_RESOURCE_LIST	Gerätetreiber-Ressourcenliste
REG_SZ	Zeichenfolge, die mit einem Null-Zeichen endet

*Tabelle 17.1:
Werttypen der
Registrierung*

17.1.2 Kapazität der Registrierung

Einträge, die größer als ein oder zwei Kbyte sind, sollten nicht in der Registrierung gespeichert werden. Verwenden Sie für große Einträge eine separate Datei, und legen Sie deren Dateinamen in der Registrierung ab.

Windows 95 begrenzt die Größe der in der Registrierung gespeicherten Werte auf 64 Kbyte. Unter Windows 98 gilt diese Einschränkung nicht mehr. Trotzdem sollten größere Einträge in separaten Dateien gespeichert werden.

Das Speichern eines Schlüssels erfordert mehr Speicherplatz als das Speichern eines Werts. Verwenden Sie deshalb, sofern möglich, lediglich einen Schlüssel, unter dem Sie Ihre Werte organisieren.

17.1.3 Vordefinierte Registrierungsschlüssel

Die Registrierung verfügt über verschiedene vordefinierte Schlüssel.

Tabelle 17.2:
Registrierungs-
schlüssel

Schlüssel	Beschreibung
HKEY_LOCAL_MACHINE	Enthält Einträge, die den Computer und dessen Konfiguration beschreiben. Dazu zählen Informationen über den Prozessor, das System-Board, den Speicher und die installierte Hard- sowie Software.
HKEY_CLASSES_ROOT	Nimmt Informationen über die Dokumenttypen und OLE/COM auf. Dieser Schlüssel ist HKEY_LOCAL_MACHINE äquivalent zu HKEY_LOCAL_MACHINE\SOFTWARE\Classes. Die hier gespeicherten Informationen werden von Kommandozeileninterpretern wie dem Programm-Manager, vom Datei-Manager, dem Explorer und OLE/ActiveX-Anwendungen verwendet.
HKEY_USERS	Der Schlüssel HKEY_USERS enthält die allgemeinen Anwendereinstellungen sowie individuelle Konfigurationen des Anwenders.
HKEY_CURRENT_USER	HKEY_CURRENT_USER ist der Basis-schlüssel zu den Informationen über die Einstellungen des gegenwärtig angemeldeten Anwenders.
HKEY_CURRENT_CONFIG	HKEY_CURRENT_CONFIG enthält Informationen über die aktuellen Systemeinstellungen. Dieser Schlüssel gleicht einem Unterschlüssel (z.B. 0001) von HKEY_LOCAL_MACHINE\Config. (Windows 95/98)
HKEY_DYN_DATA	Der Schlüssel HKEY_DYN_DATA stellt dynamische Statusinformationen zur Verfügung, z.B. über Plug&Play-Geräte. (Windows 95/98)

17.2 Manuelle Bearbeitung der Registrierung

Die Registrierung kann mit Hilfe des Registrierungseditors (*regedit.exe*) manuell bearbeitet werden.

Das Windows-NT-Programm *regedit.exe* ist eine Version des Editors, der dem Registrierungs-Editor der 16-Bit-Version von Windows gleicht. Diese Anwendung ist nicht zur Bearbeitung der Registrierung geeignet, da sie lediglich einige der Registrierungsschlüssel aufführt.



Abbildung 17.2 zeigt den Registrierungs-Editor von Windows 95.

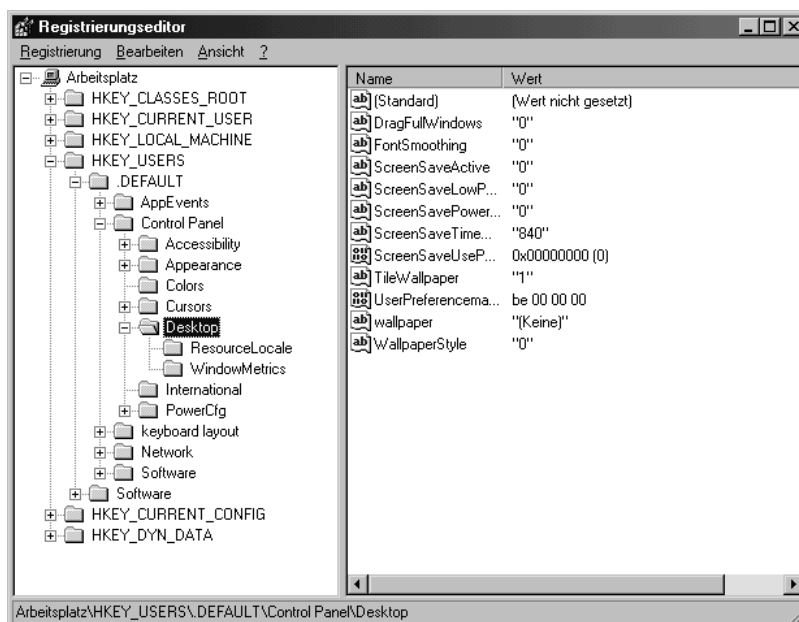


Abbildung 17.2:
Der Registrierungseditor

Programmierer werden häufig über den Registrierungseditor auf die Registrierung zugreifen müssen, um beispielsweise Schlüssel zu entfernen, die von inkorrekt ausgeführten Anwendungen im Entwicklungsstadium erzeugt wurden. Der Endanwender sollte jedoch nicht die Einstellungen der Registrierung manuell ändern müssen.

Einige Registrierungseinstellungen werden implizit von Konfigurationsanwendungen, wie z.B. der Systemsteuerung, vorgenommen. Andere Einstellungen werden während der Installation einer Anwendung ge-

speichert. OLE-Anwendungen, die mit dem Anwendungsassistenten erstellt wurden, aktualisieren ihre Registrierungseinstellungen nach jedem Start.

17.3 Allgemein verwendete Registrierungsschlüssel

Informationen über Registrierungsschlüssel sind häufig schwer zu finden. In den folgenden Abschnitten sind daher die vorwiegend von Programmierern genutzten Registrierungsschlüssel beschrieben.

17.3.1 Unterschlüssel in HKEY_LOCAL_MACHINE

Die Schlüssel in HKEY_LOCAL_MACHINE enthalten Informationen über die Konfiguration der Soft- und Hardware des Computers. `Config` und `Enum` sind spezifische Windows-95/98-Unterschlüssel, die sich auf das Plug&Play beziehen. Der `Config`-Schlüssel speichert verschiedene Hardware-Konfigurationen. `Enum` führt die Windows-95/98-Busstruktur der Hardware-Geräte auf.

Sowohl Windows 95/98 als auch Windows NT verfügen über den HKEY_LOCAL_MACHINE-Unterschlüssel `System`. `System\CurrentControlSet` enthält Konfigurationsinformationen über Dienste und Gerätetreiber.

Weitere Unterschlüssel sind `Software` und `Classes`. `Software` nimmt Informationen über die installierten Software-Pakete auf. `Classes` ist der Unterschlüssel, auf den HKEY_CLASSES_ROOT verweist.

Die `Software`-Struktur ist für Programmierer besonders interessant. In diesem Unterschlüssel legen Sie Konfigurations- und Installationsinformationen über Ihre Anwendung ab. Microsoft empfiehlt, mehrere Unterschlüssel in `HKEY_LOCAL_MACHINE\Software` anzulegen. Diese Unterschlüssel sollten den Namen Ihres Unternehmens, die Produktbezeichnung sowie die Versionsnummer der Anwendung aufnehmen:

```
HKEY_LOCAL_MACHINE\Software\CompanyName\ProductName\1.0
```

Konfigurationsinformationen, die die Version des auf meinem Computer installierten Programms Microsoft Bookshelf betreffen, sind in dem folgenden Schlüssel enthalten:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Bookshelf '95\95.0.0.39
```

Welche Informationen Sie unter solch einem Schlüssel speichern, ist abhängig von Ihrer Anwendung. Speichern Sie in dem Unterschlüssel `Software` keine spezifischen Anwenderdaten. Diese sollten unter `HKEY_CURRENT_USER` abgelegt werden.

Interessant ist auch der Schlüssel:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
```

Dieser Schlüssel beschreibt die aktuelle Windows-Konfiguration.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion
```

ist ein Schlüssel mit einer besonderen Bedeutung unter Windows 95/98. Er dient der Kompatibilität mit 32-Bit-Debugger, die für Windows NT geschrieben wurden. Debugger-Informationen, die unter

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion
\Aedebug
```

gespeichert sind, beziehen sich auf Windows-95/98-Debugger.

17.3.2 Unterschlüssel in HKEY_CLASSES_ROOT

HKEY_CLASSES_ROOT enthält zwei verschiedene Unterschlüsseltypen: Unterschlüssel, die sich auf Dateinamenserweiterungen beziehen, und Klassendefinitionsunterschlüssel.

Die Unterschlüssel für Dateinamenserweiterungen sind mit der jeweiligen Endung des Dateinamens bezeichnet (wie z.B. .doc). Der Schlüssel enthält gewöhnlich einen nicht bezeichneten Wert, der die Bezeichnung des Klassendefinitionsunterschlüssels aufnimmt.

Der Klassendefinitionsschlüssel beschreibt das Verhalten einer Dokumentklasse. Zu den hier gespeicherten Informationen zählen die Daten des Kommandozeileninterpreters sowie OLE-Eigenschaften.

Ein Unterschlüssel von HKEY_CLASSES_ROOT ist CLSID, der COM-Klassenbezeichner aufnimmt.

Wenn Sie mit dem Visual-C++-Anwendungsassistenten eine MFC-Anwendung erstellen, werden mehrere Unterschlüssel in HKEY_CLASSES_ROOT erstellt. Diese bezeichnen den Dokumenttyp und die Dateinamenserweiterungen Ihrer neuen Anwendung sowie deren OLE-Eigenschaften, wie z.B. den OLE-Klassenbezeichner. Eine MFC-Anwendung mit dem Namen *TEST* und der Dateinamenserweiterung *.tst* für die mit dieser Anwendung erzeugten Dokumentdateien führt zur Erstellung der folgenden Registrierungseinträge unter HKEY_CLASSES_ROOT:

```
.TST = Test.Document
Test.Document\shell\open\command = TEST.EXE %1
Test.Document\shell\open\ddeexec = [open("%1")]
Test.Document\shell\open\ddeexec\application = TEST
Test.Document = Test Document
Test.Document\protocol\StdFileEditing\server = TEST.EXE
Test.Document\protocol\StdFileEditing\verb\0 = &Edit
Test.Document\Insertable =
Test.Document\CLSID = {FC168A60-F1EA-11CE-87C3-00403321BFAC}
```


Die folgenden Einträge werden unter HKEY_CLASSES_ROOT\CLSID erstellt:

```
{FC168A60-F1EA-11CE-87C3-00403321BFAC} = Test Document
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\DefaultIcon = TEST.EXE,1
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\LocalServer32 = TEST.EXE
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\ProgId = Test.Document
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\MiscStatus = 32
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\AuxUserType\3 = test
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\AuxUserType\2 = Test
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\Insertable =
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\verb\1 = &Open,0,2
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\verb\0 = &Edit,0,2
{FC168A60-F1EA-11CE-87C3-00403321BFAC}\InprocHandler32 = ole32.dll
```

17.3.3 Unterschlüssel in HKEY_USERS

HKEY_USERS enthält einen Unterschlüssel mit der Bezeichnung .Default und keinen oder mehr Unterschlüssel zu jedem Anwender des Systems. Der Schlüssel .Default nimmt das Profil des Standardanwenders auf. Andere Einträge betreffen die Profile der eigentlichen Anwender.

17.3.4 Unterschlüssel in HKEY_CURRENT_USER

HKEY_CURRENT_USER entspricht dem Profil des gegenwärtig angemeldeten Anwenders. Diesem Schlüssel sind mehrere Unterschlüssel zugewiesen, die sowohl für Windows 95/98 als auch für Windows NT gelten. Einige dieser Schlüssel sind jedoch betriebssystemspezifisch.

Konfigurationsinformationen einer Anwendung, die sich auf den aktuellen Anwender beziehen, sollten unter Software abgelegt werden. Informationen über das Unternehmen, das Produkt und die Versionsnummer des Produkts sollten einzelnen Schlüsseln zugewiesen werden. Anwendereinstellungen zu Microsoft Excel 5.0 finden Sie beispielsweise unter dem folgenden Schlüssel:

```
HKEY_CURRENT_USER\Software\Microsoft\Excel\5.0
```

Die Anwendereinstellungen für Windows, dessen Komponenten und Applets, sind unter dem folgenden Schlüssel aufgeführt:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion
```

17.3.5 Die Registrierung und INI-Dateien

In neuen Anwendungen sollte die Registrierung anstelle von INI-Dateien verwendet werden. Wie verhalten sich jedoch ältere Anwendungen unter der 32-Bit-Version von Windows?

Dieses Verhalten ist für Windows 95/98 und Windows NT unterschiedlich. Um eine maximale Abwärtskompatibilität aufrechtzuerhal-

ten, unterstützt Windows 95/98 weiterhin INI-Dateien, wie z.B. *win.ini* und *system.ini*. Windows NT übernimmt diese Dateien in die Registrierung.

Welche Dateien in die Registrierung übernommen werden, bestimmt der folgende Schlüssel:

```
SOFTWARE\Microsoft\Windows NT\CurrentVersion\IniFileMapping
```

Dieser Schlüssel enthält einen Unterschlüssel für jede übernommene INI-Datei. Die Werte dieser Unterschlüssel entsprechen den Abschnitten der INI-Datei und verweisen gewöhnlich auf andere Schlüssel in der Registrierung.

Die Übernahme von INI-Dateien betrifft Funktionen, wie z.B. *ReadProfileString* oder *WriteProfileString*. Wurde eine INI-Datei übernommen, entnehmen diese Funktionen die benötigten Informationen der Registrierung, oder legen Daten dort ab.

17.4 Anwendungen und die Registrierung

Die Win32-API bietet verschiedene Funktionen zur Bearbeitung der Registrierung an.

17.4.1 Öffnen eines Registrierungsschlüssels

Jeder Zugriff auf die Registrierung geschieht über Handles. Damit eine Anwendung auf einen Schlüssel der Registrierung zugreifen kann, muß diese den Handle eines bereits bestehenden, geöffneten Schlüssels verwenden. Einige vordefinierte Schlüssel-Handles sind immer geöffnet. Dazu zählen *HKEY_LOCAL_MACHINE*, *HKEY_CLASSES_ROOT*, *HKEY_USERS* und *HKEY_CURRENT_USER*.

Mit der Funktion *RegOpenKeyEx* greifen Sie auf einen Registrierungsschlüssel zu. Um beispielsweise den Handle des Registrierungsschlüssels *HKEY_LOCAL_MACHINE\Software* zu ermitteln, ist der folgende Aufruf erforderlich:

```
RegOpenKeyEx(HKEY_LOCAL_MACHINE, "Software", 0, KEY_READ, &hKey);
```

Um auf einen Unterschlüssel von *HKEY_LOCAL_MACHINE\Software* zuzugreifen, können Sie die Funktion wie folgt aufrufen:

```
RegOpenKeyEx(HKEY_LOCAL_MACHINE, "Software\\Classes", 0, KEY_READ, &hKey);
```

Nachdem eine Anwendung einen Registrierungsschlüssel verwendet hat, muß dieser mit *RegCloseKey* geschlossen werden.

17.4.2 Abfrage eines Werts

RegQueryValueEx Ein Registrierungswert wird mit `RegQueryValueEx` ermittelt. Dazu muß zunächst der entsprechende Schlüssel mit `RegOpenKeyEx` geöffnet werden.

`RegQueryValueEx` bietet die Möglichkeit, den zum Speichern des gewünschten Werts erforderlichen Speicher zu berechnen, bevor der Wert ermittelt wird. Übergeben Sie dieser Funktion einen `NULL`-Zeiger als Datenpufferzeiger, gibt die Funktion die Länge des Datenpuffers zurück, ohne den Wert zu kopieren. Ein wiederholter Aufruf von `RegQueryValueEx` ist somit möglich: Der erste Aufruf ermittelt die Länge des Puffers, der zweite Aufruf kopiert den Wert:

```
RegQueryValueEx(hKey, "MyValue", NULL, &dwType, NULL, &dwSize);  
pData = malloc(dwSize);  
RegQueryValueEx(hKey, "MyValue", NULL, &dwType, pData, &dwSize);
```

Die Angabe verketteter Schlüssel und durch einen Backslash voneinander getrennte Werte werden von `RegQueryValueEx` nicht akzeptiert. Der folgende Aufruf ist daher fehlerhaft:

```
RegQueryValueEx(hKey, "MyKey\\Value", NULL, &dwType, pData, &dwSize); // ERROR!
```

17.4.3 Setzen eines Werts

RegSetValueEx Ein Wert kann innerhalb der Registrierung mit der Funktion `RegSetValueEx` gesetzt werden. Bevor diese Funktion verwendet werden kann, muß der entsprechende Schlüssel über die Funktion `RegOpenKeyEx` mit einem `KEY_SET_VALUE`-Zugriff geöffnet werden.

17.4.4 Erstellen eines neuen Schlüssels

RegCreateKeyEx Anwendungen können ebenfalls einen neuen Unterschlüssel in der Registrierung erstellen. Die Funktion `RegCreateKeyEx` erzeugt den neuen Schlüssel, öffnet diesen und ermittelt dessen Handle. Die Funktion wird außerdem zum Öffnen eines bereits bestehenden Schlüssels verwendet. Sie ist somit für Situationen geeignet, in denen eine Anwendung unabhängig davon auf einen Schlüssel zugreifen möchte, ob dieser existiert oder nicht. Während einer Installation ist diese Vorgehensweise üblich.

Unter Windows NT weist eine Anwendung einem Schlüssel zusätzlich einige Sicherheitsattribute zu, während dieser erstellt wird. Die Attribute bestimmen, wer auf den Schlüssel zugreifen und Daten darin ablegen oder daraus auslesen darf. Die Sicherheitsattribute des geöffneten Schlüssels werden mit `RegGetKeySecurity` ermittelt und mit `RegSetKeySecurity` gesetzt (sofern die Anwendung über die erforderlichen Rechte verfügt).

17.4.5 Weitere Registrierungsfunktionen

Die Funktionen `RegEnumKeyEx` und `RegEnumValue` führen die Unterschlüssel und Werte eines bestimmten Registrierungsschlüssels auf. Registrierungsschlüssel werden mit `RegDeleteKey` gelöscht. Andere Funktionen speichern und laden Unterschlüssel, stellen eine Verbindung zur Registrierung eines anderen Computers her und führen weitere administrative Aufgaben aus.

17.4.6 Ein Beispiel

Das folgende Kommandozeilenprogramm demonstriert die Verwendung der Registrierung aus einer Anwendung heraus. Es liest die Registrierungseinstellungen. Das in Listing 17.1 aufgeführte Programm wird mit der erweiterten API-Bibliothek über die Kommandozeile kompiliert:

CL READREG.CPP ADVAPI32.LIB

```
#include <windows.h>
#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#define STR_HKEY_LOCAL_MACHINE "HKEY_LOCAL_MACHINE"
#define STR_HKEY_CLASSES_ROOT "HKEY_CLASSES_ROOT"
#define STR_HKEY_USERS "HKEY_USERS"
#define STR_HKEY_CURRENT_USER "HKEY_CURRENT_USER"
#define LEN_HKEY_LOCAL_MACHINE (sizeof(STR_HKEY_LOCAL_MACHINE)-1)
#define LEN_HKEY_CLASSES_ROOT (sizeof(STR_HKEY_CLASSES_ROOT)-1)
#define LEN_HKEY_USERS (sizeof(STR_HKEY_USERS)-1)
#define LEN_HKEY_CURRENT_USER (sizeof(STR_HKEY_CURRENT_USER)-1)
#define SWAP_ENDIAN(x) (((x<<24)&0xFF000000)|((x<<8)&0xFF0000)|\
                        ((x>>8)&0xFF00)|((x>>24)&0xFF))

void printval(unsigned char *pBuffer, DWORD dwType, DWORD dwSize)
{
    switch (dwType)
    {
        case REG_BINARY:
            cout << "Binary data:";
            {
                for (unsigned int i = 0; i < dwSize; i++)
                {
                    if (i % 16 == 0) cout << '\n';
                    cout.fill('0');
                    cout << hex << setw(2) <<
                        (unsigned int)(pBuffer[i]) << ' ';
                }
            }
            cout << '\n';
            break;
        case REG_DWORD:
            cout.fill('0');
            cout << "Double word: " << hex << setw(8) <<
                *((unsigned int *)pBuffer) << '\n';
            break;
        case REG_DWORD_BIG_ENDIAN: // Intel-spezifisch!
            cout.fill('0');
            cout << "Big-endian double word: " << hex
                << setw(8)
```

*Listing 17.1:
Ein einfaches
Programm zum
Auslesen der
Registrierung*

```

        << SWAP_ENDIAN(*((unsigned int *)pBuffer))
        << '\n';
    break;
case REG_EXPAND_SZ:
    cout << "Expandable string: " << pBuffer << '\n';
    break;
case REG_LINK:
    cout << "Unicode link.";
    break;
case REG_MULTI_SZ:
    cout << "Multiple strings:\n";
    {
        char *pStr;
        int i;
        for (i = 0, pStr = (char *)pBuffer; *pStr!='\0';
            i++, pStr += strlen((char *)pStr) + 1)
        {
            cout << "String " << i << ": "
                << pStr << '\n';
        }
    }
    break;
case REG_NONE:
    cout << "Undefined value type.\n";
    break;
case REG_RESOURCE_LIST:
    cout << "Resource list.\n";
    break;
case REG_SZ:
    cout << "String: " << pBuffer << '\n';
    break;
default:
    cout << "Invalid type code.\n";
    break;
}
}
void main(void)
{
    char szKey[1000];
    char *pKey;
    HKEY hKey, hSubKey;
    DWORD dwType;
    DWORD dwSize;
    unsigned char *pBuffer;
    int nKey;
    while (1)
    {
        cout << "Enter key: ";
        cin.getline(szKey, 1000);
        nKey = strlen(szKey);
        hKey = NULL;
        if (!strcmp(szKey, STR_HKEY_LOCAL_MACHINE) &&
            nKey == LEN_HKEY_LOCAL_MACHINE)
            hKey = HKEY_LOCAL_MACHINE;
        if (!strcmp(szKey, STR_HKEY_CLASSES_ROOT, nKey) &&
            nKey == LEN_HKEY_CLASSES_ROOT)
            hKey = HKEY_CLASSES_ROOT;
        if (!strcmp(szKey, STR_HKEY_USERS, nKey) &&
            nKey == LEN_HKEY_USERS)
            hKey = HKEY_USERS;
        if (!strcmp(szKey, STR_HKEY_CURRENT_USER, nKey) &&
            nKey == LEN_HKEY_CURRENT_USER)
            hKey = HKEY_CURRENT_USER;
        if (hKey == NULL || szKey[nKey] != '\\')
        {

```

```

        cout << "Invalid key.\n";
        continue;
    }
    pKey = szKey + nKey + 1;
    nKey = strcspn(pKey, "\\");
    while (pKey[nKey] == '\\')
    {
        pKey[nKey] = '\\0';
        if (RegOpenKeyEx(hKey, pKey, NULL, KEY_READ, &hSubKey)
            == ERROR_SUCCESS)
        {
            RegCloseKey(hKey);
            hKey = hSubKey;
        }
        else
        {
            RegCloseKey(hKey);
            hKey = NULL;
            break;
        }
        pKey += nKey + 1;
        nKey = strcspn(pKey, "\\");
    }
    if (hKey == NULL)
    {
        cout << "Invalid key.\n";
        continue;
    }
    if (RegQueryValueEx(hKey, pKey, NULL, &dwType, NULL,
                        &dwSize) == ERROR_SUCCESS)
    {
        pBuffer = (unsigned char *)malloc(dwSize);
        if (pBuffer == NULL)
        {
            cout << "Insufficient memory.\n";
            break;
        }
        if (RegQueryValueEx(hKey, pKey, NULL, &dwType,
                            pBuffer, &dwSize) == ERROR_SUCCESS)
            printval(pBuffer, dwType, dwSize);
        else
            cout << "Error reading key.\n";
        free(pBuffer);
    }
    else
        cout << "Error reading key.\n";
    RegCloseKey(hKey);
}
}

```

Dieses Programm demonstriert einige Aspekte der Verwaltung der Registrierung. Die Ausführung beginnt mit der Schleife in der Funktion `main` (Sie verlassen das Programm mit `[Strg] + [C]`). Nachdem der Anwender einen Registrierungsschlüssel eingegeben hat, prüft das Programm zunächst, ob die Bezeichnung eines Top-Level-Schlüssels in der Eingabe enthalten ist. In diesem Fall beginnt die Iteration.

Die Eingabe kann Backslashes enthalten, die als Trennzeichen zwischen den Schlüsseln dienen. In der Iteration werden aus der Eingabe mehrere Zeichenfolgen mit Hilfe der Funktion `strcspn` generiert. Die Iteration

ist beendet, wenn die letzte Zeichenfolge extrahiert wurde, die den Namen eines Werts repräsentiert. Die Iteration erlaubt leere Namen (keine Zeichen) für Schlüssel und Werte.

Während der Iteration wird für jede extrahierte Zeichenfolge mit `RegOpenKeyEx` ein Schlüssel-Handle ermittelt. Mißlingt diese Ermittlung (weil der Anwender einen unzulässigen Schlüssel angegeben hat), wird der Schleifendurchlauf mit einem Fehler unterbrochen. Waren alle Angaben korrekt, wird der Wert, der der zuletzt extrahierten Zeichenfolge entspricht, mit Hilfe der Funktion `RegQueryValueEx` ermittelt. Diese Funktion wird zweimal aufgerufen. Der erste Aufruf ermittelt den zum Speichern des Werts erforderlichen Speicherplatz, während der zweite Aufruf den Wert selbst ausliest.

Der Wert wird in der Funktion `printval` ausgegeben. Der Funktion wird ein Zeiger auf den Wert, der Typ des Werts und dessen Länge übergeben, woraufhin eine formatierte Ausgabe vorgenommen wird.

Nachfolgend finden Sie einige Beispielausgaben, die von dem Programm erzeugt wurden:

```
Enter key:
HKEY_CURRENT_USER\Software\Microsoft\Access\7.0\Settings\Maximized
Double word: 00000001
Enter key: HKEY_LOCAL_MACHINE\Enum\Monitor\Default_Monitor\0001\ConfigFlags
Binary data:
00 00 00 00
Enter key:
```

Dieses Programm ist sehr nützlich, wenn der Typ eines Werts in der Windows-95/98-Registrierung ermittelt werden soll. Das Programm bietet weitaus mehr Informationen als der Registrierungseditor. Sie können die Anwendung derart modifizieren, daß sie ebenfalls in die Registrierung schreibt.

17.5 Zusammenfassung

In der Registrierung speichern Windows und Anwendungen Konfigurationsdaten. Die Registrierung ist ein strukturierter, hierarchisch aufgebauter Informationsspeicher. Registrierungseinträge, die auch als Schlüssel bezeichnet werden, tragen eine Bezeichnung und enthalten Unterschlüssel oder Werte.

Die Top-Level-Schlüssel der Registrierung bilden die Basisschlüssel `HKEY_USERS` und `HKEY_LOCAL_MACHINE`. Andere vordefinierte Schlüssel sind `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_USER`, `HKEY_CURRENT_CONFIG` und `HKEY_DYN_DATA`.

Ein Registrierungswert kann aus einem 4 Byte breiten Integer, einer Zeichenfolge, einer Zeichenfolgrengruppe oder binären Daten bestehen. Registrierungswerte werden gewöhnlich von Anwendungen, Installationsprozeduren oder Konfigurationshilfsmitteln, wie z.B. der Systemsteuerung, erzeugt. Darüber hinaus kann die Registrierung manuell mit dem Registrierungseditor bearbeitet werden.

Anwendungen speichern Konfigurationsinformationen gewöhnlich unter `HKEY_LOCAL_MACHINE\Software` und anwenderspezifische Daten unter `HKEY_CURRENT_USER\Software`. In beiden Fällen sollten Unterschlüssel erstellt werden, die den Unternehmensnamen, die Bezeichnung des Produkts sowie die Versionsnummer des Produkts aufnehmen.

Anwendungen, die bestimmte Dokumenttypen verwalten, speichern die Dateinamenserweiterung sowie die Klassendefinition unter `HKEY_CLASSES_ROOT`. OLE-Anwendungen legen Informationen ebenfalls unter diesem Schlüssel ab.

Die Win32-API stellt einige Funktionen für den Zugriff auf die Registrierung zur Verfügung. Mit Hilfe eines vordefinierten Registrierungsschlüssels erhalten Anwendungen einen Lese- und Schreibzugriff auf die Unterschlüssel. Anwendungen können außerdem eigene Schlüssel erzeugen oder löschen.

Ausnahmebehandlung

Kapitel 18



Die Win32-API unterstützt die STRUKTURIERTE AUSNAHMEBEHANDLUNG. Anwendungen bearbeiten mit Hilfe dieses Mechanismus verschiedene Fehler, die sich auf die Hard- und Software beziehen. Die strukturierte Ausnahmebehandlung sollte nicht mit dem Konzept der Ausnahmen in der C++-Sprache verwechselt werden, das ein Feature dieser Sprache ist. Die Win32-Ausnahmebehandlung ist nicht von ihrer Implementierung in einer bestimmten Programmiersprache abhängig. Um Mißverständnisse auszuschließen, befolgt dieses Kapitel die Konventionen der Microsoft-Dokumentation und verwendet den Ausdruck »C-Ausnahme« für die strukturierten Win32-Ausnahmen, während sich der Terminus »C++-Ausnahme« auf die Ausnahmebehandlung der C++-Sprache bezieht.

18.1 Ausnahmebehandlung in C und C++

Microsoft stellt einige Erweiterungen der C-Sprache zur Verfügung, die C-Anwendungen die Bearbeitung strukturierter Win32-Ausnahmen ermöglichen. Diese Ausnahmebehandlung unterscheidet sich von den Ausnahmen der C++-Sprache. Dieser Abschnitt bietet eine Übersicht über beide Mechanismen hinsichtlich der Ausnahmen in der Win32-Umgebung.

18.1.1 C-Ausnahmen

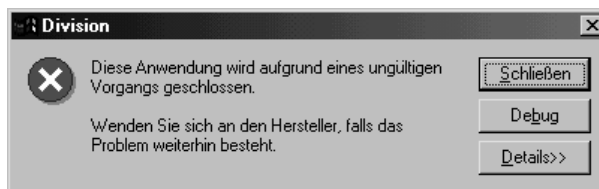
Was ist eigentlich eine Ausnahme? Wie arbeiten Ausnahmen? Betrachten Sie zunächst das Programm in Listing 18.1, bevor Sie eine Antwort auf diese Fragen erhalten.

Listing 18.1:
Ein Programm,
das eine Ausnah-
me erzeugt

```
void main(void)
{
    int x, y;
    x = 5;
    y = 0;
    x = x / y;
}
```

Natürlich führt die Integer-Division durch Null dazu, daß dieses Programm abgebrochen wird. Kompilieren Sie das Beispiel, und führen Sie die Anwendung unter Windows 95/98 aus, wird der in Abbildung 18.1 dargestellte Dialog angezeigt.

Abbildung 18.1:
Ein Fehler trat
während des
Versuchs auf,
einen Wert
durch Null zu
teilen



Was ist geschehen? Offensichtlich generiert der Prozessor einen Fehler, wenn Sie versuchen, einen Wert durch Null zu teilen (dieser Mechanismus betrifft die Hardware und kann nicht von uns beeinflusst werden). Der Fehler wird von dem Betriebssystem erkannt, das daraufhin nach einem entsprechenden AUSNAHMEBEARBEITER sucht. Wird kein Bearbeiter gefunden, führt das Betriebssystem die Standardausnahmebehandlung aus, die den abgebildeten Dialog darstellt.

Ausnahmen
abfangen Mit Hilfe der C-Ausnahmebehandlung kann diese Ausnahme abgefangen und bearbeitet werden. Betrachten Sie dazu auch das Programm in Listing 18.2.

Listing 18.2:
Bearbeiten der
Division-durch-
Null-Ausnahme

```
#include "windows.h"
void main(void)
{
    int x, y;
    __try
    {
        x = 5;
        y = 0;
        x = x / y;
    }
    __except (GetExceptionCode() == EXCEPTION_INT_DIVIDE_BY_ZERO
        ? EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
    {
        printf("Divide by zero error.\n");
    }
}
```

Starten Sie dieses Programm, wird der in Abbildung 18.1 dargestellte Dialog nicht mehr angezeigt. Statt dessen erhalten Sie die Meldung »Divide by zero error«. Die Anwendung wird anschließend beendet.

Der Anweisungsblock, der `__try` folgt, wird häufig als ÜBERWACHUNGSBLOCK bezeichnet. Dieser Block wird immer ausgeführt. Tritt ein Fehler innerhalb des Überwachungsblocks auf, wird der Ausdruck (der oft als FILTERAUSDRUCK bezeichnet wird) ausgewertet, der der Anweisung `__except` folgt. Der Ausdruck ist ein Integer, der einem der in Tabelle 18.1 aufgeführten Werte entspricht.

Symbolische Konstante	Wert	Beschreibung
<code>EXCEPTION_CONTINUE_EXECUTION</code>	-1	Setzt die Ausführung an der Position fort, an der der Fehler auftrat.
<code>EXCEPTION_CONTINUE_SEARCH</code>	0	Übergibt die Steuerung an die nächste Fehlerbehandlung.
<code>EXCEPTION_EXECUTE_HANDLER</code>	1	Führt den Ausnahmebearbeiter aus.

Tabelle 18.1:
Filterausdrücke

➡ Ist der Filterausdruck gleich -1 (`EXCEPTION_CONTINUE_EXECUTION`), wird die Programmausführung an der Position fortgesetzt, an der der Fehler auftrat. Damit ist die Programmzeile gemeint, die den Fehler erzeugte und nicht die darauf folgende Zeile. Der Fehler könnte somit erneut auftreten. Dies ist von dem Typ des Ausnahmefehlers abhängig. Bei einer Division durch Null würde die Ausnahme beispielsweise erneut erzeugt. Eine Fließkommadivision durch Null würde nicht erneut zu diesem Fehler führen. Sie sollten diesen Filterausdruck in jedem Fall mit großer Sorgfalt verwenden, um Endlosschleifen zu vermeiden. Diese Schleifen entstehen, wenn die Ausführung an der Position fortgesetzt wird, an der der Fehler auftrat, und die Ursache des Fehlers nicht beseitigt wurde.

In den verbleibenden Zuständen, die der Filterausdruck annehmen kann, wird der Geltungsbereich des Überwachungsblocks überschritten. Alle von der Ausnahme unterbrochenen Funktionsaufrufe werden beendet.

- ➡ Ergab die Auswertung des Filterausdrucks den Wert 1 (`EXCEPTION_EXECUTE_HANDLER`), wird die Steuerung an den Anweisungsblock übergeben, der `__except` folgt.
- ➡ Der dritte Wert des Filterausdrucks, 0 (`EXCEPTION_CONTINUE_SEARCH`), deutet auf verschachtelte Ausnahmen hin. Sehen Sie sich dazu

auch das Programm in Listing 18.3 an. Die Anwendung erzeugt zwei Ausnahmen für eine Fließkommadivision durch Null sowie für eine Integer-Division durch Null. Die beiden Ausnahmen werden unterschiedlich bearbeitet.

Listing 18.3:
Verschachtelte
Ausnahme-
bearbeiter

```
#include <stdio.h>
#include <float.h>
#include <windows.h>
int divzerofilter(unsigned int code, int *j)
{
    printf("Inside divzerofilter\n");
    if (code == EXCEPTION_INT_DIVIDE_BY_ZERO)
    {
        *j = 2;
        printf("Handling an integer division error.\n");
        return EXCEPTION_CONTINUE_EXECUTION;
    }
    else return EXCEPTION_CONTINUE_SEARCH;
}
void divzero()
{
    double x, y;
    int i, j;
    __try
    {
        x = 10.0;
        y = 0.0;
        i = 10;
        j = 0;
        i = i / j;
        printf("i = %d\n", i);
        x = x / y;
        printf("x = %f\n", x);
    }
    __except (divzerofilter(GetExceptionCode(), &j))
    {
    }
}
void main(void)
{
    _controlfp(_EM_OVERFLOW, _MCW_EM);
    __try
    {
        divzero();
    }
    __except (GetExceptionCode() == EXCEPTION_FLT_DIVIDE_BY_ZERO
        ? EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
    {
        printf("Floating point divide by zero error.\n");
    }
}
```

Tritt ein Fehler in der Funktion `divzero` auf, wird zunächst der Filterausdruck ermittelt. Dies führt zu einem Aufruf der Funktion `divzerofilter`. Die Funktion prüft, ob die Ausnahme aufgrund einer Integer-Division durch Null verursacht wurde. In diesem Fall wird der Wert des Divisors (`j`) korrigiert und der Wert `EXCEPTION_CONTINUE_EXECUTION` zurückgegeben. Die Ausführung wird daraufhin an der Position fortgesetzt, an der der Fehler auftrat. Jede andere Ausnahme führt dazu, daß

`divzerofilter` den Wert `EXCEPTION_CONTINUE_SEARCH` zurückgibt, so daß die Ausnahmebehandlung nach einem anderen Ausnahmearbeiter sucht.

Dieser andere Ausnahmearbeiter wurde in der `main`-Funktion installiert. Er bearbeitet Fließkommadivisionen durch Null. Die Ausführung wird jedoch nicht an der Position fortgesetzt, an der der Fehler auftrat. Statt dessen wird eine Fehlermeldung ausgegeben.

Wenn Sie das Programm starten, erhalten Sie die folgende Ausgabe:

```
Inside divzerofilter
Handling an integer division error.
i = 5
Inside divzerofilter
Floating point divide by zero error.
```

Wie Sie sehen, wird der in der Funktion `divzero` installierte Ausnahmefilter für die beiden Ausnahmen aktiviert. Die Ausnahme, die durch die Fließkommadivision durch Null erzeugt wird, bleibt unbearbeitet. Die Ausnahme wird daher dem Ausnahmearbeiter in der `main`-Funktion übergeben.

Um Fließkommaausnahmen bearbeiten zu können, muß die Funktion `_controlfp` aufgerufen werden. Diese Funktion ermöglicht Fließkommaausnahmen, die in der Intel-Architektur gewöhnlich nicht berücksichtigt werden. Die Funktion `_controlfp` ist in der Fließkommabibliothek enthalten, die zu dem IEEE-Standard kompatibel ist.



Einige der vorwiegend auftretenden C-Ausnahmen sind in Tabelle 18.2 aufgeführt.

Symbolische Konstante	Beschreibung
<code>EXCEPTION_ACCESS_VIOLATION</code>	Verweis auf einen unzulässigen Speicherbereich
<code>EXCEPTION_PRIV_INSTRUCTION</code>	Es wurde versucht, privilegierte Anweisungen auszuführen
<code>EXCEPTION_STACK_OVERFLOW</code>	Stack-Überlauf
<code>EXCEPTION_FLT_DIVIDE_BY_ZERO</code>	Fließkommadivision
<code>EXCEPTION_FLT_OVERFLOW</code>	Fließkommaergebnis zu groß
<code>EXCEPTION_FLT_UNDERFLOW</code>	Fließkommaergebnis zu klein
<code>EXCEPTION_INT_DIVIDE_BY_ZERO</code>	Integer-Division
<code>EXCEPTION_INT_OVERFLOW</code>	Integer-Ergebnis zu groß

Tabelle 18.2:
Filterwerte der
vorwiegend auf-
tretenden Aus-
nahmen

Exceptions auslösen Anwendungen können zusätzlich zu den vom System generierten Ausnahmen Software-Ausnahmen mit Hilfe der Funktion `RaiseException` erzeugen. Windows reserviert Ausnahmewerte, deren gesetztes 29. Bit benutzerdefinierte Software-Ausnahmen anzeigt.

18.1.2 C-Terminierungsbehandlung

Die C-Terminierungsbehandlung ist der Bearbeitung von C-Ausnahmen ähnlich. Sehen Sie sich bitte das Programm in Listing 18.4 an, damit Sie verstehen, zu welchem Zweck die Terminierungsbehandlung verwendet wird.

Listing 18.4:
Das Problem der
Reservierung
von Ressourcen

```
#include <stdio.h>
#include <windows.h>
void badmem()
{
    char *p;
    printf("allocating p\n");
    p = malloc(1000);
    printf("p[1000000] = %c\n", p[1000000]);
    printf("freeing p\n");
    free(p);
}

void main(void)
{
    __try
    {
        badmem();
    }
    __except (GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION ?
              EXCEPTION_EXECUTE_HANDLER :
              EXCEPTION_CONTINUE_SEARCH)
    {
        printf("An access violation has occurred.");
    }
}
```

In diesem Programm reserviert die Funktion `badmem` Speicher für das Zeichenarray `p`. Die Ausführung des Programms wird unterbrochen, wenn es auf ein unzulässiges Feldelement zugreift. In diesem Fall kann die Funktion den reservierten Bereich nicht freigeben, wie die folgende Ausgabe beweist:

```
allocating p
An access violation has occurred.
```

Sie lösen dieses Problem, indem Sie einen Terminierungsbearbeiter in der `badmem`-Funktion installieren, wie in Listing 18.5 dargestellt.

Listing 18.5:
Ein Terminie-
rungsbearbeiter

```
#include <stdio.h>
#include <windows.h>
void badmem()
{
    char *p;
    __try
    {
        printf("allocating p\n");
```

```

        p = malloc(1000);
        printf("p[1000000] = %c\n", p[1000000]);
    }
    __finally
    {
        printf("freeing p\n");
        free(p);
    }
}
void main(void)
{
    __try
    {
        badmem();
    }
    __except (GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION ?
              EXCEPTION_EXECUTE_HANDLER :
              EXCEPTION_CONTINUE_SEARCH)
    {
        printf("An access violation has occurred.");
    }
}

```

Starten Sie diese Anwendung, erhalten Sie die folgende Ausgabe:

```

allocating p
freeing p
An access violation has occurred.

```

Wie Sie sehen, sind die Anweisungen in der Funktion `badmem` nun in einem `__try`-Block enthalten, dem das Schlüsselwort `__finally` folgt. Der Anweisungsblock, der `__finally` folgt, wird unabhängig davon, in welcher Weise die Funktion beendet wird, immer ausgeführt. Wenn `badmem` den Geltungsbereich überschreitet, geben die Anweisungen in dem `__finally`-Block die Ressourcen frei, die zuvor reserviert wurden.

18.1.3 C++-Ausnahmebehandlung

Die Win32-Ausnahmebehandlung verwendet die Funktion `GetExceptionCode`, um den Ausnahmefehler zu ermitteln. Die C++-Ausnahmebehandlung ist im Gegensatz dazu typbasierend. Die Ausnahmeart wird somit über den Typ ermittelt.

Die meisten Beispiele, die die C++-Ausnahmebehandlung demonstrieren, verwenden dazu eine Klassendeklaration. Diese Vorgehensweise ist jedoch nicht notwendig und läßt die einfache Anwendung der C++-Ausnahmebehandlung nicht erkennen. Betrachten Sie bitte einmal das Beispiel in Listing 18.6. (Vergessen Sie nicht, der `cl`-Kommandozeile den Schalter `-GX` hinzuzufügen, wenn Sie dieses Beispiel und alle anderen Programme kompilieren, die die C++-Ausnahmebehandlung verwenden.)

```

#include <iostream.h>
int divide(int x, int y)
{
    if (y == 0) throw int();
}

```

*Listing 18.6:
Die C++-Ausnahmebehandlung*


```

        return x / y;
    }
    void main(void)
    {
        int x, y;
        try
        {
            x = 5;
            y = 0;
            x = divide(x, y);
        }
        catch (int)
        {
            cout << "A division by zero was attempted.\n";
        }
    }
}

```

Die Funktion `divide` erzeugt in diesem Beispiel eine Ausnahme vom Typ `int`, wenn versucht wird, eine Division durch Null auszuführen. Die Ausnahme wird von dem Ausnahmearbeiter in `main` abgefangen.

18.1.4 Terminierungsbehandlung in C++

C++-Ausnahmen können ebenfalls für die Terminierungsbehandlung verwendet werden. Ein C++-Programm verfügt über die Möglichkeit, einen Codeblock unter Verwendung eines »jeden Fehler abfangenden« Ausnahmearbeiters zu bilden und Ressourcen freizugeben, bevor alle Ausnahmen mit der Funktion `throw` einem High-Level-Bearbeiter übergeben werden. Sehen Sie sich dazu auch bitte das Programm in Listing 18.7 an, das eine C++-Variante des Beispiels in Listing 18.5 ist.

Listing 18.7:
Terminierungs-
behandlung mit
C++-Ausnahmen

```

#include <stdio.h>
#include <windows.h>
void badmem()
{
    char *p;
    try
    {
        printf("allocating p\n");
        p = (char *)malloc(1000);
        printf("p[1000000] = %c\n", p[1000000]);
    }
    catch(...)
    {
        printf("freeing p\n");
        free(p);
        throw;
    }
}
void main(void)
{
    try
    {
        badmem();
    }
    catch(...)
    {
        printf("An exception was raised.");
    }
}

```

Nach dem Programmstart wird der folgende Text ausgegeben:

```
allocating p
freeing p
An exception was raised.
```

Der Ausnahmerearbeiter in der Funktion `badmem` besitzt dieselbe Aufgabe wie der `__finally`-Block in der C-Ausnahmebehandlung.

Obwohl diese Beispiele die Leistungsfähigkeit der C++-Ausnahmebehandlung mit C-Programmcode demonstrieren, bietet die Verwendung der Ausnahmebehandlung in Klassen einige Vorteile. Wird die Ausnahme beispielsweise weitergegeben, wird ein Objekt von dem Typ der Ausnahme erzeugt. Es ist daher möglich, zusätzliche Informationen über die Ausnahme in Form von Member-Variablen zur Verfügung zu stellen. Darüber hinaus können Konstruktoren und Destruktoren den nicht sehr eleganten Mechanismus zur Freigabe von Ressourcen ersetzen, der in Listing 18.7 verwendet wird.

18.1.5 C++-Ausnahmeklassen

Visual C++ stellt eine Implementierung der `exception`-Klassenhierarchie in der Standard-C++-Bibliothek zur Verfügung. Diese Hierarchie besteht aus der `exception`-Klasse und davon abgeleiteten Klassen, die verschiedene Fehler repräsentieren, wie z.B. Laufzeitfehler. Die `exception`-Klasse sowie die davon abgeleiteten Klassen sind in der Header-Datei mit der Bezeichnung `exception` deklariert.

18.2 C- und C++-Ausnahmefehler

Der C-Compiler unterstützt keine C++-Ausnahmen. Der C++-Compiler wiederum unterstützt sowohl C++-Ausnahmen als auch die Microsoft-Erweiterung für C-Ausnahmen. Bisweilen ist die Verwendung beider Ausnahmearten erforderlich, um mit der C++-Ausnahmesyntax die strukturierten Win32-Ausnahmen abzufangen. Dazu stehen Ihnen zwei Verfahren zur Verfügung. Sie können den universellen Ausnahmerearbeiter oder eine Übersetzungsfunktion verwenden.

18.2.1 Der universelle Ausnahmerearbeiter

Die Terminierungsbehandlung des Beispiels in Listing 18.7 verwendet einen universellen Ausnahmerearbeiter. Dieser Bearbeiter, der jeden Fehler abfängt, hat die folgende Struktur:

```
catch(...)
{
}
```

*Ausnahmen
beliebigen Typs
abfangen*

Der universelle Ausnahmearbeiter fängt jede Ausnahme ab, auch C-Ausnahmen. Er bietet daher eine einfache Ausnahmebehandlung, die der in Listing 18.7 verwendeten gleicht. Leider verfügt der universelle Ausnahmearbeiter nicht über alle Informationen zu dem strukturierten Ausnahmetyp.

Sie werden nun sicherlich vermuten, daß zur Lösung dieses Problems eine Ausnahme vom Typ `unsigned int` abgefangen werden könnte (die Microsoft-Visual-C++-Dokumentation gibt darüber Aufschluß, daß dies der Typ für C-Ausnahmen ist). Sie müßten dann lediglich den Wert dieses Typs ermitteln. Betrachten Sie dazu bitte das Listing 18.8.

Listing 18.8:
Das Abfangen
von C-Ausnahmen
als C++-Ausnahmen
vom Typ
`unsigned int`

```
#include <windows.h>
#include <iostream.h>
void main(void)
{
    int x, y;
    try
    {
        x = 5;
        y = 0;
        x = x / y;
    }
    catch (unsigned int e)
    {
        if (e == EXCEPTION_INT_DIVIDE_BY_ZERO)
        {
            cout << "Division by zero.\n";
        }
        else throw;
    }
}
```

Dieses Programm wird nicht zu dem gewünschten Ergebnis führen. C-Ausnahmen können lediglich von einem universellen Ausnahmearbeiter abgefangen werden. Vielleicht wäre das Verwenden der `GetExceptionCode`-Funktion in dem C++-Block `catch` möglich, um den Typ der strukturierten Ausnahme zu ermitteln. Das Beispiel in Listing 18.9 unternimmt diesen Versuch.

Listing 18.9:
C++-Ausnahmearbeiter
können `GetExceptionCode`
nicht aufrufen

```
#include <windows.h>
#include <iostream.h>
void main(void)
{
    int x, y;
    try
    {
        x = 5;
        y = 0;
        x = x / y;
    }
    catch (...)
    {
        // Die folgende Zeile führt zu einem Compiler-Fehler
        if (GetExceptionCode() == EXCEPTION_INT_DIVIDE_BY_ZERO)
        {
            cout << "Division by zero.\n";
        }
    }
}
```

```

        else throw;
    }
}

```

Leider ist auch diese Vorgehensweise nicht möglich. Die Funktion `GetExceptionCode` wird als eine integrierte Funktion implementiert und kann lediglich als ein Bestandteil eines Filterausdrucks in der C-Anweisung `__except` aufgerufen werden. Wir benötigen somit einen anderen Mechanismus, um zwischen C-Ausnahmen und C++-Code zu unterscheiden.

Eine weitere mögliche Lösung wäre die Erzeugung eines C-Ausnahmebearbeiters, der alle C-Ausnahmen abfängt und eine C++-Ausnahme vom Typ `unsigned int` mit dem Wert des C-Ausnahmecodes weiterleitet. Ein Beispiel hierfür finden Sie in Listing 18.10.

```

#include <windows.h>
#include <iostream.h>
int divide(int x, int y)
{
    try
    {
        x = x / y;
    }
    catch(unsigned int e)
    {
        cout << "Inside C++ exception.\n";
        if (e == EXCEPTION_INT_DIVIDE_BY_ZERO)
        {
            cout << "Division by zero.\n";
        }
        else throw;
    }
    return x;
}

unsigned int catchall(unsigned int code)
{
    cout << "inside catchall: " << code << '\n';
    if (code != 0xE06D7363) throw (unsigned int)code;
    return EXCEPTION_CONTINUE_SEARCH;
}

void main(void)
{
    int x, y;
    __try
    {
        x = 10;
        y = 0;
        x = divide(x, y);
    }
    __except(catchall(GetExceptionCode())) {}
}

```

*Listing 18.10:
Aufretende C++-
Ausnahmen in ei-
nem C-Ausnah-
mefilter*

Dieses Verfahren ist mit einem Problem verbunden. Leitet die Funktion `catchall` eine C++-Ausnahme weiter, die der C++-Ausnahmebearbeiter nicht bearbeitet, wird diese Ausnahme wieder wie eine C-Ausnahme behandelt. Dies führt zu einem erneuten Aufruf der Funktion `catchall`. Dieser Vorgang wiederholte sich endlos, würde nicht der

Wert 0xE06D7363 überprüft, der eine besondere Bedeutung hinsichtlich der C++-Ausnahmen hat. Wir sollten uns jedoch nicht mit undokumentierten Themen beschäftigen. Es muß eine andere Lösung geben.

Sie werden nun möglicherweise fragen, wieso wir uns derart ausführlich mit diesem Problem beschäftigen, wenn C++-Programme doch die Microsoft-C-Ausnahmebehandlung verwenden können? Wieso rufen wir nicht einfach `__try` und `__except` auf? Dies wäre eine zulässige Lösung. Wenn Sie jedoch Ihren Programmcode portieren müssen, möchten Sie möglicherweise die C++-Ausnahmebehandlung verwenden und Microsoft-Ausnahmen ebenfalls lokalisieren.

18.2.2 Übersetzen von C-Ausnahmen

Glücklicherweise stellt die Win32-API eine Funktion zur Verfügung, die der Übersetzung einer C-Ausnahme in eine C++-Ausnahme dient. Der Name dieser Funktion lautet `_set_se_translator`. Das Beispiel in Listing 18.11 verwendet die Funktion.

Listing 18.11:
Übersetzen von
C-Ausnahmen
mit `_set_se_translator`

```
#include <windows.h>
#include <iostream.h>
#include <eh.h>
int divide(int x, int y)
{
    try
    {
        x = x / y;
    }
    catch(unsigned int e)
    {
        cout << "Inside C++ exception.\n";
        if (e == EXCEPTION_INT_DIVIDE_BY_ZERO)
        {
            cout << "Division by zero.\n";
        }
        else throw;
    }
    return x;
}

void se_translator(unsigned int e, _EXCEPTION_POINTERS* p)
{
    throw (unsigned int)(e);
}

void main(void)
{
    int x, y;
    _set_se_translator(se_translator);
    x = 10;
    y = 0;
    x = divide(x, y);
}
```

18.3 Zusammenfassung

Win32-Programmierer, die die C++-Sprache verwenden, müssen sich mit zwei separaten, nur teilweise kompatiblen Ausnahmebehandlungsmechanismen vertraut machen. Strukturierte Win32-Ausnahmen werden gewöhnlich von dem Betriebssystem erzeugt. Diese Ausnahmen beziehen sich nicht auf die implementierte Programmiersprache und übergeben der Anwendung die Fehlerursache in einem 32-Bit-Wert ohne Vorzeichen.

Im Gegensatz dazu sind C++-Ausnahmen typenorientiert. Die Ausnahme wird gewöhnlich von dem Typ des Objekts abgeleitet, das während der Weiterleitung der Ausnahme verwendet wird.

C-Programme verwenden die Schlüsselworte `__try` und `__except` (die Microsoft-Erweiterungen der C-Sprache bilden), um strukturierte Ausnahmen zu bearbeiten. Ausnahmebearbeiter können verschachtelt werden. Der Typ einer Ausnahme wird mit `GetExceptionCode` in dem Filterausdruck `__except` ermittelt. Abhängig von dem Wert des Filterausdrucks kann eine Ausnahme von dem Ausnahmebearbeiter ausgewertet werden, die Ausführung an der Stelle fortgesetzt werden, an der der Fehler auftrat, oder die Steuerung an den nächsten Ausnahmebearbeiter abgegeben werden. Eine nichtbearbeitete Ausnahme führt zu einem Anwendungsfehler.

C-Programme können außerdem Terminierungsbearbeiter verwenden, die über die Schlüsselworte `__try` und `__finally` installiert werden. Ein Terminierungsbearbeiter gewährleistet, daß eine Funktion, die aufgrund einer Ausnahme unterbrochen wurde, alle Ressourcen korrekt freigeben kann.

C++-Programme bearbeiten Ausnahmen mit `try` und `catch`. Die Deklaration des Typs der Ausnahme folgt dem Schlüsselwort `catch`. Dieses Schlüsselwort kann zusammen mit der Auslassungszeichendeklaration (...) zum Abfangen aller Fehler verwendet werden. Sie können diese Kombination, analog zu dem `__finally`-Block in der C-Ausnahmebehandlung, ebenfalls als Terminierungsbearbeiter einsetzen.

Da C++-Programme ebenfalls C-Ausnahmen benutzen können, ist eine gemeinsame Verwendung dieser beiden Ausnahmebehandlungsmechanismen möglich. C++-Anwendungen fangen C-Ausnahmen mit einem universellen Ausnahmebearbeiter ab. Dieses Verfahren ermöglicht einer Anwendung jedoch nicht, den Ausnahmecode zu ermitteln. Ein C++-Programm kann daher eine Ausnahme-Übersetzungsfunktion installieren, die strukturierte C-Ausnahmen in typenorientierte C++-Ausnahmen umwandelt.

Die MFC



- 
- 19. Microsoft Foundation Classes: Eine Übersicht
 - 20. Das MFC-Anwendungsgerüst
 - 21. Die Arbeit mit Dokumenten und Ansichten
 - 22. Dialoge und Registerdialoge
 - 23. MFC-Unterstützung für Standarddialoge und Standardsteuerelemente
 - 24. Gerätekontext und GDI-Objekte
 - 25. Serialisierung: Datei- und Archivobjekte
 - 26. Container-Klassen
 - 27. Ausnahmen, Multithreading und andere MFC-Klassen
- 

Microsoft Foundation Classes: Eine Übersicht

Kapitel 19

Die MFC-Bibliothek (MFC steht für »Microsoft Foundation Classes«) ist eine der wesentlichen Komponenten des Visual-C++-Entwicklungssystems. Diese Sammlung verschiedener C++-Klassen kapselt einen großen Bereich der Win32-API und stellt einen leistungsfähigen Anwendungsrahmen für Anwendungen zur Verfügung.

19.1 MFC und Anwendungen

Eine typische MFC-Anwendung wird mit dem Visual-C++-Anwendungsassistenten erzeugt. Dieser ist jedoch nicht unbedingt erforderlich, um eine MFC-Anwendung zu erstellen. Viele MFC-Klassen werden in einfachen Programmen verwendet. Sogar Kommandozeilenanwendungen (Konsolenanwendungen) nutzen diese Klassen.

Das MFC-Programm in Listing 19.1 wird über die Kommandozeile mit der folgenden Anweisung kompiliert.

```
CL -MT HELLOMFC.CPP
```

```
#include <afx.h>
CFile& operator<<(CFile& file, const CString& string)
{
    file.Write(string, string.GetLength());
    return file;
}

void main(void)
{
    CFile file((int)GetStdHandle(STD_OUTPUT_HANDLE));
    CString string = "Hello, MFC!";
    file << string;
}
```

*Listing 19.1:
Eine einfache
MFC-Konsolen-
anwendung*

Die MFC bildet vorwiegend einen Klassensammelbehälter für die Windows-API. Ihre wichtigsten Klassen, wie z.B. `CWnd`, `CDialog` oder `CGdiObject`, repräsentieren die Ergebnisse dieser Design-Philosophie. Eine MFC-Anwendung ruft im Idealfall niemals eine Windows-API-Funktion direkt auf. Statt dessen konstruiert die Anwendung ein Objekt des entsprechenden Typs und nutzt dessen Elementfunktionen. Der Konstruktor und der Destruktor des Objekts überwachen die Initialisierung sowie die erforderliche Freigabe von Ressourcen.

Eine Anwendung, die beispielsweise in ein Fenster zeichnen soll, erstellt dazu ein `CClientDC`-Objekt und ruft dessen Elementfunktionen auf (die den GDI-Zeichenfunktionen ähnlich sind). Der Konstruktor `CClientDC` führt die entsprechenden Aufrufe aus, um einen Gerätekontext zu erstellen, den Abbildungsmodus einzurichten und andere Initialisierungen durchzuführen. Wird der Geltungsbereich des Objekts überschritten oder das Objekt mit dem `delete`-Operator zerstört, gibt der Destruktor automatisch den Gerätekontext frei. Diese Vorgehensweise erleichtert das Schreiben von Anwendungen, sogar ohne die Vorteile des Visual Studio, des Anwendungsassistenten und anderer leistungsfähiger Features.

MFC-Programmierung erfordert Kenntnisse in OOP und Einarbeitung in die MFC-Bibliothek

Das Problem, das sich Programmierern stellt, die noch nicht mit MFC gearbeitet haben, ist die Menge der Neuerungen, die es zu lernen gilt. Selbst die einfachste Aufgabe scheint die Suche in umfangreichen Anleitungen zu erfordern. Das Schreiben der Codezeilen

```
CClientDC *pDC;  
pDC = new CClientDC(this);  
pDC->Rectangle(0, 0, 100, 100);  
delete pDC;
```

ist nur dann sehr einfach, wenn Sie wissen, was genau diese Zeilen bedeuten. Andernfalls müssen Sie

1. sicherstellen, daß eine Klasse besteht, die die Funktionalität eines Gerätekontextes zur Verfügung stellt, der sich auf den Client-Bereich eines Fensters bezieht.
2. Anschließend müssen Sie die Elementfunktionen der Klasse `CClientDC` und deren übergeordnete Klassen untersuchen, um festzustellen, daß es eine `CDC::Rectangle`-Elementfunktion gibt.
3. Schließlich prüfen Sie Ihre Vorgehensweise erneut, um zu gewährleisten, daß keine weiteren Initialisierungen notwendig sind.

Ohne Hilfsmittel kann die Einarbeitung in die MFC sehr zeitaufwendig sein. Der Programmierer muß jedoch nicht auf Anleitungen verzichten. Abgesehen von dem Buch, das Sie gerade in Ihren Händen halten, gibt es Online-Referenzen, Hilfedateien, Beispielpprogramme und natürlich den Anwendungs-Assistenten.

Eine vereinfachte Übersicht ist sehr hilfreich, wenn ein komplexes Thema verstanden werden soll. Erlauben Sie mir daher, die verbleibenden Abschnitte dieses Kapitels dieser Übersicht zu widmen, bevor Sie detailliert über die MFC informiert werden.

19.2 MFC-Grundlagen

Die Klassen der MFC sind in verschiedene Kategorien unterteilt. Die beiden wesentlichen Kategorien sind die ANWENDUNGSARCHITEKTURKLASSEN sowie die FENSTERUNTERSTÜTZUNGSKLASSEN. Andere Kategorien enthalten Klassen, die unterschiedliche System-, GDI- und andere Dienste umfassen, wie z.B. die Unterstützung des Internet.

Die überwiegende Anzahl der MFC-Klassen ist von einer allgemeinen Basis abgeleitet, der `CObject`-Klasse. Diese Klasse implementiert zwei wichtige Features:

- ➔ SERIALISIERUNG und
- ➔ LAUFZEITTYPINFORMATIONEN.

Beachten Sie bitte, daß es die `CObject`-Klasse bereits vor dem neuen C++-Mechanismus der Laufzeittypinformationen (RTTI = Runtime Type Information) gab. Die `CObject`-Klasse unterstützt RTTI nicht. Einige einfache Unterstützungsklassen sind nicht von `CObject` abgeleitet.



Die wesentlichen MFC-Kategorien sind in Abbildung 19.1 dargestellt.

Aufgrund der besonderen Bedeutung von `CObject` wird diese Klasse zunächst erläutert.

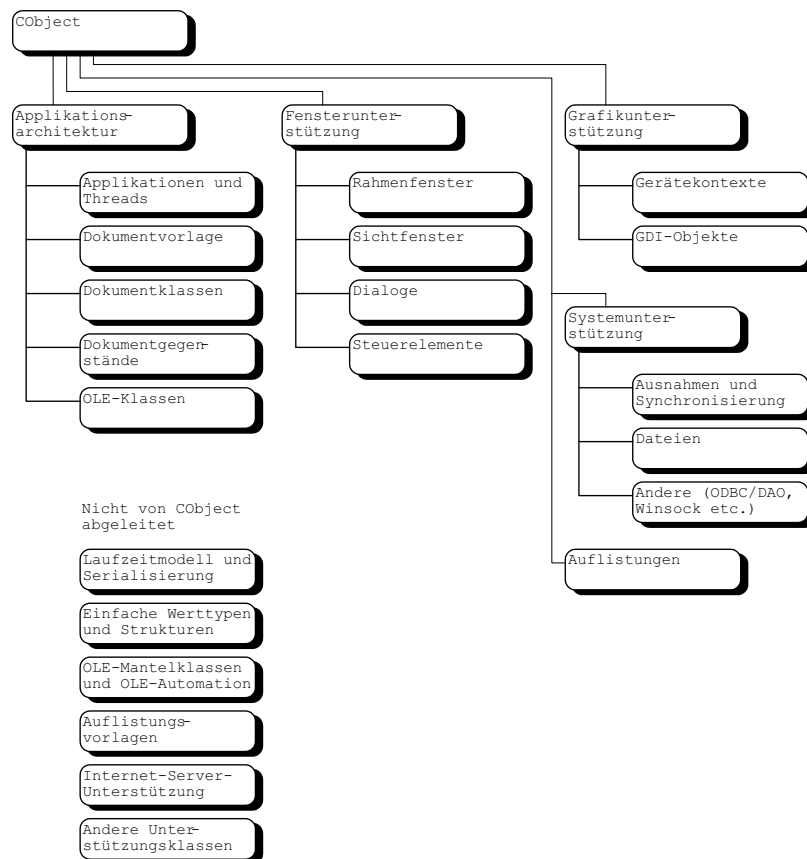
19.2.1 Die `CObject`-Klasse: Serialisierung und Typinformationen

Wie bereits erwähnt wurde, implementiert die `CObject`-Klasse Serialisierung und Laufzeittypinformationen. Doch was ist mit diesen Konzepten gemeint?

Serialisierung

Serialisierung ist die Konvertierung eines Objekts in ein beständiges Objekt, respektive die Konvertierung eines beständigen Objekts in ein gewöhnliches Objekt. Einfach beschrieben ist die Serialisierung das Speichern eines Objekts in eine Datei oder das Auslesen eines Objekts aus einer Datei.

Abbildung 19.1:
Übersicht über
die MFC



*Zeigerelemente
in Objekten
stellen ein Problem dar*

Doch wozu wird die Serialisierung benötigt? Wieso ist

```
cout << myObject;
```

nicht ausreichend? Jeder Programmierer weiß, daß das Schreiben von Objekten in eine Datei sehr kompliziert sein kann, wenn die Objekte Zeiger enthalten. Lesen Sie das Objekt später aus der Datei aus, besteht die Möglichkeit, daß die Elemente, auf die der Zeiger verweist, entfernt wurden oder nicht mehr im Speicher vorhanden sind. Doch das ist nicht alles. MFC-Objekte werden nicht ausschließlich in Dateien geschrieben. Die Serialisierung wird ebenfalls dazu verwendet, ein Objekt in die Zwischenablage zu kopieren oder auf die OLE-Einbettung vorzubereiten.

CArchive Die MFC-Bibliothek verwendet **CArchive**-Objekte für die Serialisierung. Ein **CArchive**-Objekt repräsentiert einen beständigen Speicher. Soll ein Objekt serialisiert werden, ruft **CArchive** die Elementfunktion **Serialize** für dieses Objekt auf, die eine der überschreibbaren Funktionen in

`CObject` ist. Das Objekt weiß daher selbst, wie es auf das dauerhafte Speichern vorbereitet werden muß. Das `CArchive`-Objekt hingegen weiß, wie der resultierende Datenstrom an das beständige Speichermedium übertragen wird.

Dazu ein Beispiel, das die Zeichenfolgenklasse `CMyString` implementiert. (Beachten Sie bitte, daß diese nichts mit der MFC-Klasse `CString` gemein hat. Dieses Beispiel soll lediglich die `CObject`-Serialisierung demonstrieren.) *Beispiel*

`CMyString` verfügt über zwei Datenelemente:

- ➡ ein Element repräsentiert die Länge der Zeichenfolge, während das
- ➡ andere Element auf die Zeichenfolgendaten verweist.

Im Gegensatz zu C-Zeichenfolgen kann eine `CMyString`-Zeichenfolge eingebettete Null-Zeichen enthalten und erfordert keine abschließende Null-Zeichenfolge. Die Deklaration der Klasse `CMyString` geschieht wie folgt (lediglich die Datenelemente und die Elementfunktion `Serialize` sind aufgeführt):

```
class CMyString
{
private:
    WORD m_nLength;
    LPSTR m_pString;
public:
    virtual void Serialize(CArchive &ar);
};
```

Wieso verwendet das Beispiel den Windows-Typ `WORD`, anstatt `m_nLength` als Integer zu deklarieren? Dafür gibt es einen wichtigen Grund. Windows garantiert, daß der Typ `WORD` einen 16-Bit-Integer in allen aktuellen und zukünftigen Versionen von Windows repräsentiert. Dies ist besonders für das Speichern von Daten auf einem beständigen Speicher bedeutend. Der Typ `WORD` gewährleistet, daß Datendateien, die von unserer Anwendung unter einer bestimmten Version des Betriebssystems geschrieben wurden, unter einer anderen Betriebssystemversion eingelesen werden können. Würden wir statt dessen `int` verwenden, müßten wir berücksichtigen, daß `int` unter Windows 3.1 ein 16-Bit-Typ und unter Windows 95 sowie Windows NT ein 32-Bit-Typ ist. Datendateien, die unter verschiedenen Betriebssystemen erstellt wurden, wären somit inkompatibel zueinander. *Der Datentyp WORD*

Die Elementfunktion `Serialize` liest Daten aus einem `CArchive`-Objekt und schreibt Daten in dieses Objekt. Dennoch können Sie nicht einfach `m_nLength` und `m_pString` in das Archiv schreiben. Sie verwenden statt dessen die Daten, auf die `m_pString` verweist, also die Zeichenfol- *Serialize*

ge selbst. Sollen die Daten eingelesen werden, müssen Sie zunächst die Länge der Zeichenfolge ermitteln und anschließend den erforderlichen Speicher reservieren:

```
CMyString::Serialize(CArchive &ar)
{
    if (ar.IsStoring()) // Schreiben oder Lesen?
    {
        ar << m_nLength;
        ar.Write(m_pString, m_nLength);
    }
    else
    {
        ar >> m_nLength;
        m_pString = new char[m_nLength];
        ar.Read(m_pString, m_nLength);
    }
}
```

Die Serialisierung ist an den Einsatz bestimmter MFC-Makros gebunden

Zur korrekten Kompilierung und Ausführung dieses Programmcodes sind einige Hilfsmakros erforderlich. Damit eine Klasse serialisiert werden kann, muß das Makro `DECLARE_SERIAL` in der Klassendeklaration angegeben und das Makro `IMPLEMENT_SERIAL` in der Implementierungsdatei der Klasse verwendet werden. Ein besonderes Feature, das diese Makros der Klasse hinzufügen, ist die MFC-Laufzeittypinformation.

Doch wieso ist die Laufzeittypinformation für eine erfolgreiche Serialisierung notwendig? Stellen Sie sich bitte einmal vor, was geschieht, wenn Daten aus einem beständigen Speicher gelesen werden. Bevor das Objekt eingelesen wird, besitzen wir keine Informationen über dieses Objekt. Wir wissen lediglich, daß es von `CObject` abgeleitet ist. Laufzeittypinformationen, die mit dem Objekt serialisiert wurden, ermitteln dessen Typ. Nachdem die Typinformation vorhanden ist, kann das `CArchive`-Objekt ein neues Objekt des ermittelten Typs erstellen und dessen `Serialize`-Elementfunktion aufrufen, um die spezifischen Objektdaten einzulesen. Ohne die Laufzeittypinformation wäre diese Vorgehensweise nicht möglich.

Laufzeittypinformation

Die MFC realisiert Laufzeittypinformationen mit Hilfe der Klasse `CRuntimeClass` und verschiedenen Hilfsmakros.

Die `CRuntimeClass`-Klasse verfügt über Elementvariablen, die die Bezeichnung der Klasse und die Größe eines Objekts aufnehmen, das sich auf diese Klasse bezieht. Diese Informationen dienen nicht nur der Bestimmung der Klasse, sondern werden ebenfalls für die Serialisierung verwendet.

Anwendungen setzen `CRuntimeClass` selten direkt ein. Statt dessen führen sie einige Makros aus, die ein `CRuntimeClass`-Objekt in der Deklaration der von `CObject` abgeleiteten Klasse einbetten und eine Implementierung zur Verfügung stellen.

Dazu werden drei Makrogruppen verwendet, die in Tabelle 19.1 aufgeführt sind.

Symbolische Konstanten	Beschreibung
DECLARE_DYNAMIC und IMPLEMENT_DYNAMIC	Fügt der Klasse Laufzeitinformationen hinzu. Ermöglicht die Verwendung der IsKindOf-Elementfunktion.
DECLARE_DYNCREATE und IMPLEMENT_DYNCREATE	Erzeugt eine dynamisch erstellbare Klasse über CRuntimeClass::CreateObject.
DECLARE_SERIAL und IMPLEMENT_SERIAL	Fügt einer Klasse die Möglichkeit zur Serialisierung hinzu. Ermöglicht den Einsatz der Operatoren << und >> zusammen mit CArchive.

Tabelle 19.1:
Hilfsmakros

Sie verwenden immer nur eine Makrogruppe gleichzeitig. Die Funktionalität von DECLARE_DYNCREATE / IMPLEMENT_DYNCREATE ist eine Untermenge der Funktionalität von DECLARE_DYNAMIC / IMPLEMENT_DYNAMIC. Die Funktionalität von DECLARE_SERIAL / IMPLEMENT_SERIAL ist eine Untermenge der Funktionalität von DECLARE_DYNAMIC / IMPLEMENT_DYNAMIC.

Betten Sie das gewünschte DECLARE_-Makro in die Deklaration Ihrer Klasse ein, und fügen Sie das entsprechende IMPLEMENT_-Makro Ihrer Implementierungsdatei hinzu. Um eine CMyString-Klasse zu erstellen, die sich von CObject ableitet und Serialisierung unterstützt, sollten Sie die folgende Klassendeklaration verwenden:

```
class CMyString : public CObject
{
    DECLARE_SERIAL(CMyString)
    ...
};
```

Der Implementierungsdatei fügen Sie das folgende Makro (außerhalb der Elementfunktionen) hinzu:

```
IMPLEMENT_SERIAL(CMyString, CObject, 0)
```

19.2.2 MFC und Mehrfachvererbung

Eine häufig gestellte Frage betrifft die Verwendung der Klassen der MFC unter Berücksichtigung der Mehrfachvererbung. Obwohl die Mehrfachvererbung mit MFC-Klassen möglich ist, ist diese Vorgehensweise nicht empfehlenswert.

Die CRuntimeClass-Klasse unterstützt keine Vererbung. CRuntimeClass wird von CObject zur Ermittlung von Laufzeitklasseninformationen, zur Erstellung dynamischer Objekte und für die Serialisierung verwendet.

Dies hat bedeutende Auswirkungen auf jeden Versuch, die Mehrfachvererbung in einem MFC-Programm zu verwenden.

Erfordert Ihr Projekt die Verwendung der Mehrfachvererbung mit MFC, möchte ich an dieser Stelle auf den Technischen Hinweis 16 der MFC-Dokumentation in der Visual-C++-Online-Dokumentation verweisen. Dieser Hinweis bietet eine sehr gute Übersicht über dieses Thema.

19.2.3 MFC und Windows-Objekte

Sehr viele MFC-Klassen repräsentieren Objekte unter Windows, wie z.B. Fenster, einen Gerätekontext oder ein GDI-Objekt. Beachten Sie bitte, daß ein Objekt einer derartigen MFC-Klasse (beispielsweise ein `CWnd`-Objekt) kein Windows-Objekt ist. Das `CWnd`-Objekt repräsentiert lediglich ein Fenster.

Dieselbe Aussage gilt für andere MFC-Klassen. Die Existenz eines Windows-Objekts impliziert nicht automatisch das Vorhandensein eines entsprechenden MFC-Objekts. Umgekehrt bedeutet die Existenz eines MFC-Objekts nicht automatisch, daß ein entsprechendes Windows-Objekt besteht. Häufig wird ein nicht zugewiesenes MFC-Objekt erstellt, das später einem bestehenden oder neu erstellten Windows-Objekt zugeteilt wird. Bisweilen werden temporäre MFC-Objekte erzeugt, die für kurze Zeit ein dauerhaftes Windows-Objekt repräsentieren (ein temporäres `CWnd`-Objekt kann beispielsweise den Desktop repräsentieren).

19.3 Fensterunterstützungsklassen

Fensterunterstützungsklassen stellen einen Sammelbehälter für allgemeine Fenstertypen zur Verfügung. Dazu zählen Rahmenfenster und Ansichtsfenster sowie Dialogfenster und Steuerelemente. Alle Fensterunterstützungsklassen werden von der `CWnd`-Klasse abgeleitet, die wiederum von `CObject` abgeleitet ist. Die `CWnd`-Klasse enthält die allgemeine Funktionalität aller Fenster. Die große Anzahl ihrer Elementfunktionen kann in verschiedene Kategorien unterteilt werden, die in Tabelle 19.2 aufgeführt sind.

Tabelle 19.2:
Die Kategorien
der `CWnd`-Elementfunktionen

Kategorie	Beschreibung
Initialisierung	Initialisieren und Erstellen von Fenstern
Fensterstatusfunktionen	Setzt oder ermittelt Fenstereinstellungen

Kategorie	Beschreibung
Größe und Position	Ermittelt oder ändert die Größe und Position
Fensterzugriff	Fensterbezeichnung
Aktualisierung und Zeichnen	Zeichenfunktionen
Koordinatenumwandlung	Umwandeln zwischen logischen und physikalischen Koordinaten
Fenstertext	Bearbeiten des Fenstertextes oder Ändern des Textformats
Bildlauf	Manipulation der Bildlaufleisten
Drag&Drop	Akzeptieren von Drag&Drop-Dateien
Schreibmarke	Bearbeitung der Schreibmarke
Dialogfeld	Manipulieren von Dialogfeldelementen
Menü	Bearbeiten von Menüs
QuickInfo	Bearbeiten von QuickInfos
Zeitgeber	Zeitgeber setzen und löschen
Alarm	Fensteranzeige und Nachrichtfelder
Fenster Nachrichten	Verwalten von Nachrichten
Zwischenablage	Manipulation des Inhalts der Zwischenablage
OLE-Steuerelemente	Bearbeiten der OLE-Steuerelemente
Überladbare Funktionen	Bearbeiten von Nachrichten und anderen Elementen

19.3.1 Rahmenfenster

Rahmenfenster beinhalten die Funktionalität des Hauptfensters der Anwendung und verwalten deren Menüleiste, Werkzeugleisten und der Statusleiste.

Die verschiedenen Rahmenfenster sind in Abbildung 19.2 dargestellt. Sie werden von SDI- und MDI-Anwendungen sowie für die OLE-In-Place-Bearbeitung genutzt. Alle Rahmenfenster werden von der `CFrameWnd`-Klasse abgeleitet, die wiederum von `CWnd` abstammt.

Diese Rahmenfensterklassen werden gewöhnlich als Basisklassen für benutzerdefinierte Rahmenfensterklassen verwendet.

Nahe Verwandte der Rahmenfenster sind die Steuerleisten, wie z.B. Symbolleisten und Statusleisten. Steuerleistenklassen werden von der Klasse `CControlBar` abgeleitet, die von `CWnd` abstammt. Diese Klassen sind in Abbildung 19.3 dargestellt.

Abbildung 19.2:
Rahmenfenster-
klassen

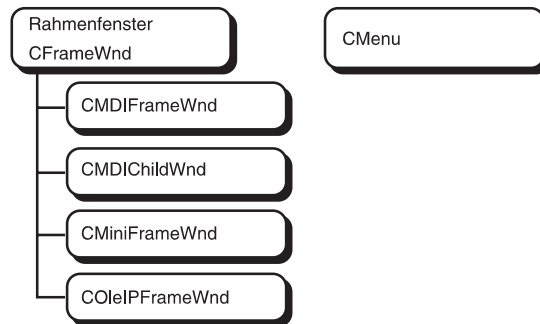
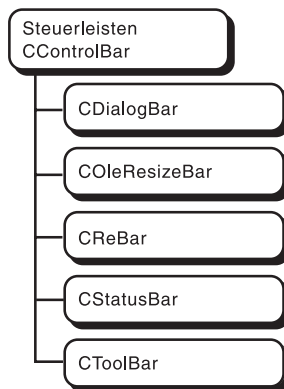


Abbildung 19.3:
Steuerleisten-
klassen



Die zusätzliche Klasse `C splitterWnd` wird zur Erstellung *teilbarer Fenster* verwendet. `C splitterWnd` wird gewöhnlich dazu verwendet, ein `C splitterWnd`-Objekt in ein Rahmenfensterobjekt einzubetten.

19.3.2 Ansichtsfenster

Ansichtsfenster beziehen sich wie Rahmenfenster auf den MFC-Anwendungsrahmen. Eine MFC-Anwendung verwendet Ansichtsfenster, um die Inhalte eines Dokuments dem Anwender zu präsentieren.

Verschiedene Ansichtsfenstertypen repräsentieren unterschiedliche Darstellungen der Ansicht eines Dokuments. Ansichtsfensterklassen unterstützen den Bildlauf, die Textbearbeitung, Listenfelder und strukturierte Listenansichten sowie Formulare, die Dialogen ähnlich sind.

Alle Ansichtsfensterklassen werden von der `CView`-Klasse abgeleitet, die von `CWnd` abstammt. Die Hierarchie der Ansichtsfensterklassen ist in Abbildung 19.4 dargestellt.

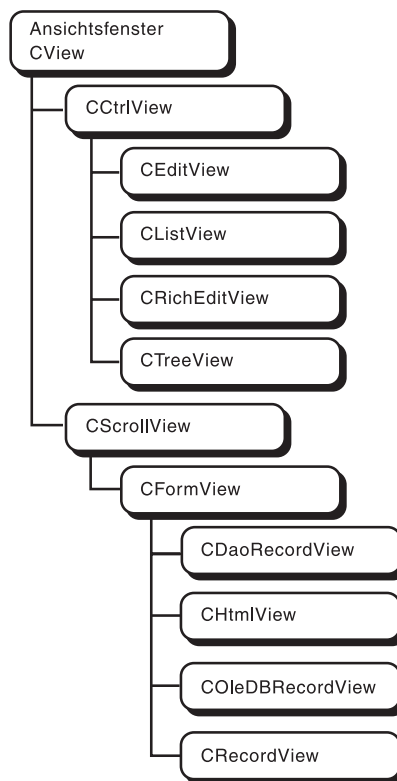


Abbildung 19.4:
Ansichtsfenster-
klassen

Wie Rahmenfensterklassen, dienen auch Ansichtsfensterklassen gewöhnlich als Basisklassen für benutzerdefinierte Klassen, die eine spezifische Ansichtsfunktionalität implementieren.

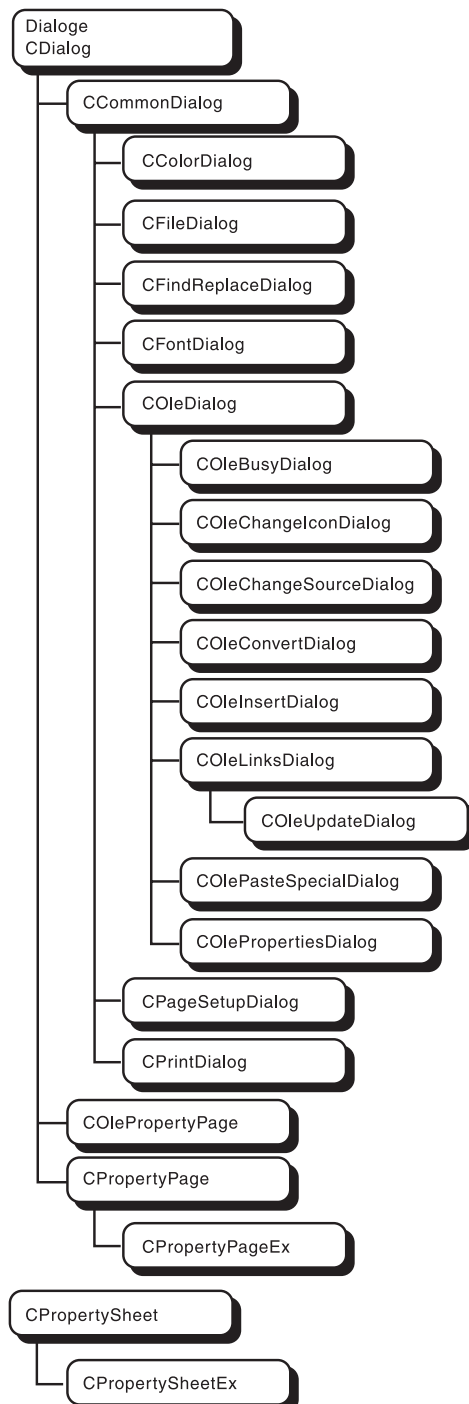
19.3.3 Dialoge

Dialogklassen enthalten die Funktionalität benutzerdefinierter Dialoge und von Standarddialogen. Die Hierarchie der Dialogklassen ist in Abbildung 19.5 dargestellt.

Dialogklassen können außerhalb von MFC-Anwendungsrahmenanwendungen verwendet werden. Das Programm in Listing 19.2 zeigt beispielsweise den Standarddialog FARBE AUSWÄHLEN mit Hilfe der `CColorDialog`-Klasse an. Sie kompilieren dieses Programm über die Kommandozeile mit der Anweisung

```
CL /MT COLORS.CPP.
```

Abbildung 19.5:
Dialogklassen



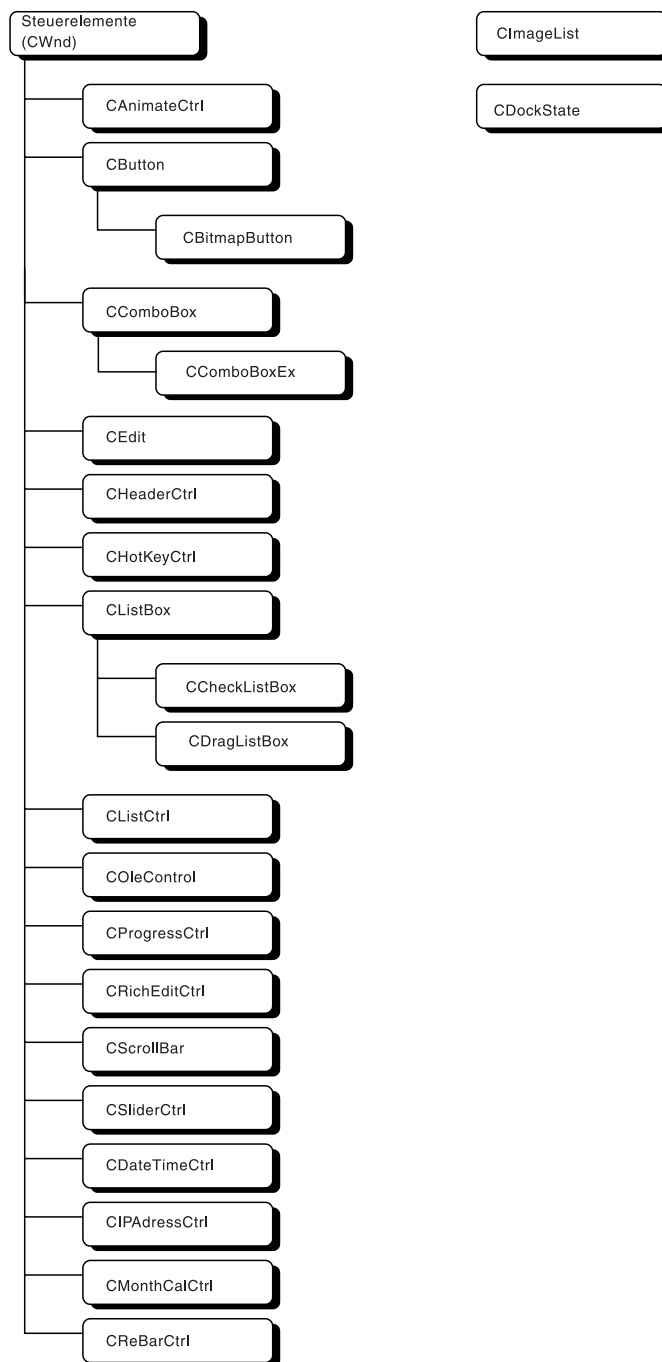


Abbildung 19.6:
Steuererelement-
klassen

Listing 19.2:
Verwenden einer
MFC-Dialogklas-
se in einer An-
wendung, die
keine MFC-An-
wendung ist

```
#include <afx.h>
#include <afxdlgs.h>
int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
    CColorDialog dlg;
    dlg.DoModal();
    return 0;
}
```

19.3.4 Steuerelemente

Steuerelementklassen umfassen die Funktionalität der Windows-Standardsteuerelemente, der allgemeinen Windows-95-Steuerelemente und der ActiveX-Steuerelemente (OCX). Die Hierarchie dieser Klassen ist in Abbildung 19.6 aufgeführt.

19.4 Anwendungsarchitekturklassen

Anwendungsarchitekturklassen werden von der Basisklasse `CCmdTarget` abgeleitet. Ein `CCmdTarget`-Objekt verfügt über eine NACHRICHTENTABELLE und kann Nachrichten bearbeiten. Da Fenster die Empfänger von Nachrichten sind, ist die `CWnd`-Klasse ebenfalls von `CCmdTarget` abgeleitet.

Zu den Anwendungsarchitekturklassen zählen

- ➔ Dokumentklassen,
- ➔ Dokumentvorlagenklassen,
- ➔ Dokumentobjektklassen,
- ➔ Anwendungsobjektklassen und
- ➔ verschiedene Klassen, die sich auf OLE beziehen.

19.4.1 Dokumentklassen

Dokumente repräsentieren Daten, die von dem Anwender geöffnet und bearbeitet werden können. Dokumentobjekte arbeiten mit Sichtobjekten zusammen, die die Präsentation der Daten sowie die Interaktion mit dem Anwender steuern. Die Hierarchie der Dokumentklassen ist in Abbildung 19.7 dargestellt.

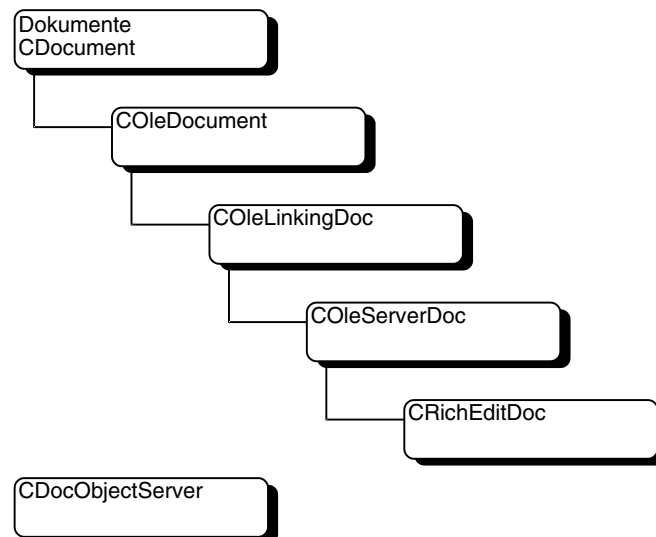


Abbildung 19.7:
Dokument-
klassen

19.4.2 Dokumentvorlagen

Dokumentvorlagen beschreiben das grundlegende Verhalten der benutzerdefinierten Dokumente sowie der Ansichtsklassen. Die Familie der Dokumentvorlagenklassen ist in Abbildung 19.8 dargestellt.

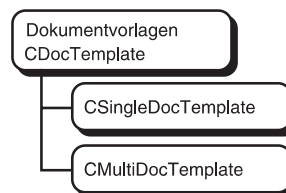


Abbildung 19.8:
Dokumentvorla-
genklassen

19.4.3 Anwendungsobjekte

Anwendungsobjekte repräsentieren Threads und Prozesse (Abbildung 19.9).

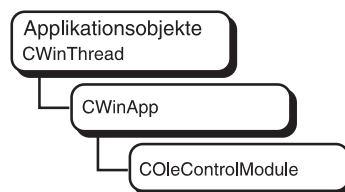


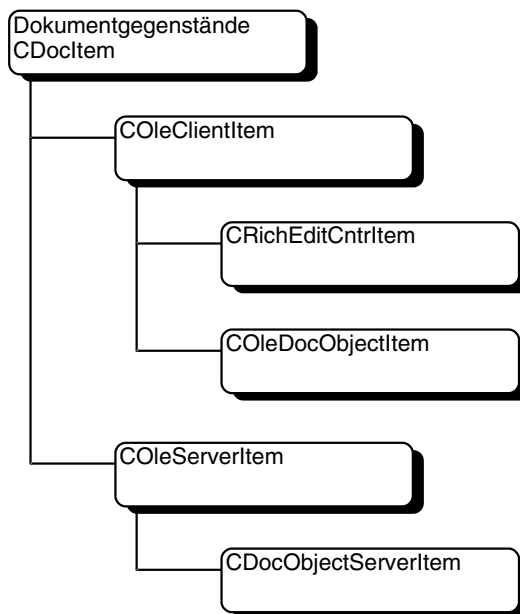
Abbildung 19.9:
Anwendungsob-
jekt-klassen

Jede Anwendung, die den MFC-Applikationsrahmen verwendet, verfügt über ein von `CWinApp` abgeleitetes Objekt, das die Hauptnachrichtenschleife Ihrer Anwendung zur Verfügung stellt.

19.4.4 Dokumentobjekte

Dokumentobjekte sind Objekte, die in einem Dokument enthalten sind. Das Dokument einer Zeichenanwendung kann beispielsweise Objekte enthalten, die Zeichenfiguren repräsentieren. Die MFC verwendet die Dokumentobjektklassen für OLE-Server und Client-Objekte. Die Hierarchie der Dokumentobjektklassen ist in Abbildung 19.10 dargestellt.

Abbildung 19.10:
Dokumentobjektklassen



19.4.5 Andere Anwendungsarchitekturklassen

Weitere Anwendungsarchitekturklassen tragen zur Implementierung von OLE innerhalb eines MFC-Anwendungsrahmens bei. Diese Klassen sind in Abbildung 19.11 aufgeführt.

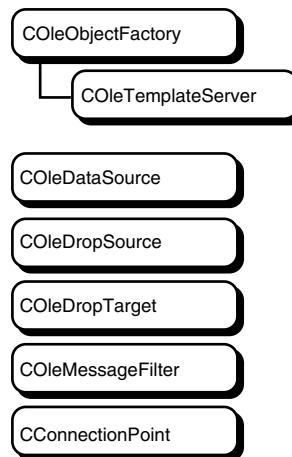


Abbildung 19.11:
OLE-bezogene
Anwendungsar-
chitekturklassen

19.5 Verschiedene Klassen

In diesem Kapitel werden Klassen, die System- und Grafikdienste unterstützen, als Auflistungen und die von `CObject` abgeleiteten Klassen als verschiedene Klassen bezeichnet.

19.5.1 Grafikunterstützungsklassen

Die GDI-Funktionalität wird von den in Abbildung 19.12 aufgeführten Gerätekontextklassen und GDI-Objektklassen zur Verfügung gestellt. Beide Klassenfamilien sind von `CObject` abgeleitet.

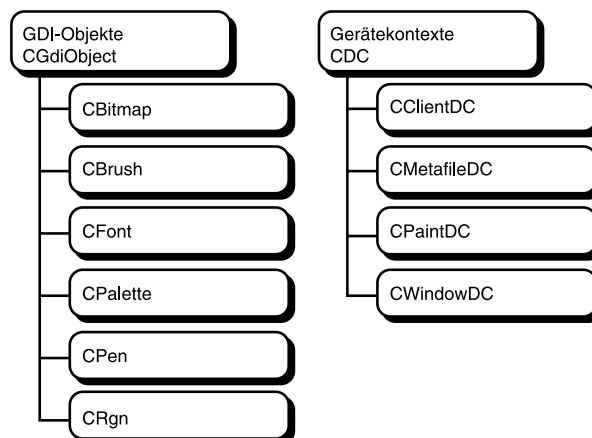
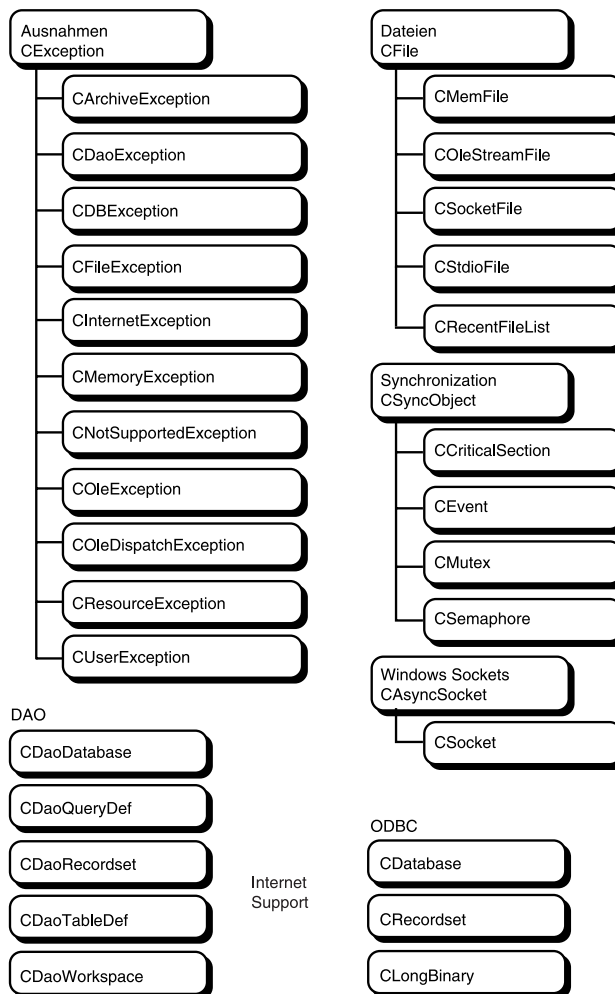


Abbildung 19.12:
Grafikunterstüt-
zungsklassen

19.5.2 Systemunterstützungsklassen

Systemunterstützungsklassen bieten die Funktionalität der Systemobjekte, wie z.B. Ausnahmen, Synchronisierungsobjekte und Dateien. Andere Systemunterstützungsklassen unterstützen ODBC, DAO, Win-Sock und Internet-Dienste. Die Hierarchie dieser Klassen ist in Abbildung 19.13 dargestellt.

Abbildung 19.13:
Systemunterstützungsklassen



19.5.3 Container-Klassen

Zu den Container-Klassen zählen Arrays, Listen und Tabellen. Arrays sind dynamisch reservierte Container, die von einem Integer-Index verwaltet werden. Listen sind geordnete Container, und Tabellen sind Auflistungen, die mit einem Schlüssel organisiert werden.

Die Hierarchie der Container-Klassen ist in Abbildung 19.14 aufgeführt.



Abbildung 19.14:
Container-Klassen

19.5.4 Nicht von CObject abgeleitete Klassen

Die MFC enthält einige Unterstützungsklassen, die nicht von der CObject-Klasse abgeleitet sind. Dazu zählen einfache Werttypen (z.B. CRect oder CString), typisierte Vorlagenauflistungen und weitere Klassen. Abbildung 19.15 zeigt diese Klassen.

Abbildung 19.15:
Klassen, die
nicht von COb-
ject abgeleitet
sind

Laufzeit-Objekt- modell

CArchive

CDumpContext

CRuntimeClass

einfache Werttypen

CPoint

CRect

CSize

CString

CTime

CTimeSpan

Strukturen

CCommandLineInfo

CCreateContext

CFileStatus

CMemoryState

CPrintInfo

Unterstützungsklassen

cCmdUI

ColeCmdUI

CDaoFieldExchange

CDataExchange

CDBVariant

CFieldExchange

COleDataObject

COleDispatchDriver

CPropExchange

CRectTracker

CWaitCursor

typisierte Vorlagen- auflistungen

CTypedPtrArray

CTypedPtrList

CTypedPtrMap

OLE-Mandelklassen

CFontHolder

CPictureHolder

OLE-Automationstypen

COleCurrency

COleDateTime

COleDateTimeSpan

COleVariant

Synchronisierung

CMultiLock

CSingleLock

Internet Server API

CHtmlStream

CHttpFilter

CHttpFilterContext

CHttpServer

CHttpServerContext

19.6 Zusammenfassung

Die MFC-Bibliothek repräsentiert einen leistungsfähigen Applikationsrahmen für die Erstellung von Windows-Anwendungen. Die Klassen der MFC umfassen die Windows-Funktionalität, die sich auf Anwendungen, Threads, Fenster, Dialoge, Steuerelemente, grafische Objekte und Gerätekontexte bezieht. Die MFC muß nicht ausschließlich für MFC-Applikationsrahmen-Anwendungen verwendet werden. Andere Windows-Programme und sogar Konsolenanwendungen können diese Bibliothek nutzen.

Die Basis der meisten MFC-Klassen ist die `CObject`-Klasse. Diese Klasse implementiert die Überprüfung der Laufzeittypen (die sich von dem neuen C++-RTTI-Feature unterscheidet) sowie die Serialisierung. Serialisierung ist ein leistungsfähiger, plattformunabhängiger Mechanismus zur Erstellung eines Objektabbildes auf einem beständigen Speicher. Außerdem können Objektdaten mit Hilfe dieses Mechanismus von solch einem Speicher eingelesen werden. Die Serialisierung ist nicht auf Dateien beschränkt. Sie wird ebenfalls für die Übertragung zur Zwischenablage und OLE verwendet.

Die wichtigsten MFC-Kategorien sind Anwendungsarchitekturklassen, Fensterunterstützungsklassen und weitere Klassen, die System-, GDI- und andere Dienste enthalten.

Fensterunterstützungsklassen entsprechen den verschiedenen Fenstertypen, die von dem System verwendet oder von der MFC-Bibliothek zur Verfügung gestellt werden. Dazu zählen Rahmen- und Ansichtsfenster, Dialoge sowie Steuerelemente. Diese Klassen werden von der `CWnd`-Klasse abgeleitet, die die von allen Fenstern angebotene grundlegende Funktionalität enthält.

`CWnd` selbst ist von der `CCmdTarget`-Klasse abgeleitet, der Basisklasse aller Klassen, die Nachrichtentabellen verwenden und Nachrichten bearbeiten sowie weiterleiten. Die Anwendungsarchitekturklassen werden ebenfalls von `CCmdTarget` abgeleitet. Dazu zählen Klassen für Dokumente, Dokumentvorlagen, Dokumentobjekte, OLE sowie Thread- und Prozeßobjekte. Der zuletzt genannte Typ wird als `CWinApp` bezeichnet. Jede MFC-Applikationsrahmen-Anwendung enthält ein von `CWinApp` abgeleitetes Objekt, das die Hauptnachrichtenschleife der Anwendung implementiert.

Das MFC- Anwendungsgerüst

Kapitel 20

Wie ist eine gewöhnliche MFC-Anwendung aufgebaut? Wie nutzt diese Anwendung die Anwendungs-, Dokumentvorlagen- und Dokumentklassen? Wie erstellen Sie solch eine Anwendung? Diese Fragen werden in den folgenden Abschnitten beantwortet.

20.1 Ein einfaches MFC- Anwendungsgerüst

Sie werden erneut ein Hello-World-Programm erstellen, das jedoch eine mit dem MFC-Anwendungsassistenten generierte Anwendung sein wird. Wir verwenden diese Anwendung zur Erörterung der MFC-Features und der Beziehungen zwischen den verschiedenen Klassen des Programms.

20.1.1 Erstellen des Projekts

Ich würde unser Projekt gerne YAHWA nennen (Abkürzung für Yet Another Hello World Application). Der Anwendungsassistent würde daraus jedoch Dateinamen generieren, die aus mehr als acht Zeichen bestünden. Diese Dateinamen könnten zu Problemen führen, wenn das Projekt auf einer ISO9660-CD-ROM gespeichert werden soll. Wir verwenden daher einen kürzeren Dateinamen.

1. Öffnen Sie daher im Visual Studio das Menü DATEI, und wählen Sie daraus den Eintrag NEU.
2. Öffnen Sie in dem Dialog NEU das Register PROJEKTE, und selektieren Sie MFC-ANWENDUNGS-ASSISTENT (EXE).

3. Geben Sie die Bezeichnung des neuen Projekts ein (YAH) und wählen Sie ein Verzeichnis aus, in dem das Projekt gespeichert werden soll. Betätigen Sie anschließend bitte die Schaltfläche OK.
4. YAH soll ein SDI-Projekt werden (Single Document Interface). Setzen Sie daher die Option EINZELNES DOKUMENT (SDI) auf der ersten Dialogseite des Anwendungsassistenten.
5. Übernehmen Sie die Voreinstellungen des Assistenten bis zum vierten Schritt. Klicken Sie dort bitte auf die Schaltfläche WEITERE OPTIONEN.

Abbildung 20.1:
Weitere Optionen des YAH-Projekts



6. Geben Sie dort unter DATEIERWEITERUNG die Endung YAH ein, und bestimmen Sie unter BESCHRIFTUNG DES HAUPTFENSTERS den neuen Titel »HELLO, WORLD!« (oder einen Titel Ihrer Wahl). Betrachten Sie dazu bitte auch Abbildung 20.1.
7. Lassen Sie den Anwendungsassistenten das Projekt nach diesen Änderungen erstellen (Schaltfläche FERTIGSTELLEN). Im Anschluß daran öffnet das Visual Studio das Projekt, und zeigt den Projektarbeitsbereich in der Klassen-Ansicht an (Abbildung 20.2).



Abbildung 20.2:
YAH-Klassen

20.1.2 Das Anwendungsobjekt

Wie Sie sehen, erzeugt der Anwendungs-Assistent fünf Klassen für das YAH-Projekt.

Die Klasse CYAHApp ist von CWinApp abgeleitet und repräsentiert die Anwendung selbst. Die CYAHApp-Klasse ist in YAH.h deklariert. Diese Datei kann entweder mit einem Doppelklick auf die CYAHApp-Klasse in der Klassen-Ansicht oder durch einen Doppelklick auf den Dateinamen in der Dateien-Ansicht geöffnet werden.

CYAHApp

Drei Elementfunktionen sind für CYAHApp deklariert (Listing 20.1):

- ➔ ein Konstruktor,
- ➔ die virtuelle überladbare Funktion InitInstance und
- ➔ die Elementfunktion CAppAbout.

```
class CYAHApp : public CWinApp
{
public:
    CYAHApp();

    // Überladungen
    // Vom Klassen-Assistenten generierte Überladungen virtueller
    // Funktionen
    //{AFX_VIRTUAL(CYAHApp)
    public:
        virtual BOOL InitInstance();
    //}AFX_VIRTUAL

    // Implementierung

    //{AFX_MSG(CYAHApp)
    afx_msg void OnAppAbout();
    // HINWEIS - An dieser Stelle werden Member-Funktionen
    // vom Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

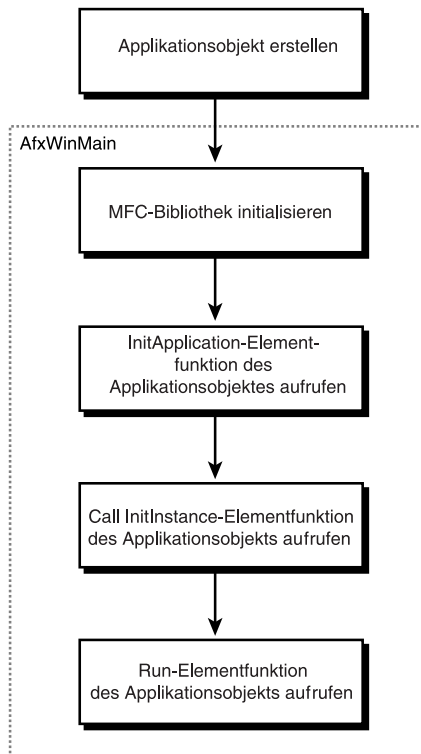
Listing 20.1:
Die CYAHApp-
Klassende-
klaration

*WinMain wird
ersetzt durch Afx-
WinMain*

In welcher Beziehung stehen diese Funktionen zu einer gewöhnlichen WinMain-Funktion in einer Anwendung, die keine MFC-Anwendung ist?

Ein Blick auf die Implementierung von `AfxWinMain` in der MFC-Source-datei `WINMAIN.CPP` gibt die Antwort. Die hier ausgeführten Initialisierungen sind in Abbildung 20.3 dargestellt.

Abbildung 20.3:
Die wesentlichen Initialisierungen



Wie aber kann das Anwendungsobjekt konstruiert werden, bevor `AfxWinMain` ausgeführt wird? Dazu wird in `YAH.cpp` ein globales Objekt vom Typ `CYAHApp` mit der Bezeichnung `theApp` deklariert. Beginnt die Ausführung, ist dieses Objekt bereits konstruiert (was bedeutet, daß dessen Konstruktor aufgerufen wurde).

Die Elementfunktionen `InitApplication` und `InitInstance` können überschrieben sein. Sie entsprechen einmaligen Initialisierungen und spezifischen Instanzinitialisierungen. Die Funktionen werden explizit von `AfxWinMain` aufgerufen, bevor die Nachrichtenschleife ausgeführt wird.

Betrachten Sie bitte einmal die erste Hälfte der Implementierungsdatei für die `CYAHApp`-Klasse, die mit `YAH.cpp` bezeichnet ist (lassen Sie dazu die untergeordneten Elemente der `CYAHApp`-Klasse in der Klassen-Ansicht anzeigen, und führen Sie einen Doppelklick auf eine der Elementfunktionen aus). Die zweite Hälfte dieser Datei enthält die Deklaration und Implementierung der Klasse `CAboutDlg`, die das Gerüst des Info-Dialogs der Anwendung bildet. Uns interessiert jedoch zunächst der in Listing 20.2 aufgeführte Bereich der Datei `YAH.CPP`.

```

////////////////////////////////////
// CYAHApp

BEGIN_MESSAGE_MAP(CYAHApp, CWinApp)
//{{AFX_MSG_MAP(CYAHApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // HINWEIS - Hier werden Mapping-Makros vom
        // Klassen-Assistenten eingefügt und entfernt.
        // Innerhalb dieser generierten Quelltextabschnitte
        // NICHTS VERÄNDERN!
//}}AFX_MSG_MAP
// Dateibasierte Standard-Dokumentbefehle
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// Standard-Druckbefehl "Seite einrichten"
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

////////////////////////////////////
// CYAHApp Konstruktion

CYAHApp::CYAHApp()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen
    // Alle wichtigen Initialisierungen in InitInstance platzieren
}

////////////////////////////////////
// Das einzige CYAHApp-Objekt

CYAHApp theApp;

////////////////////////////////////
// CYAHApp Initialisierung

BOOL CYAHApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standardinitialisierung
    // Wenn Sie diese Funktionen nicht nutzen und die Größe Ihrer
    // fertigen ausführbaren Datei reduzieren wollen, sollten Sie
    // die nachfolgenden spezifischen Initialisierungsroutinen, die
    // Sie nicht benötigen, entfernen.

#ifdef _AFXDLL
    Enable3dControls();      // Diese Funktion bei Verwendung von
                           // MFC in gemeinsam genutzten DLLs
                           // aufrufen
#else
    Enable3dControlsStatic(); // Diese Funktion bei statischen
                           // MFC-Anbindungen aufrufen

```

*Listing 20.2:
Die CYAHApp-
Klassenimple-
mentierung*

```

#endif

// Ändern des Registrierungsschlüssels, unter dem unsere
// Einstellungen gespeichert sind.
// Sie sollten dieser Zeichenfolge einen geeigneten Inhalt
// geben wie z.B. den Namen Ihrer Firma oder Organisation.
SetRegistryKey(_T("Local AppWizard-Generated Applications"));

LoadStdProfileSettings(); // Standard-INI-Dateioptionen
                          // einlesen (einschließlich MRU)

// Dokumentvorlagen der Anwendung registrieren.
// Dokumentvorlagen dienen als Verbindung zwischen Dokumenten,
// Rahmenfenstern und Ansichten.

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CYAHDoc),
    RUNTIME_CLASS(CMainFrame),      // Haupt-SDI-Rahmenfenster
    RUNTIME_CLASS(CYAHView));
AddDocTemplate(pDocTemplate);

// DDE-Execute-Open aktivieren
EnableShellOpen();
RegisterShellFileTypes(TRUE);

// Befehlszeile parsen, um zu prüfen auf
// Standard-Umgebungsbefehle DDE, Datei öffnen
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

// Verteilung der in der Befehlszeile angegebenen Befehle
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// Das einzige Fenster ist initialisiert und kann jetzt
// angezeigt und aktualisiert werden.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

// Öffnen mit Drag & Drop aktivieren
m_pMainWnd->DragAcceptFiles();

return TRUE;
}

```

InitInstance

Der Anwendungsrahmen erstellt eine überschriebene Version von `InitInstance`, nicht aber von `InitApplication`.



Da Win32-Anwendungen in separaten Speicherbereichen ausgeführt werden, sind nur wenige spezifische Anwendungsinitialisierungen möglich (im Gegensatz zu den instanzspezifischen Initialisierungen).

In `InitInstance` werden einige Initialisierungen vorgenommen.

Hauptfenster Eine der wichtigsten Aufgaben von `InitInstance` besteht darin, das Hauptfenster für die Anwendung zu erzeugen. Dies geschieht auf einem Umweg. Der Code für die Registrierung einer passenden Fensterklasse und die Erzeugung des Hauptfensters ist in der Implementierung der Klasse `CFrameWnd` versteckt, von der unsere Hauptfensterklasse `CMainFrame` abgeleitet wurde. Was bleibt, ist ein Rahmenfenster-Objekt von `CMainFrame` zu erzeugen. Dies geschieht im MFC-Applikationsgerüst nicht direkt (Einsatz des Operators `new`), sondern indirekt bei der Erstellung der Dokumentvorlage. (Ein Zeiger auf das Rahmenfenster-Objekt wird der Variablen `CWinThread::m_pMainWnd` zugewiesen, wozu es zum Hauptfenster der Anwendung wird, d. h., die Anwendung wird geschlossen, wenn das Fenster geschlossen wird.)

Dokumentvorlage Der wohl wichtigste Initialisierungsvorgang ist die Erstellung einer Dokumentvorlage. Ein Objekt vom Typ `CSingleDocTemplate` (da wir eine SDI-Anwendung selektiert haben) wird generiert und den Dokumentvorlagen der Anwendung mit der Elementfunktion `AddDocTemplate` hinzugefügt.

Die in den Dokumentvorlagen gespeicherten Informationen werden benötigt, wenn der Anwender aus dem Menü `DATEI` den Eintrag `NEU` auswählt. (Die Standardimplementierung dieser Anweisung befindet sich in der Funktion `CWinApp::OnFileNew`. Die Funktion verwendet die Vorlageninformation, um zu bestimmen, welche Objekte zur Darstellung des neuen Dokumentobjekts und der entsprechenden Ansicht erzeugt werden müssen.)

Sonstiges Hinzu kommt noch die Unterstützung für verschiedene Features, die wir im Anwendungsassistenten aktiviert haben, beispielsweise wurde die 3D-Darstellung für die YAH-Anwendung ausgewählt. Demgemäß wird diese Option in `InitInstance` aktiviert (`Enable3dControls`).



MDI-Anwendungen

Einige Anwendungen können unterschiedliche Dokumente bearbeiten. Eine grafische Anwendung könnte beispielsweise sowohl Bitmap- als auch Vektorgrafikdateien darstellen. Der Editor eines Programmierers kann möglicherweise Quelldateien (Text) und die Grafiken der Ressourcendateien bearbeiten. Doch wie realisieren MFC-Anwendungen unterschiedliche Dokumenttypen?

Dazu muß zunächst eine MDI-Anwendung (Multiple Document Interface) erstellt werden. (Eine mit dem Anwendungsassistenten generierte SDI-Anwendung unterstützt nicht mehrere Dokumenttypen.) Das Hinzufügen zusätzlicher Dokumenttypen ist ein wenig diffizil. Nachdem eine neue Dokumentklasse und die entsprechende Ansichtsklasse deklariert und implementiert wurden, müssen diese dem Anwendungsobjekt als neue Dokumentvorlagen hinzugefügt werden. Rufen Sie dazu `AddDocTemplate` in der Elementfunktion `InitInstance` Ihres Anwendungsobjekts auf. Wählt der Anwender später aus dem Menü *Datei* den Eintrag *Neu* aus, zeigt der Anwendungsrahmen automatisch einen Dialog an, in dem der Anwender das gewünschte Dokument auswählen kann.

20.1.3 Die Nachrichtentabelle

In den Dateien `YAH.h` und `YAH.cpp` sind die Makros `DECLARE_MESSAGE_MAP`, `BEGIN_MESSAGE_MAP` und `END_MESSAGE_MAP` enthalten. Was aber repräsentieren diese Makros, und wie sind sie mit der Elementfunktion `Run` in der Hauptnachrichtenschleife des Anwendungsobjekts verbunden?

Die Elementfunktion `Run`

Die `Run`-Elementfunktion leitet Nachrichten an die Zielfenster weiter. Dazu ruft es die Funktion `::DispatchMessage` auf, die auch von Programmen verwendet wird, die keine MFC-Anwendungen sind. Der erste Empfänger einer Nachricht ist immer ein Fenster.

Die Nachrichtенbearbeitungsfunktion eines Objekts, das Befehlsnachrichten entgegennimmt (ein Anweisungszielobjekt, zu dem auch Fensterobjekte zählen), leitet Nachrichten in der folgenden Reihenfolge weiter:

1. an jedes gegenwärtig aktive untergeordnete Anweisungszielobjekt,
2. an sich selbst,
3. an andere Anweisungszielobjekte

Eine Nachricht, die beispielsweise von der Dokumentklasse der Anwendung bearbeitet werden soll, könnte über deren Rahmenfenster und Ansichtsfenster weitergeleitet werden, bevor sie den Nachrichtenbearbeiter der Dokumentklasse erreicht.

Tabelle 20.1 faßt zusammen, wie Nachrichten von den wesentlichen MFC-Anweisungszielklassen bearbeitet werden.

Klasse	Reihenfolge der Weiterleitung
MDI-Rahmenfenster (CMDIFrameWnd)	1. Aktives MDI-Child-Fenster 2. Dieses Fenster 3. Anwendungsobjekt
Dokumentrahmenfenster (CMDIChildWnd, CFrameWnd)	1. Aktive Ansicht 2. Dieses Fenster 3. Anwendungsobjekt
Ansicht	1. Dieses Fenster 2. Zugewiesenes Dokumentobjekt
Dokument	1. Dieses Dokument 2. Dokumentvorlage
Dialogfeld	1. Dieses Fenster 2. Besitzendes Fenster 3. Anwendungsobjekt

*Tabelle 20.1:
Weiterleiten von
Nachrichten*

In welcher Beziehung stehen die zuvor genannten Nachrichtentabellenmakros zur Bearbeitung von Nachrichten? Hier die Antwort:

Das Makro `DECLARE_MESSAGE_MAP` deklariert ein Array in der Klassendeklaration, das aus Nachrichtentabelleneinträgen besteht. Die Makros `BEGIN_MESSAGE_MAP` und `END_MESSAGE_MAP` enthalten einige Initialisierungen für dieses Array, die individuelle Nachrichten repräsentieren, auf die Ihre Klasse reagieren kann. Mit diesen Makros kann also eine Nachrichtentabelle aufgebaut werden, in der einzelne Nachrichten mit Bearbeitungsfunktionen verbunden werden (vergleiche mit der Fensterfunktion von API-Anwendungen).

*Aufbau von
Nachrichtenta-
bellen*

Sehen Sie sich dazu auch die Nachrichtentabelleneinträge in *YAH.cpp* an. Diese Standardeinträge verbinden einige Standardbefehle in dem Menü DATEI mit den Standardimplementierungen, die von der *CWinApp*-Klasse zur Verfügung gestellt werden. `ON_COMMAND` ist eines der Makros, das das Erstellen von Nachrichtentabelleneinträgen erleichtert. Nachrichtentabelleneinträge werden gewöhnlich automatisch von dem Anwendungsassistenten oder dem Klassen-Assistenten generiert. Bisweilen müssen diese Einträge jedoch manuell definiert werden (wenn beispielsweise eine spezifische Anwendungsnachricht bearbeitet werden soll).

20.1.4 Der Rahmen, das Dokument und die Ansicht

In einer gewöhnlichen Windows-Anwendung würden Sie ein Fenster erstellen und dessen Client-Bereich verwenden, um die Ausgabe darzustellen. MFC-Anwendungen nutzen zwei Fenster:

■ ➔ RAHMENFENSTER und

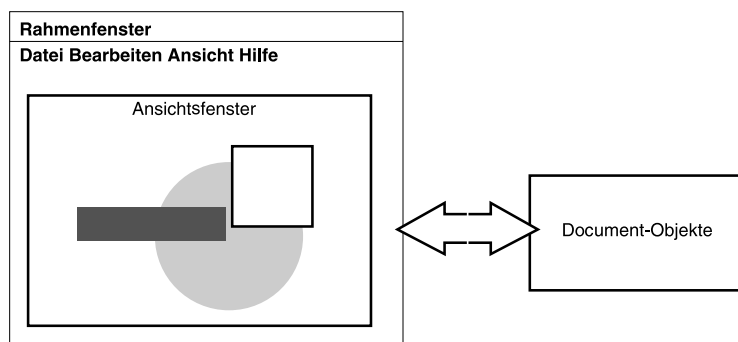
■ ➔ ANSICHTSFENSTER.

Das Rahmenfenster nimmt die Menüs, Symbolleisten und andere Elemente der Benutzeroberfläche auf.

Das Ansichtsfenster hingegen präsentiert die Daten des Anwendungsdokuments.

Abbildung 20.4:
Rahmen,
Ansichten und
Dokumente

Ansichtsfenster



Das Dokumentobjekt ist kein sichtbares Objekt. Es repräsentiert die Daten der Anwendung und entspricht gewöhnlich dem Inhalt einer Datei. Das Dokumentobjekt interagiert mit dem Ansichtsfenster, um Daten darzustellen und die Interaktion mit dem Anwender zu ermöglichen.

Die Beziehung zwischen Rahmen- und Ansichtsfenster sowie dem Dokumentobjekt ist in Abbildung 20.4 dargestellt.

Der nächste Abschnitt beschreibt die Deklaration und Implementierung dieser drei Klassen.

Die Rahmenfensterklasse

Das Rahmenfenster der Anwendung wird von der `CMainFrame`-Klasse unterstützt, die in *MainFrm.h* deklariert ist (Listing 20.3).

```
class CMainFrame : public CFrameWnd
{
protected: // Nur aus Serialisierung erzeugen
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attribute
public:

// Operationen
public:

// Überladungen
    // Vom Klassen-Assistenten generierte Überladungen virtueller
    // Funktionen
    //{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}AFX_VIRTUAL

// Implementierung
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // Eingebundene Elemente der Steuerleiste
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generierte Message-Map-Funktionen
protected:
    //{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    // HINWEIS - An dieser Stelle werden Member-Funktionen vom
    // Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

*Listing 20.3:
Die CMainFrame-
Klassendeklara-
tion*

Beachten Sie bitte die beiden Elementvariablen `m_wndStatusBar` und `m_wndToolBar`, die der einzigen Symbolleiste und der Statusleiste der Anwendung entsprechen. Möchten Sie Ihrem Programm weitere Steuerleisten hinzufügen, sollten Sie diese als Elementvariablen der Rahmenfensterklasse deklarieren und der Implementierungsdatei der Rahmenfensterklasse den unterstützenden Programmcode hinzufügen.

[illegible]

```

        || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Symbolleiste konnte nicht erstellt "
              "werden\n");
        return -1;        // Fehler bei Erstellung
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
                                      sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Statusleiste konnte nicht erstellt "
              "werden\n");
        return -1;        // Fehler bei Erstellung
    }

    // ZU ERLEDIGEN: Löschen Sie diese drei Zeilen, wenn Sie
    // nicht wollen, dass die Symbolleiste andockbar ist.
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // ZU ERLEDIGEN: Ändern Sie hier die Fensterklasse oder
    // das Erscheinungsbild, indem Sie CREATESTRUCT cs
    // modifizieren.

    return TRUE;
}

```

Die Anwender, die mit früheren Varianten von Visual C++ vertraut sind, erkennen in diesem Listing einen deutlichen Versionsunterschied. Obwohl wie bisher ein globales Array mit der Bezeichnung `indicators` verwendet wird, das die in der Statusleiste angezeigten Meldungen definiert, ist kein globales Array mehr vorhanden, das die Schaltflächen der Symbolleiste aufnimmt. Visual C++ unterstützt seit Version 4 einen Symbolleisten-Ressourcentyp in den Ressourcendateien. Diese Ressource kann mit dem Ressource-Editor des Visual Studio bearbeitet werden. Das manuelle Einrichten und der Einsatz eines Arrays, das Schaltflächen-Anweisungsbezeichner enthält, die sich auf die Schaltflächen in der Symbolleiste beziehen, ist daher nicht mehr notwendig.

Die Dokumentklasse

Die Deklaration der Dokumentklasse in *YAHDoc.h* (Listing 20.5) enthält überschriebene Versionen der beiden Funktionen `OnNewDocument` und `Serialize`.

```

class CYAHDoc : public CDocument
{
protected: // Nur aus Serialisierung erzeugen
    CYAHDoc();

```

Listing 20.5:
CYAHDoc-Klassendeklaration

```

DECLARE_DYNCREATE(CYAHDoc)

// Attribute
public:

// Operationen
public:

// Überladungen
    // Vom Klassen-Assistenten generierte Überladungen virtueller
    // Funktionen
    //{AFX_VIRTUAL(CYAHDoc)
    public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
    //}AFX_VIRTUAL

// Implementierung
public:
    virtual ~CYAHDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generierte Message-Map-Funktionen
protected:
    //{AFX_MSG(CYAHDoc)
    // HINWEIS - An dieser Stelle werden Member-Funktionen vom
    // Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

`OnNewDocument` wird aufgerufen, wenn der Anwender den Eintrag **NEU** aus dem Menü **DATEI** auswählt. Im Gegensatz zu SDI-Anwendungen wird `OnNewDocument` nur dann in MDI-Anwendungen aufgerufen, wenn ein neues Dokument erstellt wird. SDI-Programme rufen diese Funktion auf, um das einzige Dokumentobjekt der Anwendung erneut zu initialisieren. Die Initialisierungen, die gewöhnlich innerhalb des Konstruktors ausgeführt werden, sind somit statt dessen in dieser Funktion angeordnet.

Die `Serialize`-Elementfunktion wird während des Ladens und Speicherns des Dokuments aufgerufen. Diese Elementfunktion muß überschrieben sein. Sie müssen Ihren eigenen Programmcode in die überschriebene Version schreiben, um die Daten Ihres Dokuments laden und speichern zu können.

Eine kleine Unregelmäßigkeit führt zu der Frage, wieso das Makro `DECLARE_DYNCREATE` in der Klassendeklaration verwendet wird, wenn diese Klasse die Serialisierung unterstützt? Sollte dort nicht statt dessen `DECLARE_SERIAL` stehen?

Obwohl die Klasse über eine `Serialize-Elementfunktion` verfügt, verwenden wir `DECLARE_SERIAL` nicht, da der Operator `>>` niemals zum Einlesen eines Dokuments aus `CArchive` genutzt wird. Die `Serialize-Elementfunktion` wird explizit von `CDocument::OnOpenDocument` aufgerufen. `DECLARE_SERIAL` (und `IMPLEMENT_SERIAL`) ist lediglich für Klassen erforderlich, die mit dem genannten Operator aus einem `CArchive`-Objekt geladen werden sollen.

Die beiden überladenen `CYAHDoc`-Funktionen sind in der Datei `YAHDoc.cpp` implementiert (Listing 20.6). Die Standardimplementierung erfüllt keine besondere Aufgabe. Sie müssen den Programmcode zur Initialisierung Ihres Dokumenttyps und zum Speichern und Laden der Dokumentdaten zur Verfügung stellen.

```

//////////////////////////////////// CYAHDoc
IMPLEMENT_DYNCREATE(CYAHDoc, CDocument)

BEGIN_MESSAGE_MAP(CYAHDoc, CDocument)
//{{AFX_MSG_MAP(CYAHDoc)
// HINWEIS - Hier werden Mapping-Makros vom
// Klassen-Assistenten eingefügt und entfernt.
// Innerhalb dieser generierten Quelltextabschnitte
// NICHTS VERÄNDERN!
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////// CYAHDoc
Konstruktion/Destruktion

CYAHDoc::CYAHDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion einfügen
}

CYAHDoc::~CYAHDoc()
{
}

BOOL CYAHDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
    // (SDI-Dokumente verwenden dieses Dokument)

    return TRUE;
}

//////////////////////////////////// CYAHDoc
Serialisierung

void CYAHDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())

```

*Listing 20.6:
CYAHDoc-Klassen-
implementierung*

```

{
    // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
}
else
{
    // ZU ERLEDIGEN: Hier Code zum Laden einfügen
}
}

```

Die Ansichtsklasse

Die Standarddeklaration der Ansichtsklasse in *YAHView.h* (Listing 20.7) enthält verschiedene überladene Funktionen. Die wohl wichtigste dieser Funktionen ist `OnDraw`. Sie stellt die Daten des Dokuments dar, das dieser Ansicht entspricht. Die erste Aktion der `OnDraw`-Funktion ist daher, sich mit Hilfe der Funktion `GetDocument` einen Zeiger auf das Dokumentobjekt zu besorgen.

Listing 20.7:
CYAHView-Klassendeklaration

```

class CYAHView : public CView
{
protected: // Nur aus Serialisierung erzeugen
    CYAHView();
    DECLARE_DYNCREATE(CYAHView)

// Attribute
public:
    CYAHDoc* GetDocument();

// Operationen
public:

// Überladungen
    // Vom Klassen-Assistenten generierte Überladungen
    // virtueller Funktionen
    //{AFX_VIRTUAL(CYAHView)
    public:
        virtual void OnDraw(CDC* pDC); // überladen zum Zeichnen
                                         // dieser Ansicht
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}AFX_VIRTUAL

// Implementierung
public:
    virtual ~CYAHView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generierte Message-Map-Funktionen
protected:
    //{AFX_MSG(CYAHView)
    // HINWEIS - An dieser Stelle werden Member-Funktionen vom
    // Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!

```

```

    ///AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

Beachten Sie bitte, daß diese Klasse wie die Dokumentklasse mit dem `DECLARE_DYNCREATE`-Makro deklariert wird. Dieses Makro muß verwendet werden, da das Ansichtsbjekt dynamisch während der Erstellung eines neuen Dokuments erzeugt wird.

Die Implementierung der Ansichtsklasse in `YAHView.cpp` (Listing 20.8) hält einige Überraschungen bereit. Die überschriebenen Funktionen sind lediglich Gerüste. Sie müssen eine eigene Implementierung zur Verfügung stellen. Eine fehlerfrei ausgeführte Anwendung erfordert jedoch lediglich die Bearbeitung der `OnDraw`-Elementfunktion. Möchten Sie die Druckmöglichkeiten nutzen, müssen Sie hier nicht die Elementfunktionen zum Drucken einrichten. Diese benötigen Sie nur dann, wenn Ihnen die Standard-Features nicht ausreichen.

```

////////////////////////////////////// CYAHView
IMPLEMENT_DYNCREATE(CYAHView, CView)

BEGIN_MESSAGE_MAP(CYAHView, CView)
    ///AFX_MSG_MAP(CYAHView)
    // HINWEIS - Hier werden Mapping-Makros vom
    // Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    ///AFX_MSG_MAP
    // Standard-Druckbefehle
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////// CYAHView
Konstruktion/Destruktion

CYAHView::CYAHView()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen,
}

CYAHView::~CYAHView()
{
}

BOOL CYAHView::PreCreateWindow(CREATESTRUCT& cs)
{
    // ZU ERLEDIGEN: Ändern Sie hier die Fensterklasse oder das
    // Erscheinungsbild, indem Sie CREATESTRUCT cs modifizieren.

    return CView::PreCreateWindow(cs);
}

////////////////////////////////////// CYAHView
Zeichnen

void CYAHView::OnDraw(CDC* pDC)

```

*Listing 20.8:
CYAHView-Klas-
senimplementie-
rung*


```

{
    CYAHDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen
    // Daten hinzufügen
}

////////////////////////////////////// CYAHView Drucken

BOOL CYAHView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // Standardvorbereitung
    return DoPreparePrinting(pInfo);
}

void CYAHView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // ZU ERLEDIGEN: Zusätzliche Initialisierung vor dem Drucken
    // hier einfügen
}

void CYAHView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // ZU ERLEDIGEN: Hier Reinigungsarbeiten nach dem Drucken
    // einfügen
}

```

Beachten Sie bitte, daß sich verschiedene Nachrichtentabelleneinträge auf den Druckvorgang beziehen. Sie rufen die Basisklassenfunktionen auf, die den Standarddruck und die Druckvorschau implementieren.

Anwendungsgerüstressourcen

Mit einer kurzen Beschreibung der von dem Anwendungsassistenten generierten Ressourcen beenden wir die Übersicht zum MFC-Anwendungsgerüst.

Um sich die Liste der Ressourcen anzeigen zu lassen, wechseln Sie zur Ressourcen-Ansicht und öffnen dort den Knoten des übergeordneten Ordners.

Tastaturkürzel	Die ACCELERATOR-Ressource bedarf keiner umfangreichen Erläuterung. Sie enthält die Definition der Tastaturkürzel für die Standardmenüfunktionen. Die Menüleiste selbst ist in der MENU-Ressource definiert.
Dialoge	Der Anwendungsassistent erzeugte eine DIALOG-Ressource, nämlich einen INFO-Dialog. Dieser Dialog wird angezeigt, wenn der Anwender den Eintrag INFO aus dem Menü HILFE auswählt.
Symbole	Zwei Symbole wurden generiert. IDR_MAINFRAME ist das Anwendungssymbol und IDR_YAHTYPE ist das Symbol, das den Dokumenttyp der Anwendung repräsentiert.

Stringtabelle Die Zeichenfolgentabelle enthält mehrere Zeichenfolgen, die überwiegend den MFC-Anwendungsrahmennachrichten entsprechen. Andere Texte sind Statusleistenmeldungen, QuickInfos usw. Besonders interessant ist die Zeichenfolgen-Ressource IDR_MAINFRAME, die auch als *Dokumentvorlagenzeichenfolge* bezeichnet wird. Sie enthält bis zu neun untergeordnete Zeichenfolgen, die durch das Newline-Zeichen (\n) voneinander getrennt sind.

Der Anwendungsassistent hat dieser Zeichenfolge den folgenden Text zugewiesen:

```
Hello, World!\n\nYAH\nDateitypYAH(*.yah)\n.yah\nYAH.Document\nYAH Document
```

Die der Dokumentvorlagenzeichenfolge untergeordneten Zeichenfolgen sind in Tabelle 20.2 beschrieben. Die Syntax lautet wie folgt:

```
<windowTitle>\n<docName>\n<fileNewName>\n<filterName>\n<filterExt>\n<regFileTypeID>\n<regFileTypeName>
```

Untergeordnete Zeichenfolge	Beschreibung
<windowTitle>	Der Titel des Rahmenfensters der Anwendung
<docName>	Basisdokumentname für Dokumentfenster (Dieser Name wird mit einer Nummer als Fenstertitel verwendet)
<fileNewName>	Der Dokumenttyp, der in dem Dialog DATEI NEU angezeigt wird, sofern die Anwendung unterschiedliche Typen unterstützt
<filterName>	Die in den DATEI-Dialogen verwendeten Filter
<filterExt>	Die in den DATEI-Dialogen verwendeten Dateiendungen
<regFileTypeID>	Der in der Registrierung registrierte Dateityp
<regFileTypeName>	Der angezeigte Name des in der Registrierungsdatei registrierten Dateityps

*Tabelle 20.2:
Die der Dokumentvorlagen-
zeichenfolge untergeordneten
Zeichenfolgen*

Die Anwendung verfügt überdies über eine TOOLBAR-Ressource und eine VERSION-Ressource.



Beachten Sie bitte, daß sich einige Ressourcen den Bezeichner `IDR_MAINFRAME` teilen. Gemeinsame Bezeichner werden verwendet, wenn die Anwendung den Konstruktor `CSingleDocTemplate` (oder `CMultiDocTemplate`) aufruft. Dieser bezeichnet das Menü, das Symbol, die Tastenkürzeltabelle sowie die Dokumentvorlagenzeichenfolge, die sich auf einen bestimmten Dokumenttyp beziehen.

20.2 Hinzufügen von Programmcode zur Anwendung

Nun, da Sie die grundlegenden Elemente eines MFC-Gerüsts kennen gelernt haben, sollten Sie erfahren, wie Sie diesem Gerüst Ihren eigenen Programmcode hinzufügen.

20.2.1 Manuelle Bearbeitung

Wir werden nun der Dokumentklasse eine Zeichenfolgen-Elementvariable hinzufügen, die Sie aus einer Ressource beziehen. Die Ansichtsklasse erweitern Sie mit Programmcode, der diese Zeichenfolge in der Mitte des Ansichtsfensters der Anwendung ausgibt.

Hinzufügen einer Zeichenfolgen-Ressource

Abbildung 20.5:
Zeichenfolgen-
Ressource
anlegen



1. Öffnen Sie die Ressourcen-Ansicht des Arbeitsbereichsfensters und dort den Knoten des Ordners `STRING TABLE`.
2. Führen Sie einen Doppelklick auf dem untergeordneten Element mit der Bezeichnung `ZEICHENFOLGENTABELLE` aus.

3. Rufen Sie den Befehl EINFÜGEN/NEUE ZEICHENFOLGE auf (Visual Studio- oder Kontextmenü), und fügen Sie der Zeichenfolgentabelle eine Zeichenfolge mit dem Bezeichner `IDS_HELLO` hinzu, und setzen Sie den Wert dieser Zeichenfolge auf »HELLO, WORLD!« (oder einen Text Ihrer Wahl).

Modifizieren des Dokuments

Fügen Sie der Dokumentklasse zunächst eine Elementvariable hinzu.

1. Klicken Sie in der Dateienansicht auf den Knoten der Datei *YAHDoc.h*.
2. Fügen Sie dem Attribute-Abschnitt die Deklaration einer Elementvariablen vom Typ `CString` wie folgt hinzu:

```
// Attribute
public:
    CString m_sData;
```

`m_sData` muß natürlich initialisiert werden. Die Elementvariable muß außerdem der `Serialize-Elementfunktion` hinzugefügt werden, damit diese die Variable in eine Datei speichern und Daten aus einer Datei auslesen kann. Die entsprechenden Änderungen werden in der Datei *YAHDoc.cpp* vorgenommen.

3. Wir initialisieren die Zeichenfolge in der Elementfunktion `OnNewDocument`, so daß sie immer dann erneut initialisiert wird, wenn der Anwender aus dem Menü DATEI den Eintrag NEU auswählt.

```
BOOL CYAHDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
    // (SDI-Dokumente verwenden dieses Dokument)
    m_sData.LoadString(IDS_HELLO);
    return TRUE;
}
```

4. Die `Serialize-Elementfunktion` muß wie folgt modifiziert werden:

```
void CYAHDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
        ar << m_sData;
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
        ar >> m_sData;
    }
}
```

Die Zeichenfolge muß jetzt lediglich noch angezeigt werden. Dazu ergänzen Sie die Ansichtsklasse.

Modifizieren der Ansicht

Damit unsere Zeichenfolge angezeigt wird, müssen Sie die `OnDraw`-Elementfunktion der Ansichtsklasse modifizieren. Da der Anwendungsassistent bereits eine Implementierung des entsprechenden Gerüsts zur Verfügung stellt, ist eine Veränderung der Klassendeklaration nicht erforderlich.

Sie fügen den Programmcode einfach dem bestehenden Funktionsgerüst in der Datei `YAHView.cpp` hinzu:

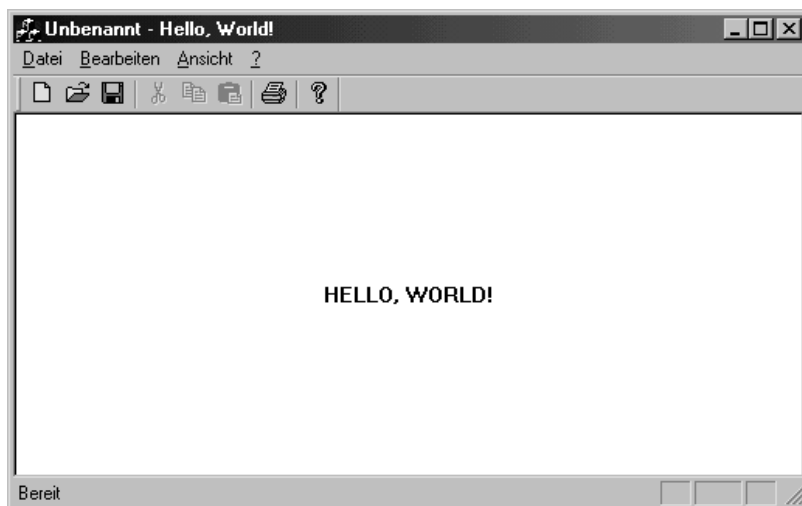
```
void CYAHView::OnDraw(CDC* pDC)
{
    CYAHDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    CRect rect;
    GetClientRect(&rect);
    pDC->DPtoLP(&rect);
    pDC->DrawText(pDoc->m_sData, &rect,
        DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}
```

Programm testen

Kompilieren und starten Sie das Programm.

Haben Sie die Anleitung korrekt befolgt, sollte sich Ihnen das Anwendungsfenster, wie in Abbildung 20.6 dargestellt, präsentieren.

Abbildung 20.6:
Die Hello-World-
Anwendung



20.2.2 Bearbeitung mit dem Klassen-Assistenten

Als nächstes soll der angezeigte Text geändert werden, wenn der Anwender mit der Maus in den Client-Bereich des Hauptfensters klickt. Dazu ist es erforderlich, eine entsprechende Bearbeitungsfunktion für die WM_LBUTTONDOWN-Nachricht zu implementieren.

Bearbeitungsfunktion mit Klassen-Assistent einrichten

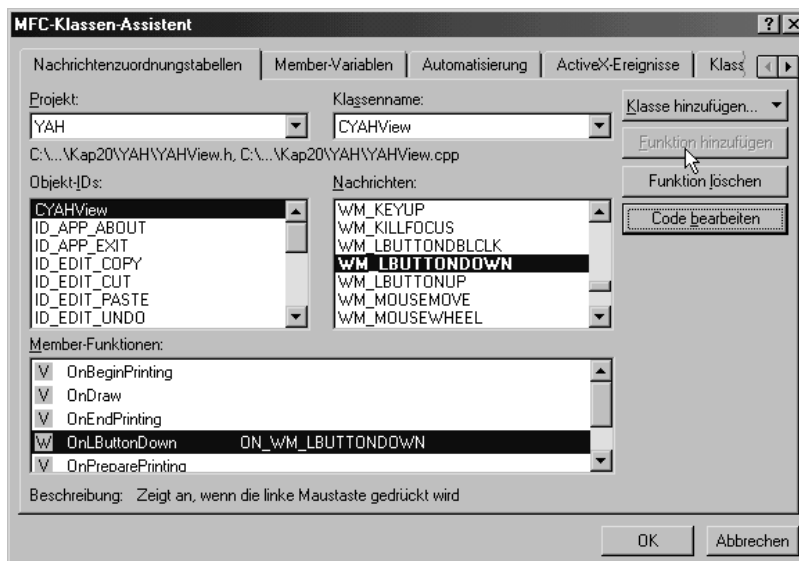


Abbildung 20.7:
Einrichtung
einer Nachrichten-
bearbeitungs-
funktion

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT / KLASSEN-ASSISTENT).
2. Wählen Sie im Feld KLASSENNAME der Registerseite NACHRICHTENZUORDNUNGSTABELLEN die Ansichtsclass CYAHView aus.
3. Achten Sie darauf, daß im Feld OBJEKT-IDS ebenfalls CYAHView ausgewählt ist, und scrollen Sie im Feld NACHRICHTEN bis zur Nachricht WM_LBUTTONDOWN.
4. Klicken Sie auf den Schalter FUNKTION HINZUFÜGEN.
5. Klicken Sie danach auf den Schalter CODE BEARBEITEN, um in den Editor zu wechseln.
6. Ändern Sie den Inhalt der Variablen m_sData und sorgen Sie dafür, daß der Client-Bereich neu gezeichnet wird.

```
void CYAHView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten hier
    // einfügen und/oder Standard aufrufen
}
```

```
CYAHDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);

pDoc->m_sData = "HALLO DIRK!";
Invalidate();
UpdateWindow();

CView::OnLButtonDown(nFlags, point);
}
```

20.3 Zusammenfassung

MFC-Anwendungen werden mit dem Anwendungsassistenten erstellt. Den Kern jeder MFC-Anwendung bildet ein von `CWinApp` abgeleitetes Objekt, das die Initialisierung sowie die Hauptnachrichtenschleife des Programms implementiert.

Nachrichten werden über Nachrichtentabellen verteilt und weitergeleitet, die ein Feature der Anweisungsbearbeitungsobjekte sind (wie z.B. Fenster). Die `CWinApp::Run`-Elementfunktion verteilt Nachrichten über `::DispatchMessage`. Weiterleitungen erfolgen nach den Regeln der MFC-Nachrichtenweiterleitung. Ein Anweisungsbearbeitungsobjekt leitet eine Nachricht gewöhnlich zunächst an alle untergeordneten Anweisungsbearbeitungsobjekte, anschließend an sich selbst und daraufhin an zusätzliche Nachrichtenbearbeitungsobjekte weiter.

Die visuelle Präsentation der Anwendung sowie die Verwaltung der Anwendungsdaten sind ein Ergebnis der Kooperation zwischen Rahmenfenstern, Ansichtsfenstern und Dokumentobjekten. Das Dokumentobjekt verfügt über die Anwendungsdaten, die gewöhnlich in einer Datei gespeichert sind. Das Ansichtsfenster stellt die Inhalte eines Dokuments dar und ermöglicht die Interaktion mit dem Anwender. Das Ansichtsfenster arbeitet mit dem Rahmenfenster zusammen, das andere Elemente der Benutzeroberfläche verwaltet, wie z.B. die Menüleiste, Symbolleisten oder die Statusleiste.

Während der Implementierung einer MFC-Anwendung werden die Dokument- und Ansichtsklassen gewöhnlich gleichzeitig bearbeitet. Neue Dokumente werden als Elemente der Dokumentklasse deklariert. Die visuelle Schnittstelle, die die neuen Elemente berücksichtigt, wird in der Ansichtsklasse implementiert.

Anwendungen, die mit dem Anwendungsassistenten erstellt wurden, eignen sich auch für die Weiterbearbeitung mit dem Klassen-Assistenten.

Die Arbeit mit Dokumenten und Ansichten

Kapitel 21

Den Kern einer Doc/View-Anwendung bildet das Konzept eines DOKUMENTOBJEKTS und des entsprechenden ANSICHTSFENSTERS. Das Dokumentobjekt repräsentiert (gewöhnlich) eine Datei, die von einer Anwendung geöffnet wurde. Das Ansichtsfenster stellt die Daten des Dokuments dar und ermöglicht die Interaktion mit dem Anwender. Eine Ansicht kann sich auf lediglich ein Dokument beziehen, während einem Dokument mehrere Ansichten zugewiesen werden können – die Daten eines Dokuments können also über verschiedene Views in unterschiedlicher Weise angezeigt werden. (Beispielsweise könnte eine Webseite als formatiertes Dokument oder reiner HTML-Code angezeigt werden.)

Dokumentobjekte sind Instanzen von Klassen, die von `CDocument` abgeleitet sind. Ansichtsfensterklassen werden von `CView` abgeleitet. Dieses Kapitel erläutert diese Klassen und beschreibt, wie sie zur Erstellung verschiedener Darstellungen Ihrer Daten verwendet werden. Wir erörtern außerdem eine effiziente Benutzeroberfläche.

21.1 Die CDocument-Klasse

Die `CDocument`-Klasse stellt die grundlegende Funktionalität für die Dokumentobjekte Ihrer Anwendung zur Verfügung. Dazu zählt die Fähigkeit, ein neues Dokument zu erstellen und Dokumentdaten zu serialisieren sowie die Kooperation zwischen einem Dokument und einem Ansichtsfenster zu ermöglichen. MFC bietet außerdem einige `CDocument`-Klassen an, die eine OLE-Funktionalität implementieren.

21.1.1 Deklarieren einer Dokumentklasse in einer Anwendung

Unterstützung durch den Anwendungsassistenten In einer mit dem Anwendungsassistenten generierten Anwendung müssen Sie Ihre Dokumentklasse gewöhnlich nicht mehr deklarieren. Der Anwendungsassistent übernimmt diese Aufgabe für Sie. Sie sollten sich dennoch mit dieser Klasse befassen. Das dadurch gewonnene Wissen ermöglicht Ihnen nicht nur, das von dem Anwendungsassistenten erstellte Anwendungsgerüst zu ergänzen, sondern hilft Ihnen ebenfalls, zusätzliche Dokumenttypen zu definieren, die Ihre Anwendung unterstützen sollen. (Der Anwendungsassistent erzeugt Anwendungen, die lediglich einen Dokumenttyp verwalten.)

Wenn Sie eine einfache MFC-Anwendung erstellen, müssen Sie gewöhnlich nur wenige Modifizierungen an der von dem Anwendungsassistenten bereitgestellten Dokumentklasse vornehmen. Sie benötigen lediglich einige Elementvariablen und möglicherweise Elementfunktionen, die den Zugriff auf diese Variablen ermöglichen.

Beispiel Stellen Sie sich beispielsweise ein einfaches Kommunikationsprogramm vor (Terminal-Emulator). Die Dokumentobjekte dieses Programms bilden die Einstellungen (Telefonnummer, Geschwindigkeit, Parität usw.) zu einer Verbindung. Diese Einstellungen werden durch einige Dateneinträge in der Dokumentklasse repräsentiert, wie im folgenden dargestellt.

```
class CTerminalDoc : public CDocument
{
protected: // Nur aus Serialisierung erstellen
    CTerminalDoc();
    DECLARE_DYNCREATE(CTerminalDoc)
// Attribute
public:
    CString m_sPhone;
    DWORD m_dwSpeed;
    WORD m_nParity;
    WORD m_nBits;
    ...
}
```

Zusätzlich zur Deklaration der Elementvariablen müssen diese mit Standardwerten in der `OnNewDocument`-Elementfunktion Ihrer Dokumentklasse initialisiert werden. Eine korrekte Serialisierung ist ebenfalls erforderlich:

```
...
BOOL CTerminalDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    m_sPhone = _T("555-1212");
    m_dwSpeed = 2400;
    m_nParity = 0;
}
```

```
m_nBits = 8;
return TRUE;
}
...
void CTerminalDoc::Serialize(CArchive &ar)
{
    if (ar.IsStoring())
    {
        ar << m_sPhone;
        ar << m_dwSpeed;
        ar << m_nParity;
        ar << m_nBits;
    }
    else
    {
        ar >> m_sPhone;
        ar >> m_dwSpeed;
        ar >> m_nParity;
        ar >> m_nBits;
    }
}
```

Weitere Schritte sind nicht notwendig, um eine vollständige Dokumentklasse zu erstellen.

21.1.2 Die CDocument-Elementfunktionen

Die CDocument-Klasse enthält Elementfunktionen, die häufig von Anwendungen verwendet werden.

Auf Ansichten zugreifen

Einige dieser Elementfunktionen ermöglichen den Zugriff auf die entsprechenden Ansichtsobjekte. Jedem Dokument sind mehrere Ansichtsobjekte zugewiesen. Zu dieser Liste kann eine Zählervariable vom Typ POSITION mit GetFirstViewPosition ermittelt werden.

Zähler

Werte vom Typ POSITION werden in der MFC vorwiegend in Verbindung mit Auflistungsklassen verwendet. Anwendungen, die eine Auflistung bearbeiten müssen, ermitteln gewöhnlich einen Zähler, dem das erste Objekt der Auflistung zugewiesen ist. Mit Hilfe einer Iterationsfunktion wird anschließend sukzessiv auf die Elemente der Auflistung zugegriffen. Für CDocument und die entsprechenden Ansichten gilt die gleiche Aussage. Nachdem ein Zähler für die Auflistung mit GetFirstViewPosition ermittelt wurde, kann die Funktion GetNextView wiederholt auf die Elemente der Auflistung zugreifen.



Zur Bearbeitung aller Ansichten, die einem Dokument zugewiesen sind, könnten Sie somit den folgenden Programmcode verwenden:

```
POSITION pos = GetFirstViewPosition();
while (pos != NULL)
{
    CView *pView = GetNextView(pos);
    // pView kann nun verwendet werden
}
```

Ansichten aktualisieren

Möchten Sie lediglich die Ansichten ermitteln, die von dem Dokument verändert wurden, müssen Sie keine Schleife verwenden. Sie können statt dessen die Elementfunktion `UpdateAllViews` aufrufen. Während des Aufrufs dieser Funktion können Sie ebenfalls spezifische Anwendungsdaten angeben, die dem Ansichtsobjekt die Aktualisierung von Bereichen des Ansichtsfensters ermöglichen. Zu diesem Thema erhalten Sie später weitere Informationen, wenn wir die `CView::OnUpdate`-Elementfunktion erörtern.

Ansichten hinzufügen und entfernen

Weniger häufig verwendete Ansichtsfunktionen sind `AddView` und `RemoveView`. Diese Funktionen erlauben Ihnen das manuelle Hinzufügen und Entfernen von Ansichten zu respektive aus der entsprechenden Dokumentliste. Der Grund für den seltenen Gebrauch dieser Funktionen besteht darin, daß die meisten Anwendungen die MFC-Standardimplementierung nutzen und diese zur Verwaltung ihrer Fenster modifizieren.

SetModifiedFlag Sobald sich die Dokumentdaten ändern, sollten Sie die Elementfunktion `SetModifiedFlag` setzen. Der kontinuierliche Einsatz dieser Funktion gewährleistet, daß der Anwendungsrahmen den Anwender benachrichtigt, bevor ein verändertes, nicht gespeichertes Dokument zerstört wird. Der Status dieses Flags wird mit der Elementfunktion `IsModified` ermittelt.

SetTitle Die `SetTitle`-Elementfunktion setzt den Titel des Dokuments. Dieser Titel wird als Dokumenttitel in dem Rahmenfenster angezeigt (bei einer SDI-Anwendung im Hauptrahmenfenster und bei einer MDI-Anwendung im untergeordneten Rahmenfenster).

GetPathName Der vollständige Pfadname des Dokuments wird mit `SetPathName` gesetzt und mit `GetPathName` ermittelt.

GetDocTemplate Das mit dem aktuellen Dokument verknüpfte Dokumentvorlagenobjekt wird mit `GetDocTemplate` ermittelt.

21.1.3 Dokumente, Nachrichten und virtuelle Funktionen

Obwohl ein `CDocument`-Objekt nicht direkt einem Fenster zugewiesen wird, ist es dennoch ein Anweisungszielobjekt, das Nachrichten erhält. Nachrichten werden von den entsprechenden Ansichtsobjekten an `CDocument`-Objekte weitergeleitet.

Sie müssen einige Richtlinien beachten, wenn Sie entscheiden, welche Nachrichten von Ihrem Dokumentobjekt bearbeitet und welche Nachrichten zur Bearbeitung an das Ansichtsfenster (oder Rahmenfenster) weitergeleitet werden sollen.

*Wem welche
Nachricht?*

- ➡ Denken Sie immer daran, daß das Dokument eine abstrakte Darstellung Ihrer Daten ist, die unabhängig von der visuellen Repräsentation innerhalb des Ansichtsfensters ist. Einem Dokument können mehrere Ansichten zugeteilt sein. Alle Nachrichten, auf die das Dokument reagiert, sollten global sein. Diese Nachrichten wirken sich umgehend auf die Dokumentdaten aus, die in allen Ansichten verfügbar sein sollten.
- ➡ Im Gegensatz dazu sollten Ansichten auf alle Nachrichten reagieren, die für das entsprechende Fenster bestimmt sind.

Wie läßt sich diese Aussage in die Praxis umsetzen? Betrachten Sie dazu die Befehlsmeldung, die generiert wird, wenn der Anwender den Eintrag `SPEICHERN` aus dem Menü `DATEI` wählt. Nach dieser Auswahl wird das Dokument gespeichert, nicht dessen visuelle Darstellung. Diese Nachricht wird daher von der Dokumentklasse bearbeitet.

Wenn Sie sich hingegen fragen, was in die Zwischenablage kopiert wird, wenn Sie aus dem Menü `BEARBEITEN` den Eintrag `KOPIEREN` auswählen, werden Sie schnell feststellen, daß die zu kopierenden Daten der Ansicht des Dokuments entnommen werden. Bestehen mehrere Ansichten für dasselbe Dokument, können diese unterschiedliche Selektionen enthalten. Die Anweisung `KOPIEREN` führt somit für jede Ansicht zu einem anderen Ergebnis. Daraus schließen wir, daß diese Anweisung von der Ansichtsklasse bearbeitet werden sollte.

Außerdem gilt es, einige irreguläre Situationen zu berücksichtigen. Sollte beispielsweise die Anweisung `EINFÜGEN` von der Dokumentklasse oder von der Ansichtsklasse bearbeitet werden? Natürlich betrifft diese Anweisung das gesamte Dokument und nicht eine einzelne Ansicht, da sie Daten in Ihr Dokumentobjekt kopiert. Außerdem wirkt sich die Anweisung ebenfalls auf die aktuelle Ansicht aus, da die aktuelle Selektion beispielsweise durch die einzufügenden Daten ersetzt werden muß. Die Entscheidung, welche Klasse diese Anweisung bearbeiten soll, liegt daher bei Ihnen.

Einige Anweisungen sollten weder von der Ansichtsklasse noch von der Dokumentklasse, sondern von dem Rahmenfenster bearbeitet werden. Beispiele hierfür sind Anweisungen, die der Aktivierung und Deaktivierung der Anzeige von Symbolleisten dienen. Die Darstellung einer Symbolleiste ist nicht die Aufgabe eines Dokuments oder einer Ansicht. Diese Einstellung betrifft die gesamte Anwendung.

Doch kehren wir zur `CDocument`-Klasse zurück. Der MFC-Anwendungsrahmen stellt eine Standardimplementierung einiger Befehle zur Verfügung. Diese Implementierungen rufen überschriebene Elementfunktionen aus `CDocument` auf. (Die Funktionen sind überschreibbar, da sie als `virtual` deklariert sind. Sie können Ihre eigenen Versionen in einer von `CDocument` abgeleiteten Klasse bereitstellen, so daß diese anstelle der Funktionen in der Basisklasse aufgerufen werden.)

Tabelle 21.1:
CDocument-
Element-
funktionen

Funktion	Beschreibung
<code>OnNewDocument</code>	Die <code>OnNewDocument</code> -Elementfunktion wird während der Initialisierung eines neuen Dokumentobjekts aufgerufen (oder wenn ein bereits bestehendes Dokument erneut in einer SDI-Anwendung verwendet wird). Der Aufruf dieser Funktion wird gewöhnlich nach einer Auswahl des Eintrags NEU aus dem Menü DATEI vorgenommen.
<code>OnCloseDocument</code>	Die Elementfunktion <code>OnCloseDocument</code> wird aufgerufen, wenn ein Dokument geschlossen wird. Sie sollten diese Funktion überschreiben, sofern eine Freigabe von Ressourcen erforderlich ist, bevor das Dokument zerstört wird.
<code>OnOpenDocument</code> , <code>OnSaveDocument</code>	<code>OnOpenDocument</code> und <code>OnSaveDocument</code> lesen ein Dokument aus einer Datei ein oder speichern dieses. Sie sollten diese Funktionen nur dann ergänzen oder ersetzen, wenn die Standardimplementierung (die die <code>Serialize</code> -Funktion Ihrer Dokumentklasse aufruft) nicht Ihren Anforderungen entspricht.
<code>DeleteContents</code>	Die <code>DeleteContents</code> -Funktion wird aus der Standardimplementierung von <code>OnCloseDocument</code> und <code>OnOpenDocument</code> aufgerufen, um den Inhalt des Dokuments zu löschen, bevor eine neue Datei geöffnet wird. Diese Funktion löscht die Daten des Dokuments, ohne das Dokumentobjekt zu zerstören.

Funktion	Beschreibung
OnFileSendMail	Die OnFileSendMail-Elementfunktion versendet die Dokumentobjekte als mit der Mail verbundene Anlagen. Sie ruft OnSaveDocument auf, um eine Kopie des Dokuments in einer temporären Datei zu speichern. Die Datei wird anschließend einer MAPI-Mail-Nachricht zugeteilt. Die OnUpdateFileSendMail-Elementfunktion gibt den Zugriff auf den durch ID_FILE_SEND_MAIL bezeichneten Eintrag im Anwendungsmenü frei oder entfernt diesen, wenn keine MAPI-Unterstützung besteht. Sowohl OnFileSendMail als auch OnUpdateFileSendMail sind überladbare Funktionen, die Ihnen das Implementieren einer angepaßten Nachrichtenbearbeitung ermöglichen.

21.1.4 Dokumentdaten

Einfache von CDocument abgeleitete Klassen, deren Daten als Elementvariablen implementiert werden können, wurden bereits erwähnt. Echte Anwendungen sind jedoch anspruchsvoller. Die Datenanforderungen dieser Anwendungen begnügen sich nicht mit wenigen Variablen einfacher Datentypen.

Die beste Möglichkeit, eine Anwendung mit komplexen Datenelementen zu implementieren, besteht möglicherweise darin, mehrere von CObject abgeleitete Klassen zur Darstellung der Datenelemente zu verwenden. Eine Standard- oder eine benutzerdefinierte Container-Klasse bettet diese Elemente in Ihre Dokumentklasse ein. Nachfolgend finden Sie den Abschnitt einer Anwendung aufgeführt, die diese Klassen verwendet:

```
class CMyObject : public CObject
{
// ...
};
class CMyFirstSubObject : public CObject
{
// ...
};
class CMySecondSubObject : public CObject
{
// ...
};
```

In die Deklaration der Dokumentklasse wurde ein CObList-Element eingefügt:

```
class CMyDocument : public CDocument
{
// ...
// Attribute
```

```
public:
    CObList m_obList;
    // ...
};
```

In einer derart komplexen Situation genügt das Deklarieren der Elementvariablen häufig nicht. Elementfunktionen werden ebenfalls benötigt, die Methoden für den Zugriff auf die Daten des Dokuments zur Verfügung stellen. So könnten Sie in dem Programm beispielsweise verhindern wollen, daß andere Klassen (wie z.B. eine Ansichtsklasse) die `m_obList`-Elementvariable direkt manipulieren. Vielmehr könnte Ihre Absicht darin bestehen, Elementfunktionen bereitzustellen, die Daten der Liste hinzufügen oder daraus entfernen.

Solche Elementfunktionen sollten gewährleisten, daß alle Ansichten des Dokuments korrekt aktualisiert werden. Sie sollten außerdem die `SetModified`-Elementfunktion des Dokuments aufrufen, um Änderungen an den Dokumentdaten anzeigen zu lassen. Unterstützt Ihre Anwendung eine Rückgängig-Anweisung, können Sie mit Hilfe von `SetModified` die gepufferten Daten dieser Anweisung aktualisieren.

Das nachfolgende Beispiel aktualisiert die Dokumentobjektauftistung, indem sie dieser ein neues Objekt hinzufügt:

```
BOOL CMyDocument::AddObject(CMyObject *pObject)
{
    try
    {
        m_obList.AddTail((CObject *)pObject);
        SetModifiedFlag(TRUE);
        UpdateAllViews(NULL, UPDATE_OBJECT, pObject);
        return TRUE;
    }
    catch(CMemoryException *e)
    {
        TRACE("CMyDocument::AddObject Fehler in "
            "Speicherallokation.\n");
        e->Delete();
        return FALSE;
    }
}
```

Beachten Sie, wie die Steuerung zwischen dem Dokument und dessen Ansichten übergeben wird. Der Anwender interagiert zunächst mit der Ansicht, so daß ein neues Objekt erzeugt wird. Das Ansichtsbjekt ruft dazu die `AddObject`-Elementfunktion des Dokuments auf. Nachdem das neue Objekt dem Container hinzugefügt wurde, ruft das Dokumentobjekt `UpdateAllViews` auf. Innerhalb dieser Funktion wird die Elementfunktion `OnUpdate` für jede Ansicht aufgerufen, die dem Dokument zugewiesen ist. Die Daten, die `UpdateAllViews` über die Konstante `UPDATE_OBJECT` und einen Zeiger auf ein `CObject`-Objekt übergeben werden, dienen der effizienten Fensteraktualisierung. Diese wird ausge-

führt, indem lediglich die Bereiche erneut gezeichnet werden, die von der Darstellung des neuen Objekts betroffen sind. Der Mechanismus zur Übergabe der Steuerung ist in Abbildung 21.1 dargestellt.

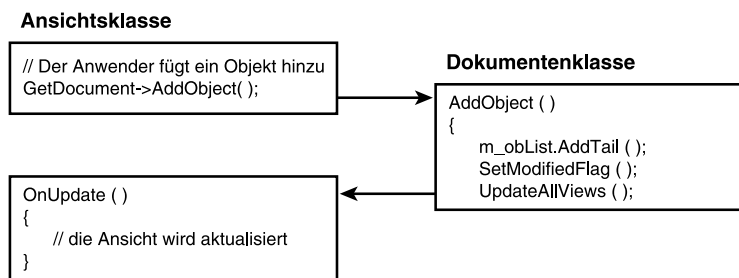


Abbildung 21.1:
Interaktion zwischen der
Ansicht und
dem Dokument

Ein weiterer Vorteil, der sich aus der Verwendung von MFC-Container-Klassen ergibt, besteht darin, daß diese die Serialisierung unterstützen. Um beispielsweise die in einem `CObList`-Container gespeicherten Dokumentdaten zu laden und zu speichern, müssen Sie die `SerializeElement`-Funktion wie folgt ergänzen:

Container-Klassen unterstützen die Serialisierung

```

void CTerminalDoc::Serialize(CArchive &ar)
{
    if (ar.IsStoring())
    {
        // Alle nicht von CObject abgeleiteten Daten
        // serialisieren
    }
    else
    {
        // Alle nicht von CObject abgeleiteten Daten
        // serialisieren
    }
    m_obList.Serialize(ar);
}

```

Damit dieser Programmcode korrekt ausgeführt wird, müssen Sie die `SerializeElement`-Funktion in jeder Ihrer Objektklassen implementieren. Eine von `CObject` abgeleitete Klasse kann sich nicht selbst serialisieren.

Beabsichtigen Sie, allgemeine Container-Templates zu verwenden, sollte Ihre besondere Aufmerksamkeit ebenfalls der Serialisierung gelten. Die Container-Templates `CArray`, `CList` und `CMap` verwenden die `SerializeElements`-Funktion, um Objekte im Container zu serialisieren. Diese Funktion wird wie folgt deklariert:

```

template <class TYPE> void AFXAPI
    SerializeElements(CArchive &ar, TYPE *pElements, int nCount);

```

Da die Container-Templates `TYPE` nicht benötigen, um von `CObject` abgeleitet zu werden, wird die `SerializeElement`-Funktion nicht von den Elementen der Templates aufgerufen (das Vorhandensein dieser Funk-

tion ist nicht gewährleistet). Statt dessen führt die Standardimplementierung von `SerializeElements` ein BITWEISES Lesen und Schreiben aus. Diese Vorgehensweise entspricht jedoch nicht unseren Anforderungen. (Die MFC sollte keine Standardimplementierung zur Verfügung stellen. Der Programmierer müßte die Funktion `SerializeElements` in diesem Fall selbst schreiben, so daß keine Mißverständnisse entstehen könnten.) Nachfolgend finden Sie ein Beispiel aufgeführt, das beschreibt, wie Sie `SerializeElements` für einen von Ihnen definierten Objekttyp implementieren, der eine `Serialize-Elementfunktion` unterstützt:

```
void SerializeElements(CArchive &ar, CMyObject **pObs,
                     int nCount)
{
    for (int i = 0; i < nCount; i++, pObs++)
        (*pObs)->Serialize(ar);
}
```

21.1.5 CCmdTarget und CDocItem

Häufig genügt das Ableiten des Dokumentobjekts von der `CObject`-Klasse nicht. So z.B., wenn Sie die OLE-Automation verwenden möchten. Die OLE-Automation verlangt, daß Ihre Objekte Anweisungsziele sind. `CObject` wird dieser Anforderung nicht gerecht. Sie sollten daher in diesem Fall `CCmdTarget` als Basisklasse Ihrer Objekte verwenden.

Eine weitaus bessere Lösung dieses Problems bietet die `CDocItem`-Klasse. Sie können entweder selbst eine `CDocItem`-Objektauflistung erstellen oder die `COleDocument`-Klasse verwenden. Dazu müssen Sie Ihr Dokument von `COleDocument` und nicht von `CDocument` ableiten. `COleDocument` wird in OLE-Anwendungen verwendet, in denen entweder diese Klasse oder eine davon abgeleitete Klasse als Basisklasse für die Dokumentklasse der OLE-Anwendung dient. `COleDocument` unterstützt eine Auflistung mit `CDocItem`-Objekten. Diese sind vom Typ `COleServerItem` und `COleClientItem`. Die Unterstützung einer Auflistung mit `CDocItem`-Objekten ist in `COleDocument` generisch. Sie können der Auflistung eigene von `CDocItem` abgeleitete Objekte hinzufügen, ohne befürchten zu müssen, daß dies zu Komplikationen mit dem gewöhnlichen OLE-Verhalten führt.

Wie aber deklarieren Sie zusätzliche `CDocItem`-Elemente in `COleDocument`? Nun, eine Deklaration ist nicht erforderlich. Sie müssen lediglich die `COleDocument`-Elementfunktionen, wie z.B. `AddItem`, `RemoveItem`, `GetStartPosition` und `GetNextItem`, verwenden, um Dokumentgegenstände zu ermitteln, hinzuzufügen und zu entfernen.

Um Ihre Dokumentgegenstände und die `COleClientItem`- sowie `COleServerItem`-OLE-Objekte ableiten zu können, müssen Sie jedoch einige Besonderheiten in bestimmten Funktionen implementieren. Sie könnten Ihre Objekte beispielsweise wie folgt deklarieren:

```
class CMyDocItem : public CDocItem
{
    // ...
    CRect m_rect;
};
```

Sie möchten außerdem die `m_rect`-Elementvariable in Ihren OLE-Client-Objekten verwenden:

```
class CMyClientItem : public COleClientItem
{
    // ...
    CRect m_rect;
};
```

Wie erstellen Sie mit dieser Klassendeklaration eine Funktion, die das `m_rect`-Element eines Gegenstands aus Ihrem Dokument verwenden kann?

Die folgende Vorgehensweise ist falsch:

```
MyFunc(CDocItem *pItem)
{
    AnotherFunc(pItem->m_rect); // Fehler!
}
```

Das Kompilieren dieses Programmabschnitts ist nicht möglich, da die `CDocItem`-Klasse über keine Elementvariable mit der Bezeichnung `m_rect` verfügt. Auch der Einsatz eines Zeigers auf Ihre von `CDocItem` abgeleitete Klasse führt nicht zu dem gewünschten Ergebnis:

```
MyFunc(CMyDocItem *pItem)
{
    AnotherFunc(pItem->m_rect);
}
```

Diese Version von `MyFunc` unterstützt nicht OLE-Client-Objekte vom Typ `CDocItem`, sondern lediglich Ihre abgeleitete Klasse. Natürlich könnten Sie zwei überladene Versionen von `MyFunc` erzeugen, diese Vorgehensweise entspricht jedoch nicht dem Prinzip der effektiven Programmierung. Sie müssen somit eine Mantelfunktion erstellen, die einen Zeiger auf das `CDocItem`-Objekt und die MFC-Laufzeittypinformationen verwendet, um die Elementvariable zu ermitteln:

```
CRect GetRect(CDocItem *pDocItem)
{
    if (pDocItem->IsKindOf(RUNTIME_CLASS(CMyDocItem)))
        return ((CMyDocItem *)pDocItem)->m_rect;
    else if (pDocItem->IsKindOf(RUNTIME_CLASS(CMyClientItem)))
        return ((CMyClientItem *)pDocItem)->m_rect;
    ASSERT(FALSE);
    return CRect(0, 0, 0, 0);
}
```

```
MyFunc(CDocItem *pItem)
{
    AnotherFunc(GetRect(pItem));
}
```

Diese Lösung verlangt das Deklarieren und Implementieren von `CMyDocItem` und `CMyClientItem` mit Hilfe der Makros `DECLARE_DYNAMIC` und `IMPLEMENT_DYNAMIC`. Dies ist gewöhnlich möglich, da Ihre Anwendung wahrscheinlich die Serialisierung dieser Gegenstände unterstützt. Die Objekte werden deshalb mit `DECLARE_SERIAL` sowie `IMPLEMENT_SERIAL` deklariert und implementiert (dieses Verfahren impliziert `DECLARE_DYNAMIC` und `IMPLEMENT_DYNAMIC`).

21.2 Die CView-Klasse

Für jede von `CDocument` abgeleitete Klasse besteht eine von `CView` abgeleitete Klasse, die Ihre Dokumentdaten visuell präsentiert und die Interaktion mit dem Anwender über das Ansichtsfenster bearbeitet.

Das Ansichtsfenster ist dem Rahmenfenster untergeordnet. Bei SDI-Anwendungen ist das Ansichtsfenster dem Hauptrahmenfenster untergeordnet, bei MDI-Anwendungen einem eigenen MDI-Rahmen. Während einer OLE-In-Place-Bearbeitung kann es das In-Place-Rahmenfenster sein (sofern Ihre Anwendung dies unterstützt).

Ein Rahmenfenster kann mehrere Ansichtsfenster enthalten (indem es beispielsweise geteilte Fenster verwendet).

21.2.1 Deklarieren einer Ansichtsklasse

Alle Dokumentdaten sollten in der Dokumentklasse deklariert werden. Einige Datenelemente, die sich auf eine bestimmte Ansicht beziehen und nicht beständig sind, werden jedoch nicht als Elemente des Dokuments gespeichert.

Stellen Sie sich beispielsweise eine Anwendung vor, die Daten in verschiedenen Vergrößerungen darstellen kann. Der Vergrößerungsfaktor kann für jede Ansicht unterschiedlich sein. (Unterschiedliche Ansichten können auch dann unterschiedliche Vergrößerungsfaktoren verwenden, wenn sie dieselben Daten darstellen.)

In diesem Fall sollte der Vergrößerungsfaktor als Elementvariable Ihrer Ansichtsklasse deklariert werden:

```
class CZoomView : public CView
{
protected: // Nur aus Serialisierung erstellen
    CZoomView();
    DECLARE_DYNCREATE(CZoomView)
```

```
// Attribute
public:
    CZoomDoc* GetDocument();
    double m_dZoom;
...
```

Wichtiger als jede Elementvariable, die eine Einstellung repräsentiert, ist eine Elementvariable, die die AKTUELLE AUSWAHL aufnimmt. Damit ist die Auflistung der Objekte in Ihrem Dokument gemeint, die der Anwender zur Bearbeitung ausgewählt hat. Welche Bearbeitung vorgenommen werden kann, ist von der Anwendung abhängig. Einige Funktionen sind jedoch für alle Anwendungen gleich. Dazu zählen das Ausschneiden und Kopieren in die Zwischenablage sowie OLE-Drag&Drop.

Die einfachste Möglichkeit der Implementierung einer Selektion besteht darin, wie in der Dokumentklasse eine Auflistungsklasse zu verwenden. Sie könnten die Auflistung, die die aktuelle Selektion repräsentieren soll, wie folgt deklarieren:

```
class CMyView : public CView
{
    // ...
    CList<CDocItem *, CDocItem *> m_selList;
    // ...
}
```

Zusätzlich zur Deklaration der Ansichtsklasse müssen Sie eine OnDraw-Elementfunktion schreiben, um der Klasse die gewünschte Funktionalität zuzuweisen. Der Standardimplementierung dieser Funktion sind keine Aufgaben zugeordnet. Sie müssen den Programmcode definieren, der Ihre Dokumentdaten darstellt.

Ist Ihre Dokumentklasse beispielsweise von COLEDocument abgeleitet und verwendet sie CDocItem-Objekte für Ihre Dokumentdaten, könnte Ihre OnDraw-Elementfunktionsimplementierung wie folgt aufgebaut sein:

```
void CMyView::OnDraw(CDC *pDC)
{
    CMyDoc *pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    POSITION pos = pDoc->GetStartPosition();
    while (pos != NULL)
    {
        CDocItem *pObject = pDoc->GetNextItem(pos);
        if (pObject->IsKindOf(RUNTIME_CLASS(CMyDocItem)))
        {
            ((CMyDocItem *)pObject)->Draw(pDC);
        }
        else if (pObject->IsKindOf(RUNTIME_CLASS(CMyClientItem)))
        {
            ((CMyClientItem *)pObject)->Draw(pDC);
        }
        else
            ASSERT(FALSE);
    }
}
```

21.2.2 CView-Elementfunktionen

Die `CView`-Klasse bietet eine große Auswahl verschiedener Elementfunktionen.

Dokumentobjekt ermitteln

Die vorwiegend verwendete Elementfunktion ist `GetDocument`, die einen Zeiger auf das Dokumentobjekt zurückgibt, das der Ansicht zugeordnet ist.

Drucken

`DoPreparePrinting` zeigt den Drucken-Dialog an und erstellt gemäß der Auswahl des Anwenders einen Drucker-Gerätekontext.

Elementfunktionen zum Überschreiben

Die verbleibenden `CView`-Elementfunktionen sind zum Überschreiben gedacht. Sie ergänzen die große Zahl überschreibbarer Funktionen, die für die `CWnd`-Klasse erhältlich sind (die Basis von `CView`) und bearbeiten die meisten Benutzeroberflächenereignisse. Diese Funktionen sind derart zahlreich, daß sie in diesem Buch nicht aufgeführt werden können. Zu den Funktionen zählen Nachrichtenbearbeiter für die Tastatur, die Maus, den Zeitgeber sowie System- und andere Nachrichten. Einige Funktionen bearbeiten Zwischenablage- und MDI-Ereignisse und initialisieren respektive löschen Nachrichten. Ihre Anwendung sollte eigene Implementierungen für diese Funktionen zur Verfügung stellen, sofern diese benötigt werden. Ermöglicht Ihr Programm beispielsweise dem Anwender, ein Objekt in einem Dokument zu platzieren, indem er darauf klickt und es mit der Maus auf das Ziel zieht, sollten Sie eine Überladung für die Elementfunktion `CWnd::OnLButtonDown` bereitstellen.

OnDraw

Eine der interessantesten Elementfunktionen wurde bereits genannt. Das Überschreiben von `OnDraw` ist für ein von `CView` abgeleitetes Objekt erforderlich, um Daten darstellen zu können.

IsSelected

Die `IsSelected`-Elementfunktion muß für OLE-Anwendungen implementiert werden. Diese Funktion gibt `TRUE` zurück, wenn das Objekt, auf das der Parameter der Funktion verweist, ein Abschnitt der gegenwärtigen Selektion in der Ansicht ist. Wenn Sie Ihre Selektion mit Hilfe der `CList`-Vorlagenauflistung als eine `CDocItem`-Objektauflistung implementieren, sollte `IsSelected` wie folgt übernommen werden:

```

BOOL CMyView::IsSelected(const CObject* pDocItem) const
{
    return (m_selList.Find((CDocItem *)pDocItem) != NULL);
}

```

OnUpdate

Eine weitere wichtige überschreibbare Elementfunktion ist `OnUpdate`. Diese wird von `UpdateAllViews` aus der Dokumentklasse aufgerufen, die der Ansicht zugeordnet ist. Die Standardimplementierung löscht den gesamten Client-Bereich des Ansichtsfensters. Sie können die Performance Ihrer Anwendung erhöhen, indem Sie diese Funktion überschreiben und lediglich die Bereiche löschen, die aktualisiert werden müssen. Das folgende Beispiel zeigt eine Implementierung von `OnUpdate`:

```

void CMyView::OnUpdate(CView *pView, LPARAM lHint, CObject *pObj)
{
    if (lHint == UPDATE_OBJECT) // von Ihnen definierte Hint-Konstante
        InvalidateRect(((CMyObject *)pObj)->m_rect);
    else
        Invalidate();
}

```

Sie sollten gewöhnlich keine Zeichenvorgänge in `OnUpdate` ausführen lassen. Verwenden Sie dazu die `OnDraw`-Elementfunktion Ihrer Ansicht.

OnPrepareDC

Die `OnPrepareDC`-Elementfunktion ist besonders bedeutend, wenn Ihre Ansicht keinen Standardumwandlungsmodus wie die Vergrößerung unterstützt. In dieser Funktion können Sie den Umwandlungsmodus des Ansichtsfensters setzen, bevor ein Zeichenvorgang ausgeführt wird. Denken Sie daran, wenn Sie einen Gerätekontext für Ihr Ansichtsfenster erstellen, `OnPrepareDC` aufzurufen, um zu gewährleisten, daß dem Gerätekontext die korrekten Einstellungen zugewiesen werden.

Bisweilen ist es erforderlich, einen Gerätekontext zu erstellen, um den aktuellen Umwandlungsmodus mit `OnPrepareDC` zu ermitteln. Die `OnLButtonDown`-Elementfunktion Ihrer Ansicht könnte beispielsweise die Position des Mausklicks von physikalische in logische Koordinaten umwandeln müssen:

```

void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    OnPrepareDC(&dc);
    dc.DPtoLP(&point);
    // ...
}

```

Andere `CView`-Elementfunktionen unterstützen die Initialisierung und das Beenden, OLE-Drag&Drop, den Bildlauf, die Aktivierung und Deaktivierung der Ansicht sowie das Drucken. Ob diese Funktionen über-

geschrieben werden müssen, ist davon abhängig, ob Sie ein besonderes Feature unterstützen möchten und ob die Standardimplementierung (sofern vorhanden) Ihren Anforderungen genügt.

21.2.3 Ansichten und Nachrichten

Zusätzlich zu Nachrichten, für die bereits Standardbearbeiter in `CView` oder der übergeordneten Klasse `CWnd` existieren, bearbeitet eine gewöhnliche Ansichtsklasse viele andere Nachrichten. Dazu zählen Befehlsnachrichten, die eine Auswahl des Anwenders aus einem Menü repräsentieren, Symbolleistenschaltflächen und andere Benutzeroberflächenobjekte.

Wenn Sie entscheiden, ob eine Ansicht oder das Dokument (oder der Rahmen) eine bestimmte Nachricht bearbeiten soll, sind das bestimmende Kriterium der Geltungsbereich und die Auswirkungen der Nachricht oder der Anweisung.

- ➡ Betrifft die Anweisung das gesamte Dokument, sollte es von der Dokumentklasse bearbeitet werden (sofern sich die Anweisung nicht wie in einigen Implementierungen des Einfügen-Befehls auf eine bestimmte Ansicht bezieht).
- ➡ Wirkt sich die Anweisung lediglich auf eine Ansicht aus (so wie das Setzen eines Vergrößerungsfaktors), sollte sie von einem Ansichtsobjekt bearbeitet werden.

21.2.4 Varianten von `CView`

Die MFC-Bibliothek stellt einige von `CView` abgeleitete Klassen zur Verfügung, die besonderen Zwecken dienen. Diese Klassen sind in Tabelle 21.2 aufgeführt.

Tabelle 21.2:
Die Varianten
von `CView`

Klassenname	Beschreibung
<code>CCtrlView</code>	Unterstützt Ansichten, die auf einem Steuerelement basieren (zum Beispiel eine Strukturansicht oder Textfeld)
<code>CDaoRecordView</code>	Stellt Datensätze in Dialogfeld-Steuerelementen dar
<code>CEditView</code>	Stellt ein Editor-Fenster mit Hilfe eines mehrzeiligen Textfeld-Steuerelements zur Verfügung
<code>CFormView</code>	Basiert auf einer Dialogfeldvorlage; zeigt Dialogfeld-Steuerelemente an
<code>CHtmlView</code>	Ansicht für Programme mit Webbrowser-Funktionalität, unterstützt auch dynamisches HTML.

Klassenname	Beschreibung
CListView	Stellt ein Listefeld-Steuerelement dar
CRecordView	Stellt Datensätze in Dialogfeld-Steuerelementen dar
CRichEditView	Stellt ein Rich-Text-Steuerelement dar
CScrollView	Ermöglicht die Verwendung von Bildlaufleisten
CTreeView	Stellt ein Strukturansicht-Steuerelement dar

Eine selten überladene Variante von `CView` ist `CPreviewView`. Diese Klasse wird zur Bereitstellung einer Druckvorschau von dem MFC-Anwendungsrahmen verwendet.

All diese Klassen verfügen über Elementfunktionen, die bestimmte Aufgaben ausführen. Die Elementfunktionen der von `CCtrlView` abgeleiteten Klassen bearbeiten Windows-Nachrichten, die sich auf die Steuerelementklasse beziehen, die sie repräsentieren.

`CFormView` und davon abgeleitete Klassen (`CDaoRecordView` und `CRecordView`) unterstützen den Dialog-Datenaustausch. Sie verwenden diese Klassen in einer ähnlichen Weise wie die von `CDialog` abgeleiteten Klassen.

21.2.5 Dialogbasierte Anwendungen

Dialogbasierte Anwendungen unterstützen kein Doc/View. Wenn Sie mit Hilfe des Anwendungsassistenten eine dialogbasierende Anwendung erstellen, verfügt das daraus resultierende Programm über keine Dokument- oder Ansichts-klasse. Statt dessen wird die gesamte Funktionalität in einer einzelnen Dialogklasse implementiert, die sich von `CDialog` ableitet.

Obwohl dies für die meisten Anwendungen ausreichend ist, werden einige MFC-Features nicht unterstützt. Eine dialogbasierende Anwendung erhält kein Menü, keine Symbolleiste und auch keine Statusleiste. Sie unterstützt weder OLE noch die MAPI und keine Druckfunktionen.

Eine Alternative zur Verwendung einer dialogbasierten Anwendung besteht darin, eine Anwendung zu erstellen, die die `CFormView`-Klasse als Basisklasse für das Ansichtsfenster verwendet und das SDI-Anwendungsmodell nutzt. Auf diese Weise verfügen Sie über alle Vorteile einer MFC-Anwendung und präsentieren überdies einen Dialog, indem Sie eine Dialogvorlage für die Definition der Ansichtsinhalte verwenden und den Dialog-Datenaustausch nutzen.

21.3 Zusammenfassung

Die meisten MFC-Anwendungen basieren auf dem Dokument/Ansicht-Modell. Das Dokument, ein abstraktes Objekt, repräsentiert die Anwendungsdaten und entspricht gewöhnlich dem Inhalt einer Datei. Die Ansicht hingegen stellt die Daten dar und bearbeitet Benutzeroberflächenereignisse.

Einem Dokument können mehrere Ansichten zugewiesen sein. Eine Ansicht ist jedoch immer nur einem Dokument zugeteilt.

Dokumentklassen werden von `CDocument` abgeleitet. Diese Klasse enthält die Funktionalität eines Dokumentobjekts. Anwendungen müssen der Klasse lediglich einige Elementvariablen hinzufügen, die die spezifischen Anwendungsdaten aufnehmen, und eigene Implementierungen für die Elementfunktionen `OnNewDocument` (Initialisierung) und `Serialize` (Speichern und Laden) zur Verfügung stellen, um eine funktionsfähige Dokumentklasse zu erhalten.

Komplexe Anwendungen verwenden gewöhnlich Auflistungsklassen, um die Objekte zu implementieren, die ein Dokument bilden. Anwendungen verwenden die Klasse `COleDocument` und deren Funktionen, um eine `CDocItem`-Objektauflistung zu verwalten, die nicht auf OLE-Client- und OLE-Server-Objekte beschränkt ist.

Ansichtsklassen werden von `CView` abgeleitet. Ansichtsfenster, die `CView`-Objekte repräsentieren, sind Child- oder Rahmenfenster. Ein Rahmenfenster kann über mehrere Child-Ansichtsfenster verfügen. Dies ist der Fall, wenn geteilte Fenster verwendet werden.

Ein Ansichtsobjekt enthält Elementvariablen, die spezifische Anzeigeeinstellungen aufnehmen. Diese Objekte implementieren häufig die aktuelle Selektion. Die gegenwärtige Selektion besteht aus den Dokumentobjekten, die der Anwender in der aktuellen Ansicht zur weiteren Bearbeitung ausgewählt hat. Anwendungen verwenden zum Implementieren der Selektion Auflistungsklassen.

Eine Ansichtsklasse muß eine Implementierung der `OnDraw`-Elementfunktion zur Verfügung stellen, um die Objekte des entsprechenden Dokuments zeichnen zu können. OLE-Anwendungen nutzen außerdem die `IsSelected`-Elementfunktion. Andere häufig überladene Elementfunktionen sind `OnPrepareDC` und `OnUpdate`.

Verschiedene Varianten der `CView`-Klasse bearbeiten den Bildlauf, Ansichten, die auf Dialogfeldern basieren, Steuerelemente und Ansichten,

in denen Datensätze dargestellt werden. Sie sollten die Klasse als Basisklasse Ihrer Ansichtsklasse verwenden, die den Anforderungen Ihrer Anwendung gerecht wird.

Sowohl Dokumente als auch Ansichten (sowie Rahmenobjekte) bearbeiten Nachrichten. Welche Klasse eine bestimmte Nachricht bearbeiten sollte, ist von der Auswirkung der Nachricht abhängig. Betrifft die Nachricht das gesamte Dokument, sollte die Dokumentklasse die Nachricht bearbeiten. Bezieht sich die Nachricht auf eine bestimmte Ansicht, ist natürlich die Ansichtsklasse für deren Bearbeitung zuständig. Nachrichten, die sich global auf die Anwendung auswirken, sollten von der Rahmenklasse bearbeitet werden.

Dialoge und Registerdialoge

Kapitel 22

Anwendungen verwenden sehr häufig Dialoge. Die MFC-Bibliothek unterstützt Dialoge mit der `CDialog`-Klasse und den davon abgeleiteten Klassen.

Ein `CDialog`-Objekt entspricht einem Dialogfenster, dessen Inhalt auf einer Dialogvorlagenressource basiert. Die Dialogvorlagenressource kann mit Hilfe eines Dialog-Editors bearbeitet werden. Dazu verwenden Sie gewöhnlich den Dialog-Editor des Visual Studios.

Ein interessantes Feature von `CDialog` ist der Dialog-Datenaustausch. Dieser Mechanismus ermöglicht den einfachen und effizienten Datenaustausch zwischen den Steuerelementen eines Dialogs und den Elementvariablen in der Dialogklasse.

Die `CDialog`-Klasse wird von `CWnd` abgeleitet. Sie können daher viele `CWnd`-Elementfunktionen verwenden, um Ihren Dialog zu erweitern. Außerdem kann Ihre Dialogklasse über Nachrichtentabellen verfügen. Komplexen Dialogen müssen häufig Nachrichtentabelleneinträge hinzugefügt werden, damit diese bestimmte Steuernachrichten bearbeiten können.

Aktuelle Anwendungen verwenden oft mit Registerreitern versehene Dialoge, die auch als `REGISTERDIALOGS` bezeichnet werden. Ein Registerdialog besteht gewissermaßen aus mehreren Dialogen. Der Anwender selektiert einen Registerreiter, um eine der Registerdialogseiten zu öffnen.

Die nächsten Abschnitte beschreiben, wie Dialoge in einer MFC-Anwendung erstellt werden.

22.1 Erstellen von Dialogen

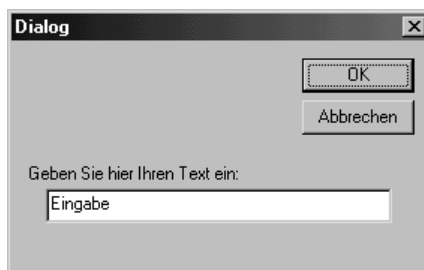
Um einen Dialog konstruieren und diesen Ihrer Anwendung hinzufügen zu können, muß

- die Dialogvorlagenressource generiert werden,
- eine von `CDialog` abgeleitete Klasse erstellt werden, die sich auf diese Ressource bezieht,
- innerhalb Ihrer Anwendung ein Objekt dieser Klasse erzeugt werden,
- die Kommunikation zwischen Dialog und Anwendung eingerichtet werden.

Für unsere Dialogbeispiele verwenden wir eine einfache SDI-Anwendung, die mit dem Anwendungsassistenten erstellt wird und mit DLG1 bezeichnet ist. Andere Einstellungen als die Auswahl der Option EINZELNES DOKUMENT (SDI), sind in dem Dialog des Anwendungsassistenten nicht erforderlich.

Der nächste Abschnitt beschreibt, wie Sie einen einfachen Dialog erstellen. Der Dialog enthält ein Eingabefeld und wird mit dem Menübefehl ANSICHT/DIALOG der DLG-Anwendung verbunden. Der von DLG1 angezeigte Dialog ist in Abbildung 22.1 dargestellt.

Abbildung 22.1:
Ein einfacher
Dialog



22.1.1 Hinzufügen einer Dialogvorlage

Zunächst muß die Dialogvorlagenressource generiert werden. Diese Ressource wird mit dem integrierten Dialog-Editor des Visual Studios erzeugt.

1. Rufen Sie – nachdem Sie vom Anwendungsassistenten die SDI-Anwendung haben erstellen lassen – den Menübefehl EINFÜGEN/RESOURCE auf.

2. Wählen Sie im erscheinenden Dialogfeld den Ressourcentyp DIALOG aus und klicken Sie auf den Schalter NEU.
3. Ziehen Sie aus der Steuerelemente-Leiste ein statisches Textfeld und ein Eingabefeld in Ihren Dialog.

Abbildung 22.1 zeigt den Dialog während des Entwurfs. Die Schaltflächen OK und ABBRECHEN werden automatisch von dem Dialog-Editor angeordnet. Das Eingabefeld erhält den Bezeichner IDC_EDIT1, der Dialog selbst wird mit IDD_DIALOG1 bezeichnet.

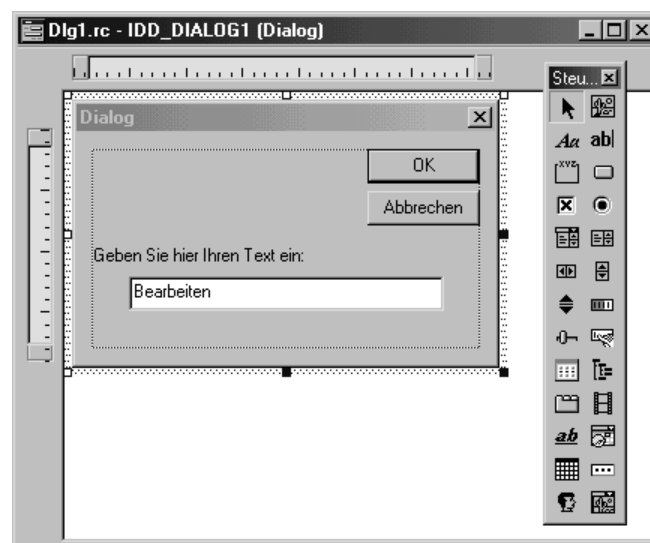


Abbildung 22.2:
Entwurf eines
einfachen Dia-
logs

Während die Dialogvorlage zur Bearbeitung im Dialog-Editor geöffnet ist, können Sie den Klassen-Assistenten aufrufen, um die Dialogklasse zu definieren, über die auf die Vorlage zugegriffen werden soll.

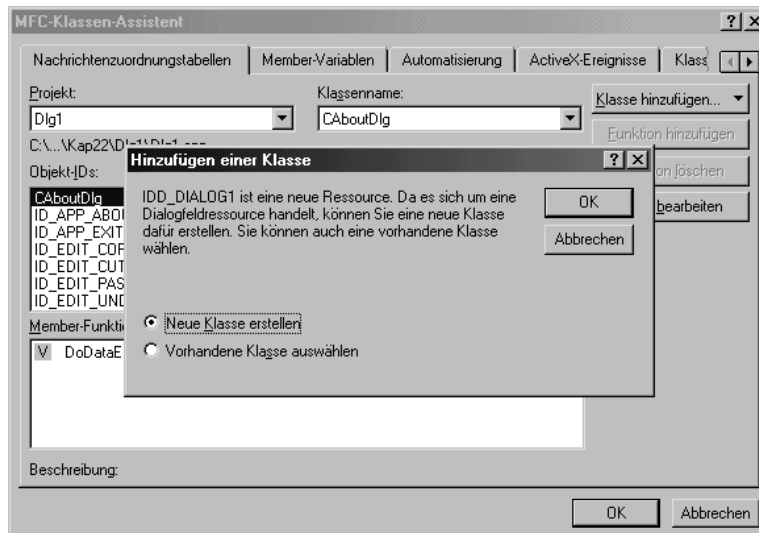
22.1.2 Erstellen der Dialogklasse

Natürlich können Sie eine Dialogklasse manuell erstellen. Sie sollten jedoch zu diesem Zweck den Klassen-Assistenten verwenden. Der Assistent erleichtert diese Aufgabe. Um eine Dialogklasse für den in Abbildung 22.1 dargestellten Dialog zu erstellen, sind folgende Schritte notwendig:

1. Bewegen Sie den Mauszeiger auf das Fenster des Dialog-Editors, und betätigen Sie die rechte Maustaste.
2. Wählen Sie aus dem anschließend geöffneten Kontextmenü den Eintrag KLASSEN-ASSISTENT aus.

3. Wird der Klassenassistent für eine neue Ressource aufgerufen, wird der in Abbildung 22.2 dargestellte Dialog HINZUFÜGEN EINER KLASSE geöffnet. Selektieren Sie dort die Option NEUE KLASSE ERSTELLEN, und betätigen Sie anschließend die Schaltfläche OK.

Abbildung 22.3:
Hinzufügen
einer Dialog-
Klasse



In dem nun angezeigten Dialog NEUE KLASSE (Abbildung 22.3) können Sie den Namen des Dialoges angeben und weitere Optionen setzen, wie zum Beispiel den Dateinamen, den Resourcebezeichner oder Automatisationseinstellungen. Sie können die Klasse außerdem der Komponentengalerie hinzufügen, so daß diese später von anderen Anwendungen verwendet werden kann.

4. Geben Sie der neuen Klasse einen Namen, zum Beispiel `CMyDialog`.
5. Lassen Sie die von dem Klassenassistenten generierten Dateinamen `MYDIALOG.H` und `MYDIALOG.CPP` für unser Beispiel unverändert.
6. Betätigen Sie die Schaltfläche OK, um die neue Klasse zu erstellen und zu dem Dialog des Klassenassistenten zurückzukehren.



Sollten Sie den Dateinamen ändern, den der Klassenassistent für die neuen Header- und Implementierungsdateien Ihrer Klasse vorschlägt? Sollten Sie eine separate Header- und Implementierungsdatei für jeden neuen Dialog verwenden, den Sie erstellen? Dies ist eine interessante Frage. Oberflächlich betrachtet sollte die Antwort ja lauten. Doch selbst der Anwendungsassistent handelt anders.

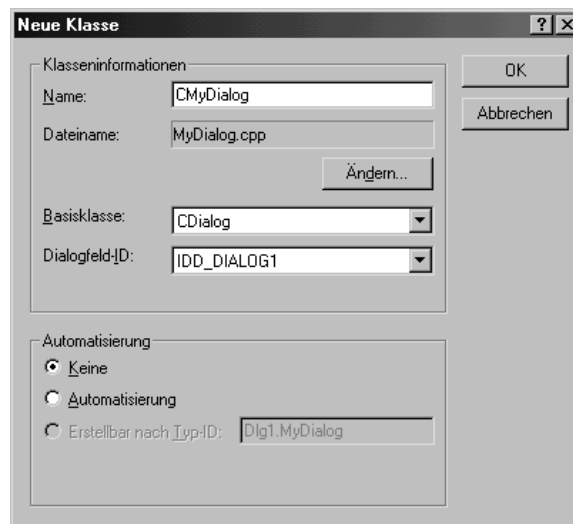


Abbildung 22.4:
Der Dialog neue
Klasse

Er nimmt sowohl die Deklaration als auch die Implementierung des Info-Dialogs Ihrer Anwendung in der Implementierungsdatei des Anwendungsobjekts vor. Die Entscheidung bleibt daher Ihnen überlassen. Einige Anwender fassen Dialogklassen in Gruppen zusammen, wenn diese nicht umfangreich und einfach aufgebaut sind. Wären diese in separaten Dateien enthalten, könnte dies die Übersichtlichkeit des Arbeitsbereichs der Anwendung einschränken. Unter Visual C++ ist dieses Argument jedoch irrelevant, da Sie hier nicht mehr mit der Dateiansicht arbeiten müssen, um auf Ihren Quellcode zugreifen zu können.

Sie müssen der Klasse nun eine Elementvariable hinzufügen, die mit dem Textfeld in der Dialogvorlage kongruiert.

22.1.3 Hinzufügen einer Elementvariable

1. Selektieren Sie das Register MEMBER-VARIABLEN im Dialog des Klassen-Assistenten, um eine neue Elementvariable zu definieren (Abbildung 22.4).
2. Sie definieren die Elementvariable für das IDC_EDIT-Steuerelement, indem Sie einen Doppelklick auf dessen Bezeichner in dem Listenfeld unter STEUERELEMENT-IDS ausführen. Daraufhin wird ein weiterer Dialog geöffnet, der in Abbildung 22.5 dargestellt ist.
3. Geben Sie den Namen der neuen Variablen ein (`m_sEdit`), und betätigen Sie bitte die Schaltfläche OK. Schließen Sie anschließend den Dialog des Klassen-Assistenten mit einem Klick auf OK.

Abbildung 22.5:
Elementvariablen im Klassen-Assistenten

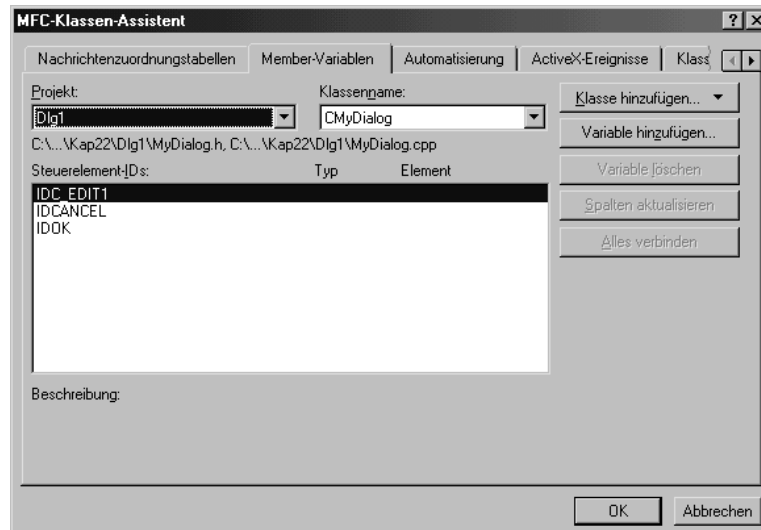
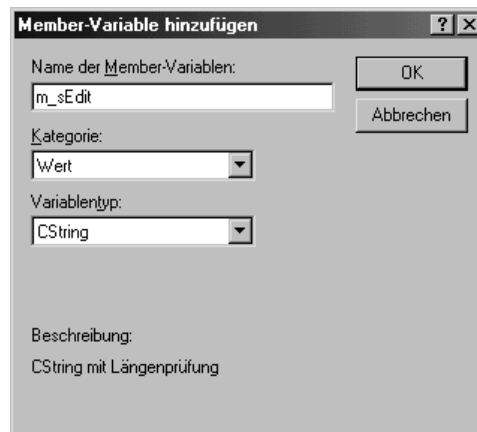


Abbildung 22.6:
Der Dialog
Member-Vari-
ble hinzufügen



Schließen Sie auch das Fenster, in dem die Vorlagenressource angezeigt wird. Sie werden gleich den Programmcode schreiben, der unseren Dialog aufruft. Betrachten Sie jedoch zunächst den Code, den der Klassenassistent bisher erzeugt hat.

22.1.4 Die Ergebnisse des Klassenassistenten

Die Deklaration von `CMyDialog` (in `MYDIALOG.H`) ist in Listing 22.1 aufgeführt. Ein Abschnitt der Klassendeklaration ist die Deklaration von `IDC`, die die Dialogvorlage bezeichnet. Die Klassendeklaration enthält ebenfalls die Elementvariable `m_sEdit`, die wir mit Hilfe des Klassenassistenten erstellt haben.

```

class CMyDialog : public CDialog
{
// Konstruktion
public:
    CMyDialog(CWnd* pParent = NULL); // Standardkonstruktor

// Dialogfelddaten
//{{AFX_DATA(CMyDialog)
enum { IDD = IDD_DIALOG1 };
CString m_sEdit;
//}}AFX_DATA

// Überschreibungen
// Vom Klassen-Assistenten generierte virtuelle
// Funktionsüberschreibungen
//{{AFX_VIRTUAL(CMyDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV-
                                                    // Unterstützung
//}}AFX_VIRTUAL

// Implementierung
protected:

    // Generierte Nachrichtenzuordnungsfunktionen
    //{AFX_MSG(CMyDialog)
    // HINWEIS: Der Klassen-Assistent fügt hier
    // Member-Funktionen ein
    //{AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

Das Listing enthält auch die Deklarationen des Konstruktors und der überschriebenen DoDataExchange-Funktion.

Diese beiden Funktionen sind in *MYDIALOG.CPP* (Listing 22.2) definiert. Beachten Sie bitte, daß der Klassenassistent Programmcode in diese Funktionen eingefügt hat. Die Elementvariable `m_sEdit` wird in dem Konstruktor initialisiert. Ein Verweis auf diese Variable befindet sich in DoDataExchange.

```

CMyDialog::CMyDialog(CWnd* pParent /*=NULL*/)
: CDialog(CMyDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CMyDialog)
    m_sEdit = _T("");
    //}}AFX_DATA_INIT
}

void CMyDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMyDialog)
    DDX_Text(pDX, IDC_EDIT1, m_sEdit);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMyDialog, CDialog)
    //{{AFX_MSG_MAP(CMyDialog)

```

*Listing 22.1:
Die CMyDialog-
Klassendeklara-
tion*

*Listing 22.2:
Die CMyDialog-
Elementfunktio-
nen*

```
// HINWEIS: Der Klassen-Assistent fügt hier Zuordnungsmakros
// für Nachrichten ein
//))AFX_MSG_MAP
END_MESSAGE_MAP()
```

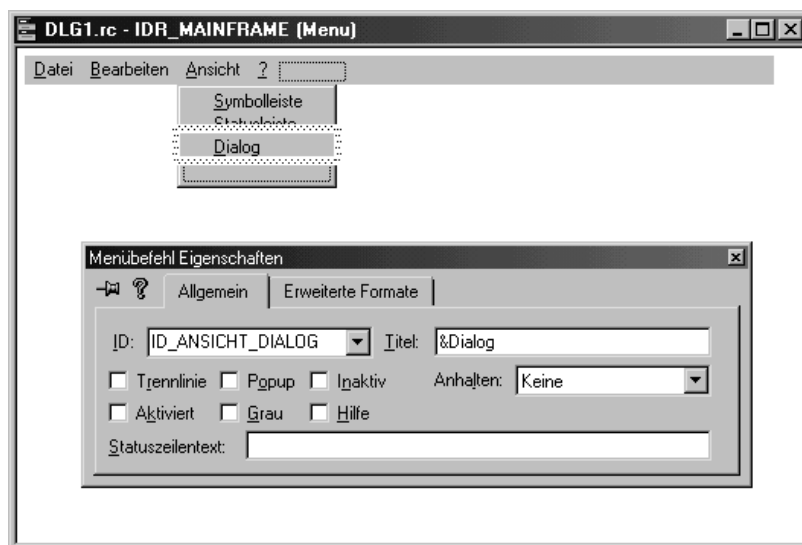
DoDataExchange DoDataExchange ist die Funktion, die den Datenaustausch zwischen den Elementvariablen und den Dialogfeld-Steuer-elementen ermöglicht. Sie wird aufgerufen, wenn der Dialog geöffnet und geschlossen wird. Die von dem Klassen-Assistenten eingefügten Makros (wie zum Beispiel das Makro `DDX_Text`) erlauben den Datenaustausch in beide Richtungen. Die Richtung wird mit der Elementvariable `m_bSaveAndValidate` des `CDataExchange`-Objekts bestimmt, auf das der Parameter `pDX` verweist. Wir werden diese Funktion und die verschiedenen Hilfsmakros zum Datenaustausch in Kürze erörtern.

22.1.5 Aufrufen des Dialogs

Der Dialog wurde erstellt. Wie rufen wir diesen aus der DLG1-Anwendung auf?

Der neue Dialog soll geöffnet werden, wenn der Anwender einen neuen Befehl mit der Bezeichnung `DIALOG` aus dem Menü `ANSICHT` auswählt.

Abbildung 22.7:
Hinzufügen des
Menüeintrags



Dieser Menüeintrag muß nun dem Anwendungsmenü mit Hilfe des Ressourceneditors hinzugefügt werden (Abbildung 22.6).

1. Doppelklicken Sie dazu in der Ressourcen-Ansicht auf die `IDR_MAINFRAME`-Ressource unter dem Ordner `MENU`.
2. Doppelklicken Sie im Menü `ANSICHT` auf die leere Menübefehls-schablone.
3. Geben Sie im erscheinenden Dialogfeld einen `TITEL` für den Menü-befehl ein. (Durch ein vorangestelltes kaufmännisches Und (&) können Sie einen Buchstaben als `[ALT]`-Tastenkombination kenn-zeichnen.

Rufen Sie jetzt den Klassenassistenten auf, um den Programmcode zu definieren, der den neuen Menüeintrag bearbeitet. Hier müssen Sie der `CMainFrame`-Klasse eine Anweisungsbearbeitungsfunktion für den Bezeichner `ID_ANSICHT_DIALOG` hinzufügen.

Doch wieso der `CMainFrame`-Klasse? Das Darstellen des Dialoges bezieht sich nicht auf ein bestimmtes Dokument oder auf eine der Ansichten. `CMainFrame` ist daher für unsere Zwecke geeignet.

1. Bewegen Sie den Mauszeiger auf den neuen Menüeintrag, und klicken Sie auf die rechte Maustaste.
2. Wählen Sie aus dem anschließend dargestellten Kontextmenü den Eintrag `KLASSEN-ASSISTENT` aus. (Sollten Sie dazu aufgefordert werden, für die neue Ressource eine Klasse zu bestimmen, ent-scheiden Sie sich bitte für die bereits vorhandene Klasse `CMyDia-log`.)
3. Wählen Sie auf der Registerseite `NACHRICHTENZUORDNUNGSTA-BELLEN` des Klassen-Assistenten den Bezeichner `ID_ANSICHT_DIALOG`.
4. Wählen Sie aus dem Listefeld unter Nachrichten die Nachricht `COMMAND` aus, und betätigen Sie bitte die Schaltfläche `FUNKTION HIN-ZUFÜGEN`.
5. Über den Schalter `CODE BEARBEITEN` gelangen Sie zur Definition der neuen Funktion.

Der Bezeichner `ID_ANSICHT_DIALOG` wird automatisch von dem Klas-senassistenten aus dem Namen des Eintrags (Dialog) und der Be-zeichnung des Menüs (ANSICHT) erzeugt. Verwenden Sie andere Begriffe zur Bezeichnung dieses Menüs oder des Eintrags, wird der Bezeichner ebenfalls anders lauten. Die von dem Klassen-Assisten-ten generierten Funktionen erhalten Namen, die auf dieselbe Wei-se bestimmt werden.



Dialog erzeugen und anzeigen Die Implementierung von `CMainFrame::OnAnsichtDialog` ist in Listing 22.3 aufgeführt. Nach der Erstellung des Dialogobjekts weisen wir der Elementvariablen `m_sEdit` einen initialisierenden Wert zu. Wir greifen direkt auf diese Variable zu, da sie ein öffentliches Element der `CMyDialog`-Klasse ist. Wäre die Variable hingegen vom Typ `Private` oder `Protected`, müßten wir über zwei `CMyDialog`-Elementfunktion (`Get/Set`-Methoden) darauf zugreifen.

DoModal Der Dialog wird anschließend mit der Funktion `DoModal` aufgerufen. Nachdem der Dialog von dem Anwender geschlossen wurde, überprüfen wir den neuen Wert von `m_sEdit`, indem wir diesen in einem Nachrichtenfeld ausgeben lassen.

Listing 22.3:
Die `CMainFrame::OnAnsichtDialog`-Elementfunktion

```
void CMyDialog::OnAnsichtDialog()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    CMyDialog myDialog;
    myDialog.m_sEdit = "Eingabe";
    if (myDialog.DoModal() == IDOK)
        MessageBox(myDialog.m_sEdit);
}
```

Damit in `CMainFrame::OnAnsichtDialog` ein Objekt vom Typ `CMyDialog` deklariert werden kann, muß die Header-Datei `MYDIALOG.H` in `MAINFRM.CPP` eingebunden (`#include`-Anweisung) werden.

Die Anwendung kann nun kompiliert und ausgeführt werden.

22.1.6 Nicht-modale Dialoge

Die `DoModal`-Elementfunktion stellt einen modalen Dialog dar. Einige Anwendungen müssen jedoch nicht-modale Dialoge darstellen. Das Erstellen und die Anzeige eines nicht-modalen Dialoges unterscheidet sich von der entsprechenden Vorgehensweise für einen modalen Dialog.

Von modal zu nicht-modal **Nicht-modalen Dialog erzeugen**

Um den Dialog in DLG in einen nicht-modalen Dialog zu konvertieren, müssen wir zunächst die Konstruktorfunktion modifizieren. Dort muß zur Erstellung des Dialogobjekts die `Create`-Elementfunktion aufgerufen werden. Außerdem ist der Aufruf einer anderen Version des Basiskonstruktors erforderlich, wie in Listing 22.4 aufgeführt.

Listing 22.4:
Die nicht-modale Version von `CMyDialog`

```
CMyDialog::CMyDialog(CWnd* pParent /*=NULL*/)
: CDialog()
{
    Create(CMyDialog::IDD, pParent);
    //{{AFX_DATA_INIT(CMyDialog)
    m_sEdit = _T("");
    //}}AFX_DATA_INIT
}
```

Nicht-modalen Dialog anzeigen

Auch der Aufruf des Dialogs aus `CMainFrame::OnAnsichtDialog` unterscheidet sich von dem ersten Beispiel. Hier wird nicht die `DoModal`-Elementfunktion aufgerufen, da das Dialogobjekt bereits mit `Create` erzeugt wurde.

Beachten Sie bitte, daß unser nicht-modaler Dialog nicht über den Stack zur Verfügung gestellt wird. Da der Dialog sehr lange und auch dann dargestellt werden kann, wenn die Funktion `CMainFrame::OnAnsichtDialog` bereits beendet ist, müssen wir den Speicher für das `CDialog`-Objekt auf eine andere Weise reservieren. Die neue Version von `CMainFrame::OnAnsichtDialog` ist in Listing 22.5 aufgeführt (`MAIN-FRM.CPP`).

```
void CMainFrame::OnAnsichtDialog()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    CMyDialog *pMyDialog;
    pMyDialog = new CMyDialog;
    pMyDialog->m_sEdit = "Eingabe";
    pMyDialog->UpdateData(FALSE);
    pMyDialog->ShowWindow(SW_SHOW);
}
```

`UpdateData` wird in dieser Funktion aufgerufen, da wir den Wert von `m_sEdit` setzen, nachdem das Dialogobjekt erstellt und der Datenaustausch eingerichtet wurde. Durch den Aufruf von `UpdateData` gewährleisten wir, daß die Steuerelemente in dem Dialogobjekt mit den Einstellungen in den Elementvariablen des `CDialog`-Objekts aktualisiert werden. Dieses Beispiel zeigt einmal mehr, daß C++-Objekte und Windows-Objekte zwei unterschiedliche Gebilde sind.

Wir müssen außerdem die `ShowWindow`-Elementfunktion aufrufen, um den neuen Dialog anzuzeigen. Alternativ dazu hätten wir die Dialogvorlagenressource mit dem Stil `WS_VISIBLE` erstellen können.

Behandlung der Schaltflächen

Der Dialog wird angezeigt, bis der Anwender die Schaltfläche OK oder ABBRECHEN betätigt. Die Standardimplementierung von `CDialog::OnOK` respektive `CDialog::OnCancel` zerstören den Dialog nicht, sondern zeigen diesen lediglich nicht mehr an. Wir müssen diese Funktionen überschreiben, um den Dialog zu zerstören. Dazu wird in beiden Funktionen die Elementfunktion `DestroyWindow` aufgerufen.

Außerdem muß die `OnOK`-Funktion des Dialogs überschrieben werden, um zu gewährleisten, daß jede Eingabe des Anwenders bearbeitet wird. Ein Aufruf von `DoModal` ist dazu nicht geeignet.

Nicht-modale Dialogobjekte werden auf dem Heap angelegt

*Listing 22.5:
Erstellen eines ungebundenen Dialogs in CMainFrame::OnAnsichtDialog*

Der Aufruf von `DestroyWindow` in `OnOK` und `OnCancel` zerstört das Dialogobjekt. Doch wie wird das C++-Objekt zerstört? Wir verwenden zu diesem Zweck eine Überschreibung der `PostNcDestroy`-Elementfunktion. In dieser Funktion löschen wir das von `CDialog` abgeleitete Objekt.

Zum Überschreiben der Standardimplementierungen von `OnOK`, `OnCancel` und `PostNcDestroy` müssen Sie diese zunächst der `CMyDialog`-Klasse über den Klassen-Assistenten hinzufügen. Öffnen Sie die Implementierungsdatei `MYDIALOG.CPP`, und verwenden Sie die Assistentenleiste, um die Funktionen zu definieren.

Die Implementierungen von `CMyDialog::OnOK`, `CMyDialog::OnCancel` und `CMyDialog::PostNcDestroy` sind in Listing 22.6 aufgeführt (`MYDIALOG.CPP`).

Listing 22.6:
Elementfunktionen in der unbundenen Version von CMyDialog

```
void CMyDialog::OnCancel()
{
    // TODO: Zusätzliche Freigabe hier einfügen
    CDialog::OnCancel();
    DestroyWindow();
}
void CMyDialog::OnOK()
{
    // TODO: Zusätzliche Überprüfung hier einfügen
    MessageBox(m_sEdit);
    CDialog::OnOK();
    DestroyWindow();
}
void CMyDialog::PostNcDestroy()
{
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen
    CDialog::PostNcDestroy();
    delete this;
}
```

Wenn Ihr nicht-modaler Dialog dem Rahmen, dem Dokument oder der Ansicht eine Nachricht senden muß, verwenden Sie dazu eine Elementfunktion. Die Dialogklasse erhält dazu eine Elementvariable, die einen Zeiger auf das Rahmen-, Dokument- oder Ansichtobjekt aufnimmt, das den Dialog erzeugt hat. Natürlich können Sie weitere Mechanismen zur Kommunikation zwischen nicht-modalen Dialogen und anderen Elementen Ihrer Anwendung verwenden, so daß der Dialog beispielsweise eine Nachricht an ein Fenster oder die Anwendung hinterlegt.

22.2 Dialog-Datenaustausch

In dem letzten Beispiel verwendeten wir den Dialog-Datenaustausch, um den Inhalt eines Eingabe-Steuerelementes einer `CString`-Elementvariable der Dialogklasse zuzuweisen. Der Dialog-Datenaustausch bietet überdies die Möglichkeit, einfache Variablen oder Steuerelement-Objekte den Steuerelementen eines Dialogs zuzuordnen.

Wenngleich der Dialog-Datenaustausch und die Dialog-Datenüberprüfung im Zusammenhang mit Dialogfeldern beschrieben wird, können diese Mechanismen nicht ausschließlich für Dialoge verwendet werden. Die genannten Elementfunktionen, wie zum Beispiel `DoDataExchange` und `UpdateData`, sind Elementfunktionen von `CWnd` und nicht von `CDialog`. `CFormView` und davon abgeleitete Klassen nutzen den Dialog-Datenaustausch ebenfalls.



22.2.1 Dialog-Datenaustausch

Der Dialog-Datenaustausch geschieht in der `DoDataExchange`-Elementfunktion der Dialogklasse. In dieser Funktion werden Aufrufe für alle Elementvariablen vorgenommen, deren Inhalt Steuerelementen zugewiesen ist. Die Aufrufe betreffen MFC-Funktionen, deren Bezeichnung mit `DDX_` beginnt. Diese Funktionen führen den eigentlichen Datenaustausch aus.

Um beispielsweise einen Datenaustausch zwischen einem Eingabe-Steuerelement und einer Elementvariablen vom Typ `CString` auszuführen, muß die folgende Funktion aufgerufen werden:

```
DDX_Text(pDX, IDC_EDIT, m_sEdit);
```

22.2.2 Dialog-Datenüberprüfung

Die MFC bietet zusätzlich zu dem einfachen Austausch von Daten zwischen Elementvariablen und Dialog-Steuerelementen einen Dialog-Datenüberprüfungsmechanismus. Die Dialog-Datenüberprüfung wird mit Hilfe von Funktionen ausgeführt, deren Namen mit `DDV_` beginnen. Diese Funktionen nehmen die entsprechenden Überprüfungen vor und geben eine Meldung aus, wenn ein Überprüfungsfehler auftritt. Außerdem wird in diesem Fall eine Ausnahme vom Typ `CUserException` erzeugt. Die Funktionen rufen überdies die `Fail`-Elementfunktion des `CDataExchange`-Objekts auf, das `DoDataExchange` übergeben wird. Dieses Objekt setzt den Fokus auf das fehlerhafte Steuerelement.

Ein Beispiel für eine Dialog-Datenüberprüfungsfunktion ist `DDV_MaxChars`, die die Länge einer Zeichenfolge in einem Eingabe-Steuerelement überprüft. Möchten Sie zum Beispiel ermitteln, ob die Länge einer derartigen Zeichenfolge nicht mehr als einhundert Zeichen beträgt, rufen Sie die Funktion wie folgt auf:

```
DDV_MaxChars(pDX, m_sEdit, 100);
```

Die Datenüberprüfung eines Steuerelements muß dem Aufruf der Funktion zum Datenaustausch mit diesem Steuerelement direkt folgen.

22.2.3 Verwenden einfacher Typen

Der Dialog-Datenaustausch mit einfachen Typen wird für Eingabe-Steuerelemente, Bildlaufleisten, Kontrollkästchen, Optionsschaltflächen, Listenfelder und Kombinationsfelder unterstützt.

Tabelle 22.1 führt die verschiedenen Typen auf, die für den Dialog-Datenaustausch mit Eingabe-Steuerelementen verwendet werden können.

*Tabelle 22.1:
Dialog-Daten-
austausch und
Dialog-Daten-
überprüfung*

Steuerelement	Datentyp	DDX-Funktion	DDV-Funktion
Eingabefeld	BYTE	DDX_Text	DDV_MinMaxByte
Eingabefeld	short	DDX_Text	DDV_MinMaxInt
Eingabefeld	int	DDX_Text	DDV_MinMaxInt
Eingabefeld	UINT	DDX_Text	DDV_MinMaxUnsigned
Eingabefeld	long	DDX_Text	DDV_MinMaxLong
Eingabefeld	DWORD	DDX_Text	DDV_MinMaxDWord
Eingabefeld	float	DDX_Text	DDV_MinMaxFloat
Eingabefeld	double	DDX_Text	DDV_MinMaxDouble
Eingabefeld	CString	DDX_Text	DDV_MaxChars
Eingabefeld	ColeDateTime	DDX_Text	
Eingabefeld	ColeCurrency	DDX_Text	
Kontrollkästchen	BOOL	DDX_Check	
Optionsschaltfläche	int	DDX_Radio	
Listenfeld	int	DDX_LBIndex	
Listenfeld	CString	DDX_LBString	
Listenfeld	Cstring	DDX_LBStringExact	
Kombinationsfeld	int	DDX_CBIndex	
Kombinationsfeld	CString	DDX_CBString	DDV_MaxChars
Kombinationsfeld	Cstring	DDX_CBStringExact	
Bildlaufleiste	int	DDX_Scroll	

Die MFC stellt zusätzliche Versionen der DDX-Funktionen zur Verfügung, um den Datenaustausch zwischen einem Dialog und den Datensätzen einer Datenbank zu ermöglichen. Die Bezeichnungen dieser Funktionen beginnen mit `DDX_Field`. Die Datenbankvariante von `DDX_Text` trägt somit den Namen `DDX_FieldText`.

Datenbankunterstützung

22.2.4 Verwenden von Steuerelement-Datentypen

Sie können einem Steuerelement nicht nur eine Elementvariable zuweisen, die den Wert des Steuerelements repräsentiert. Sie verfügen außerdem über die Möglichkeit, Elementvariablen zu definieren, die das Steuerelementobjekt selbst bilden. So können Sie beispielsweise einem Eingabe-Steuerelement eine Variable vom Typ `CEdit` zuweisen.

Der Dialog-Datenaustausch verwendet die `DDX_Control`-Funktion, um Daten zwischen einem Dialog-Steuerelement und einem von `CWnd` abgeleiteten Steuerelementobjekt auszutauschen.

Mit einem Steuerelementobjekt erhalten Sie eine bessere Kontrolle über den Aufbau und das Verhalten der Dialog-Steuerelemente. Dazu ein Beispiel. Da Steuerelementobjekte von `CWnd` abgeleitet werden, kann Ihre Anwendung die `CWnd`-Elementfunktionen nutzen, um die Größe und Position des Steuerelements zu verändern. Mit Hilfe des Steuerelementobjekts können außerdem Nachrichten an das Steuerelement gesendet werden. Des weiteren manipulieren diese Objekte das Verhalten bestimmter Steuerelementtypen. Sie können beispielsweise `CEdit::SetReadOnly` verwenden, um den Nur-Lese-Status eines Eingabe-Steuerelements zu verändern.

Eine große Anzahl verschiedener Steuerelementtypen (einschließlich die neuen allgemeinen Steuerelemente) verlangen ein Steuerelementobjekt für den Datenaustausch. Das Verwenden einfacher Datentypen wird nicht unterstützt.

22.2.5 Implementierung von benutzerdefinierten Datentypen

Der Dialog-Datenaustausch ist äußerst flexibel. Für benutzerdefinierte Datentypen ist dieser Mechanismus jedoch nicht geeignet. Der Klassen-Assistent bietet jedoch die Möglichkeit der Bearbeitung von benutzerdefinierten DDX- und DDV-Routinen.

Die Vorgehensweise zur Implementierung benutzerdefinierter DDX- und DDV-Unterstützung ist sehr zeitintensiv und lediglich für Datentypen zu empfehlen, die Sie häufig verwenden. (Weiterführende Informationen finden Sie in der MSDN-Dokumentation.)

22.3 Dialoge und Nachrichtenbearbeitung

Die von `CDialog` abgeleiteten Objekte können Nachrichten bearbeiten. In jedem Fall ist es erforderlich, Ihrer von `CDialog` abgeleiteten Klasse einen Nachrichtenbearbeiter hinzuzufügen.

Die Nachrichtenbearbeitung in einem Dialog unterscheidet sich nicht von der Nachrichtenbearbeitung in einer Ansicht oder einem Rahmenfenster. Nachrichtenbehandlungsfunktionen können der Nachrichten-zuordnungstabelle der Dialogklasse einfach mit Hilfe des Klassen-Assistenten hinzugefügt werden. In den ersten Beispielen geschah dies, indem der Klasse die überschriebenen Versionen von `OnOK` und `OnCancel` hinzugefügt wurden. Diese Elementfunktionen bearbeiten `WM_COMMAND`-Nachrichten. (Die dritte Funktion mit der Bezeichnung `PostNcDestroy` ist kein Nachrichtenbearbeiter. Sie wird von dem `OnNcDestroy`-Bearbeiter der `WM_NCDESTROY`-Nachricht aufgerufen.)

Die am häufigsten in Dialogklasse verwendeten Nachrichtenbearbeiter reagieren auf Nachrichten, die dem Dialogfenster von einem der darin enthaltenen Steuerelemente gesendet werden. Dazu zählen von einer Schaltfläche gesendete `BN_CLICKED`-Nachrichten, die verschiedenen `CBN_-`Nachrichten von einem Kombinationsfeld sowie `EN_-`Nachrichten, die ein Eingabe-Steuerelement versendet. Weitere Nachrichten, die von Dialogklassen bearbeitet werden, sind `WM_DRAWITEM` und `WM_MEASUREITEM` für besitzergezeichnete Steuerelemente.

Selbst gezeichnete Steuerelemente Die besitzergezeichneten Steuerelemente führen zu einer interessanten Frage. Sollten Sie die Nachrichten dieser Steuerelemente in Ihrer Dialogklasse bearbeiten, oder sollten Sie dem Steuerelement ein von der Steuerelementklasse abgeleitetes Objekt zuweisen und die Nachrichten darin bearbeiten? Verfügen Sie beispielsweise über eine besitzergezeichnete Schaltfläche, können Sie Ihrer Dialogklasse einen Bearbeiter für `WM_DRAWITEM`-Nachrichten hinzufügen oder eine Klasse von `CButton` ableiten und deren `DrawItem`-Elementfunktion überschreiben.

Leider gibt es keine definitive Antwort auf die eben gestellten Fragen. Sie sollten jedoch Ihre eigene Steuerelementklasse ableiten, wenn Sie das Steuerelement in mehreren Dialogen verwenden möchten. Andernfalls genügt die Bearbeitung von `WM_DRAWITEM` in der Dialogklasse (und ist dort überdies einfacher).

22.4 Registerdialoge

Registerdialoge sind mehrere in einem Dialog zusammengefaßte, überlappende Dialoge. Der Anwender selektiert einen dieser Dialoge, die auch als Registerseiten bezeichnet werden, indem er auf den entsprechenden Reiter des Registers klickt.

Die MFC unterstützt Registerdialoge durch vier Klassen: `CPropertySheet` (`CPropertySheetEx`) und `CPropertyPage` (`CPropertyPageEx`).

- ➡ `CPropertySheet` entspricht dem Registerdialog.
- ➡ Eine von `CPropertyPage` abgeleitete Klasse repräsentiert eine einzelne Registerseite in dem Registerdialog.

Um einen Registerdialog verwenden zu können, muß dieser zunächst entworfen und anschließend erstellt werden.

Das folgende Beispiel beschreibt die dazu notwendige Vorgehensweise. Sie verwenden dazu eine neue Anwendung mit der Bezeichnung PRP, die den in Abbildung 22.8 dargestellten Registerdialog anzeigen wird. PRP ist eine vom Anwendungsassistenten erzeugte SDI-Anwendung.



Abbildung 22.8:
Ein Beispiel für
einen Register-
dialog

22.4.1 Entwerfen von Registerdialogen

Das Entwerfen eines Registerdialogs entspricht weitgehend dem Entwurf eines gewöhnlichen Dialogs. Sie erstellen zunächst die Dialogvorlagenressource für die einzelnen Registerseiten, die der Registerdialog erhalten soll.

Beim Entwurf einer Dialogvorlagenressource für einen Registerdialog müssen einige Besonderheiten berücksichtigt werden:

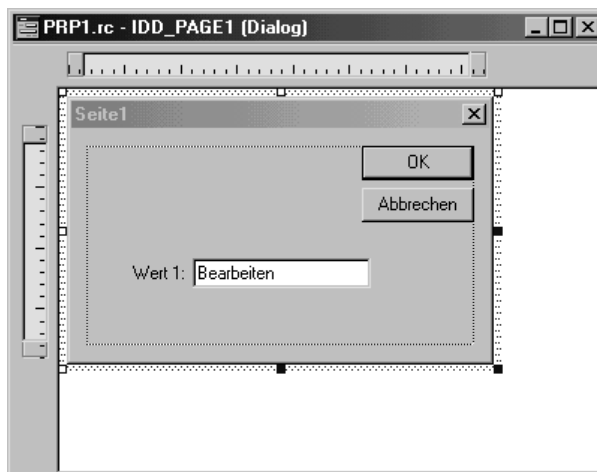
- ➔ Der besseren Übersicht halber sollte die Bezeichnung des Dialoges mit dem Titel übereinstimmen, der in dem Reiter der Registerdialogseite angezeigt werden soll.
- ➔ Das Format des Dialoges muß auf UNTERGEORDNET gesetzt werden.
- ➔ Wählen Sie einen dünnen Rand aus.
- ➔ Geben Sie an, daß der Dialog eine Titelleiste erhalten soll.
- ➔ Das erweiterte Format DEAKTIVIERT sollte ebenfalls gesetzt werden.

Wenngleich sich die einzelnen Registerseiten später im Registerdialog überlappen werden, müssen diese nicht in derselben Größe erstellt werden. Die MFC-Bibliothek setzt die Größe der Registerseiten auf die der größten Seite.

Registerseiten anlegen

In unserem Beispiel erstellen wir zwei Registerseiten für unsere Anwendung. In beiden Seiten wird ein Eingabefeld angezeigt. Die erste Registerseite mit dem Titel SEITE 1 ist in Abbildung 21.9 dargestellt.

Abbildung 22.9:
Entwerfen einer
Registerseite



1. Klicken Sie in der Ressourcen-Ansicht Ihres Projekts mit der rechten Maustaste auf den Eintrag **DIALOG** und wählen Sie in dem anschließend dargestellten Kontextmenü den Eintrag **DIALOG EINFÜGEN** aus.

2. Fügen Sie dem Dialog die geforderten Steuerelemente hinzu.
 3. Der Bezeichner der Dialogvorlagenressource sollte auf `IDD_PAGE1` gesetzt sein, der Bezeichner des Eingabefelds auf `IDC_EDIT1`.
 4. Überprüfen Sie, ob die Eigenschaften der Dialogvorlage korrekt eingerichtet sind.
- ➔ Sie setzen den Titel des Dialoges, indem Sie einen Doppelklick darauf ausführen. Daraufhin wird der in Abbildung 22.9 dargestellte Registerdialog `DIALOG EIGENSCHAFTEN` angezeigt.
 - ➔ Selektieren Sie bitte das Register `FORMATE` in dem Eigenschaften-Registerdialog, um das Format, den Rand und die Titelzeileneinstellung zu bestimmen (Abbildung 22.10).
 - ➔ In dem Register `WEITERE FORMATE` setzen Sie die Option `DEAKTIVIERT` (Abbildung 22.11).



Abbildung 22.10:
Die Bezeichnung des Registerdialogs



Abbildung 22.11:
Das Format des Registerdialogs

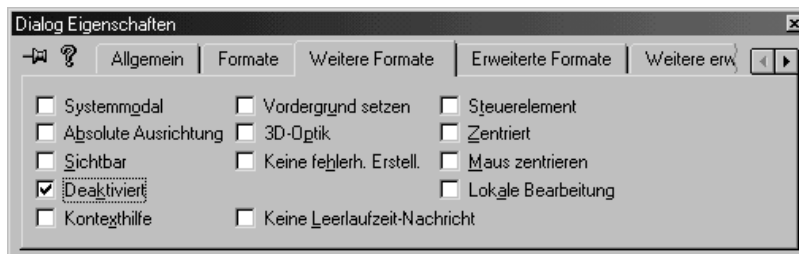


Abbildung 22.12:
Deaktivieren des Registerdialogs

Die zweite Registerseite unseres Beispiels ist mit der ersten Seite beinahe identisch.

1. Erzeugen Sie eine Kopie der ersten Registerseite als Vorlage für die Dialogressource der zweiten Seite: ID der Registerseite: `IDD_PAGE2`; ID des Eingabefelds: `IDC_EDIT2`. (Kopieren Sie über die Zwischenablage.)
2. Ändern Sie außerdem den Titel des Dialoges und den Text des statischen Steuerelements.



Das Verwenden der selben Bezeichner für Steuerelemente in separaten Registerseiten ist übrigens zulässig, da diese wie eigenständige Dialoge agieren. Die Autoren verwenden jedoch unterschiedliche Bezeichner, um die Möglichkeit einer Verwechslung auszuschließen.

Erzeugen der Registerseiten

1. Nachdem die Registerdialogressourcen entworfen wurden, rufen Sie den Klassen-Assistent auf und generieren Klassen, die die Registerseiten repräsentieren.
2. Starten Sie den Klassen-Assistent während der Fokus auf dem ersten Registerseitedialog liegt und dieser zur Bearbeitung geöffnet ist. Der Klassen-Assistent erkennt automatisch, daß der Dialogvorlage keine entsprechende Klasse zugewiesen ist und bietet Ihnen an, eine neue Klasse zu erstellen.
3. Bestimmen Sie in dem Dialog NEUE KLASSE eine Bezeichnung für die Klasse, die der Dialogvorlage entspricht (zum Beispiel `CMyPage1`). Achten Sie darauf, die Basisklasse `CPropertyPage` (und nicht die Voreinstellung `CDialog`) für diese neue Klasse auszuwählen. Erzeugen Sie die Klasse, nachdem alle erforderlichen Einstellungen vorgenommen wurden.
4. Sie sollten der neuen Klasse mit Hilfe des Klassenassistenten ebenfalls eine Elementvariable hinzufügen, über die auf das Textfeld-Steuerelement der Registerseite zugegriffen werden kann. Nennen Sie diese Variable `m_sEdit1`.
5. Wiederholen Sie diese Schritte für die zweite Registerseite. Nennen Sie die Klasse `CMyPage2` und die Elementvariable `m_sEdit2`.

Das Erstellen der Registerseiten ist nun abgeschlossen. Betrachten Sie kurz den vom Klassenassistenten erzeugten Programmcode. Die Deklaration von `CMyPage1` ist in Listing 22.7 aufgeführt. (Die Deklaration von `CMyPage2` ist beinahe identisch.)

```

class CMyPage1 : public CPropertyPage
{
    DECLARE_DYNCREATE(CMyPage1)

// Konstruktion
public:
    CMyPage1();
    ~CMyPage1();

// Dialogfelddaten
//{{AFX_DATA(CMyPage1)
enum { IDD = IDD_PAGE1 };
CString    m_sEdit1;
//}}AFX_DATA

// Überschreibungen
// Der Klassen-Assistent generiert virtuelle
// Funktionsüberschreibungen
//{{AFX_VIRTUAL(CMyPage1)
protected:
// DDX/DDV-Unterstützung
    virtual void DoDataExchange(CDataExchange* pDX);
//}}AFX_VIRTUAL

// Implementierung
protected:
// Generierte Nachrichtenzuordnungsfunktionen
//{{AFX_MSG(CMyPage1)
// HINWEIS: Der Klassen-Assistent fügt hier
// Member-Funktionen ein
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
}

```

Listing 22.7:
CMyPage1-
Deklaration

Wie Sie sehen, besteht kaum ein Unterschied zwischen dieser Deklaration und der Deklaration, die von dem Klassenassistenten für eine von `CDialog` abgeleitete Dialogklasse generiert wird. Die von `CPropertyPage` abgeleiteten Klassen können ebenso wie `CDialog`-Klassen die Funktionen zum Dialog-Datenaustausch nutzen.

Die Implementierung der `CMyPage1`-Elementfunktionen (Listing 22.8) unterscheidet sich unwesentlich von der Implementierung ähnlicher Funktionen in einer `CDialog`-Klasse. Der einzige Unterschied besteht darin, daß diese Klasse mit den Makros `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE` als dynamisch erstellbar deklariert wurde.

```

IMPLEMENT_DYNCREATE(CMyPage1, CPropertyPage)

CMyPage1::CMyPage1() : CPropertyPage(CMyPage1::IDD)
{
    //{{AFX_DATA_INIT(CMyPage1)
    m_sEdit1 = _T("");
    //}}AFX_DATA_INIT
}

CMyPage1::~CMyPage1()
{
}

void CMyPage1::DoDataExchange(CDataExchange* pDX)

```

Listing 22.8:
CMyPage1-
Implementierung


```

{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMyPage1)
    DDX_Text(pDX, IDC_EDIT1, m_sEdit1);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMyPage1, CPropertyPage)
    //{{AFX_MSG_MAP(CMyPage1)
    // HINWEIS: Der Klassen-Assistent fügt hier
    // Zuordnungsmakros für Nachrichten ein
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

Die Implementierung von `CMyPage2` entspricht der von `CMyPage1`.

22.4.2 Hinzufügen eines Registerdialogobjekts

Nachdem die Registerseiten erstellt wurden, erstellen Sie nun den Registerdialog. Dieser wird aufgerufen, wenn der Anwender den neuen Eintrag `EIGENSCHAFTSDIALOG` aus dem Menü `ANSICHT` der Anwendung auswählt.

1. Fügen Sie dem Menü diesen Eintrag mit dem Ressourceneditor hinzu (siehe Kapitel 22.1.5).
2. Starten Sie anschließend den Klassen-Assistenten, um der `CMainFrame`-Klasse die entsprechende Elementfunktion `CMainFrame::OnAnsichtEigenschaftsdialo` hinzuzufügen.

In dieser Elementfunktion wird zunächst ein Registerdialogobjekt erstellt. Im Anschluß daran müssen dem Objekt die Registerseiten hinzugefügt werden. Dies geschieht mit Hilfe der `AddPage`-Elementfunktion. Schließlich wird der Registerdialog mit der `DoModal`-Elementfunktion aufgerufen.

Listing 22.9 enthält die Implementierung von `CMainFrame::OnAnsichtEigenschaftsdialo`, die all diese Aufgaben ausführt.

Listing 22.9:
Die `CMainFrame::OnAnsichtPropertySheet`-Funktion

```

void CMainFrame::OnAnsichtPropertySheet()
{
    // Zu erledigen: Fügen Sie den Befehlsbearbeiter hier ein
    CPropertySheet myPropSheet;
    CMyPage1 myPage1;
    CMyPage2 myPage2;
    myPage1.m_sEdit1 = "First";
    myPage2.m_sEdit2 = "Second";
    myPropSheet.AddPage(&myPage1);
    myPropSheet.AddPage(&myPage2);
    myPropSheet.DoModal();
}

```

Vergessen Sie nicht, die Header-Dateien *MYPAGE1.H* und *MYPAGE2.H* in *MAINFRM.CPP* einzubinden. Andernfalls können Sie keine Objekte vom Typ *CMyPage1* oder *CMyPage2* deklarieren, so daß die Funktion in Listing 22.9 nicht kompiliert wird.

Die Anwendung kann jetzt kompiliert und gestartet werden.

Obwohl wir in diesem Beispiel keinen Gebrauch von den Elementvariablen der Registerseiten machen, nachdem der Registerdialog geschlossen wurde, können wir über die Registerseitenobjekte *myPage1* und *myPage2* darauf zugreifen.

22.4.3 CPropertyPage-Elementfunktionen

Das Beispiel nutzt kaum die erweiterten Möglichkeiten der *CPropertyPage*-Klasse.

In einer echten Anwendung möchten Sie beispielsweise die *CancelToClose*-Elementfunktion überschreiben, wenn eine Änderung in der Registerseite vorgenommen wird. Diese Funktion führt dazu, daß der Registerdialog nach Betätigung der Schaltfläche OK geschlossen und der Zugriff auf die Schaltfläche ABBRECHEN gesperrt wird. Die Funktion sollte nach einer unwiderruflichen Änderung in einer Registerseite verwendet werden.

Eine weitere häufig benutzte Registerseitenfunktion ist *SetModified*. Diese Funktion gibt den Zugriff auf die Schaltfläche ÜBERNEHMEN in dem Registerdialog frei.

Andere überschreibbare Registerseitenfunktionen sind

- ➡ *OnOK* (wird nach Betätigung der Schaltflächen OK, ÜBERNEHMEN oder SCHLIESSEN aufgerufen),
- ➡ *OnCancel* (wird nach Betätigung der Schaltfläche ABBRECHEN aufgerufen) und
- ➡ *OnApply* (wird nach Betätigung der Schaltfläche ÜBERNEHMEN aufgerufen).

Registerdialoge werden ebenfalls dazu verwendet, das Verhalten eines Assistenten zu implementieren. Dieses Verhalten entspricht dem der Assistenten, die einige Microsoft-Anwendungen zur Verfügung stellen. Sie aktivieren den Assistentenmodus, indem Sie die *SetWizardMode*-Funktion des Registerdialogs aufrufen. Überschreiben Sie in den Registerseiten die Elementfunktionen *OnWizardBack*, *OnWizardNext* und *OnWizardFinish*.

22.4.4 Nicht-modale Registerdialoge

Verwenden Sie die `DoModal`-Elementfunktion, wird der Registerdialog modal dargestellt. Sie können ihn jedoch ebenso nicht-modal anzeigen lassen.

Dazu müssen Sie zunächst Ihre eigene Registerdialogklasse ableiten. Dies ist wichtig, da Sie die `PostNcDestroy`-Elementfunktion überschreiben müssen, um zu gewährleisten, daß die Objekte dieser Klasse zerstört werden, wenn der nicht-modale Dialog geschlossen wird.

Die neue Registerdialogklasse kann mit dem Klassenassistenten erstellt werden. Erzeugen Sie eine neue von `CPropertySheet` abgeleitete Klasse, und bezeichnen Sie diese mit `CMySheet`. Fügen Sie der Klasse im Klassenassistenten die `PostNcDestroy`-Elementfunktion hinzu.

Die von dem Klassenassistenten generierte Deklaration der Klasse `CMySheet` (in `MYSHEET.H`) ist in Listing 22.10 aufgeführt.

Listing 22.10:
Die CMySheet-
Deklaration

```
class CMySheet : public CPropertySheet
{
    DECLARE_DYNAMIC(CMySheet)

// Konstruktion
public:
    CMySheet(UINT nIDCaption, CWnd* pParentWnd = NULL,
        UINT iSelectPage = 0);
    CMySheet(LPCTSTR pszCaption, CWnd* pParentWnd = NULL,
        UINT iSelectPage = 0);

// Attribute
public:

// Operationen
public:

// Überschreibungen
// Vom Klassen-Assistenten generierte virtuelle
// Funktionsüberschreibungen
//{{AFX_VIRTUAL(CMySheet)
protected:
    virtual void PostNcDestroy();
//}}AFX_VIRTUAL

// Implementierung
public:
    virtual ~CMySheet();

// Generierte Nachrichtenzuordnungsfunktionen
protected:
    ///{AFX_MSG(CMySheet)
    // HINWEIS - Der Klassen-Assistent fügt hier
    // Member-Funktionen ein und entfernt diese.
    ///{AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

In der Implementierungsdatei *MYSHEET.CPP* muß die `PostNcDestroy`-Elementfunktion modifiziert werden, um nicht nur das Registerdialogobjekt, sondern ebenfalls die entsprechenden Registerseiten zu zerstören. Die Implementierung dieser Funktion ist zusammen mit anderen vom Klassen-Assistenten zur Verfügung gestellten Funktionsimplementierungen für die *CMySheet*-Klasse in Listing 22.11 aufgeführt.

```

////////////////////
// CMySheet

IMPLEMENT_DYNAMIC(CMySheet, CPropertySheet)

CMySheet::CMySheet(UINT nIDCaption, CWnd* pParentWnd,
                  UINT iSelectPage)
    :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
}

CMySheet::CMySheet(LPCTSTR pszCaption, CWnd* pParentWnd,
                  UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
}

CMySheet::~CMySheet()
{
}

BEGIN_MESSAGE_MAP(CMySheet, CPropertySheet)
    //{AFX_MSG_MAP(CMySheet)
    // HINWEIS - Der Klassen-Assistent fügt hier
    // Zuordnungsmakros ein und entfernt diese.
    //}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////
Behandlungsroutinen für Nachrichten CMySheet

void CMySheet::PostNcDestroy()
{
    // TODO: Speziellen Code hier einfügen und/oder
    // Basisklasse aufrufen

    CPropertySheet::PostNcDestroy();
    for (int i = 0; i < GetPageCount(); i++)
        delete GetPage(i);
    delete this;
}

```

*Listing 22.11:
Die CMySheet-
Implementierung*

Ein nicht-modaler Registerdialog verfügt nicht automatisch über die Schaltflächen OK, ABBRECHEN und ÜBERNEHMEN. Diese müssen dem Dialog manuell hinzugefügt werden. Natürlich kann ein nicht-modaler Registerdialog über das Systemmenü geschlossen werden.

Um den nicht-modalen Registerdialog aufzurufen, müssen Sie die `On-AnsichtPropertySheet`-Elementfunktion in der *CMainFrame*-Klasse modifizieren, da `DoModal` zur Anzeige des Dialogs nicht verwendet werden kann. Erstellen Sie weder den Registerdialog noch dessen Registersei-

ten auf dem Stack, da der Dialog und die Seiten nicht zerstört werden sollen, wenn die `OnAnsichtPropertySheet`-Funktion beendet ist. Die neue `OnAnsichtPropertySheet`-Funktion ist in Listing 22.12 aufgeführt.

Listing 22.12:
*Aufruf eines
nicht-modalen
Registerdialogs*

```
void CMainFrame::OnAnsichtPropertySheet()
{
    // Zu erledigen: Fügen Sie den Befehlsbearbeiter hier ein
    CMySheet *pMyPropSheet;
    CMyPage1 *pMyPage1;
    CMyPage2 *pMyPage2;
    pMyPropSheet = new CMySheet("");
    pMyPage1 = new CMyPage1;
    pMyPage2 = new CMyPage2;
    pMyPage1->m_sEdit1 = "Erste Seite";
    pMyPage2->m_sEdit2 = "Zweite Seite";
    pMyPropSheet->AddPage(pMyPage1);
    pMyPropSheet->AddPage(pMyPage2);
    pMyPropSheet->Create(this);
}
```

Damit die neue Version von `CMainFrame::OnAnsichtPropertySheet` korrekt kompiliert wird, müssen Sie die Datei `MYSHEET.H` in `MAIN-FRM.CPP` einbinden. Andernfalls schlägt der Versuch fehl, ein Objekt vom Typ `CMySheet` zu deklarieren.

22.5 Zusammenfassung

In der MFC werden Dialoge durch Klassen repräsentiert, die sich von `CDialog` ableiten.

Um einen Dialog zu erstellen, der in einer MFC-Anwendung verwendet werden soll, gehen Sie wie folgt vor:

- ➔ Erstellen Sie die Dialogvorlagenressource.
- ➔ Rufen Sie den Klassen-Assistenten auf, und erstellen Sie die Dialogklasse zur Ressource.
- ➔ Fügen Sie der Klasse mit Hilfe des Klassen-Assistenten Elementvariablen für die Steuerelemente hinzu.
- ➔ Verwenden Sie den Klassen-Assistenten, um der Klasse Nachrichtenbearbeiter hinzuzufügen, sofern erforderlich.
- ➔ Fügen Sie Ihrer Anwendung Programmcode hinzu, der ein Dialogobjekt erstellt, aufruft (mit der `DoModal`-Elementfunktion) und das Ergebnis des Aufrufs ermittelt.

MFC-Anwendungen können außerdem nicht-modale Dialoge darstellen. Diese Dialoge werden auf eine andere Weise als gewöhnliche Dialoge erstellt. Die Konstruktorfunktion Ihrer Dialogklasse ruft dazu die `Create-Elementfunktion` auf.

Überdies muß eine eigene Version des Konstruktors implementiert werden. Der nicht-modale Dialog muß explizit mit einem Aufruf der Funktion `ShowWindow` aufgerufen werden.

Klassen, die sich auf nicht-modale Dialoge beziehen, sollten die `OnOK`- und `OnCancel`-Elementfunktion überschreiben und die `DestroyWindow`-Elementfunktion aufrufen. Das Überschreiben von `PostNcDestroy` und das Zerstören des C++-Objekts (zum Beispiel mit `delete this`) ist ebenfalls erforderlich.

Die Steuerelemente eines Dialoges werden häufig durch Elementvariablen in der entsprechenden Dialogklasse repräsentiert. Der Dialog-Datenaustausch ermöglicht den Datenaustausch zwischen den Steuerelementen in dem Dialogobjekt und den Elementvariablen in der Dialogklasse. Dieser Mechanismus stellt eine einfache Methode für die Zuweisung der Elementvariablen zur Verfügung. Elementvariablen können von einem einfachen Werttyp sein oder Steuerelementobjekte repräsentieren. So ist es möglich, eine Elementvariable eines einfachen Typs zur Ermittlung des Werts eines Steuerelements zu verwenden, während ein Steuerelementobjekt alle Aspekte des Steuerelements verwaltet. Der Dialog-Datenaustausch bietet außerdem die Möglichkeit zur Datenüberprüfung.

Der Dialog-Datenaustausch kann für häufig verwendete Typen, die nicht dem Standard entsprechen, erweitert werden. Dazu werden einem Projekt oder allen Projekten neue Datenaustausch- und Datenüberprüfungsroutinen hinzugefügt.

Registerdialoge bestehen aus mehreren überlappten Dialogen, die als Registerseiten bezeichnet werden. Der Anwender öffnet eine Registerseite, indem er auf den entsprechenden Reiter des Registerdialogs klickt.

Das Erstellen eines Registerdialoges geschieht in zwei Phasen. Zunächst werden die Registerseiten und anschließend das Registerdialogobjekt erstellt. Dem Objekt werden die Registerseiten hinzugefügt, so daß der Registerdialog im Anschluß daran angezeigt werden kann.

Nachfolgend ist die Vorgehensweise zum Erstellen eines Registerdialoges beschrieben:

- ➔ Erstellen der Dialogvorlagenressource für jede Registerseite. Selektieren Sie für diese Seiten das Format `UNTERGEORDNET`, einen dünnen Rahmen, eine Titelzeile und die Option `DEAKTIVIERT`. Setzen Sie die Beschriftung auf den Text, der in dem Reiter angezeigt werden soll.

- ➔ Rufen Sie den Klassen-Assistenten auf, und erstellen Sie eine von `CPropertyPage` abgeleitete Klasse, die die Dialogvorlagenressource repräsentiert.
- ➔ Fügen Sie der Dialogklasse mit Hilfe des Klassen-Assistenten Elementvariablen hinzu, über die auf die Steuerelemente der Registerseite zugegriffen wird.
- ➔ Fügen Sie der Klasse Nachrichtensarbeiter hinzu, sofern erforderlich.

Nachdem die Registerseite erstellt wurden, beginnen Sie mit der zweiten Phase:

- ➔ Erstellen Sie ein `CPropertySheet`-Objekt oder ein Objekt aus der von `CPropertySheet` abgeleiteten Klasse.
- ➔ Generieren Sie ein Registerseitenobjekt für jede Registerseite, die dem Registerdialog hinzugefügt werden soll.
- ➔ Fügen Sie dem Registerdialog die Registerseiten mit einem Aufruf von `AddPage` hinzu.
- ➔ Rufen Sie den Registerdialog mit `DoModal` auf.

Sie können ebenfalls nicht-modale Registerdialoge erzeugen. Leiten Sie dazu eine Klasse von `CPropertySheet` ab, und überschreiben Sie deren `PostNcDestroy`-Elementfunktion, um nicht nur das Registerdialogobjekt, sondern ebenfalls alle Registerseiten des Dialogs zu löschen. Der ungebundene Registerdialog wird mit der `Create`-Elementfunktion angezeigt.

MFC-Unterstützung für Standarddialoge und Standardsteuerelemente

Kapitel

23

Die MFC-Bibliothek stellt Ihnen eine umfangreiche Unterstützung von Standarddialogen und den neuen Windows-95/98-Standardsteuerelementen zur Verfügung.

Standarddialoge sind seit Version 3.1 ein Feature von Microsoft-Windows. Sie werden zum Öffnen und Schließen von Dateien, zur Auswahl von Farben und Schriftarten, zur Suche nach Text sowie zum Einrichten und Gebrauch des Druckers verwendet. Die Anwendung dieser Dialoge hat sich bisher kaum verändert. Ist eine umfassende Konfiguration der Dialoge nicht erforderlich, kann ein Standarddialog mit Hilfe einiger C-Strukturelemente aufgerufen werden. Die Auswahl des Anwenders wird ebenfalls über diese Elemente bearbeitet. Die MFC-Bibliothek automatisiert das Erstellen der notwendigen Strukturen. Die Standarddialogklassen der MFC vereinfachen das Einrichten der Dialoge.

Windows bietet ebenfalls eine Unterstützung für einige Standardsteuerelemente, wie zum Beispiel Eingabe-Steuerelemente, statische Steuerelemente oder Schaltflächen. Die neuen Windows-95/98-Standardsteuerelemente repräsentieren leistungsfähige Hilfsmittel zur Erweiterung des Erscheinungsbildes der Anwendungen. Die MFC-Version integriert diese Steuerelemente mit anderen MFC-Anwendungsrahmen-Features – sofern dies möglich ist.

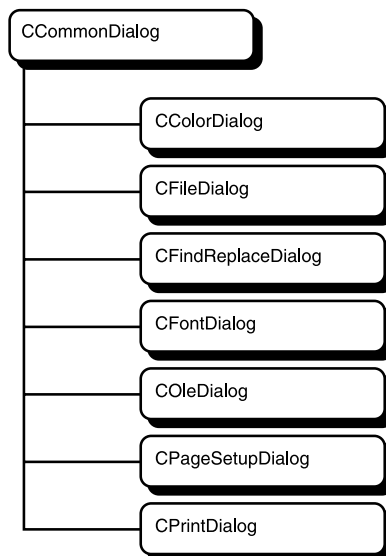
*Standarddialoge
gibt es seit
Windows 3.1*

23.1 Standarddialoge

CCommonDialog Die MFC-Unterstützung für Standarddialoge wird durch von *CCommonDialog* abgeleitete Klassen zur Verfügung gestellt. *CCommonDialog* selbst ist von *CDialog* abgeleitet. Sie leiten eine Klasse niemals direkt von *CCommonDialog* ab und erstellen auch kein *CCommonDialog*-Objekt. Statt dessen erstellen Sie ein Objekt der entsprechenden Dialogklasse. Sie können außerdem eine angepaßte Standarddialogvorlage benutzen und davon eine Klasse ableiten.

Die Klassen, die in der MFC der Unterstützung von Standarddialogen dienen, sind in Abbildung 23.1 aufgeführt.

Abbildung 23.1:
Standarddialoge in der MFC



Einige Standarddialoge bieten erweiterte Fehlerinformationen an. Diese werden mit *CommDlgExtendedError* ermittelt.

Die folgenden Abschnitte beschreiben die von der MFC-Bibliothek zur Verfügung gestellten Standarddialoge. Dazu verwenden wir ein vom Anwendungsassistenten generiertes Anwendungsgerüst. Die Anwendung CDLG kann als eine SDI-Anwendung mit den Voreinstellungen des Anwendungsassistenten erstellt werden. Wir verwenden das Ansichtsmenü der Anwendung, um zusätzliche Einträge darin aufzunehmen, die unterschiedliche Dialoge aufrufen. Das Erzeugen eines neuen Eintrags wird lediglich einmal erläutert, um Wiederholungen zu vermeiden.

23.1.1 CColorDialog

Die `CColorDialog`-Klasse unterstützt den Windows-Standarddialog zur Auswahl einer Farbe.

Erstellen Sie ein `CColorDialog`-Objekt, und rufen Sie dessen `DoModal`-Funktion auf, um den Dialog anzeigen zu lassen.

Sie können den Dialog initialisieren, bevor Sie `DoModal` aufrufen, indem Sie die `CColorDialog`-Elementvariable `m_cc` einrichten. Diese Variable ist eine Win32-Struktur vom Typ `CHOOSECOLOR`.

Nachdem der Dialog geschlossen wurde, kann die Farbauswahl des Anwenders mit der `GetColor`-Elementfunktion ermittelt werden.

Listing 23.1 zeigt eine mögliche Implementierung des Farbauswahldialogs unter Verwendung der `CColorDialog`-Klasse.

```
void CMainFrame::OnViewColordialog()
{
    // TODO: Nachrichtensarbeiter hier einfügen
    CColorDialog dlg;
    if (dlg.DoModal() == IDOK)
    {
        TCHAR temp[100];
        wsprintf(temp, _T("Color selected: %#8.8X"),
            dlg.GetColor());
        AfxMessageBox(temp);
    }
}
```

*Listing 23.1:
Verwenden von
CColorDialog in
CDLG*

Natürlich können Sie den Dialog zur Farbauswahl konfigurieren. Leiten Sie dazu Ihre eigene Klasse von `CColorDialog` ab, und verwenden Sie eine benutzerdefinierte Dialogvorlage. Sie finden diese Vorlage in der Datei `COLOR.DLG`, die in dem VisualC++-Verzeichnis `\INCLUDE` enthalten ist. Wenn Sie der Vorlage neue Steuerelemente hinzufügen, werden Sie vermutlich auch eine Nachrichtentabelle verwenden, um die Nachrichten der Steuerelemente bearbeiten zu können.

23.1.2 CFileDialog

Die `CFileDialog`-Klasse unterstützt die Windows-Dialoge **ÖFFNEN** und **SPEICHERN UNTER**.

Sie erstellen zunächst ein `CFileDialog`-Objekt, bevor Sie einen Dateidialog in einer MFC-Anwendung benutzen können. Der Konstruktor benötigt einige Parameter. Der erste Parameter ist ein obligatorischer Boolescher Wert, der bestimmt, welcher der beiden Dialoge **ÖFFNEN** und **SPEICHERN UNTER** verwendet werden soll.

Nachdem der Dialog erstellt wurde, wird dessen `DoModal`-Elementfunktion aufgerufen, die den Dialog anzeigt.

Die meisten Initialisierungseinstellungen des Dialoges werden in dem `CFileDialog`-Konstruktor vorgenommen. Sie können außerdem die Werte der `m_ofn`-Struktur setzen, die eine Win32-`OPENFILENAME`-Struktur ist.

Wird der Dialog geschlossen, ermitteln Sie den vom Anwender angegebenen Dateinamen mit Hilfe der `GetPathName`-Elementfunktion. Ist die Auswahl mehrerer Dateinamen zulässig, erhalten Sie einen Zähler für die Dateinamensliste mit `GetStartPosition`. Dateinamen können in dieser Liste mit einem wiederholten Aufruf von `GetNextPathName` ermittelt werden. Der Funktion muß dazu der Zähler übergeben werden.

Listing 23.2 demonstriert die Anwendung des Dialogs ÖFFNEN.

Listing 23.2:
Verwenden von
CFileDialog in
CDLG

```
void CMainFrame::OnViewFileDialog()
{
    // TODO: Nachrichtensarbeiter hier einfügen
    CFileDialog dlg(TRUE);
    if (dlg.DoModal() == IDOK)
    {
        CString temp = _T("File selected: ") + dlg.GetPathName();
        AfxMessageBox(temp);
    }
}
```

Leiten Sie eine Klasse von `CFileDialog` ab, können Sie eine angepaßte Bearbeitung für die Verletzung des gemeinsamen Zugriffs, für fehlerhafte Dateinamen und für die Benachrichtigungen über Veränderungen von Listenelementen implementieren.

Der Dateidialog kann ebenfalls konfiguriert werden. Leiten Sie dazu Ihre eigene Klasse von `CFileDialog` ab, und verwenden Sie eine benutzerdefinierte Dialogvorlage. Sie finden diese Vorlage in der Datei `FILEOPEN.DLG`, die in dem VisualC++-Verzeichnis `\INCLUDE` enthalten ist. Fügen Sie der Klasse Ihre Nachrichtentabelle hinzu, um die Nachrichten der Steuerelemente bearbeiten zu können.

23.1.3 CFindReplaceDialog

Die `CFindReplaceDialog`-Klasse unterstützt die Verwendung der Windows-Dialoge SUCHEN und ERSETZEN in MFC-Anwendungen.

Der Einsatz der Dialoge SUCHEN und ERSETZEN unterscheidet sich wesentlich von dem anderer Dialoge. Während alle anderen Standarddialoge modal angezeigt werden, sind die Dialoge SUCHEN und ERSETZEN nicht modal und müssen entsprechend erstellt und verwendet werden.

Erstellen Sie zunächst ein `CFindReplaceDialog`-Objekt. Dieses Objekt kann nicht auf dem Stack erzeugt werden. Verwenden Sie statt dessen den `new`-Operator, so daß der Dialog auch dann noch angezeigt wird, wenn die Funktion, in der dieser generiert wurde, bereits beendet ist.

Die Dialoge SUCHEN und ERSETZEN kommunizieren über spezielle Nachrichten mit den Fenstern, die diese Dialoge besitzen. Damit Sie diese Nachrichten nutzen können, müssen Sie die `::RegisterWindowMessage`-Funktion verwenden. Der während der Registrierung der Suchen- und Ersetzen-Nachrichten ermittelte Wert kann mit dem `ON_REGISTERED_MESSAGE`-Makro in einer Fensternachrichtentabelle verwendet werden.

Der eigentliche Dialog wird mit einem Aufruf der `CreateElementfunktion` erstellt. Beachten Sie bitte, daß Sie in dem Aufruf von `CFindReplaceDialog::Create` das Fenster angeben müssen, das Nachrichten von dem Dialog empfangen soll.

Listing 23.3 zeigt, wie die Dialoge SUCHEN und ERSETZEN in der CDLG-Anwendung implementiert werden. Der in dem Listing aufgeführte Programmcode ist in der Datei `MAINFRM.CPP` enthalten.

```
// MainFrm.cpp : Implementierung der CMainFrame-Klasse
//
...
static UINT WM_FINDREPLACE = RegisterWindowMessage(FINDMSGSTRING);
//////////////////////////////////// CMainFrame
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ...
    //}}AFX_MSG_MAP
    ON_REGISTERED_MESSAGE(WM_FINDREPLACE, OnFindReplace)
END_MESSAGE_MAP()
...
void CMainFrame::OnViewFindreplacedialog()
{
    // TODO: Nachrichtenbearbeiter hier einfügen
    CFindReplaceDialog *pDlg = new CFindReplaceDialog;
    pDlg->Create(TRUE, _T("Findme"), NULL, FR_DOWN, this);
}
LRESULT CMainFrame::OnFindReplace(WPARAM wParam, LPARAM lParam)
{
    if (((LPFINDREPLACE)lParam)->Flags & FR_FINDNEXT)
    {
        CString temp = _T("Search string: ");
        temp = temp + ((LPFINDREPLACE)lParam)->lpstrFindWhat;
        AfxMessageBox(temp);
    }
    return 0;
}
```

*Listing 23.3:
Verwenden von
CFindReplace-
Dialog in CDLG*

Die Funktion `OnFindReplace` muß ebenfalls in `MAINFRM.H` deklariert werden. Fügen Sie dazu am Ende des Listings, hinter der vom Klassenassistenten generierten Nachrichtentabelle und vor dem Aufruf des Makros `DECLARE_MESSAGE_MAP` die folgende Zeile ein:

```
afx_msg LRESULT OnFindReplace(WPARAM wParam, LPARAM lParam);
```

Die Konfiguration der Dialoge SUCHEN und ERSETZEN erfordert das Ableiten einer Klasse von `CFinReplaceDialog`, das Generieren einer benutzerdefinierten Dialogvorlage und das Verwenden einer Nachrichtentabelle, um die Nachrichten der von Ihnen hinzugefügten Steuerelemente bearbeiten zu können. Die Dialogvorlage befindet sich in der Datei *findtext.dlg* im VisualC++-Verzeichnis `\INCLUDE`.

23.1.4 CFontDialog

Die `CFontDialog`-Klasse unterstützt die Auswahl einer Schriftart in MFC-Anwendungen mit Hilfe des entsprechenden Standarddialogs.

Erstellen Sie ein `CFontDialog`-Objekt, und rufen Sie dessen `DoModal`-Elementfunktion auf, um den Dialog zur Auswahl einer Schriftart verwenden zu können.

Sie können den Dialog initialisieren, bevor Sie `DoModal` aufrufen, indem Sie den Wert der `CFontDialog`-Elementvariable `m_cf` setzen. Dieses Element ist eine Struktur vom Typ `CHOOSEFONT`.

Wird der Dialog geschlossen, können Sie mit Hilfe der `CFontDialog`-Elementfunktionen, wie zum Beispiel `GetFaceName`, `GetSize` oder `GetColor`, die von dem Anwender ausgewählte Schriftart ermitteln. Alternativ dazu können Sie die Werte der `m_cf`-Struktur überprüfen. Besonders interessant ist `m_cf.lpLogFont`. Nachdem `CFontDialog::DoModal` beendet ist, verweist dieser Zeiger auf eine `LOGFONT`-Struktur, die in den folgenden Aufrufen von `::CreateFontIndirect` oder `CFont::CreateFontIndirect` verwendet werden kann, um eine logische Schriftart zu erzeugen.

`CFontDialog` wird in Listing 23.4 verwendet.

Listing 23.4:
CFontDialog in
CDLG

```
void CMainFrame::OnViewFontdialog()
{
    // TODO: Nachrichtensarbeiter hier einfügen
    CFontDialog dlg;
    if (dlg.DoModal() == IDOK)
    {
        CString temp = _T("Font selected: ") + dlg.GetFaceName();
        AfxMessageBox(temp);
    }
}
```

Der Dateidialog kann konfiguriert werden. Leiten Sie dazu Ihre eigene Klasse von `CFontDialog` ab, und verwenden Sie eine benutzerdefinierte Dialogvorlage. Sie finden diese Vorlage in der Datei *FONT.DLG*, die in dem VisualC++-Verzeichnis `\INCLUDE` enthalten ist. Fügen Sie der Klasse Ihre Nachrichtentabelle hinzu, um die Nachrichten der Steuerelemente bearbeiten zu können.

23.1.5 CPageSetupDialog

Der Dialog SEITE EINRICHTEN wird unter Windows 95 und Windows NT 3.51 dazu verwendet, die zu druckende Seite einzurichten. Dieser Dialog ersetzt den Dialog DRUCKER EINRICHTEN.

Sie nutzen diesen Dialog, indem Sie ein Objekt vom Typ `CPageSetupDialog` erstellen und dessen `DoModal`-Elementfunktion aufrufen.

Die Informationen über die Einstellungen des Anwenders werden mit einigen Elementfunktionen ermittelt, wie zum Beispiel `GetDeviceName`, `GetMargins` oder `GetPaperSize`. Alternativ dazu können Sie die `m_psd`-Elementvariable überprüfen (die vom Typ `PAGESETUPDLG` ist). Sie haben die Möglichkeit, die Elemente dieser Struktur vor dem Aufruf von `DoModal` zu modifizieren, um das Standardverhalten zu überschreiben.

Listing 23.5 führt ein Beispiel auf, das den Dialog SEITE EINRICHTEN verwendet.

```
void CMainFrame::OnViewPagesetupdialog()
{
    // TODO: Nachrichtenerarbeiter hier einfügen
    CPageSetupDialog dlg;
    if (dlg.DoModal() == IDOK)
    {
        char temp[100];
        sprintf(temp, "Paper size: %g\" x %g\".",
            (double)dlg.m_psd.ptPaperSize.x / 1000.0,
            (double)dlg.m_psd.ptPaperSize.y / 1000.0);
        AfxMessageBox(temp);
    }
}
```

*Listing 23.5:
Verwenden von
CPageSetupDia-
log in CDLG*

Natürlich können Sie auch hier einen angepaßten Dialog erstellen. Leiten Sie dazu eine Klasse von `CPageSetupDialog` ab, verwenden Sie Ihre eigene Dialogvorlage und stellen Sie eine Nachrichtentabelle zur Verfügung, um die Nachrichten der von Ihnen hinzugefügten Steuerelemente bearbeiten zu können. Die originale Dialogvorlage für den Dialog SEITE EINRICHTEN ist in der Datei `PRNSETUP.DLG` im VisualC++-Verzeichnis `\INCLUDE` enthalten.

Leiten Sie Ihre eigene Klasse von `CPageSetupDialog` ab, können Sie dessen Elementfunktionen `OnDrawPage` und `PreDrawPage` überschreiben, um eine angepaßte Seitenansicht der zu druckenden Seite zu zeichnen.

23.1.6 CPrintDialog

Obwohl die von dem Anwendungsassistenten generierten Anwendungsgerüste die Möglichkeit zum Einrichten einer Seite und zum Drucken zur Verfügung stellen, ist es bisweilen erforderlich, die Dialoge DRUCKEN, DRUCKER EINRICHTEN und SEITE EINRICHTEN zu verwenden.

Die `CPrintDialog`-Klasse unterstützt die Dialoge `DRUCKEN` und `DRUCKER EINRICHTEN` in MFC-Anwendungen. Anwendungen, die für Windows 95 oder Windows NT ab Version 3.51 geschrieben wurden, sollten jedoch von einem Gebrauch des Dialoges `DRUCKER EINRICHTEN` absehen. Statt dessen sollte der neue Dialog `SEITE EINRICHTEN` verwendet werden.

Erstellen Sie ein `CPrintDialog`-Objekt und rufen Sie dessen `DoModal`-Elementfunktion auf, um einen `DRUCKEN`-Dialog zu generieren. Der erste Parameter des Konstruktors ist ein Boolescher Wert, der bestimmt, welcher der Dialoge `DRUCKEN` und `DRUCKER EINRICHTEN` angezeigt werden soll. Setzen Sie den Parameter auf `TRUE`, wenn Sie den Dialog `DRUCKER EINRICHTEN` verwenden möchten.

Sie initialisieren den Dialog, indem Sie die Werte der `m_pd`-Elementstruktur setzen. Diese Struktur ist vom Typ `PRINTDLG`. Die Einstellungen des Anwenders können mit Hilfe einiger Elementfunktionen ermittelt werden, wie zum Beispiel `GetDeviceName` oder `GetPrinterDC`.

Die `CPrintDialog`-Klasse kann ebenfalls zur Ermittlung eines Drucker-Gerätekontexts verwendet werden, der sich auf den Standarddrucker bezieht. Verwenden Sie dazu die `CreatePrinterDC`-Funktion. Beachten Sie bitte, daß diese Funktion den zuvor in `m_pd.hDC` gespeicherten Gerätekontext-Handle überschreibt, ohne das Gerätekontext-Objekt zu löschen.

Das Gerätekontext-Objekt muß nicht nur dann gelöscht werden, wenn es mit einem Aufruf von `CreatePrinterDC` erstellt wurde, sondern auch in dem Fall, daß `CPrintDialog` für einen `DRUCKEN`-Dialog erzeugt wurde (wozu der erste Parameter des Konstruktors auf `FALSE` gesetzt wird). Anwendungen müssen außerdem jede `DEVMODE`- und `DEVNAMES`-Struktur löschen, die von `CPrintDialog` zur Verfügung gestellt wird. Dies geschieht mit einem Aufruf der Windows-Funktion `GlobalFree` für die Handle `m_pd.hDevMode` und `m_pd.hDevNames`.

Listing 23.6 zeigt, wie ein `DRUCKEN`-Dialog in unserer Anwendung `CDLG` erstellt wird.

Listing 23.6:
CPrintDialog in
CDLG

```
void CMainFrame::OnViewPrintdialog()
{
    // TODO: Nachrichtensarbeiter hier einfügen
    CPrintDialog dlg(FALSE);
    if (dlg.DoModal() == IDOK)
    {
        CString temp = "Device selected: " + dlg.GetDeviceName();
        AfxMessageBox(temp);
    }
    GlobalFree(dlg.m_pd.hDevMode);
    GlobalFree(dlg.m_pd.hDevNames);
    DeleteDC(dlg.GetPrinterDC());
}
```

Sie konfigurieren die Dialoge DRUCKEN und DRUCKER EINRICHTEN, indem Sie eine Klasse von `CPrintDialog` ableiten, eine benutzerdefinierte Dialogvorlage erstellen und der Klasse eine Nachrichtentabelle hinzufügen, um die Nachrichten der Steuerelemente zu bearbeiten. Verwenden Sie eine der Dialogvorlagen in der Datei `PRNSETUP.DLG` im Visual C++-Verzeichnis `\INCLUDE` als Basis für jede neue Dialogvorlage, die Sie generieren.

Beachten Sie, daß Sie möglicherweise zwei separate Klassen von `CPrintDialog` ableiten müssen, wenn Sie sowohl den Dialog DRUCKEN als auch den Dialog DRUCKER EINRICHTEN anpassen möchten.

23.1.7 COleDialog

OLE unterstützt mehrere Standarddialoge. Diese Dialoge werden von der MFC-Bibliothek über verschiedene Dialogklassen zur Verfügung gestellt. Die Klasse `COleDialog` dient als Basisklasse für alle OLE-Standarddialogklassen.

Abbildung 23.2 führt die unterschiedlichen OLE-Standarddialoge auf.

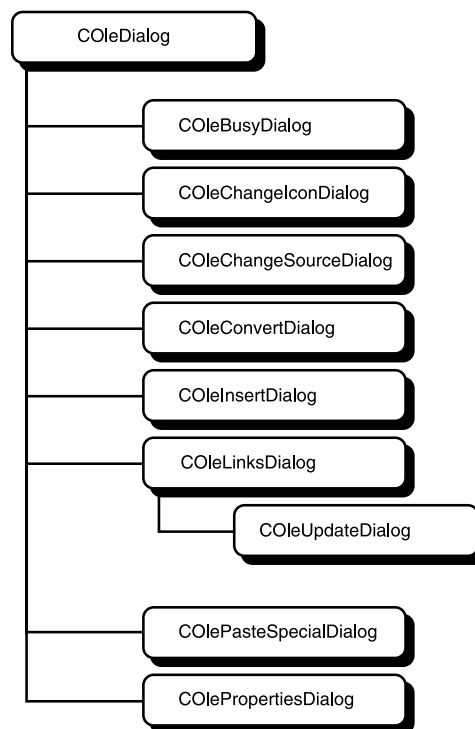


Abbildung 23.2:
OLE-Standard-
dialoge in der
MFC

23.2 Standardsteuerelemente

Die MFC verfügt über eine Mantelklasse für jedes der neuen Windows-95/98-Standardsteuerelemente. Eine große Zahl dieser Steuerelemente kann nun mit Hilfe des Dialog-Editors im Visual Studio erstellt werden. Sie müssen der Klasse lediglich eine Elementvariable mit Hilfe des Klassen-Assistenten hinzufügen, um einem neu erstellten Steuerelement ein Objekt der entsprechenden Steuerelementklasse zuzuweisen.

Wir verwenden eine einfache Anwendung, um die Anwendung der Standardsteuerelemente zu erörtern. Erstellen Sie dieses Programm mit Hilfe des Anwendungsassistenten (DIALOGFELDBASIEREND), übernehmen Sie alle Voreinstellungen und nennen Sie die Anwendung CTRL. Die Anwendung nutzt Schieberegler, Fortschrittsleisten, Zugriffstasten, Auf-/Ab-Steuerelemente, Listfelder, Strukturansichten, Animations-Steuerelemente und Registerreiter. Wir verwenden das Register-Steuerelement, um manuell einen Registerdialog zu erzeugen. Gewöhnlich würden Sie dazu den Dialog-Editor verwenden. Die manuelle Erstellung eines Registerdialoges demonstriert jedoch die Möglichkeiten des Register-Steuerelements.

23.2.1 Animation-Steuerelement

Abbildung 23.3:
Das Animation-
Steuerelement



CAnimateCtrl Das Animation-Steuerelement kann Videodateien im einfachen AVI-Format abspielen. Sie erstellen ein Animation-Steuerelement in einem Dialog mit Hilfe des Dialog-Editors. Fügen Sie der Klasse anschließend mit Hilfe des Klassenassistenten eine Elementvariable vom Typ `CAnimateCtrl` zu. Das Ableiten Ihrer eigenen Klasse von `CAnimateCtrl` ist gewöhnlich nicht erforderlich.

Sie laden ein Video in das Animation-Steuerelement, indem Sie die `CAnimateCtrl::Load`-Elementfunktion aufrufen. Diese Funktion akzeptiert entweder einen Dateinamen oder einen Ressourcenbezeichner als Parameter. Ich habe beispielsweise der CTRL-Anwendung eine AVI-Datei als benutzerdefinierte Ressource in einer externen Datei hinzugefügt, und diese dem Textbezeichner »VIDEO« zugewiesen. In der `OnInitDialog`-Elementfunktion des CTRL-Dialoges wird anschließend die folgende Funktion aufgerufen:

```
m_cAnimate.Open(_T("VIDEO"));
```

Nachdem das Animation-Steuerelement mit dem `Autoplay`-Stil erstellt wurde, starten Sie das Video mit einem Aufruf der `CAnimateCtrl::Play`-Elementfunktion. Indem Sie die entsprechenden Parameter dieser Funktion setzen, bestimmen Sie das erste und letzte Bild der Vorführung und die Anzahl der Wiederholungen des Clips. Zum Abspielen und kontinuierlichen Wiederholen des gesamten Videos rufen Sie die folgende Funktion auf:

```
m_cAnimate.Play(0, -1, -1);
```

Abbildung 23.3 stellt das Animation-Steuerelement in der CTRL-Anwendung dar.

Das Animation-Steuerelement wurde nicht als Video-Abspielement entwickelt. Es dient der Anzeige einfacher Animationen, wie zum Beispiel ein animiertes Symbol während eines zeitintensiven Vorgangs.



23.2.2 Spaltenüberschriften-Steuerelement

Das Spaltenüberschriften-Steuerelement wird zur Erstellung von Überschriften für die Spalten eines Listenfeldes verwendet. Sie finden dieses Steuerelement vorwiegend in der Berichtansicht einer Listenansicht, wie in Abbildung 23.5 dargestellt.

Verwenden Sie die `CHeaderCtrl`-Klasse, um ein Spaltenüberschriften-Steuerelement zu erstellen. Rufen Sie dazu die `Create`-Elementfunktion der Klasse auf. Um dem Spaltenüberschriften-Steuerelement eine Überschrift hinzuzufügen, lassen Sie `CHeaderCtrl::InsertItem` ausführen. *CHeaderCtrl*

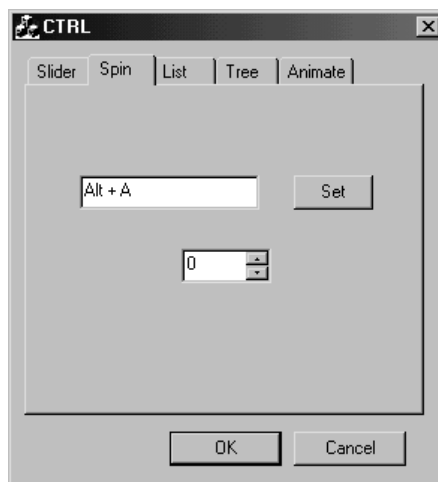
Wird das Spaltenüberschriften-Steuerelement innerhalb eines Ansichtsfensters angeordnet, nehmen Sie die genannten Initialisierungen in der `OnInitialUpdate`-Elementfunktion des Ansichtsobjekts vor.

Ein Spaltenüberschriften-Steuerelement kann einem Dialog nicht mit Hilfe des Dialog-Editors zugewiesen werden. Möchten Sie ein Spaltenüberschriften-Steuerelement in Ihrem Dialog verwenden, fügen Sie der

Dialogklasse eine Elementvariable vom Typ `CHeaderCtrl` hinzu. Initialisieren Sie das Steuerelement anschließend in der `OnInitDialog`-Elementfunktion des Dialoges.

23.2.3 Zugriffstasten-Steuerelement

Abbildung 23.4:
Ein Zugriffstasten-Steuerelement und ein Drehfeld



Ein Zugriffstasten-Steuerelement akzeptiert einzelne Tastenanschläge, zeigt Tasten symbolisch an (zum Beispiel STRG+S) und gibt den virtuellen Code der betätigten Taste sowie die Shift-Codes zurück. Das Steuerelement wird gewöhnlich mit einer `WM_SETHOTKEY`-Nachricht verwendet, um einem Fenster eine Zugriffstaste zuzuweisen.

Eine Verwendungsmöglichkeit des Zugriffstasten-Steuerelements ist in der CTRL-Anwendung (Abbildung 23.4) aufgeführt. Tippt der Anwender eine Tastenkombination und anschließend die Schaltfläche SET, wird eine `WM_SETHOTKEY`-Nachricht an das Dialogfenster gesendet. Der CTRL-Dialog kann im Anschluß daran mit dieser Tastenkombination geöffnet werden.

CHotKeyCtrl Sie fügen einem Dialog ein Zugriffstasten-Steuerelement mit Hilfe des Dialog-Editors hinzu. Das entsprechende Objekt vom Typ `CHotKeyCtrl` wird mit dem Klassenassistenten als Elementvariable der Dialogklasse erzeugt.

Die aktuellen Einstellungen zu einem Tastenkürzel werden ermittelt, wenn der Anwender zu verstehen gibt, daß die Auswahl beendet ist. Dazu kann der Anwender beispielsweise eine Schaltfläche betätigen. In der CTRL-Anwendung erfüllt die Schaltfläche SET diesen Zweck. Der Wert des Tastenkürzels wird mit der Elementfunktion `CHotKey::GetHot-`

Key ermittelt. Der von dieser Funktion zurückgegebene DWORD-Wert kann anschließend in einem Aufruf von `SendMessage` als WPARAM-Parameter der `WM_SETHOTKEY`-Nachricht verwendet werden. Listing 23.7 demonstriert diese Vorgehensweise in der CTRL-Anwendung.

```
void CCTRLDlg::OnButton()
{
    // TODO: Nachrichtenbearbeiter hier einfügen
    SendMessage(WM_SETHOTKEY, m_cHotKey.GetHotKey());
}
```

*Listing 23.7:
Bearbeiten eines
Tastenkürzels*

Tastenkürzel können ebenfalls in Verbindung mit der Win32-Funktion `RegisterHotKey` verwendet werden, um spezifische Thread-Zugriffstasten zu definieren.

23.2.4 Listenansicht-Steuerelement

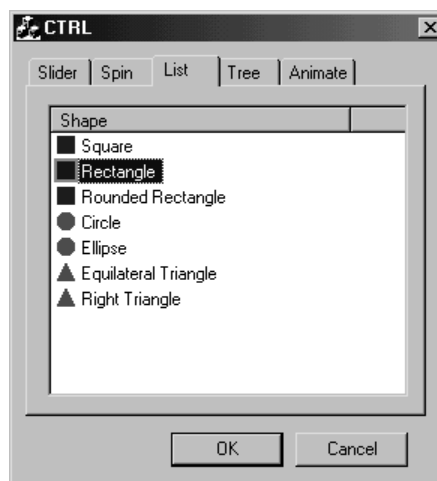


*Abbildung 23.5:
Eine Listenansicht*

Eine Listenansicht enthält Einträge, die mit einem Symbol und einer Beschriftung bezeichnet sind. Die Einträge in einer Listenansicht können unterschiedlich angeordnet werden.

- ➡ In der Ansicht GROSSE SYMBOLE werden die Einträge als große Symbole mit einem darunter angeordneten Text angezeigt. Sie können innerhalb des Steuerelements an eine beliebige Position verschoben werden.
- ➡ In der Ansicht KLEINE SYMBOLE erscheinen die Einträge als kleine Symbole. Die Beschriftung befindet sich rechts des Symbols. Auch hier ist ein Verschieben der Einträge möglich.

- In der Ansicht LISTE werden die Einträge in Spalten angeordnet und mit kleinen Symbolen dargestellt. Rechts neben den Symbolen befinden sich die Beschriftungen. Die Position der Einträge kann in dieser Ansicht nicht verändert werden.
- In der Ansicht DETAILS werden die Einträge ebenfalls in mehreren Spalten angeordnet. Sie können den Einträgen Untereinträge zuweisen, die rechts neben dem übergeordneten Eintrag positioniert werden.

CListView Die MFC hält zwei Klassen für das Listenansicht-Steuerelement bereit. Die *CListView*-Klasse wird dazu benutzt, ein Ansichtsfenster mit einem Listenansicht-Steuerelement zu erstellen, das den gesamten Client-Bereich einnimmt.

CListCtrl Möchten Sie die Listenansicht auf eine andere Weise verwenden, zum Beispiel in einem Dialog, steht Ihnen die *CListCtrl*-Klasse zur Verfügung. Soll einem Dialog ein Listenansicht-Steuerelement hinzugefügt werden, geschieht dies mit dem Dialog-Editor des Visual Studios.

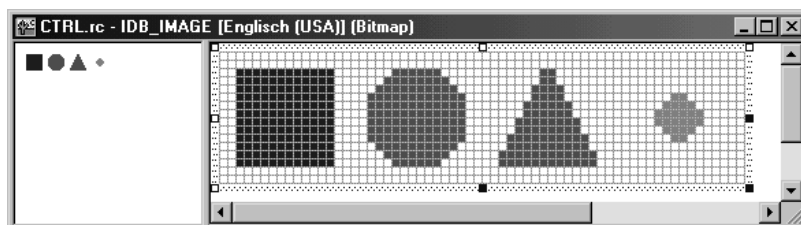
Das Erstellen einer Listenansicht wird in mehreren Schritten ausgeführt. Das Steuerelement wird zunächst erzeugt, anschließend werden seine Spalten mit einem Aufruf der *InsertColumn*-Elementfunktion eingerichtet, und schließlich fügen Sie der Listenansicht die gewünschten Einträge mit der *InsertItem*-Funktion hinzu.

Die Symbole der Einträge müssen in einer *BILDERLISTE* bereitgestellt werden. Bilderlisten sind eine Sammlung kleiner Grafikobjekte derselben Größe. Die *CImageList*-Klasse unterstützt Bilderlisten in der MFC.

Abbildung 23.5 zeigt die in der CTRL-Anwendung verwendete Listenansicht. Das Listenansicht-Steuerelement in der Dialogvorlage wurde für die Reportansicht und Einzelauswahl konfiguriert.

Die für die Listenansicht benutzte Bitmap besteht aus vier Bildern mit einer Größe von 16×16 Pixeln. Die Bitmap ist in Abbildung 23.6 dargestellt.

Abbildung 23.6:
Die Bitmap der
Listenansichts-
symbole



Erstellen Sie zur Verwaltung der Listenansicht in CTRL mit Hilfe des Klassen-Assistenten eine Elementvariable vom Typ `CListCtrl` für Ihren Dialog. Initialisieren Sie das Listenansicht-Steuerelement anschließend in `OnInitDialog`, wie in Listing 23.8 dargestellt.

```
m_cImageList.Create(IDB_IMAGE, 16, 10, 0);
m_cList.InsertColumn(0, _T("Shape"), LVCFMT_LEFT, 200);
m_cList.SetImageList(&m_cImageList, LVSIL_SMALL);
PSTR pszListItems[] = {_T("Square"), _T("Rectangle"),
                       _T("Rounded Rectangle"), _T("Circle"),
                       _T("Ellipse"),
                       _T("Equilateral Triangle"),
                       _T("Right Triangle"), NULL};
int nListTypes[] = {0, 0, 0, 1, 1, 1, 2, 2};
for (int i = 0; pszListItems[i] != NULL; i++)
    m_cList.InsertItem(i, pszListItems[i], nListTypes[i]);
```

*Listing 23.8:
Einrichten der Li-
stenansicht in
CTRL*

Zur Bearbeitung der in der Listenansicht vorgenommenen Auswahl des Anwenders können Sie den Status eines Eintrages ermitteln, indem Sie die `CListCtrl::GetItemState`-Elementfunktion aufrufen.

23.2.5 Fortschrittsleiste

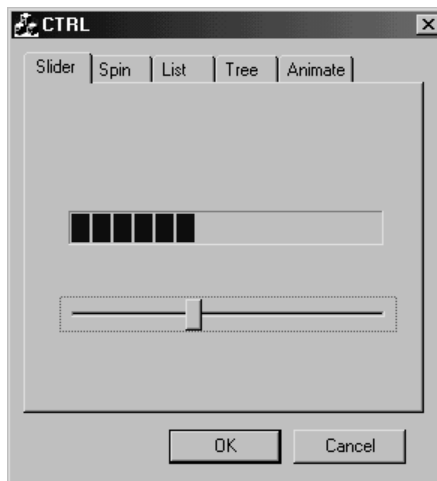


*Abbildung 23.7:
Ein Fortschritts-
leiste- und ein
Schieberegler-
Steuerelement*

Eine Fortschrittsleiste informiert den Anwender mit einer grafischen Anzeige über den Fortschritt eines bestimmten Vorgangs. Dieses Steuerelement unterscheidet sich von anderen Steuerelementen, da es lediglich den Anwender informiert, von diesem jedoch keine Eingaben entgegennimmt.

Um eine Fortschrittsleiste in einem Dialog zu verwenden, fügen Sie dem Dialog ein Fortschrittsleiste-Steuerelement mit Hilfe des Visual Studios hinzu. Erstellen Sie anschließend mit dem Klassen-Assistenten eine Elementvariable vom Typ `CProgressCtrl`. Die Initialisierung des

CProgressCtrl

Steuerelements verlangt das Einrichten eines Minimum- und Maximum-Wertes sowie, optional, der aktuellen Position der Leiste. Setzen Sie das Fortschrittsleiste-Steuerelement in einem Dialog ein, nehmen Sie die Initialisierung in der `OnInitDialog`-Elementfunktion der Dialogklasse vor.

Abbildung 23.7 zeigt eine Fortschrittsleiste mit einem Schieberegler in CTRL.

Die Initialisierung der Fortschrittsleiste in `CCTRLDlg::OnInitDialog`, ist in Listing 23.9 aufgeführt:

Listing 23.9:
Initialisierung
der Fortschritts-
leiste

```
m_cProgress.SetRange(1, 100);
m_cProgress.SetPos(1);
```

Die Position der Fortschrittsleiste wird mit der `SetPos`-Elementfunktion gesetzt.

23.2.6 Rich-Text-Steuerelement

Das Rich-Text-Steuerelement oder RTF-Steuerelement erweitert die Fähigkeiten des Standard-Textfeld-Steuerelements. RTF-Steuerelemente stellen formatierten Text dar und verwalten diesen.

Diese Steuerelemente bilden eine komplexe Programmierschnittstelle, die aus einer großen Zahl verschiedener Elementfunktionen der Klassen besteht, die diesen Steuerelementtyp unterstützen. Die MFC-Bibliothek stellt vier Klassen für das Rich-Text-Steuerelement zur Verfügung.

CRichEditCtrl Eine einfache Unterstützung des Rich-Text-Steuerelements wird von der `CRichEditCtrl`-Klasse angeboten. Die MFC-Bibliothek unterstützt jedoch außerdem Anwendungen mit Dokumenten, die auf einem solchen Steuerelement basieren. Zu diesem Zweck werden die Klassen `CRichEditDoc`, `CRichEditView` und `CRichEditCntrItem` verwendet.

Text kann in Rich-Text-Feldern mehrzeilig angeordnet werden. Dem Text können Zeichen- und Absatzformate zugewiesen werden. Das Formatieren eines Textes innerhalb eines Rich-Text-Steuerelements, das durch ein `CRichEditCntr`-Objekt repräsentiert wird, geschieht mit Hilfe der Elementfunktionen `SetSelectionCharFormat`, `SetWordCharFormat` und `SetParaFormat`. Das Rich-Text-Steuerelement verfügt nicht über die Funktionen der Steuerelementklasse. Es stellt keine Benutzeroberfläche für die Textformatierung zur Verfügung. Die Anwendung muß diese Oberfläche in Form von Menüs, Symbolleisten und anderen Steuerelementen implementieren.

Rich-Text-Steuerelemente unterstützen außerdem die Zwischenablage, OLE, das Drucken und Dateioperationen. Sie laden und speichern Text sowohl im Text- (ASCII) als auch im Rich-Text-Format.

Das Windows-95/98-Programm WordPad demonstriert den Gebrauch eines Rich-Text-Steuerelements in einer MFC-Anwendung. Microsoft stellt Entwicklern den Quellcode dieser Anwendung zur Verfügung. Dieser Programmcode ist für die Anwender eine ausgezeichnete Referenz, die Rich-Text-Steuerelemente in einer MFC-Anwendung nutzen möchten.

23.2.7 Schieberegler-Steuerelement

Schieberegler-Steuerelemente werden häufig mit den Lautstärkeregler an Stereo-Anlagen verglichen. Sie ersetzen die bedenkliche Praxis, Bildlaufleisten für diesen Zweck zu verwenden.

In der MFC werden Schieberegler mit der `CSliderCtrl`-Klasse gebildet. Ein Schieberegler-Steuerelement kann einer Dialogvorlage mit Hilfe des Dialog-Editors hinzugefügt werden. Erstellen Sie anschließend eine entsprechende Elementvariable in Ihrer Dialogklasse. Der Klassenassistent hilft Ihnen dabei. Das Schieberegler-Steuerelement kann über diese Elementvariable in der `OnInitDialog`-Elementfunktion Ihrer Dialogklasse initialisiert werden. Die Initialisierung umfaßt das Einrichten eines Minimum- und Maximum-Wertes sowie, optional, der anfänglichen Position und der Darstellungsweise.

Abbildung 23.7 zeigt ein Schieberegler-Steuerelement. Mit dem Schieberegler bestimmen Sie die Position der Fortschrittsleiste. Die Initialisierung des Steuerelements besteht aus einem einzigen Aufruf:

```
m_cSlider.SetRange(1, 100);
```

Schieberegler-Steuerelemente senden `WM_HSCROLL`-Nachrichten an das übergeordnete Fenster, wenn die Position des Reglers verändert wird. Bearbeiter für diese Nachrichten können mit dem Klassenassistenten in der MFC-Anwendung installiert werden. In `CTRL` wird eine Bearbeiterfunktion verwendet, die die Position des Reglers ermittelt und dazu nutzt, die Position der Fortschrittsleiste zu aktualisieren. Die Bearbeiterfunktion ist in Listing 23.10 aufgeführt.

```
void CCTRLDlg::OnHScroll(UINT nSBCode, UINT nPos,
                        CScrollBar* pScrollBar)
{
    // TODO: Nachrichtensbearbeiter hier einfügen
    if ((CSliderCtrl *)pScrollBar == &m_cSlider)
    {
        m_cProgress.SetPos(m_cSlider.GetPos());
    }
    else
        CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

*Listing 23.10:
Bearbeiten der
WM_HSCROLL-
Nachrichten des
Schieberegler-
Steuerelements*

23.2.8 Drehregler

Eine Drehfeld-Schaltfläche besteht aus zwei Pfeilen, die zum Inkrementieren und Dekrementieren eines Wertes verwendet werden. Drehregler werden vorwiegend mit Eingabefeldern eingesetzt, die einen numerischen Wert enthalten.

Das Buddy-Element Mit Hilfe des Dialog-Editors fügen Sie einem Dialog eine Drehfeld-Schaltfläche hinzu. Eine Drehfeld-Schaltfläche setzt den Wert eines Eingabe-Steuerelements, dem es zugewiesen wurde. Sie können jedoch alternativ dazu einen Drehregler erstellen, der automatisch (Auto-Buddy-Format) dem Steuerelement zugewiesen wird, das in der Tabulatorreihenfolge vor der Schaltfläche angeordnet ist.

CSpinButtonCtrl Bestimmen Sie das Auto-Buddy-Format für einen Drehregler, müssen Sie keinen Programmcode schreiben. Müssen Sie den Drehregler jedoch manipulieren, können Sie eine Elementvariable vom Typ `CSpinButtonCtrl` in Ihrer Dialogklasse anlegen.

Die CTRL-Anwendung verwendet einen Drehregler (Abbildung 23.4). Für dieses Steuerelement wurde eine Elementvariable erstellt, die die Schaltfläche versteckt oder anzeigt.

23.2.9 Statusfenster

MFC-Anwendungen zeigen Statusfenster, die auch als Statusleisten bezeichnet werden, im unteren Bereich des Hauptrahmenfensters an.

CStatusBarCtrl Das Windows-95/98-Statusfenster-Steuerelement wird in der MFC von der `CStatusBarCtrl`-Klasse unterstützt. Anwendungen können diese Klasse nutzen, sofern Statusleisten benötigt werden, deren Funktionalität die der von dem MFC-Anwendungsrahmen zur Verfügung gestellten Statusleisten überbieten soll.

Erstellen Sie zunächst ein `CStatusBarCtrl`-Objekt. Verwenden Sie die `Create-Elementfunktion` des Objekts, um das Statusfenster-Steuerelement zu erzeugen. Statusfenster werden nicht von dem Dialog-Editor unterstützt.

Die `CStatusBar`-Klasse integriert Statusleisten in MFC-Rahmenfenstern.

23.2.10 Register-Steuerelement

Register-Steuerelemente bilden überlappende Objekte in einer kompakten grafischen Oberfläche. Diese Steuerelemente werden gewöhnlich in Registerdialogen verwendet.

Sie erstellen Register-Steuerelemente mit dem Dialog-Editor des Visual Studios. Mit Hilfe des Klassen-Assistenten erzeugen Sie anschließend in der Dialogklasse eine Elementvariable vom Typ `CTabCtrl`. Über diese Elementvariable setzen Sie die Beschriftung und das äußere Erscheinungsbild des Steuerelements.

Register-Steuerelemente senden Nachrichten an das übergeordnete Fenster. Die Bearbeiter dieser Nachrichten werden mit dem Klassenassistenten installiert.

Die CTRL-Anwendung verwendet ein Register-Steuerelement, um einen Registerdialog zu simulieren. Abbildung 23.3 und die folgenden Abbildungen zeigen das Steuerelement während dessen Anwendung.

Die dem Steuerelement zugewiesene Elementvariable `m_cTab` wird in der `CCTRLDlg::OnInitDialog`-Elementfunktion initialisiert. Die Initialisierung ist in Listing 23.11 aufgeführt.

```
TC_ITEM tcItem;
PSTR pszTabItems[] = {_T("Slider"), _T("Spin"), _T("List"),
                     _T("Tree"), _T("Animate"), NULL};
for (i = 0; pszTabItems[i] != NULL; i++)
{
    tcItem.mask = TCIF_TEXT;
    tcItem.pszText = pszTabItems[i];
    tcItem.cchTextMax = strlen(pszTabItems[i]);
    m_cTab.InsertItem(i, &tcItem);
}
```

*Listing 23.11:
Die Initialisierung des Register-Steuerelements*

Ein Nachrichtensarbeiter vom Typ `TCN_SELCHANGE` reagiert auf die von dem Anwender vorgenommene Auswahl eines Registerreiters in dem Register-Steuerelement. In diesem in Listing 23.12 aufgeführten Bearbeiter werden andere Steuerelemente des Dialogs gemäß des selektierten Registers gesperrt oder freigegeben. Die Funktion setzt außerdem den Fokus auf das entsprechende Steuerelement und startet die Video-Wiedergabe, sofern das Animation-Steuerelement nach Auswahl des Registers ANIMATE sichtbar gemacht wurde.

```
void CCTRLDlg::OnSelchangeTab(NMHDR* pNMHDR, LRESULT* pResult)
{
    // TODO: Nachrichtensarbeiter hier einfügen
    m_cSlider.ShowWindow(m_cTab.GetCurSel() == 0
        ? SW_SHOW : SW_HIDE);
    m_cProgress.ShowWindow(m_cTab.GetCurSel() == 0
        ? SW_SHOW : SW_HIDE);
    m_cEdit.ShowWindow(m_cTab.GetCurSel() == 1
        ? SW_SHOW : SW_HIDE);
    m_cSpin.ShowWindow(m_cTab.GetCurSel() == 1
        ? SW_SHOW : SW_HIDE);
    m_cHotKey.ShowWindow(m_cTab.GetCurSel() == 1
        ? SW_SHOW : SW_HIDE);
    m_cButton.ShowWindow(m_cTab.GetCurSel() == 1
        ? SW_SHOW : SW_HIDE);
    m_cList.ShowWindow(m_cTab.GetCurSel() == 2
        ? SW_SHOW : SW_HIDE);
}
```

*Listing 23.12:
Bearbeiten von TCN_SELCHANGE-Nachrichten in CTRL*

```

m_cTree.ShowWindow(m_cTab.GetCurSel() == 3
                  ? SW_SHOW : SW_HIDE);
switch (m_cTab.GetCurSel())
{
    case 0: m_cSlider.SetFocus(); break;
    case 1: m_cEdit.SetFocus(); break;
    case 2: m_cList.SetFocus(); break;
    case 3: m_cTree.SetFocus(); break;
    case 4: m_cAnimate.SetFocus(); break;
}
if (m_cTab.GetCurSel() == 4)
{
    m_cAnimate.ShowWindow(SW_SHOW);
    m_cAnimate.Play(0, (UINT)-1, (UINT)-1);
}
else
{
    m_cAnimate.Stop();
    m_cAnimate.ShowWindow(SW_HIDE);
}
*pResult = 0;
}

```

23.2.11 Symbolleisten

Symbolleisten sind Fenster, die mehrere Schaltflächen enthalten. Betätigt der Anwender eine dieser Schaltflächen, sendet die Symbolleiste eine Anweisungsnachricht an das übergeordnete Fenster. Symbolleisten werden sehr häufig in MFC-Anwendungen verwendet.

CToolBarCtrl Die `CToolBarCtrl`-Klasse unterstützt Symbolleisten in der MFC. Sie erzeugen eine Symbolleiste, indem Sie zunächst ein Objekt vom Typ `CToolBarCtrl` erzeugen und anschließend dessen `Create`-Elementfunktion aufrufen. Das Steuerelement unterstützt Schaltflächen, die mit einer Bitmap, einem Text oder sowohl einer Bitmap als auch einem Text beschriftet sind. Rufen Sie dazu eine der `CToolBarCtrl`-Elementfunktionen `AddBitmap` oder `AddString` auf. Die Schaltflächen der Symbolleiste werden mit einem Aufruf von `AddButtons` generiert.

Symbolleisten und Quickinfos Symbolleisten können zusammen mit `QuickInfos` verwendet werden. Erstellen Sie eine Symbolleiste mit dem `TBSTYLE_TOOLTIPS`-Stil, sendet die Symbolleiste `QuickInfo`-Nachrichten an das übergeordnete Fenster. Diese Nachrichten fordern die `QuickInfo`-Texte an, die in den `QuickInfo`-Steuerelementen dargestellt werden sollen. `QuickInfo`-Steuerelemente werden von dem Symbolleiste-Steuerelement erzeugt und verwaltet.

Symbolleisten unterstützen eine umfassende Konfiguration über einen Systemdialog, der dem Anwender das Hinzufügen, Löschen und Anordnen von Symbolleistenschaltflächen ermöglicht. Während dieser Konfiguration sendet die Symbolleiste Nachrichten an das übergeordnete Fenster.

Der Status einer Symbolleiste kann mit der `CToolBarCtrl::SaveState`-Elementfunktion in der Registrierung gespeichert werden. Der gespeicherte Status wird mit `CToolBarCtrl::RestoreState` ermittelt.

Die MFC unterstützt Symbolleisten, die über die `CToolBar`-Klasse in einem Rahmenfenster integriert werden.

23.2.12 QuickInfo-Steuerelement

QuickInfos sind kleine Pop-Up-Fenster, die eine einzelne Zeile Text anzeigen. QuickInfos werden gewöhnlich als visuelle Hilfe zur Erläuterung der Funktion eines Steuerelements oder eines Werkzeugs in einer Anwendung verwendet.

QuickInfos werden automatisch von Symbolleiste-Steuerelementen erstellt und verwaltet. Die `CWnd`-Klasse unterstützt QuickInfos ebenfalls über die Funktionen `EnableToolTips`, `CancelToolTips`, `FilterToolTipMessage` und `OnToolTipHitTest`.

Möchten Sie QuickInfos verwenden, erstellen Sie zunächst ein Objekt vom Typ `CToolTipCtrl` und rufen dessen Elementfunktion `Create` auf. Registrieren Sie mit `AddTool` ein Werkzeug für das QuickInfo. Geben Sie dazu das entsprechende Fenster und einen rechteckigen Bereich innerhalb dieses Fensters an. Wird der Mauszeiger später für mehr als eine Sekunde über diesem Bereich angeordnet, erscheint das QuickInfo. Sie können mehrere Werkzeuge für dasselbe QuickInfo registrieren.

Sie aktivieren das QuickInfo mit einem Aufruf der Elementfunktion `CToolTipCtrl::Activate`.

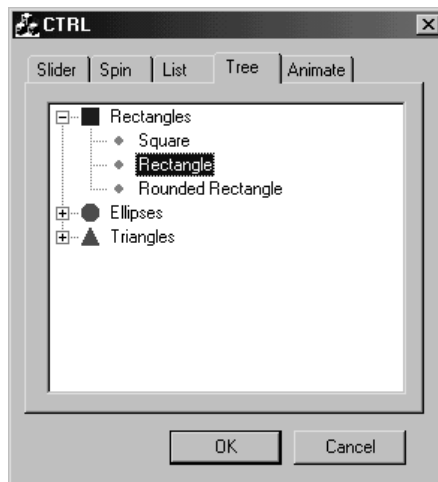
QuickInfos senden Nachrichten an das übergeordnete Fenster. Benötigt ein QuickInfo Informationen über den darzustellenden Text, sendet es die Nachricht `TTN_NEEDTEXT`.

23.2.13 Strukturansicht-Steuerelement

Ein Strukturansicht-Steuerelement führt Listeneinträge hierarchisch auf. Die MFC unterstützt die Strukturansicht mit den Klassen `CTreeView` und `CTreeCtrl`. `CTreeView` wird von Anwendungen verwendet, die Ansichten zur Verfügung stellen, die aus einem einzelnen Strukturansicht-Steuerelement bestehen.

Sie fügen einem Dialog eine Strukturansicht mit Hilfe des Dialog-Editors hinzu. Für jedes Strukturansicht-Steuerelement muß ein Objekt vom Typ `CTreeCtrl` mit Hilfe des Klassen-Assistenten erstellt werden. Über dieses Objekt wird die Strukturansicht initialisiert.

Abbildung 23.8:
Ein Strukturan-
sicht-Steuer-
element



Dem Strukturansicht-Steuererelement werden mit der `CTreeCtrl::InsertItem`-Elementfunktion neue Einträge hinzugefügt. Strukturansichten verwenden Bilderlisten zur Anzeige der Symbole, die mit den Einträgen verknüpft sind. Mit der Klasse `CImageList` erstellen Sie eine Bilderliste.

Auch Strukturansicht-Steuerelemente senden Nachrichten an das übergeordnete Fenster. Wird beispielsweise ein Eintrag in einer Strukturansicht selektiert, sendet das Steuerelement eine `TVN_SELCHANGED`-Nachricht.

Abbildung 23.8 zeigt das in der CTRL-Anwendung verwendete Strukturansicht-Steuerelement. Dieses wurde dem Dialog der Anwendung mit dem Dialog-Editor hinzugefügt. Drei Stil-Einstellungen (Plus-/Minus-Zeichen, Linien, Wurzellinien) ermöglichen die Darstellung von Linien und Knoten, die auch Plus-/Minus-Zeichen genannt werden.

Das Steuerelement wird in der `CCTRLDlg::OnInitDialog`-Elementfunktion initialisiert (Listing 23.13).

*Listing 23.13:
Initialisierung ei-
nes Bauman-
sicht-Steuer-
elements*

```

PSTR pszTreeRoots[] = {_T("Rectangles"), _T("Ellipses"),
                        _T("Triangles")};
m_cTree.SetImageList(&m_cImageList, TVSIL_NORMAL);
HTREEITEM rootitems[3];
for (i = 0; i < 3; i++)
    rootitems[i] = m_cTree.InsertItem(TVIF_PARAM | TVIF_TEXT |
                                     TVIF_IMAGE |
                                     TVIF_SELECTEDIMAGE,
                                     pszTreeRoots[i], i, i, 0, 0,
                                     -1, TVI_ROOT, TVI_LAST);
for(i = 0; pszListItems[i] != NULL; i++)
    m_cTree.InsertItem(TVIF_PARAM | TVIF_TEXT | TVIF_IMAGE |
                     TVIF_SELECTEDIMAGE, pszListItems[i], 3, 3,
                     0, 0, i, rootitems[i], ListTypes[i], TVI_LAST);

```

Die CTRL-Anwendung bearbeitet die von diesem Struktursicht-Steuer-element kommenden Nachrichten. Wird ein Eintrag in dem Steuer-element selektiert, setzt der Bearbeiter der TVN_SELCHANGED-Nachricht den Selektionsstatus des entsprechenden Eintrags in der Struktursicht der Anwendung. Diese Bearbeiterfunktion ist in Listing 23.14 aufgeführt.

```
void CCTRLDlg::OnSelchangedTree(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_TREEVIEW* pNMTreeView = (NM_TREEVIEW*)pNMHDR;
    // TODO: Nachrichtenbearbeiter hier einfügen
    int i = m_cTree.GetItemData(m_cTree.GetSelectedItem());
    if (i != -1)
        m_cList.SetItemState(i, LVIS_SELECTED, LVIS_SELECTED);
    *pResult = 0;
}
```

*Listing 23.14:
Bearbeitung der
TVN_SEL-
CHANGED-Nach-
richt in CTRL*

23.2.14 Die neuen Windows-98-Steuerelemente

Windows 95/98 verwendet einige neue Standardsteuerelemente. Neu seit Windows 98 ist die Unterstützung der Internet-Explorer-4.0-Steuerelemente (IE-Werkzeugleiste, IP-Adresse etc.)

In den Klammern stehen die Namen der zugehörigen MFC-Klassen.

Datums-/Zeitauswahl Zur Einstellung von Datum und Zeit. (CDateTimeCtrl).

Monatskalender Zur Auswahl eines Datums. (CMonthCalCtrl)

Werkzeugleiste In der Größe veränderbare Werkzeugleiste, die andere Kindfenster aufnehmen kann. (CReBarCtrl)

IP-Adresse Eingabefeld für IP-Adressen. (CIPAddressCtrl)



23.3 Zusammenfassung

Die MFC-Bibliothek unterstützt Standarddialoge und Standardsteuerelemente unter Windows.

Standarddialoge sind von CCommonDialog abgeleitete Klassen. Diese Klassen bieten die Funktionalität der Windows-Standarddialoge und dienen der Konfiguration, Darstellung und Bearbeitung dieser Dialoge. Die folgenden Dialoge werden angeboten:

- ➡ Dialoge zur Auswahl einer Farbe werden mit der CColorDialog-Klasse erzeugt.
- ➡ Die Dialoge ÖFFNEN und SPEICHERN UNTER werden von der CFileDialog-Klasse unterstützt.

- ➡ Mit Hilfe der Klasse `CFindReplaceDialog` erstellen Sie die Dialoge SUCHEN und ERSETZEN.
- ➡ Der Dialog zur Auswahl einer Schriftart wird von der Klasse `CFontDialog` unterstützt.
- ➡ Die Klasse `CPageSetupDialog` dient der Erzeugung des Dialoges SEITE EINRICHTEN.
- ➡ Die Dialoge DRUCKEN und DRUCKER EINRICHTEN werden von der Klasse `CPrintDialog` unterstützt.
- ➡ Die von `COleDialog` abgeleiteten Klassen werden zur Erstellung von OLE-Standarddialogen verwendet.

Standardsteuerelemente sind ein neues Feature von Windows 95/98. Sie erweitern die Werzeugsammlung des Anwendungsprogrammierers und werden vorwiegend in Dialogen und Ansichtsfenstern verwendet.

Die Standardsteuerelemente sowie ihre Unterstützung durch die MFC und den Dialog-Editor des Visual Studios sind in Tabelle 23.1 aufgeführt.

Tabelle 23.1:
MFC- und Visual
Studio-Unter-
stützung für
Standardsteuer-
elemente

Steuerelement	Steuerelement- klasse	Ansichtsklasse	Im Dialog- Editor?
Animation	<code>CAnimateCtrl</code>		Ja
Spaltenüber- schrift	<code>CHeaderCtrl</code>		Nein
Tastenkürzel	<code>CHotKeyCtrl</code>		Ja
Listenansicht	<code>CListCtrl</code>	<code>CListView</code>	Ja
Fortschrittsleiste	<code>CProgressCtrl</code>		Ja
Rich-Text	<code>CRichEditCtrl</code>	<code>CRichEditView</code>	Nein
Schieberegler	<code>CSliderCtrl</code>		Ja
Drehregler	<code>CSpinButtonCtrl</code>		Ja
Statusfenster	<code>CStatusBarCtrl</code>		Nein
Register	<code>CTabCtrl</code>		Ja
Symbolleiste	<code>CToolBarCtrl</code>		Nein
QuickInfo	<code>CToolTipCtrl</code>		Nein
Baumansicht	<code>CTreeCtrl</code>	<code>CTreeView</code>	Ja

Steuerelement	Steuerelement- klasse	Ansichtsklasse	Im Dialog- Editor?
Die neuen Win98-Steuerlemente:			
Erweitertes Kombinationsfeld	CComboBoxEx		Ja
Datums-/Zeitauswahl	CDateTimeCtrl		Ja
Monatskalender	CMonthCalCtrl		Ja
Werkzeugleiste	CReBarCtrl		Nein
IP-Adresse	CIPAddressCtrl		Ja

Eine zusätzliche Unterstützung der Rich-Text-Steuerelemente bilden die Klassen `CRichEditDoc` und `CRichEditCntrItem`. Über diese Klassen und `CRichEditView` können Anwendungen erstellt werden, die Rich-Text-Dokumente verwalten.

Eine integrierte Unterstützung von Symbolleisten und Statusfenstern bieten die Klassen `CToolBar` und `CStatusBar`. Diese Klassen integrieren Symbolleisten und Statusfenster in MFC-Rahmenfenstern.

Gerätekontext und GDI-Objekte

Kapitel 24

Zeichnungen und Grafiken haben in jeder Windows-Anwendung eine besondere Bedeutung. Für MFC-Anwendungen gilt diese Aussage ebenfalls.

Die Funktionalität der Grafik-Geräteschnittstelle (GDI – Graphics Device Interface) wird von zwei MFC-Klassenfamilien zur Verfügung gestellt.

- Gerätekontextklassen enthalten GDI-Gerätekontexte und sehr viele Zeichenfunktionen.
- Aus GDI-Objektklassen werden GDI-Objekte erzeugt, wie zum Beispiel Stifte, Pinsel, Bitmaps oder Schriftarten.

Anwendungen, die keine MFC-Anwendungen sind, zeichnen auf einem Ausgabegerät, indem sie

1. den entsprechenden Gerätekontext ermitteln,
2. die GDI-Zeichenobjekte einrichten,
3. die Zeichenoperationen ausführen und
4. die verwendeten Ressourcen wieder freigeben.

Der MFC-Anwendungsrahmen erleichtert diese Schritte, indem er einige einfache Aufgaben übernimmt, die der Anwender gewöhnlich ausführen müßte. Sie können beispielsweise ein Stift-Objekt erstellen, indem Sie die erforderlichen Parameter dem `CPen`-Konstruktor übergeben:

```
CPen myPen(PS_SOLID, 0, RGB(255, 0, 0));
```

Das GDI-Objekt wird automatisch von dem `CPen`-Destruktor zerstört, wenn das `CPen`-Objekt den Gültigkeitsbereich verläßt.

Tabelle 24.1:
Grafikausgabe
für verschiedene
Ereignisbehand-
lungen

Operation	Quelltext	Quelltext (On Draw)
Gerätekontext ermitteln	<pre> CMyClass::OnLButtonDown(UINT nFlags, CPoint point) { CClientDC dc(this) </pre>	<pre> CMyClass::OnDraw(CDC* pDC) { </pre>
GDI-Objekt einrichten	<pre> CPen neuerStift(PS_SOLID ,0,RGB(255,0,0)); CPen* palterStift = dc.SelectObject(neuerStift); </pre>	<pre> CPen neuerStift(PS_SOLID ,0,RGB(255,0,0)); CPen* palterStift = pDC- >SelectObject(neuerStift); </pre>
Zeichnen	<pre> dc.MoveTo(10,10); dc.LineTo(100,100); </pre>	<pre> pDC->MoveTo(10,10); pDC->LineTo(100,100); </pre>
Ressourcen freigeben	<pre> dc.SelectObject(palterStift) } </pre>	<pre> pDC- >SelectObject(palterStift) } </pre>

Zwischen dem MFC-Objekt (von `CGdiObject` abgeleitet) und dem eigentlichen Gerätekontext oder GDI-Objekt besteht unter Windows ein Unterschied. Das Erstellen des MFC-Objekts führt nicht automatisch dazu, daß ein zugrundeliegendes Windows-Objekt generiert wird. Sie können jedoch zunächst ein leeres MFC-Objekt erstellen und dieses später mit dem Windows-Objekt verknüpfen.

In diesem Kapitel erörtern wir Gerätekontexte, die als »Zeichenflächen« dienen. Die Zeichenfunktionen selbst (zum Beispiel `Rectangle`, `Ellipse` oder `DrawText`) sind in der `CDC`-Klasse enthalten und werden ebenfalls besprochen. Der zweite Abschnitt des Kapitels erläutert Klassen, die GDI-Objekte enthalten. Diese Objekte repräsentieren Zeichenwerkzeuge.

24.1 Gerätekontexte

Obwohl das Erstellen eines Gerätekontextes mit Hilfe der MFC sehr einfach ist, ist diese Vorgehensweise nicht in jedem Fall erforderlich. Gewöhnliche Zeichenfunktionen, wie zum Beispiel die `OnDraw`-Elementfunktion einer Ansichtsklasse, werden von dem MFC-Anwendungsrahmen mit einem Zeiger auf ein Gerätekontext-Objekt aufgerufen. Dieses Objekt repräsentiert einen Gerätekontext, der bereits vorhanden und konfiguriert ist.

MFC-Klassen, die einen Gerätekontext repräsentieren, werden von der CDC-Basisklasse abgeleitet. Abbildung 24.1 stellt die Hierarchie der Gerätekontextklassen dar.

Die CDC-Klasse wird häufig als Mantelklasse für Gerätekontexte verwendet. Die anderen von CDC abgeleiteten Klassen unterscheiden sich von der Basisklasse darin, daß ihre Konstrukturfunktionen anders aufgebaut sind und sie zusätzliche Funktionalität zur Verfügung stellen. Möchten Sie ein MFC-Objekt erstellen, dem ein bereits bestehender Gerätekontext-Handle zugewiesen ist, sollten Sie die CDC-Basisklasse und nicht die davon abgeleiteten Klassen benutzen.

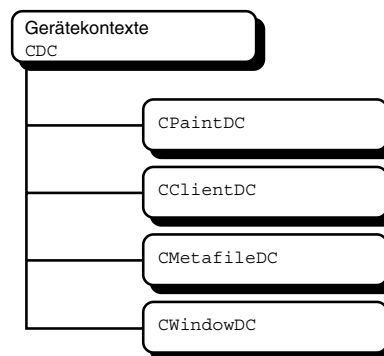


Abbildung 24.1:
Hierarchie der
Gerätekontext-
klassen

24.1.1 Die CDC-Basisklasse

Die CDC-Klasse bietet die Funktionalität des Windows-Gerätekontextes. Sie verfügt jedoch nicht ausschließlich über Funktionen, mit deren Hilfe ein Gerätekontext konfiguriert und verwaltet werden kann. Die CDC-Klasse enthält auch die GDI-Zeichenfunktionen. Nahezu 180 dokumentierte Elementfunktionen werden von dieser Klasse zur Verfügung gestellt.

Der folgende Abschnitt beschreibt, wie ein CDC-Objekt erstellt und einem GDI-Gerätekontext zugewiesen wird.

24.1.2 Erstellen eines Gerätekontextes

Ein GDI-Gerätekontext wird nicht automatisch erstellt, wenn ein CDC-Objekt über den Konstruktor generiert wird. Statt dessen muß ein Gerätekontext über die `CreateDC`-Funktion erzeugt oder einem CDC-Objekt zugewiesen werden, das bereits erstellt wurde.

Die `CreateDC`-Elementfunktion nimmt mehrere Parameter entgegen, die das Gerät, den Gerätetreiber und die Schnittstelle spezifizieren, über die auf das Gerät zugegriffen wird. *CreateDC*

Ein CDC-Objekt verfügt über zwei Elementvariablen, die GDI-Gerätekontext-Handles sind:

➡ `m_hDC` und

➡ `m_hAttribDC`.

Diese Variablen verweisen auf dasselbe Gerätekontextobjekt:

➡ Der `m_hDC`-Handle, auch Ausgabegerätekontext-Handle genannt, wird für alle Ausgabeoperationen verwendet.

➡ Der `m_hAttribDC`-Handle, der auch als Attributegerätekontext-Handle bezeichnet wird, dient der Anforderung von Informationen von dem Gerätekontext.

Attach Um einem CDC-Objekt einen Gerätekontext-Handle zuzuweisen, der bereits vorhanden ist, verwenden Sie die `Attach`-Elementfunktion. Um diese Zuweisung aufzuheben, rufen Sie die `Detach`-Elementfunktion auf.

Gerätekontext löschen Beachten Sie bitte, daß weder `Detach`, noch die Elementfunktionen `ReleaseOutputDC` und `ReleaseAttributeDC` (die die Werte von `m_hDC` und `m_hAttribDC` auf `NULL` zurücksetzt) das GDI-Gerätekontextobjekt löschen. Wurde das Gerätekontextobjekt mit der GDI-Funktion `::CreateDC` erstellt, müssen Sie `::DeleteDC` aufrufen. Der Aufruf dieser Funktion ist nicht erforderlich, wenn Sie die Zuweisung des einem CDC-Objekt zugeordneten Gerätekontextes nicht aufheben. Die CDC-Destruktorfunktion ruft `::DeleteDC` automatisch auf.

Die CDC-Klasse enthält die Elementfunktion `DeleteDC`, die ebenfalls für die Aufhebung der Zuweisung und zum Löschen des Gerätekontextes verwendet werden kann. Die Funktion sollte jedoch lediglich dann aufgerufen werden, wenn der Gerätekontext mit der `CreateDC`-Elementfunktion erzeugt wurde.

Kompatiblen Gerätekontext erzeugen `CreateCompatibleDC` ist eine weitere Funktion, die ein Gerätekontextobjekt erstellt. Diese Funktion generiert einen Speichergerätekontext, der kompatibel mit einem bereits vorhandenen Gerätekontext ist. Anwendungen verwenden die Funktion beispielsweise zusammen mit der `CClientDC`-Klasse, um einen Speichergerätekontext zu erzeugen, der mit dem Gerätekontext kompatibel ist, der den Client-Bereich des aktuellen Fensters repräsentiert:

```
CClientDC clientDC(&myWnd);
CDC memDC;
memDC.CreateCompatibleDC(&clientDC);
```

Nun können Funktionen wie `CDC::BitBlt` verwendet werden, um Pixelblöcke zwischen den beiden Gerätekontexten zu transferieren. Ähnliche Techniken werden häufig in Programmen eingesetzt, die eine fließende Animation darstellen. Indem ein Animationsrahmen im Spei-

chergerätekontext generiert wird und lediglich vollständige Rahmen zur Anzeige übertragen werden, können Sie Animationen ohne Flimmern erzeugen.

Eine statische CDC-Elementfunktion ist `CDC::FromHandle`. Diese Funktion ermittelt die Adresse eines CDC-Objekts (sofern dies besteht), dem ein Gerätekontext-Handle zugewiesen ist. Ist ein derartiges CDC-Objekt nicht vorhanden, wird ein temporäres Objekt erstellt. Die Funktion kann wie folgt aufgerufen werden: *CFromHandle*

```
CDC *pDC = CDC::FromHandle(hdc);
```

Beachten Sie, daß der von dieser Funktion zurückgegebene Zeiger nicht für jeden Zweck geeignet ist. Da dieser Zeiger auf ein CDC-Objekt verweist, das eventuell von einem anderen Abschnitt Ihrer Anwendung verwendet wird, wissen Sie nicht, wann das CDC-Objekt zerstört werden muß. Der von `CDC::FromHandle` zurückgegebene Zeiger ist somit nutzlos. Temporäre CDC-Objekte, die von `CDC::FromHandle` zurückgegeben wurden, werden von der `CDC::DeleteTmpMap`-Funktion gelöscht. Diese Funktion wird gewöhnlich von dem Inaktivitätszustandsbearbeiter in Ihrem `CWinApp`-Anwendungsobjekt aufgerufen.

Die Funktion `GetSafeHdc` gibt das `m_hDC`-Element des CDC-Objekts zurück. Diese Funktion kann auch mit `NULL`-Zeigern verwendet werden. Der folgende Programmcode ist zulässig und führt zu keiner Ausnahme: *GetSafeHdc*

```
CDC *pDC = NULL;
HDC hdc = pDC ->GetSafeHdc();
```

24.1.3 Gerätekontexttypen

Zeichengerätekontexte

Der Konstruktor und der Destruktor der `CPaintDC`-Klasse enthalten Aufrufe an `BeginPaint` und `EndPaint`. Die Klasse reagiert auf `WM_PAINT`-Nachrichten. *CPaintDC*

Beachten Sie bitte, daß die meisten Anwendungen kein `CPaintDC`-Objekt direkt erstellen. Die Standardimplementierung von `CView::OnPaint` erzeugt einen derartigen Gerätekontext und übergibt diesen der `OnDraw`-Elementfunktion der Klasse (die Funktion wird gewöhnlich überschrieben, um die Ansicht anwendungsspezifisch zu zeichnen).

Clientbereichsgerätekontexte

Die `CClientDC`-Klasse erstellt ein Gerätekontextobjekt, das dem Client-Bereich des angegebenen Fensters entspricht. Der Konstruktor und der Destruktor von `CClientDC` enthalten Aufrufe der Funktionen `GetDC` und `ReleaseDC`. `CClientDC`-Objekte werden gewöhnlich verwendet, um außerhalb einer `OnDraw`-Funktion in einen Gerätekontext zu zeichnen. *CClientDC*

`CClientDC`-Objekte werden besonders für die Koordinatenumwandlung benötigt. Bisweilen muß eine Anwendung auch dann logische Koordinaten in physikalische Koordinaten und umgekehrt umwandeln, wenn keine Zeichenoperation ausgeführt wird. In diesem Fall wird häufig ein `CClientDC`-Objekt erstellt und eine der Objektfunktionen zur Konvertierung der Koordinaten aufgerufen, wie nachfolgend dargestellt:

```
CClientDC dc(myView);
myView->OnPrepareDC(&dc);
dc.LPtoDP(&point);
```

Fenstergerätekontexte

CWindowDC Fenstergerätekontexte sind Client-Bereichskontexten sehr ähnlich. Sie werden durch die `CWindowDC`-Klasse repräsentiert. Der Konstruktor und der Destruktor von `CWindowDC` enthalten Aufrufe der Funktionen `GetWindowDC` und `ReleaseDC`.

Metadatei-Gerätekontexte

CMetaFileDC Die `CMetaFileDC`-Klasse repräsentiert Metadatei-Gerätekontexte. Mit Hilfe dieser Gerätekontexte ist das Zeichnen in eine Windows-Metadatei oder in eine der neuen erweiterten Metadateien möglich.

Metadatei-Gerätekontexte unterscheiden sich von anderen Gerätekontexten. Das `m_hAttribDC`-Element eines Metadatei-Gerätekontextes, das gewöhnlich so eingerichtet wird, daß es auf dasselbe Gerät wie `m_hDC` verweist, wird statt dessen auf `NULL` gesetzt. Aufrufe, die gewöhnlich Informationen über den Gerätekontext ermitteln, bleiben daher für ein Objekt vom Typ `CMetaFileDC` ergebnislos.

Es ist möglich, dem `m_hAttribDC`-Element einen Wert zuzuweisen. So könnten Sie der Variablen beispielsweise den Inhalt eines anderen von Ihnen erstellten Gerätekontextes zuordnen, der zum Beispiel den Bildschirm, Drucker oder ein anderes Ausgabegerät repräsentiert.

Metadatei-Gerätekontext erzeugen Das Erzeugen eines Metadatei-Gerätekontextes geschieht in zwei Schritten.

- Sie generieren zunächst das `CMetaFileDC`-Objekt.
- Im Anschluß daran rufen Sie die Elementfunktion `Create` oder `CreateEnhanced` auf.

Abhängig davon, ob Sie `Create` respektive `CreateEnhanced` einen Dateinamen übergeben, wird die Metadatei entweder auf einer Datei oder dem Speicher basieren. Eine speicherbasierende Metadatei ist lediglich temporär vorhanden.

Nachdem Sie in die Metadatei gezeichnet haben, schließen Sie das Metadatei-Objekt mit einem Aufruf von `CMetaFileDC::Close` oder `CMetaFi-`

`leDC::CloseEnhanced` (abhängig von dem Typ der Metadatei). Diese Funktionen geben ein Handle auf ein Metadatei-Objekt zurück, der beispielsweise während des Aufrufs von `CDC::PlayMetafile` verwendet werden kann. Diese Funktion überträgt den Inhalt der Metadatei in einen anderen Gerätekontext. Der Handle kann ebenfalls der Windows-Funktion `::CopyMetaFile` (oder `::CopyEnhMetaFile`) übergeben werden, um die Metadatei auf einem Datenträger zu speichern.

Wurde eine der Elementfunktionen `Close` oder `CloseEnhanced` aufgerufen, kann das `CMetaFileDC`-Objekt gelöscht werden. Benötigen Sie den mit der Funktion `Close` oder `CloseEnhanced` ermittelten Handle nicht mehr, löschen Sie das Windows-Metadatei-Objekt mit `DeleteMetaFile` oder `DeleteEnhMetaFile`.

24.1.4 CDC-Attribute

Ein Gerätekontextobjekt verfügt über Attribute, die mit Hilfe einiger CDC-Elementfunktionen gesetzt oder ermittelt werden können. Attribute, die sich auf die Koordinatenumwandlung und die Abbildungsmodi beziehen, werden in dem nächsten Abschnitt erörtert. In diesem Abschnitt werden andere Attribute beschrieben.

Hintergrundfarbe	Die Hintergrundfarbe, die freie Bereiche in Linien, schraffierte Pinsel und den Hintergrund der Textzeichen ausfüllt, wird mit der <code>SetBkColor</code> -Funktion gesetzt. Die aktuelle Hintergrundfarbe kann mit <code>GetBkColor</code> ermittelt werden.
Hintergrundmodus	Der Hintergrundmodus, der bestimmt, ob der Hintergrund transparent oder ausgefüllt dargestellt wird, wird mit <code>SetBkMode</code> gesetzt. <code>GetBkMode</code> ermittelt den aktuellen Hintergrundmodus.
Zeichenmodus	Die <code>SetROP2</code> -Elementfunktion setzt den Zeichenmodus. Dieser Modus bestimmt, wie die Bits des Zeichenwerkzeugs mit den Bits auf der Zeichenoberfläche kombiniert werden. Der Standardzeichenmodus ist <code>R2_COPYPEN</code> . In diesem Modus werden die Pixel des Zeichenwerkzeugs über die bereits vorhandenen Pixel in der Geräte-Bitmap kopiert. Wenn Sie einen bestimmten Stift oder Pinsel verwenden, zeichnen Sie über die bereits auf der Gerätekontext-Zeichenoberfläche vorhandenen Zeichnungen.

Weitere Zeichenmodi, die mit `SetROP2` gesetzt werden können, sind `R2_BLACK` (das Ziel-Pixel ist immer schwarz), `R2_NOTCOPYPEN` (das Ziel-Pixel wird in der invertierten Zeichenwerkzeugfarbe dargestellt) und `R2_XORPEN` (die Farbe des Ziel-Pixels wird aus einer OR-Operation des Ziel-Pixels mit dem Pixel des Zeichenwerkzeugs berechnet).

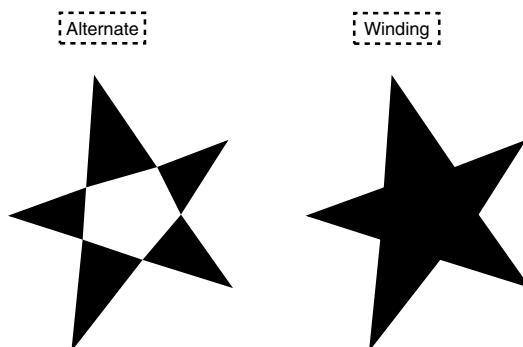
Die Zeichenmodi sind nicht auf diese voreingestellten Werte beschränkt. Die Zeichenmoduseinstellung kann eine beliebige binäre Operation zwischen den Pixeln der Zeichenoberfläche und denen des Zeichenwerkzeugs bestimmen.

Sie ermitteln den aktuellen Zeichenmodus mit einem Aufruf von `GetROP2`. Beachten Sie bitte, daß der Zeichenmodus lediglich für Rastergeräte gesetzt werden kann und keine Auswirkung auf Vektorgeräte, wie zum Beispiel Plotter, hat.

Füllmodus für Polygone

Die `SetPolyFillMode`-Funktion bestimmt den Füllmodus für Polygone. Der Unterschied zwischen den Füllmodi `ALTERNATE` und `WINDING` ist in Abbildung 24.2 dargestellt. Der aktuelle Füllmodus wird mit `GetPolyFillMode` ermittelt.

Abbildung 24.2:
Füllmodi



24.1.5 Koordinatenumwandlung und Ansichten

Die Koordinaten der meisten Grafikoperationen sind logische Koordinaten. Diese werden mit Hilfe der Koordinatenumwandlung in Gerätekoordinaten konvertiert.

Die Umwandlung definiert eine lineare Beziehung zwischen dem logischen und dem physikalischen Koordinatenraum. Sie weist dem Ursprung des logischen Koordinatenraums den Ursprung des physikalischen Koordinatenraums zu. Außerdem werden logischen Koordinaten physikalische Koordinaten zugeordnet. Die Umwandlung in der horizontalen Richtung kann von der Umwandlung in der vertikalen Richtung unabhängig sein.

Auf einem Rastergerät, wie zum Beispiel dem Bildschirm oder Drucker, werden die Gerätekoordinaten in Pixeln angegeben. Der oberen linken Ecke sind die Koordinaten [0,0] zugewiesen. Die horizontalen Koordinaten werden von links nach rechts größer. Die vertikalen Koordinaten nehmen von oben nach unten zu.



Windows definiert verschiedene Abbildungsmodi. Diese Modi werden mit der `SetMapMode`-Elementfunktion gesetzt.

Abbildungsmodus mit `SetMapMode` setzen

Eine der vordefinierten Abbildungsmodi ist `MM_TEXT`. Dieser Modus wandelt logische Koordinaten in physikalische Koordinaten um. Andere Abbildungsmodi kehren die Richtung der horizontalen Koordinaten um, so daß diese von unten nach oben größer werden. Mehr zu den verschiedenen Abbildungsmodi erfahren Sie in Kapitel 11 Zeichnen und Gerätekontexte.

Die folgenden Formeln beschreiben, wie Gerätekoordinaten (D_x und D_y) von logischen Koordinaten (L_x und L_y) und umgekehrt abgeleitet werden. Dazu werden die Ursprungswerte des Viewports (x_{VO} und y_{VO}) und Fensters (x_{WO} und y_{WO}) sowie deren Größenwerte (x_{WE} , y_{WE} , x_{VE} und y_{VE}) verwendet:



$$\begin{aligned} D_x &= (L_x - x_{WO}) * x_{VE} / x_{WE} + x_{VO} \\ D_y &= (L_y - y_{WO}) * y_{VE} / y_{WE} + y_{VO} \\ L_x &= (D_x - x_{VO}) * x_{WE} / x_{VE} + x_{WO} \\ L_y &= (D_y - y_{VO}) * y_{WE} / y_{VE} + y_{WO} \end{aligned}$$

Die CDC-Klasse stellt die Koordinatenumwandlungsfunktionen `DPtoLP` und `LPtoDP` zur Verfügung, die zum Konvertieren von logischen Koordinaten in physikalische Koordinaten und umgekehrt verwendet werden können. Zu beiden Funktionen bestehen überladene Versionen,

CDC-Umwandlungsfunktionen

die für Punkte, Rechtecke und `SIZE`-Objekte oder MFC-Klassen verwendet werden können, die diese Objekte enthalten (`CPoint`, `CRect`, `CSize`).

OLE-Unterstützung Zusätzliche Umwandlungsfunktionen sind `DpToHIMETRIC`, `HIMETRICToDP`, `LpToHIMETRIC` und `HIMETRICToLP`. Diese Funktionen sind besonders für OLE-Anwendungen geeignet. OLE-Objekte sind gewöhnlich in `HIMETRIC`-Einheiten unterteilt. Die genannten Funktionen wandeln diese Einheiten direkt in physikalische oder logische Koordinaten um.



WindowsNT

Die CDC-Klasse unterstützt nicht die unter Windows NT angebotene World-Koordinatenumwandlung. Sollen Anwendungen diese Art der Umwandlung ausführen, muß die Windows-Funktion `SetWorldTransform` direkt aufgerufen werden.

Die Koordinatenumwandlung wird umfassend in Ansichten verwendet (also in Klassen, die von `CView` abgeleitet werden). Die Elementfunktion `OnPrepareDC` wird in diesen Klassen verwendet, um die Koordinatenumwandlung für die aktuelle Ansicht einzurichten. In Ansichten, die beispielsweise Bildlaufleisten verwenden, wird `OnPrepareDC` vom Anwendungsrahmen verwendet, um den Fensterursprung und/oder den Viewport-Ursprung auf den Bereich zu verschieben, der in dem Client-Bereich der Ansicht über die Bildlaufleisten ausgewählt wurde. Anwendungen, die Features, wie zum Beispiel das Vergrößern von Bereichen, verwenden möchten, überschreiben dazu `CView::OnPrepareDC` und ändern die Größe des Fensters oder Viewports.

24.1.6 Einfache Zeichenfunktionen

Die CDC-Klasse bietet einige Elementfunktionen an, die den Low-Level-GDI-Zeichenoperationen entsprechen. Dazu zählen die Funktionen

- ➔ `FillRect` (füllt ein Rechteck mit einem bestimmten Pinsel),
- ➔ `FillSolidRect` (füllt ein Rechteck mit der angegebenen Farbe),
- ➔ `FrameRect` (zeichnet den Rahmen des Rechtecks) und
- ➔ `InvertRect` (invertiert den Inhalt des Rechtecks).

Funktionen, die Bereiche als Parameter akzeptieren, sind

- ➔ `FillRgn`, `FrameRgn` und `InvertRgn`.

Weitere Funktionen sind

- ➡ `DrawIcon` (zeichnet ein Symbol) und
- ➡ `DrawDragRect` (löscht und zeichnet ein Rechteck, das verschoben werden kann).

Andere einfache Zeichenfunktionen zeichnen Steuerelemente in verschiedenen Zuständen (selektiert und nicht selektiert) und mit unterschiedlichen Randeinstellungen.

24.1.7 GDI-Objekte selektieren

Eine große Anzahl verschiedener Funktionen, die auf Gerätekontexten ausgeführt werden, erfordern zunächst die Selektion eines GDI-Objekts in dem Gerätekontext. Verwenden Sie dazu die `CDC::SelectObject`-Elementfunktion. Zu dieser Funktion bestehen verschiedene überladene Versionen, die das Selektieren eines Objekts vom Typ `CPen`, `CBrush`, `CFont`, `CBitmap` oder `CRgn` ermöglichen.

<code>CPen</code>	Selektieren Sie in einem Gerätekontext einen Stift für Funktionen, die Linien zeichnen. Dazu zählen sowohl einfache Zeichenfunktionen (wie zum Beispiel <code>Line</code> , <code>Arc</code>) als auch Funktionen, die Figuren darstellen, deren Konturen mit dem aktuellen Stift gezeichnet werden.
<code>CBrush</code>	Selektieren Sie einen Pinsel, wenn Sie eine Figur zeichnen möchten (beispielsweise <code>Ellipse</code> , <code>Rectangle</code>). Der Pinsel füllt die Innenfläche der Figur.
<code>CFont</code>	Möchten Sie Text in einer bestimmten Schriftart in einen Gerätekontext zeichnen, selektieren Sie ein Objekt vom Typ <code>CFont</code> .
<code>CBitmap</code>	Um einen Speichergerätekontext verwenden zu können, müssen Sie ein <code>CBitmap</code> -Objekt darin selektieren. Dieses Objekt repräsentiert entweder eine monochrome Bitmap oder eine Bitmap, die kompatibel mit dem Gerätekontext ist.
<code>CRgn</code>	Die Auswahl eines <code>CRgn</code> -Objekts in einem Gerätekontext setzt den Clipping-Bereich des Kontextes auf den angegebenen Abschnitt. Die Selektion entspricht einem Aufruf der <code>CDC::SelectClipRgn</code> -Elementfunktion.

Häufig müssen Sie einen Gerätekontext in den vorherigen Zustand zurücksetzen, nachdem Sie ihn verwendet haben. Auch das zuvor ausgewählte GDI-Objekt muß wieder selektiert werden. Sie können dazu den Rückgabewert von `SelectObject` speichern (der gewöhnlich ein Zeiger

*Gerätekontext
zurücksetzen*

auf ein zuvor selektiertes `CPen`-, `CBrush`-, `CFont`- oder `CBitmap`-Objekt ist). Verwenden Sie diesen Wert im Anschluß daran in einem Aufruf von `SelectObject`. Alternativ dazu können Sie die Elementfunktionen `SaveDC` und `RestoreDC` verwenden.



In jedem Fall sind Sie für das Löschen der von Ihnen erstellten GDI-Objekte zuständig, wenn Sie diese nicht mehr benötigen.

Eine Variante von `SelectObject` ist `SelectStockObject`. Diese CDC-Elementfunktion ermöglicht Ihnen, ein GDI-Standardobjekt in einem Gerätekontext zu selektieren.

24.1.8 Linien und Figuren

Die CDC-Klasse stellt einige Zeichenfunktionen zur Verfügung, die den Windows-GDI-Zeichenfunktionen entsprechen. Zu den grundlegenden Zeichenfunktionen zählen solche, die verschiedene Linien (Geraden und Kurven) und Figuren zeichnen.

- MoveTo* Eine große Anzahl der Linienfunktionen nutzen das Konzept der aktuellen Position in dem Gerätekontext. Die aktuelle Position ist ein Koordinatenpaar, das gewöhnlich den Endpunkt der letzten Zeichenoperation kennzeichnet. Sie wird mit der `MoveTo`-Elementfunktion gesetzt und mit der `GetCurrentPosition`-Elementfunktion ermittelt.
- LineTo* Linienfunktionen verwenden den aktuellen Stift zum Zeichnen einer Linie. Möchten Sie eine gerade Linie zeichnen, verwenden Sie die Elementfunktionen `MoveTo` und `LineTo`. Rufen Sie `Arc` oder `ArcTo` auf, um einen elliptischen Kreisbogen zu zeichnen. Die Richtung des Kreisbogens wird mit der `SetArcDirection`-Elementfunktion gesteuert. (Verwenden Sie `GetArcDirection`, um die aktuelle Einstellung zu ermitteln).
- Polyline* Die Funktionen `Polyline` und `PolylineTo` zeichnen miteinander verbundene Liniensegmente. `PolyPolyline` zeichnet mehrere Polylinien in einer einzigen Operation. Windows kann ebenfalls Bézier-Kurven zeichnen. Verwenden Sie dazu `PolyBezier` oder `PolyBezierTo`. Schließlich haben Sie die Möglichkeit, eine Serie verschiedener Liniensegmente und Bézier-Kurven mit `PolyDraw` zu zeichnen.
- PolyBezier*
- Figuren* Figurenfunktionen sind `Rectangle`, `RoundRect`, `Ellipse`, `Chord`, `Pie` und `Polygon`. Die Figuren, die von diesen Funktionen generiert werden, sind in Abbildung 24.4 aufgeführt. `PolyPolygon` ermöglicht das Zeichnen mehrerer Polygone in einer einzigen Operation.
- PaintRgn* Die `PaintRgn`-Funktion zeichnet einen Bereich mit dem aktuellen Pinsel.

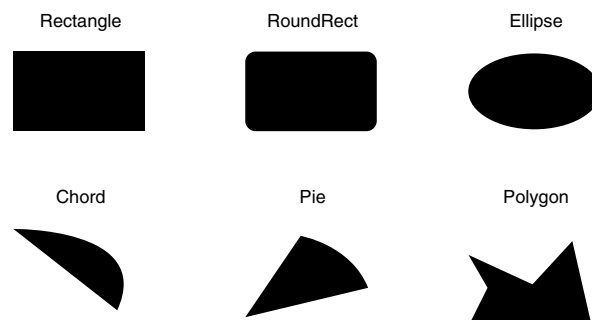


Abbildung 24.3:
Einige Grundfi-
guren

Die `DrawFocusRect` stellt einen Rahmen um ein Objekt dar, der anzeigt, daß dieses Objekt fokussiert ist. Der Fokusrahmen wird unter Verwendung der logischen `OR`-Funktion gezeichnet. Ein erneuter Aufruf von `DrawFocusRect` löscht daher den Rahmen. Beachten Sie bitte, daß Sie den Fokusrahmen zunächst löschen müssen, bevor Sie einen Bildlauf durchführen. Zeichnen Sie den Rahmen erneut, nachdem der Bildlauf ausgeführt wurde.

DrawFocus

24.1.9 Bitmaps und Bildlauf

Einige Elementfunktionen der `CDC`-Klasse führen bitweise Operationen auf Pixeldarstellungen und Bitmaps aus.

<code>SetPixel</code>	Die wohl einfachste Pixel-Operation ist <code>SetPixel</code> , die einen Pixel an der angegebenen logischen Koordinate in der angegebenen Farbe einfärbt. Die aktuelle Farbe eines Pixels kann mit <code>GetPixel</code> ermittelt werden. Eine etwas schnellere Variante von <code>SetPixel</code> ist <code>SetPixelV</code> . Diese Version der Funktion gibt nicht die gegenwärtige Farbe des Pixels zurück.
<code>BitBlt</code>	Die <code>BitBlt</code> -Elementfunktion verschiebt einen rechteckigen Bereich an eine andere Position. <code>BitBlt</code> kann ebenfalls dazu verwendet werden, Pixelblöcke zwischen Gerätekontexten zu transferieren. Sie haben beispielsweise die Möglichkeit, Pixel von dem Bildschirm zu einer kompatiblen Speicher-Bitmap oder umgekehrt zu verschieben.
<code>StretchBlt</code>	Eine Variante von <code>BitBlt</code> ist <code>StretchBlt</code> . Diese Funktion transferiert ebenfalls Pixelblöcke von einer Position zu einer anderen. Sie komprimiert oder vergrößert den Pixelblock jedoch, um diesen in das Zielrechteck einzupassen. Der Vergrößerungsmodus (die Methode, die zum Löschen und/oder Hinzufügen von Pixeln

	verwendet wird) wird mit <code>SetStretchMode</code> (<code>GetStretchMode</code>) und <code>SetColorAdjustment</code> (<code>GetColorAdjustment</code>) gesteuert.
<code>PatBlt</code>	Die <code>PatBlt</code> -Elementfunktion kombiniert die Pixel des Geräts mit den Pixeln des selektierten Pinsels. Sie verwendet dazu eine bitweise logische Operation. Die <code>MaskBlt</code> -Operation kombiniert in einer bitweisen logischen Operation die Quellen- und Ziel-Bitmaps mit einer Masken-Bitmap.
<code>FloodFill</code>	Um einen Bereich in einer Bitmap mit dem aktuellen Pinsel zu füllen, rufen Sie die Elementfunktion <code>FloodFill</code> oder <code>ExtFloodFill</code> auf.
Bildlauf	Möchten Sie einen Bildlauf für einen Bereich innerhalb eines Gerätekontextes ausführen, verwenden Sie dazu die <code>ScrollDC</code> -Elementfunktion. Diese Funktion informiert Sie außerdem über die Bereiche, die nicht von der Bildlauf-Operation überdeckt wurden. Verwenden Sie diese Informationen, um diese Bereiche erneut zu zeichnen. Soll ein Bildlauf für den gesamten Client-Bereich eines Fensters ausgeführt werden, rufen Sie die <code>CWnd::ScrollWindow</code> -Funktion auf.

24.1.10 Text- und Schriftartenfunktionen

Für die Ausgabe von Text stehen den Anwendungen verschiedene Textausgabefunktionen und Funktionen zur Bearbeitung von Schriftarten zur Verfügung.

Textausgabe	<code>CDC::TextOut</code> ist eine einfache Textausgabefunktion. Sie ordnet eine Zeichenfolge in der aktuellen Schriftart an der angegebenen Position an. Die Variante <code>CDC::ExtTextOut</code> gibt die Zeichenfolge in einem bestimmten Rechteck aus.
Tabulatoren	<code>TabbedTextOut</code> ordnet Tabstops in dem auszugebenden Text an, die in einem Array definiert sind.
Farbe	Die Farbe des Textes wird mit <code>SetTextColor</code> bestimmt (verwenden Sie <code>GetTextColor</code> , um die aktuelle Einstellung zu ermitteln).
Ausrichtung	Die horizontale und vertikale Textausrichtung wird mit <code>SetTextAlign</code> (<code>GetTextAlign</code>) festgesetzt. Mit dieser Funktion können Sie ebenfalls bestimmen, daß Textausgabefunktionen die aktuelle Position (die mit Funk-

tionen, wie zum Beispiel `MoveTo`, gesetzt wird) anstelle der in den Funktionsaufrufen angegebenen Koordinaten für die Textpositionierung verwenden sollen.

- Blockgröße** Die Größe eines Textblocks kann ermittelt werden, ohne diesen Block zu zeichnen. Die Funktion `GetTextExtent` berechnet die Breite und Höhe einer Textzeile mit Hilfe eines Attribute-Gerätekontextes. Um dieselbe Berechnung unter Verwendung des Ausgabe-Gerätekontextes auszuführen, verwenden Sie `GetOutputTextExtent`. Die Funktionen `GetTabbedTextExtent` und `GetOutputTabbedTextExtent` führen diese Berechnung für einen Text aus, der darzustellende Tabulatoren enthält.
- Blocksatz** Die Funktion `SetTextJustification` wird zusammen mit `GetTextExtent` aufgerufen, um einen Text im Blocksatz anzuordnen. `SetTextJustification` ordnet Füllzeichen (gewöhnlich Leerzeichen) in einem Text an. `SetTextCharacterSpacing` ist eine ähnliche Funktion (`GetTextCharacterSpacing` ermittelt die aktuelle Einstellung).
- Mehrzeilig** Eine komplexe Textausgabefunktion ist `DrawText`. Diese Funktion gibt mehrzeiligen Text aus. Beachten Sie bitte, daß `DrawText` nicht für Windows-Standard-Metadateien, sondern ausschließlich für erweiterte Metadateien verwendet werden kann.
- Grauschrift** Die `GrayString`-Funktion stellt Text abgeheilt (grau) dar.
- Font** Informationen über die aktuelle Schriftart können mit `GetTextFace` (der Name der Schriftart), `GetTextMetrics` (ermittelt eine `TEXTMETRIC`-Struktur, die Informationen über die gegenwärtig im Attribute-Gerätekontext selektierte Schriftart enthält) und `GetOutputTextMetrics` (ermittelt eine `TEXTMETRIC`-Struktur, die Informationen über die gegenwärtig im Ausgabe-Gerätekontext selektierte Schriftart enthält) ermittelt werden.

Weitere CDC-Elementfunktionen bearbeiten skalierbare Schriftarten (True-Type-Schriftarten) und Informationen, die über diese Schriftarten ermittelt werden können.

24.1.11 Clipping-Operationen

Ein besonders wichtiges GDI-Feature ist das Anordnen der Ausgabe in einem Rechteck oder bestimmten Bereich. Clipping wird beispielsweise dazu verwendet, lediglich Bereiche eines Fensters erneut zu zeichnen, die nicht von anderen Fenstern bedeckt wurden.

Anwendungen können das Clipping explizit über verschiedene CDC-Elementfunktionen verwenden, die Mantelfunktionen für ähnliche GDI-Funktionen sind. Dazu zählen `SelectClipRgn`, `ExcludeClipRect`, `ExcludeUpdateRgn`, `IntersectClipRect` und `OffsetClipRgn`.

Um das Rechteck zu ermitteln, das den vollständigen Clipping-Bereich optimal umschließt, rufen Sie `GetClipBox` auf.

Möchten Sie bestimmen, ob ein Punkt oder Bereiche eines Rechtecks innerhalb des Clipping-Bereichs liegen, verwenden Sie die Elementfunktion `PtVisible` oder `RectVisible`.

Windows verfügt außerdem über ein Begrenzungsrechteck, in dem Informationen über die Begrenzungen der nachfolgenden Zeichenoperationen gesammelt werden. Mit den Elementfunktionen `SetBoundsRect` und `GetBoundsRect` greifen Sie auf das Begrenzungsrechteck zu.

24.1.12 Drucken

Wenngleich zwischen dem Drucken und dem Zeichnen in andere Gerätekontexte kein Unterschied besteht, bietet die CDC-Klasse einige Drucker-Elementfunktionen an, die bestimmte Aspekte des Druckens steuern.

Das Drucken des gesamten Dokuments und einzelner Seiten wird von den Elementfunktionen `StartDoc`, `StartPage`, `EndPage` und `EndDoc` gesteuert. Um den Druckvorgang abubrechen (und alle Daten zu löschen, die seit dem letzten Aufruf von `StartDoc` an den Gerätekontext gesendet wurden), rufen Sie `AbortDoc` auf. Beachten Sie bitte, daß Ihre Anwendung nicht die Funktion `EndDoc` oder `AbortDoc` aufrufen sollte, wenn der Druck unterbrochen wurde oder der Drucker-Gerätetreiber einen Fehler zurückgibt.

Verwenden Sie die `SetAbortDoc`-Elementfunktion, um eine Rückruffunktion zu erstellen, die Windows nach einem Abbruch oder nach dem Beenden des Druckauftrags aufruft. Die `QueryAbort`-Elementfunktion fragt diese Rückruffunktion ab, um zu ermitteln, ob der Druckvorgang abgebrochen werden soll.

Auf spezifische Gerätetreiber-Features kann über die `Escape`-Elementfunktion zugegriffen werden. Da Win32 weitaus mehr Drucker-Steuerungsfunktionen anbietet, ist der Nutzen dieser Funktion, verglichen mit früheren Windows-Versionen, relativ gering.

24.1.13 Pfad-Funktionen

Die `CDC`-Klasse stellt einige Elementfunktionen zur Verfügung, die eine Pfad-Funktionalität enthalten. Ein Pfad – eine komplexe Figur, die mit mehreren GDI-Funktionsaufrufen erstellt wurde – wird erstellt, indem die `BeginPath`-Elementfunktion und die entsprechenden Zeichenfunktionen aufgerufen werden. Zuletzt rufen Sie `EndPath` auf. Diese Funktion selektiert den Pfad automatisch zur nachfolgenden Bearbeitung in dem Gerätekontext.

Funktionen, die der Bearbeitung von Pfaden dienen, sind `CloseFigure`, `FlattenPath`, `StrokePath` und `WidenPath`. Der Pfad kann mit `StrokePath` in den Gerätekontext gezeichnet werden. Um die Innenfläche des Pfades zu zeichnen, verwenden Sie `FillPath`. `StrokeAndFillPath` zeichnet sowohl die Kontur des Pfades als auch die Innenfläche. Die Funktion verwendet dazu den aktuellen Stift und den aktuellen Pinsel.

Ein Pfad kann mit `SelectClipPath` in einen Clipping-Bereich konvertiert werden.

24.2 Unterstützung von GDI-Objekten in der MFC

Einige GDI-Zeichenfunktionen verwenden GDI-Objekte, wie zum Beispiel Stifte, Pinsel oder Schriftarten. Die MFC-Bibliothek verfügt über einige Mantelklassen, in denen die Funktionalität dieser GDI-Objekte implementiert ist.

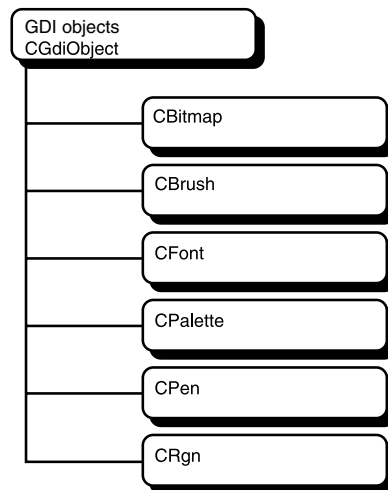
Alle GDI-Objektklassen werden von `CGdiObject` abgeleitet. Abbildung 24.5 stellt die GDI-Objektklassen der MFC dar.

Die `GdiObject`-Klasse unterstützt GDI-Objekte über verschiedene Elementfunktionen. Die Funktionen `Attach` und `Detach` weisen einem von `CGdiObject` abgeleiteten Objekt ein GDI-Objekt zu oder heben diese Zuweisung auf. Der in der `m_hObject`-Elementvariablen gespeicherte Handle des Objekts wird mit der »sicheren« Funktion `GetSafeHandle` ermittelt. (Diese Funktion kann außerdem mit `CGdiObject`-Zeigern verwendet werden, die auf kein Objekt verweisen.) Ein Zeiger auf ein `CGdiObject`, der einem Windows-GDI-Objekt-Handle entspricht, kann mit einem Aufruf der statischen Elementfunktion `FromHandle` ermittelt werden. `GetObjectType` gibt über den Typ des GDI-Objekts Aufschluß.

Die `CreateStockObject`-Elementfunktion erstellt einen Standardstift, einen Standardpinsel, eine Standardschriftart oder eine Standardpalette.

Beachten Sie bitte, daß diese Funktion mit einem von `CGdiObject` abgeleiteten Objekt der entsprechenden Klasse (`CPen`, `CBrush`, `CFont` oder `CPalette`) aufgerufen werden sollte.

Abbildung 24.4:
GDI-Objekt-
klassen



Die `UnrealizeObject`-Elementfunktion richtet den ursprünglichen Pinsel respektive die anfängliche Palette wieder ein.

`DeleteObject` löscht das GDI-Objekt, das dem von `CGdiObject` abgeleiteten MFC-Objekt zugewiesen ist. `DeleteTempMap` wird gewöhnlich von dem Inaktivitätszustandsbearbeiter des `CWinApp`-Objekts Ihrer Anwendung aufgerufen, um alle temporären `CGdiObject`-Objekte zu löschen, die von der `FromHandle`-Elementfunktion erstellt wurden.

24.2.1 Stifte

Die `CPen`-Klasse unterstützt GDI-Stiftobjekte in der MFC. Ein Stift kann in einem oder in zwei Schritt(en) erzeugt werden.

1. Rufen Sie die überladene Version des `CPen`-Konstruktors auf, um einen Stift in einem Schritt zu erstellen. Möchten Sie beispielsweise einen linierten schwarzen Stift verwenden, deklarieren Sie das Stiftobjekt wie folgt:

```
CPen myPen(PS_DASH, 0, RGB(0, 0, 0));
```

Alternativ dazu können Stifte in zwei Schritten generiert werden, indem

1. zunächst das MFC-CPen-Objekt unter Verwendung eines Konstruktors generiert wird, dem keine Parameter übergeben werden.
2. Anschließend wird `CreatePen` oder `CreatePenIndirect` aufgerufen, um ein entsprechendes GDI-Stiftobjekt wie in dem nachfolgenden Beispiel zu erzeugen:

```
CPen *pPen;  
pPen = new CPen;  
pPen ->CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
```

Um die `LOGPEN`-Struktur eines `CPen`-Objektes zu ermitteln, verwenden Sie die `GetLogPen`-Funktion. Sie können ein `CPen`-Objekt außerdem einem GDI-Funktionsaufruf übergeben, der einen Stift-Handle vom Typ `HPen` verlangt. Dies ist möglich, da die `CPen`-Klasse die `operator_HPEN`-Operatorfunktion definiert.

24.2.2 Pinsel

Die MFC unterstützt GDI-Pinsel über die `CBrush`-Klasse. Wie Stifte, können auch Pinsel in einem oder zwei Schritt(en) erstellt werden.

1. Um einen GDI-Pinsel während des Erstellens des `CBrush`-Objekts zu generieren, verwenden Sie eine der überladenen Versionen des `CBrush`-Konstruktors.

Möchten Sie beispielsweise einen gelben Pinsel verwenden, können Sie das `CBrush`-Objekt wie folgt definieren:

```
CBrush *pBrush;  
pBrush = new CBrush(RGB(255, 255, 0));
```

Eine weitere Möglichkeit besteht darin,

1. zunächst das `CBrush`-MFC-Objekt über den Konstruktor zu erstellen, ohne diesem Parameter zu übergeben.
2. Anschließend erstellen Sie das GDI-Pinselobjekt, indem Sie `CreateSolidBrush`, `CreateHatchBrush`, `CreatePatternBrush`, `CreateDIBPatternBrush`, `CreateSysColorBrush` oder `CreateBrushIndirect` aufrufen.

```
CBrush cyanBrush;  
cyanBrush.CreateSolidBrush(RGB(0, 255, 255));
```

Um die `LOGBRUSH`-Struktur eines `CBrush`-Objektes zu ermitteln, verwenden Sie die `GetLogBrush`-Funktion. Sie können ein `CBrush`-Objekt außerdem einem GDI-Funktionsaufruf übergeben, der eine Pinselzugriffsnummer vom Typ `HBrush` verlangt. Dies ist möglich, da die `CBrush`-Klasse die `operator_HBRUSH`-Operatorfunktion definiert.

24.2.3 Bitmaps

GDI-Bitmaps werden in der MFC über die Klasse `CBitmap` unterstützt.

Das Erstellen einer Bitmap geschieht in zwei Schritten.

1. Zunächst wird das MFC-`CBitmap`-Objekt erzeugt.
2. Anschließend rufen Sie eine der Initialisierungsfunktionen auf, um das GDI-Bitmap-Objekt zu generieren.

Zu den Initialisierungsfunktionen zählen `LoadBitmap`, `LoadOEMBitmap`, `LoadMappedBitmap`, `CreateBitmap`, `CreateBitmapIndirect`, `CreateCompatibleBitmap` und `CreateDiscardableBitmap`.

Um einen Zeiger auf eine `BITMAP`-Struktur zu ermitteln, die die dem `CBitmap`-Objekt zugewiesene GDI-Bitmap repräsentiert, rufen Sie die `GetBitmap`-Elementfunktion auf. Sie können `CBitmap`-Objekte außerdem anstelle des Handles vom Typ `HBITMAP` in GDI-Funktionsaufrufen verwenden, da eine `operator_HBITMAP`-Operatorfunktion besteht.

Die Bits einer Bitmap können mit `SetBitmapBits` und `GetBitmapBits` gesetzt oder ausgelesen werden. Sie haben außerdem die Möglichkeit, der Bitmap eine Breite und Höhe zuzuweisen (in `LOMETRIC`-Einheiten). Verwenden Sie dazu die Elementfunktion `SetBitmapDimension`. Diese Werte werden jedoch von `GetBitmapDimension` lediglich als Rückgabewerte verwendet.

Das folgende Beispiel demonstriert, wie eine Bitmap aus einer Resource (`IDB_BITMAP1`) geladen und in einem Fenster angezeigt werden kann:

```
void CMyView::OnDraw(CDC* pDC) {
    CMyDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    BITMAP bm;
    CBitmap bitmap;
    bitmap.LoadBitmap(IDB_BITMAP1);
    bitmap.GetObject(sizeof(bm), &bm);

    CDC speicherDC;
    speicherDC.CreateCompatibleDC(pDC);
    speicherDC.SelectObject(&bitmap);

    RECT rect;
    GetClientRect(&rect);
    pDC->StretchBlt(
        0, 0, rect.right - rect.left, rect.bottom - rect.top,
        &speicherDC, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
}
```

24.2.4 Schriftarten

Die MFC-Unterstützung logischer GDI-Schriftarten ist in der `CFont`-Klasse enthalten.

Das Erstellen einer Bitmap geschieht in zwei Schritten.

1. Zunächst wird das MFC-`CFont`-Objekt erzeugt.
2. Anschließend rufen Sie eine der Initialisierungsfunktionen auf, um das GDI-Schriftartenobjekt zu generieren.

Zu den Initialisierungsfunktionen zählen `CreateFont`, `CreateFontIndirect`, `CreatePointFont` und `CreatePointFontIndirect`.

Um einen Zeiger auf eine `LOGFONT`-Struktur zu ermitteln, verwenden Sie die `GetLogFont`-Elementfunktion. Sie können `CFont`-Objekte außerdem anstelle des Handles vom Typ `HFONT` in GDI-Funktionsaufrufen verwenden, da eine `operator_HFONT`-Operatorfunktion besteht.

24.2.5 Paletten

Die `CPalette`-Klasse unterstützt Paletten in der MFC. Wie Schriftarten und Bitmaps werden auch Paletten in zwei Schritten generiert.

1. Sie erstellen zunächst das `CPalette`-Objekt.
2. Im Anschluß daran rufen Sie eine Initialisierungsfunktion auf, um das zugrundeliegende GDI-Palettenobjekt zu erzeugen.

Zu diesen Initialisierungsfunktionen zählen `CreatePalette` und `CreateHalftonePalette`.

Palettenoperationen sind `AnimatePalette`, `GetNearestPaletteIndex`, `GetEntryCount`, `GetPaletteEntries`, `SetPaletteEntries` und `ResizePalette`.

`CPalette`-Objekte können in GDI-Funktionsaufrufen einen Handle vom Typ `HPALETTE` ersetzen, da die `CPalette`-Klasse die Operatorfunktion `operator_HPALETTE` definiert.

24.2.6 Bereiche

GDI-Bereiche werden in der MFC über die `CRgn`-Klasse unterstützt.

Ein Bereich wird erstellt, indem Sie

1. zunächst das `CRgn`-Objekt generieren und
2. anschließend eine der Initialisierungsfunktionen aufrufen.

Die große Anzahl der `CRgn`-Initialisierungsfunktionen ist in Tabelle 24.1 aufgeführt.

Tabelle 24.2:
CRgn-Initialisierungs-
funktionen

Elementfunktion	Beschreibung
CreateRectRgn	Erstellt einen rechteckigen Bereich.
CreateRectRgnIndirect	Erstellt einen Bereich aus einer RECT-Struktur.
CreateEllipticRgn	Erstellt einen elliptischen Bereich.
CreateEllipticRgnIndirect	Erstellt einen elliptischen Bereich aus einer RECT-Struktur.
CreatePolygonRgn	Erstellt einen Polygonbereich.
CreatePolyPolygonRgn	Erstellt einen Bereich, der aus mehreren Polygonen besteht (die möglicherweise nicht miteinander verbunden sind).
CreateRoundRectRgn	Erstellt einen Bereich in Form eines abgerundeten Rechtecks.
CombineRgn	Erstellt einen Bereich, der aus einer Kombination zwei verschiedener Bereiche besteht.
CopyRgn	Erstellt eine Kopie eines Bereiches.
CreateFromPath	Erstellt einen Bereich aus einem Pfad.
CreateFromData	Erstellt einen Bereich aus einer RGNDATA-Struktur und einer XFORM-Matrix.

Um zu überprüfen, ob zwei CRgn-Objekte einander gleichen, verwenden Sie die `EqualRgn`-Elementfunktion. Mit `GetRegionData` ermitteln Sie die RGNDATA-Struktur des CRgn-Objekts. Das Begrenzungsrechteck des Objekts wird mit `GetRgnBox` ermittelt.

Um einen Bereich auf ein bestimmtes Rechteck zu setzen, rufen Sie `SetRectRgn` auf. Mit Hilfe der Funktion `OffsetRgn` verschieben Sie einen Bereich.

Verwenden Sie die `PtInRegion`-Elementfunktion oder die Funktion `RectInRegion`, um zu ermitteln, ob der angegebene Punkt oder Abschnitte eines Rechtecks in dem Bereich liegen.

Ein CRgn-Objekt kann anstelle eines HRGN-Handles in GDI-Funktionsaufrufen eingesetzt werden, da eine `operator_HRGN`-Operatorfunktion besteht.

24.3 Zusammenfassung

Die GDI-Funktionalität (Graphics Device Interface) wird von den Klassen `CDC` (repräsentiert Gerätekontexte) und `CGdiObject` (repräsentiert GDI-Objekte) sowie den davon abgeleiteten Klassen angeboten.

Die von `CDC` abgeleiteten Klassen sind

- `CClientDC` (repräsentiert die Innenfläche eines Fensters),
- `CWindowDC` (repräsentiert ein Fenster) und
- `CPaintDC` (repräsentiert einen Gerätekontext während der Bearbeitung einer `WM_PAINT`-Nachricht).

Diese abgeleiteten Klassen unterscheiden sich lediglich darin von Ihrer Basisklasse, daß ihre Konstruktoren und Destruktoren Aufrufe enthalten, die entsprechende GDI-Gerätekontexte erstellen oder zerstören (zum Beispiel mit den Aufrufen `GetDC`, `ReleaseDC`, `BeginPaint` oder `EndPaint`). Im Gegensatz dazu wird ein `CDC`-Basisobjekt nicht automatisch einem GDI-Gerätekontext zugewiesen. Der Gerätekontext muß explizit mit `CreateDC` erstellt werden (oder dem Objekt mit der `AttachElement`-funktion zugewiesen werden).

Eine weitere von `CDC` abgeleitete Klasse ist

- `CMetafileDC`, die Metadatei-Gerätekontexte repräsentiert.

Ein `CDC`-Objekt verfügt über zwei GDI-Gerätekontext-Handles.

- Der Ausgabegerätekontext-Handle wird in Zeichenoperationen verwendet.
- Der Attributegerätekontext-Handle wird für Operationen benötigt, die Informationen über den Gerätekontext ermitteln.

Die beiden Handles sind gewöhnlich identisch. Eine Ausnahme bilden `CMetafileDC`-Objekte, in denen der Attribute-Gerätekontext auf `NULL` gesetzt wird.

Die `CDC`-Klasse umfaßt die GDI-Zeichenfunktionalität. Dazu zählen einfache Funktionen, die Linien, Figuren, Text, Schriftarten und Bitmaps zeichnen sowie Clipping-Operationen und den Bildlauf ausführen. Bereichsfunktionen und Pfadfunktionen zählen ebenfalls dazu. Die `CDC`-Klasse stellt außerdem verschiedene Abbildungsmodi zur Verfügung. Beachten Sie bitte, daß die Windows-NT-World-Koordinatenumwandlung nicht von der `CDC`-Klasse angeboten wird.

Die meisten Zeichenoperationen nutzen Zeichenwerkzeuge, die mit `SelectObject` oder `SelectStockObject` in dem Gerätekontext selektiert werden. Diese Werkzeuge sind GDI-Objekte, wie zum Beispiel Stifte, Pinsel, Paletten, Bitmaps, Schriftarten und Bereiche. Diese GDI-Objekte werden in der MFC über verschiedene von `CGdiObject` abgeleitete Klassen unterstützt. Die Klassen `CPen`, `CBrush`, `CFont`, `CBitmap`, `CPalette` und `CRgn` repräsentieren Stifte, Pinsel, Schriftarten, Bitmaps, Paletten und Bereiche.

Serialisierung: Datei- und Archivobjekte

Kapitel 25

Ein in der MFC-Bibliothek wichtiges Konzept ist die Serialisierung.

Die Serialisierung legt die von `CObject` abgeleiteten Objekte beständig ab. Die Serialisierung dient jedoch nicht ausschließlich dem Speichern und Laden von Daten in und aus einer Datei. `CObject`-Objekte können über die Serialisierung ebenfalls in die respektive aus der Zwischenablage oder über OLE an andere Anwendungen übertragen werden.

Die Serialisierung bildet eine Beziehung zwischen Objekten, die von der `CObject`-Klasse abgeleitet wurden, der `CArchive`-Klasse, die ein Archiv repräsentiert, und der `CFile`-Klasse, über die auf den physikalischen Speicher zugegriffen wird (Abbildung 25.1).

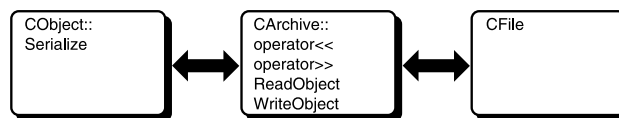


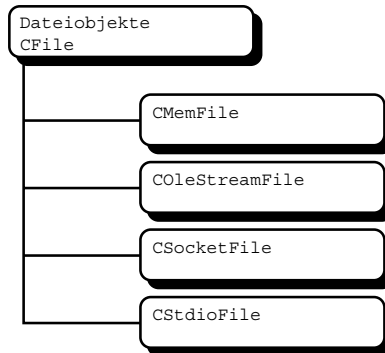
Abbildung 25.1:
Beziehung zwischen `CObject`,
`CArchive` und
`CFile`

Trotz dieser Beziehung übersteigt der Nutzen der `CFile`-Klasse den der `CObject`-Serialisierung. Der nächste Abschnitt erläutert die `CFile`-Klasse und zeigt einfache Anwendungsmöglichkeiten.

25.1 Die `CFile`-Klasse

`CFile` ist die Basisklasse für MFC-Dateidienste. `CFile` unterstützt die nicht gepufferte binäre Ein- und Ausgabe in Dateien. Über abgeleitete Klassen werden Textdateien, Speicherdateien und Windows-Sockets unterstützt. Die Hierarchie von `CFile` und den davon abgeleiteten Klassen ist in Abbildung 25.2 dargestellt.

Abbildung 25.2:
Die CFile-Klassenhierarchie



Ein wiederholt angesprochenes Thema, das MFC-Klassen betrifft, die als Mantelklassen für Windows-Objekte dienen, ist die Dualität des C++-Objekts. Ein `CFile`-Objekt ist nicht mit einem Dateiobjekt unter Windows identisch. Es repräsentiert lediglich dieses Objekt.

Das Erstellen eines `CFile`-Objekts bedingt nicht die Erzeugung des entsprechenden Dateiobjekts (erstellen Sie ein `CFile`-Objekt, ist die entsprechende Datei möglicherweise bereits geöffnet).

In einem `CFile`-Objekt enthält die Elementvariable `m_hFile` (gewöhnlich) den Handle der Datei, die das `CFile`-Objekt repräsentiert. Dieser Handle muß in dem `CFile`-Konstruktor oder über eine explizit aufgerufene Initialisierungsfunktion initialisiert werden.



Der Datei-Handle `m_hFile` ist ein Win32-Handle. Verwechseln Sie diesen bitte nicht mit den Datei-Handles oder Dateibeschreibungen in den Low-Level-C/C++-Dateibibliotheken für die Ein- und Ausgabe.

25.1.1 Dateiiinitialisierung

Dateien öffnen

Das Erstellen eines `CFile`-Objekts geschieht entweder in einem oder in mehreren Schritt(en).

1. Möchten Sie das Objekt in einem Schritt erstellen, verwenden Sie den Konstruktor, der einen Handle auf eine bereits geöffnete Datei oder auf eine Datei akzeptiert, die mit dem `CFile`-Objekt geöffnet werden soll.

Alternativ dazu können Sie

1. den Konstruktor verwenden, dem keine Parameter übergeben werden, und
2. die `Open`-Elementfunktion aufrufen.

Öffnen Sie eine Datei mit dem `CFile`-Konstruktor oder der `Open`-Elementfunktion, können Sie mehrere Flags angeben. Dateien können im Text- oder Binärmodus zum Lesen oder Schreiben geöffnet werden. Sowohl der Konstruktor als auch die `Open`-Elementfunktion können ebenfalls Dateien erstellen. Zusätzliche Modus-Flags bestimmen die gemeinsame Dateiverwendung und andere Attribute.

Eine geöffnete Datei wird mit der `Close`-Elementfunktion geschlossen. Die `Abort`-Elementfunktion dient ebenso diesem Zweck. Im Gegensatz zu `Close` schließt `Abort` die Datei in jedem Fall und ignoriert alle Fehler. *Dateien schließen*

25.1.2 Lesen aus und Schreiben in ein CFile-Objekt

Der Lese- und Schreibvorgang wird für ein `CFile`-Objekt mit den Elementfunktionen `Read` und `Write` ausgeführt. Natürlich muß die Datei in dem entsprechenden Modus geöffnet sein, damit die Operation erfolgreich durchgeführt werden kann.

Flush	Die <code>Flush</code> -Elementfunktion schreibt alle gepufferten Daten in die Datei.
Seek	Der Direktzugriff auf Dateien geschieht mit der <code>Seek</code> -Elementfunktion. Diese setzt die Position für den nächsten Lese- oder Schreibvorgang innerhalb einer Datei. Zwei Varianten der Funktion, die mit <code>SeekToBegin</code> und <code>SeekToEnd</code> bezeichnet sind, setzen die Position auf den Anfang respektive auf das Ende der Datei. Die aktuelle Position kann mit <code>GetPosition</code> ermittelt werden.
GetLength	Die Länge einer Datei kann mit <code>GetLength</code> ermittelt werden. Die <code>SetLength</code> -Funktion setzt die Länge der Datei. Die Datei wird dazu mit nicht initialisierten Daten aufgefüllt oder gekürzt, sofern dies erforderlich ist.

25.1.3 Dateiverwaltung

Zwei statische `CFile`-Elementfunktionen können verwendet werden, ohne ein `CFile`-Objekt zu erstellen:

- ➡ `CFile::Rename` benennt eine Datei um.
- ➡ `CFile::Remove` löscht eine Datei.

Der Status einer Datei wird mit `GetStatus` ermittelt. Diese Funktion setzt den Wert eines `CFileStatus`-Objekts. `GetStatus` verfügt außerdem über eine statische Variante, die den Status einer nicht geöffneten Datei ermittelt. *Dateistatus*

Um den Status einer Datei mit einem `CFileStatus`-Objekt zu setzen, rufen Sie die `SetStatus`-Elementfunktion auf.

25.1.4 Fehlerbearbeitung

Dateioperationen können inkorrekt ausgeführt werden. Während einige `CFile`-Elementfunktionen (zum Beispiel `Open`) aufgetretene Fehler in ihren Rückgabewerten anzeigen, lösen andere Funktionen eine Ausnahme aus, um über die Fehlerursache zu informieren. Die Ausnahme ist immer vom Typ `CFileException`. Bearbeiten Sie die Fehler mit einem Programmcode, der dem folgenden gleicht:

```
CFile myFile("filename.txt", CFile::modeWrite)
try
{
    CFile.Write("Data", 4);
    CFile.Close();
}
catch (CFileException *e)
{
    if (e->m_cause == CFileException::diskFull)
        cout << "The disk is full!";
    else
    {
        // Andere Fehler bearbeiten
    }
    e->Delete();
}
```

25.1.5 Sperren

Die `CFile`-Klasse unterstützt auch das Sperren von Dateien oder Dateibereichen. Der Bereich einer Datei, der über die Angabe der Startposition und die Anzahl der betroffenen Bytes bestimmt wird, kann mit der Elementfunktion `LockRange` gesperrt werden. Mit `UnlockRange` lösen Sie die Sperrung des Bereichs auf.

Das Sperren mehrerer Bereiche ist ebenfalls möglich. Sie können jedoch keine sich überlappenden Bereiche sperren. Ein Aufruf von `UnlockRange` betrifft den zuvor mit `LockRange` gesperrten Bereich. Sperren Sie beispielsweise zwei Bereiche einer Datei mit `LockRange`, müssen Sie `UnlockRange` auch dann zweimal aufrufen, wenn die Bereiche aneinander grenzen.

Der Versuch, einen bereits gesperrten Bereich zu sperren, resultiert in einem Fehler.

25.1.6 Verwenden von CFile in einer einfachen Anwendung

Die CFile-Klasse kann in vielen Situationen angewendet werden. Auch Konsolenanwendungen können diese Klasse nutzen. Solch ein einfaches Programm ist in Listing 25.1 aufgeführt. Kompilieren Sie die Anwendung über die Kommandozeile mit der Anweisung

```
CL /MT HELLO.CPP
```

```
#include <afx.h>

#define HELLOSTR "Hello, World!\n"
#define HELLOLEN (sizeof(HELLOSTR)-1)

void main(void)
{
    CFile file((int)GetStdHandle(STD_OUTPUT_HANDLE));
    file.Write(HELLOSTR, HELLOLEN);
}
```

*Listing 25.1:
Verwenden von
CFile in einer
Konsolenanwen-
dung*

Das Beispiel zeigt, daß die dort gewählte Anwendung von CFile nicht sehr vorteilhaft ist. Die wahren Vorteile der Klasse offenbaren sich erst in Verbindung mit der CArchive-Klasse für die MFC-Objektserialisierung.

25.1.7 Die CStdioFile-Klasse

Die CStdioFile-Klasse wird dazu verwendet, ein von CFile abgeleitetes Objekt mit einem C-Standardstrom (einem FILE-Zeiger) zu verknüpfen. Das Beispielprogramm in Listing 25.2 verwendet diese Klasse.

```
#include <afx.h>

#define HELLOSTR "Hello, World!\n"
#define HELLOLEN (sizeof(HELLOSTR)-1)

void main(void)
{
    CStdioFile file(stdout);
    file.Write(HELLOSTR, HELLOLEN);
}
```

*Listing 25.2:
Verwenden von
CStdioFile*

Der mit dem CStdioFile-Objekt verknüpfte Streamzeiger ist in der Elementvariablen m_pStream gespeichert.

CStdioFile-Objekte werden vorwiegend für die Text-Ein- und -Ausgabe verwendet. Zwei zusätzliche Elementfunktionen, ReadString und WriteString, unterstützen die Ein- und Ausgabe von CString-Objekten und Zeichenfolgen, die mit einem Null-Zeichen enden.

Die CStdioFile-Klasse unterstützt nicht die CFile-Elementfunktionen Duplicate, LockRange und UnlockRange. Versuchen Sie dennoch, diese Funktionen aufzurufen, wird ein Fehler vom Typ CNotSupportedException ausgelöst.

25.1.8 Die CMemFile-Klasse

Die CMemFile-Klasse bietet die Funktionalität der CFile-Klasse, jedoch lediglich für den Speicher. Eine Anwendungsmöglichkeit der CMemFile-Objekte ist das schnelle temporäre Speichern.

Wenn Sie ein CMemFile-Objekt erstellen, können Sie einen Parameter bestimmen, der angibt, um welchen Wert ein CMemFile-Objekt seinen Speicher mit jeder der nachfolgenden Speicherreservierungen vergrößern soll. Die CMemFile-Klasse verwendet die C-Standardbibliothekfunktionen malloc, realloc, free und memcpy, um Speicher zu reservieren, diesen freizugeben und Daten zu oder von den reservierten Speicherblöcken zu übertragen.

Sie können eine Klasse von CMemFile ableiten und deren Standardverhalten während der Speicherreservierung überschreiben. Zu den überschreibbaren Elementfunktionen zählen Alloc, Free, Realloc, Malloc und GrowFile.

Einem CMemFile-Objekt kann außerdem ein zuvor reservierter Speicherblock zugewiesen werden. Verwenden Sie dazu die Attach-Elementfunktion oder die Version des CMemFile-Konstruktors, der drei Parameter verlangt. Beachten Sie bitte, daß die Parameter, die das Vergrößern des reservierten Speichers kontrollieren, auf den Wert Null gesetzt werden müssen, damit das CMemFile-Objekt die Inhalte des zugewiesenen Speicherblocks verwenden kann. Ein auf diese Weise zugewiesener Speicherblock kann nicht vergrößert werden.

Verwenden Sie die Detach-Elementfunktion, um die Zuweisung des Speicherblocks aufzuheben und einen Zeiger darauf zu ermitteln. Um die Größe des Speicherblocks zu ermitteln, verwenden Sie GetLength, bevor Sie Detach aufrufen.

CMemFile unterstützt nicht die CFile-Elementfunktionen Duplicate, LockRange und UnlockRange. Versuchen Sie dennoch, diese Funktionen aufzurufen, wird ein Fehler vom Typ CNotSupportedException ausgelöst.

25.1.9 Die COleStreamFile-Klasse

Die COleStreamFile-Klasse ist mit der OLE-IStream-Schnittstelle verknüpft. Sie stellt eine Funktionalität für OLE-Ströme zur Verfügung, die der von CFile ähnlich ist.

Um ein COleStreamFile-Objekt zu erstellen, übergeben Sie dem Konstruktor dieses Objekts einen Zeiger auf eine IStream-Schnittstelle. Alternativ dazu können Sie ein COleStreamFile-Objekt mit dem Standardkonstruktor generieren und anschließend eine der Initialisierungsfunktionen aufrufen.

Zu den Initialisierungsfunktionen zählen `Attach` (weist einem `COLEStreamFile`-Objekt eine `IStream`-Schnittstelle zu), `CreateMemoryStream`, `CreateStream` und `OpenStream`. Um die Zuweisung der `IStream`-Schnittstelle aufzuheben und einen Zeiger darauf zu ermitteln, verwenden Sie die `DetachElement`-Funktion.

25.1.10 Die CSocketFile-Klasse

Die `CSocketFile`-Klasse stellt eine Schnittstelle für Windows-Socket-Objekte (`CSocket`) zur Verfügung. Ein `CSocketFile`-Objekt kann einem `CArchive`-Objekt zugewiesen werden, um die Serialisierung über ein Socket auszuführen. Es kann außerdem als ein selbstständiges Dateiojekt verwendet werden.

Beachten Sie bitte, daß `CSocketFile` einige `CFile`-Elementfunktionen nicht unterstützt (wie zum Beispiel die `Seek`-Funktionen). Der Versuch, diese Funktionen dennoch aufzurufen, führt zu einem Fehler. Die `CEditView`-Elementfunktion `SerializeRaw` kann nicht mit `CSocketFile`-Objekten verwendet werden.

25.2 Die CArchive-Klasse

Was ist ein `CArchive`-Objekt? Was sind seine Besonderheiten? Warum können `CObject`-Objekte nicht direkt in `CFile`-Objekte geschrieben werden?

Während die `CFile`-Klasse eine generische Mantelklasse für Win32-Dateiobjekte ist, bildet `CArchive` die Verknüpfung zwischen dem beständigen Speicher und den Serialisierungsfunktionen in `CObject`. `CArchive` ermöglicht den Objekten, sich selbst zu serialisieren. In einigen Fällen genügt es (zum Beispiel, wenn Sie ein Integer-Array serialisieren), lediglich das Speicherabbild des Objekts permanent zu speichern. In vielen anderen Fällen (wenn die Objekte beispielsweise Zeiger enthalten) ist diese Vorgehensweise nicht empfehlenswert. Die `CArchive`-Klasse delegiert die Aufgabe des Speicherns an die Objekte und bietet somit eine geeignete Lösung des Problems.

Ein `CArchive`-Objekt ist eine einzelne Einheit und wird dazu verwendet, Objekte von oder zu einem permanenten Speicher zu übertragen. Sie können Lese- und Schreibvorgänge nicht im Direktzugriff ausführen. Dasselbe `CArchive`-Objekt kann außerdem nicht sowohl zum Lesen als auch zum Schreiben verwendet werden. Möchten Sie beispielsweise einige Objekte einlesen, nachdem diese permanent gespeichert wurden, müssen Sie dazu ein separates `CArchive`-Objekt erstellen. Des weiteren müssen die Objekte in der Reihenfolge eingelesen werden, in der sie ursprünglich in das Archiv geschrieben wurden.

25.2.1 Erstellen eines CArchive-Objekts

1. Bevor Sie ein CArchive-Objekt erstellen können, müssen Sie ein CFile-Objekt anlegen, das eine mit den entsprechenden Attributen geöffnete Datei repräsentiert.
2. Nachdem das CFile-Objekt erzeugt wurde, kann das CArchive-Objekt erstellt werden. Übergeben Sie dazu dem Konstruktor einen Zeiger auf das Objekt.
3. In dem Konstruktor bestimmen Sie, ob das Archiv zum Lesen oder Schreiben verwendet werden soll.

m_pDocument Jedes CArchive-Objekt verfügt über die Elementvariable `m_pDocument`, die ein Zeiger auf ein CDocument-Objekt ist. Dieser Zeiger verweist gewöhnlich auf das Dokument, das in MFC-Applikationsrahmen-Anwendungen serialisiert wird. Die Elementvariable muß jedoch nicht zu diesem Zweck eingesetzt werden, wenn die von Ihnen serialisierten Objekte von der Präsenz eines zulässigen `m_pDocument`-Dokuments unabhängig sind.

Möchten Sie einen Zeiger auf ein CFile-Objekt ermitteln, das mit einem CArchive-Objekt verknüpft ist, rufen Sie die `GetFile`-Elementfunktion auf.

Das CFile-Objekt wird mit einem Aufruf von `Close` geschlossen. Diese Funktion löst ebenfalls die Verknüpfung mit dem Archiv. Der Aufruf der Funktion ist gewöhnlich nicht notwendig, da CFile automatisch geschlossen wird, nachdem das Archiv zerstört wurde. Rufen Sie `Close` auf, können Sie keine weiteren Operationen auf dem Archiv ausführen.

25.2.2 Lesen und Schreiben von Objekten

Die CArchive-Klasse liest und schreibt sowohl einfache Datentypen als auch von CObject abgeleitete Objekte.

Lesen oder Schreiben? Sie ermitteln, ob ein CArchive-Objekt zum Lesen oder Schreiben erstellt wurde, indem Sie `IsLoading` oder `IsStoring` aufrufen.

Binäre Daten

Um binäre Daten zu lesen oder zu schreiben, verwenden Sie die Elementfunktionen `Read` respektive `Write`.

Zeichenfolgen

Um Zeichenfolgen einzulesen oder zu schreiben, die mit einer Null-Zeichenfolge enden, rufen Sie `ReadString` oder `WriteString` auf.

Objekte

Möchten Sie ein von `CObject` abgeleitetes Objekt in dem Archiv speichern, verwenden Sie die Funktion `WriteObject`. Die `ReadObject`-Funktion erstellt ein von `CObject` abgeleitetes Objekt und liest die entsprechenden Daten aus dem Archiv ein. Diese Funktion verwendet Laufzeittypinformationen während der Erstellung des Objekts. Die von `CObject` abgeleitete Klasse muß daher mit `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE` deklariert und implementiert werden.

`CArchive` unterstützt das Konzept der `SCHEMANUMMER` mit den Elementfunktionen `GetObjectSchema` und `SetObjectSchema`. Schemanummern ermöglichen einer Anwendung, zwischen unterschiedlichen Versionen desselben Archivs zu unterscheiden. Wenn Sie Schemanummern verwenden, sind Ihre Anwendungen aufwärtskompatibel.

25.2.3 Die überladenen Operatoren << und >>

Anwendungen rufen die `CArchive`-Elementfunktionen nicht immer direkt auf, um Objekte in ein Archiv zu übertragen oder von dort einzulesen. Statt dessen verwenden Sie oft die überladenen Ein- und Ausgabeoperatoren.

Diese überladenen Operatoren wurden sowohl für viele einfache Typen als auch für den `CObject`-Typ definiert. Zu den einfachen Typen zählen

- `BYTE`,
- `WORD`,
- `LONG`,
- `DWORD`,
- `float` und
- `double`.

Doch warum wurden die Operatoren nicht für wesentliche C-Typen definiert, wie zum Beispiel `int`, `short` oder `long`? Der Grund hierfür besteht darin, daß die Größe dieser Typen von der jeweiligen Implementierung abhängig ist. Würden sie in `CArchive`-Operationen verwendet, wäre das daraus resultierende gespeicherte Objekt ebenfalls von der Version des Betriebssystems abhängig, unter der es erstellt wurde. In einer 16-Bit-Windows-Anwendung ist eine `int`-Variable beispielsweise zwei Byte groß. Eine 32-Bit-Windows-Variable beansprucht vier Byte.



Definieren Sie in einem `CArchive`-Objekt nicht Ihre eigenen Versionen der Operatoren `<<` und `>>` für die grundlegenden C-Typen. Verwenden Sie statt dessen die Typenkonvertierung und die bereits bestehenden Operatoren, um eine Abhängigkeit von der Betriebssystemversion zu vermeiden.

Wenn Sie einen einfachen Typ archivieren, werden die Daten lediglich in das Archiv kopiert. Wird jedoch ein `CObject`-Objekt archiviert, verweisen die Operatoren `<<` und `>>` auf die `Serialize`-Elementfunktion des Objekts. Die Operatoren übergeben der Funktion einen Zeiger auf das Archivobjekt. Das Objekt ist daher selbst für seine Serialisierung verantwortlich.

25.2.4 Die Elementfunktion `CObject::Serialize`

Die `Serialize`-Elementfunktion in Objekten vom Typ `CObject` schreibt oder liest ein Objekt in ein respektive aus einem `CArchive`-Objekt.

Diese Funktion wird mit einem Verweis auf das `CArchive`-Objekt aufgerufen. Die Implementierung von `Serialize` sollte die Elementfunktion `CArchive::IsLoading` oder `CArchive::IsStoring` verwenden, um zu ermitteln, ob das Archiv zum Lesen oder Schreiben verwendet wird. Eine typische `Serialize`-Elementfunktionsimplementierung ist nachfolgend aufgeführt:

```
void CMyClass::Serialize(CArchive &ar)
{
    if (ar.IsLoading())
    {
        // Daten laden
    }
    else
    {
        // Daten speichern
    }
}
```

In der `Serialize`-Elementfunktion werden häufig die Operatoren `<<` und `>>` und die `Serialize`-Elementfunktion anderer Objekte aufgerufen. Enthält Ihre Klasse `CMyClass` beispielsweise die Elementvariable `m_other` vom Typ `COtherClass` (eine von `CObject` abgeleitete Klasse), könnte Ihre `Serialize`-Elementfunktion wie folgt aufgebaut sein:

```
void CMyClass::Serialize(CArchive &ar)
{
    m_other.Serialize(ar);
    if (ar.IsLoading())
    {
        // Daten laden
    }
    else
    {

```

```

    // Daten speichern
}

```

25.2.5 Fehlerbearbeitung

Während der Ausführung von Archivoperationen können einige Fehler auftreten. Dazu zählen Dateioperationsfehler, unbeständige Archive und Probleme während der Speicherreservierung. Die meisten `CArchive`-Elementfunktionen verwenden Ausnahmen, um auf diese Fehler zu reagieren.

`CArchive`-Elementfunktionen können drei Ausnahmetypen auslösen:

- ➡ `CFileException`-Ausnahmen werden für Dateifehler eingesetzt.
- ➡ `CArchiveException`-Ausnahmen werden ausgelöst, wenn Archivprobleme auftreten (wenn beispielsweise ein Objekt eines falschen Typs eingelesen wird).
- ➡ `CMemoryException`-Ausnahmen zeigen Fehler während der Speicherreservierung an (wenn `CArchive` zum Beispiel versucht, Speicher für ein Objekt zu reservieren, das eingelesen wird).

25.2.6 Verwenden von CArchive in einfachen Anwendungen

Bevor wir `CArchive` in MFC-Applikationsrahmen-Anwendungen erörtern, finden Sie nachfolgend ein Beispiel aufgeführt, das die Anwendung von `CArchive` in einfachen Situationen demonstriert.

Das in Listing 25.3 aufgeführte Programm verwendet ein `CArchive`-Objekt, um die Inhalte einer Liste zu speichern. Die Liste wird mit der Klasse `CList` erzeugt. Da `CList` von `CObject` abgeleitet ist, unterstützt die Klasse die `Serialize`-Elementfunktion. Sie unterstützt jedoch nicht die Operatoren `<<` und `>>`. Wünschen Sie dies, können Sie explizit eine `operator<<`-Funktion deklarieren. Diese Art der Deklaration führen wir ebenfalls für Objekte vom Typ `CList<WORD, WORD>` aus.

```

#include <afx.h>
#include <afxtempl.h>
#include <iostream.h>

CArchive& operator<<(CArchive& ar, CList<WORD, WORD> &lst)
{
    lst.Serialize(ar);
    return ar;
}

void main(void)
{
    CList<WORD, WORD> myList;

```

*Listing 25.3:
Speichern
einer Liste mit
CArchive*

```

cout << "Creating list: ";
for (int i = 0; i < 10; i++)
{
    int n = rand();
    cout << n << ' ';
    myList.AddTail(n);
}
CFile myFile("mylist.dat", CFile::modeCreate |
               CFile::modeWrite);
CArchive ar(&myFile, CArchive::store);
ar << myList;
}

```

Wenn Sie berücksichtigen, daß dieser Programmcode überwiegend zum Erstellen der Liste verwendet wird, merken Sie, wie leistungsfähig CArchive ist. Lediglich zwei Zeilen generieren das Archivobjekt. Die gesamte Liste wird mit einer Programmcode-Zeile in das Archiv geschrieben. Natürlich geschieht auch das Einlesen der Liste mit nur einer Zeile, wie Listing 25.4 zeigt.

Listing 25.4:
Einlesen aus einem CArchive-Objekt

```

#include <afx.h>
#include <afxtempl.h>
#include <iostream.h>

CArchive& operator>>(CArchive& ar, CList<WORD, WORD> &lst)
{
    lst.Serialize(ar);
    return ar;
}

void main(void)
{
    CList<WORD, WORD> myList;

    CFile myFile("mylist.dat", CFile::modeRead);
    CArchive ar(&myFile, CArchive::load);
    ar >> myList;
    POSITION pos = myList.GetHeadPosition();
    cout << "Reading list: ";
    while (pos)
    {
        int n = myList.GetNext(pos);
        cout << n << ' ';
    }
}

```

Beide Programme können über die Kommandozeile kompiliert werden, zum Beispiel mit

```
CL /MT READLST.CPP.
```

Beachten Sie bitte, daß dieses einfache Beispiel zu Mißverständnissen führen könnte. Verwenden Sie ein Container-Template, wie zum Beispiel CList, müssen Sie möglicherweise die Helferfunktion `SerializeElements` verwenden. Die Standardimplementierung führt ein bitweises Lesen oder Schreiben der Container-Elemente aus. Dies ist zweckmäßig, wenn die Elemente vom Typ `WORD` sind. Für komplexe Typen (wie zum Beispiel von `CObject` abgeleitete Typen) ist diese Vorgehensweise

nicht möglich. (Warum verwenden die Container-Templates nicht die `Serialize`-Elementfunktionen der Objekte, die in dem Container enthalten sind? Der Grund hierfür besteht darin, daß diese Containerklassen nicht ausschließlich auf `CObject`-Objekte beschränkt sind.)

25.3 Serialisierung in MFC-Applikationsrahmen-Anwendungen

Die Konzepte, die sich hinter der Archivierung und Serialisierung verbergen, offenbaren ihr vollständiges Potential erst in MFC-Applikationsrahmen-Anwendungen.

25.3.1 Serialisierung in Dokumenten

In einer MFC-Applikationsrahmen-Anwendung bilden die von `CDocument` abgeleiteten Klassen die wesentlichen Elemente. Diese Klassen repräsentieren die Faktoren, die von Ihrer Anwendung manipuliert werden. Die von `CDocument` abgeleiteten Objekte werden über die in diesem Kapitel vorgestellten Serialisierungsmechanismen gespeichert.

Ein vom Anwendungsassistenten erstelltes Anwendungsgerüst enthält bereits die Menüeinträge `DATEI ÖFFNEN`, `DATEI SPEICHERN` und `DATEI SPEICHERN UNTER`. Diese Implementierungen erzeugen ein `CArchive`-Objekt und rufen die `Serialize`-Elementfunktion der Dokumentklasse auf. Sie müssen diese Elementfunktion derart implementieren, daß alle beständigen Datenelemente Ihrer Dokumentklasse serialisiert werden.

25.3.2 Helfermakros

Die MFC stellt verschiedene Helfermakros zur Verfügung, die eine Serialisierung der von `CObject` abgeleiteten Klassen ermöglichen.

➡ `DECLARE_DYNCREATE` / `IMPLEMENT_DYNCREATE`. Wenn `CArchive` die Daten eines neuen Objekts aus einer Datei einliest, muß ein Mechanismus vorhanden sein, der ein Objekt des gegebenen Typs erstellt. Dazu wird der entsprechenden Klasse eine statische Elementfunktion mit der Bezeichnung `CreateObject` hinzugefügt. Sie müssen diese Funktion nicht manuell deklarieren und implementieren. Verwenden Sie dazu die Makros `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE`.

- ➡ `DECLARE_DYNAMIC` / `IMPLEMENT_DYNAMIC`. Wie erfährt `CArchive`, von welchem Typ das zu erstellende Objekt ist? Die Antwort ist sehr einfach: Mit dem Objekt werden gleichzeitig die Laufzeittypinformationen gespeichert. Damit eine von `CObject` abgeleitete Klasse diese Informationen (über `CRuntimeClass`) auswerten kann, verwenden Sie die Makros `DECLARE_DYNAMIC` und `IMPLEMENT_DYNAMIC`. Da die Funktionalität dieser Makros jedoch auch in `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE` implementiert wird, ist das Hinzufügen der Makros zu der Klassendeklaration nicht erforderlich.
- ➡ `DECLARE_SERIAL` / `IMPLEMENT_SERIAL`. Die Dokumentation gibt an, daß die Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` für die Serialisierung verwendet werden. Sie werden jedoch feststellen, daß die in dem vom Anwendungsassistenten generierten Anwendungsgerüst enthaltene serialisierbare `CDokument`-Klasse keines dieser Makros nutzt. Der Grund hierfür besteht darin, daß `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` nur dann benötigt werden, wenn Sie die Operatoren `<<` und `>>` mit Ihrer Klasse und einem `CArchive`-Objekt verwenden möchten. (`DECLARE_SERIAL` und `IMPLEMENT_SERIAL` deklarieren und implementieren den überladenen Operator `>>` für Ihre Klasse. `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` beinhalten die Funktionalität von `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE`, so daß Sie diese Makros nicht verwenden müssen, wenn Sie `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` einsetzen.

25.3.3 Serialisierung, OLE und die Zwischenablage

Bisher wurde die Serialisierung lediglich hinsichtlich der Speicher- und Ladeoperationen für eine Datei beschrieben. Die MFC verwendet die Serialisierung jedoch ebenfalls für OLE-Operationen.

MFC-Applikationsrahmen-Anwendungen, die als OLE-Server dienen, verwenden die von `COleServerItem` abgeleitete Klasse, um eine Serverschnittstelle zur Verfügung zu stellen. Die `Serialize-Elementfunktion` dieser Klasse verfügt über einen Mechanismus, der spezifische Anwendungsdaten eingebetteter oder verknüpfter OLE-Objekte speichert.

In der einfachsten Implementierung delegiert die `Serialize-Funktion` die Serialisierung des Dokuments an die `Serialize-Elementfunktion` der von `CDokument` abgeleiteten Dokumentklasse. Soll die Anwendung jedoch die Serialisierung einzelner Bereiche des Dokuments unterstützen, ist eine separate Implementierung erforderlich.

Die Serialisierung von Dokumentbereichen kann unter zwei Umständen geschehen. Sie kann zunächst für verknüpfte Objekte ausgeführt werden. Außerdem wird die von `COleServerItem` abgeleitete Klasse für

Zwischenablageoperationen verwendet. Bietet die Anwendung das Kopieren der Anwenderdaten (und nicht des gesamten Dokuments) in die Zwischenablage an, muß die `Serialize-Elementfunktion` dieser Klasse eine Implementierung zur Verfügung stellen, die lediglich die Selektion des Anwenders serialisiert.

25.4 Zusammenfassung

Die Serialisierung in MFC-Anwendungen ist eine Synthese von `CObject`-Objekten (die serialisiert werden müssen), von `CFile`-Objekten, die einen beständigen Speicher repräsentieren, und von `CArchive`-Objekten, die die Serialisierungsschnittstelle zur Verfügung stellen.

Die `CFile`-Klasse enthält die Funktionalität eines Win32-Dateiobjekts. Die Elementfunktionen dieser Klasse öffnen, lesen, schreiben und bearbeiten Dateien.

Varianten der `CFile`-Klasse sind

- ➡ `CStdioFile`,
- ➡ `CMemFile`,
- ➡ `COleStreamFile` und
- ➡ `CSocketFile`.

Diese Klassen realisieren die Ein- und Ausgabe über C-Stromobjekte (`FILE`-Zeiger), Speicherblöcke, OLE-`IStream`-Schnittstellen und Windows-Sockets.

Die `CArchive`-Klasse stellt die Schnittstelle für die Serialisierung zur Verfügung.

Serialisierung ist ein Mechanismus, der den von `CObject` abgeleiteten Klassen das Schreiben oder Lesen der eigenen Daten in und aus einem beständigen Speicher ermöglicht. Dazu ruft `CArchive` die `Serialize-Elementfunktion` der von `CObject` abgeleiteten Objekte auf, wenn ein Datentransfer ausgeführt werden soll.

Die `CObject::Serialize-Elementfunktion` muß für Klassen implementiert werden, die sich von `CObject` ableiten. Die Funktion schreibt oder liest Daten in respektive aus einem `CArchive`-Objekt. Dazu wird der Funktion ein Zeiger auf dieses Objekt übergeben. Rufen Sie die `IsLoading-Elementfunktion` des `CArchive`-Objekts auf, um zu bestimmen, ob ein Speicher- oder Ladevorgang ausgeführt werden soll.

Innerhalb von `Serialize` werden Elementvariablen der Klasse zu dem Archiv übertragen oder daraus ausgelesen. Dies geschieht mit Hilfe der Operatoren `<<` und `>>`. Rufen Sie für den bitweisen Datentransfer die `Serialize-Elementfunktion` der Elementvariablen oder eine der Funktionen `CArchive::Read` und `CArchive::Write` auf.

Die Serialisierung wird in beinahe jeder MFC-Applikationsrahmen-Anwendung verwendet. Der Anwendungsrahmen stellt eine Standardimplementierung der Menüeinträge `DATEI ÖFFNEN` und `DATEI SPEICHERN` zur Verfügung. Diese Implementierungen rufen die `Serialize-Elementfunktion` Ihrer Dokumentklasse auf. Die Funktion muß von Ihnen implementiert werden und sollte alle beständigen Daten des Dokuments serialisieren.

Damit ein von `CObject` abgeleitetes Objekt serialisiert werden kann, muß es mit dem `DECLARE_SERIAL`-Makro deklariert und mit `IMPLEMENT_SERIAL` implementiert werden. Möchten Sie den Operator `>>` nicht in Ihrer Klasse verwenden, sollten Sie das Objekt mit `DECLARE_DYNCREATE` und `IMPLEMENT_DYNCREATE` deklarieren. Betrachten Sie dazu auch eine von dem Anwendungsassistenten generierte Dokumentklasse.

Container-Klassen

Kapitel 26

Die MFC-Bibliothek verfügt über einige Container-Klassen. Diese Klassen implementieren Listen, Arrays und Tabellen.

Die älteren Klassen sind reine Klassen, die neueren Container basieren auf Templates.

Wenn Sie bisher die Container-Klassen in Ihrem Programmcode verwendet haben, die nicht auf Templates basieren, können Sie diese weiterhin einsetzen. In aktuellen Anwendungen sollten Sie jedoch die neuen Klassen nutzen, da deren Typensicherheit einen stabilen Programmcode gewährleistet.

Die vorwiegend in der MFC verwendeten Container sind Klassen vom Typ `CObject`. Dieser Umstand und der relativ einfache Aufbau der `CObject`-Container führt dazu, daß wir unsere Übersicht über die Container-Klassen mit `CObList` und `CObArray` beginnen.

Seit dem neuen ANSI-C++-Standard stehen die MFC-Container in Konkurrenz zu den STL-Containern der C++-Bibliothek.

26.1 CObject-Container

Die MFC-Bibliothek bietet zwei `CObject`-Container an.

- ➡ Der `CObList`-Container ist eine aus `CObject`-Objekten bestehende Liste,
- ➡ während der `CObArray`-Container ein Array aus `CObject`-Objekten darstellt.

Doch worin besteht der Unterschied zwischen diesen Containern? Welche Vorteile ergeben sich aus deren Verwendung?

COBList

Die COBList-Klasse organisiert COBject-Zeiger in einer verketteten Liste. Diese Liste ist so beschaffen, daß das Einfügen und Entfernen von Elementen sehr schnell ausgeführt wird. Die Liste ist jedoch nicht indiziert. Das Ermitteln eines Elements anhand eines numerischen Indexes ist ein zeitintensiver Vorgang. Listen-Container werden verwendet, wenn eine Liste mit Hilfe eines Zählers bearbeitet werden soll. Eine Anwendungsmöglichkeit eines COBList-Containers besteht beispielsweise darin, alle Elemente in einem Dokument zu organisieren. Wird auf die Elemente zugegriffen, um das Dokument zu serialisieren oder zu zeichnen, geschieht dieser Zugriff sequentiell.

COBArray

Die COBArray-Klasse indiziert Elemente mit einem Integer-Wert. Das Einfügen und Entfernen von Elementen ist sehr zeitintensiv, da große Datenblöcke verschoben werden müssen. Das Ermitteln von Daten anhand des Indexes geschieht jedoch äußerst schnell.

Wenngleich beide Klassen gewöhnliche Container sind, die aus COBject-Zeigern bestehen, werden Sie sie möglicherweise niemals mit der COBject-Klasse verwenden. COBList und COBArray werden als Container für Objekte benötigt, deren Klassen sich von COBject ableiten, wie zum Beispiel Objekte vom Typ CWnd, CView oder CDocItem.



In den folgenden Abschnitten werden Ihnen einige Programmbeispiele vorgestellt, in denen Objekte vom Typ COBject deklariert werden. Beachten Sie bitte, daß die Programmfragmente in dieser Form nicht kompiliert werden können, da die Erstellung eines COBject-Objekts durch die Deklaration eines geschützten Konstruktors in der MFC-Bibliothek verhindert wird. Sollen die Beispiele in konkreten Situationen eingesetzt werden, müssen Sie ein von COBject abgeleitetes Objekt in dem entsprechenden Programmcode verwenden und alle erforderlichen Typenkonvertierungen vornehmen.

26.1.1 Die COBList-Klasse

Möchten Sie einen COBList-Container verwenden, müssen Sie diesen zunächst erstellen. Anschließend fügen Sie dem Container Objekte hinzu und greifen auf diese Objekte mit Hilfe von Zählerfunktionen zu.

COBList-Container erstellen

Sie erstellen einen COBList-Container, indem Sie entweder eine Variable vom Typ COBList deklarieren oder sich mit dem `new`-Operator einen Zeiger auf einen neuen COBList-Container zurückliefern lassen.

Elemente in Container einfügen

Objekte können dem COBList-Container mit einem Aufruf von `AddHead` oder `AddTail` hinzugefügt werden. Diese Funktionen ordnen ein Element am Anfang respektive am Ende der Liste an. Die Einträge der Liste müssen sich nicht voneinander unterscheiden. Derselbe CObject-Zeiger kann mehreren Elementen der Liste zugewiesen sein.

Um das erste oder letzte Element der Liste zu ermitteln, verwenden Sie die `GetHead`- oder `GetTail`-Elementfunktion.

Außerdem können Sie der Liste ein neues Element an einer bestimmten Position hinzufügen. Dies geschieht mit den Elementfunktionen `InsertBefore` und `InsertAfter`, die einen neuen Eintrag vor oder nach einem bestimmten Element einfügen. Die Position des Elements wird mit einer Variable vom Typ `POSITION` angegeben. Dieser Typ wird für alle Container-Klassen als Zähler verwendet.

Ein Wert vom Typ `POSITION` kann mit Hilfe der Elementfunktionen `GetHeadPosition` und `GetTailPosition` ermittelt werden. Der zurückgegebene `POSITION`-Wert wird anschließend in Aufrufen von `GetNext` oder `GetHead` verwendet werden, um sequentiell auf die Elemente der Liste zuzugreifen.

*Der Typ
POSITION*

Container durchlaufen

Um beispielsweise alle Elemente in einem COBList-Container ab dem ersten Element nacheinander zu bearbeiten, könnten Sie den folgenden Programmcode verwenden:

```
COBList myList;
...
// Liste mit Elementen erstellen
...
POSITION pos = myList.GetHeadPosition();
while (pos != NULL)
{
    CObject *pObject = myList.GetNext(pos);
    ...
    // *pObject bearbeiten
    ...
}
```

Natürlich können Sie die Elemente der Liste auch vom letzten bis zum ersten Element bearbeiten, wie nachfolgend dargestellt:

```
COBList myList;
...
```

```
// Liste mit Elementen erstellen
...
POSITION pos = myList.GetTailPosition();
while (pos != NULL)
{
    CObject *pObject = myList.GetPrev(pos);
    ...
    // *pObject bearbeiten
    ...
}
```

Wie Sie an diesem Beispiel erkennen, können die Bezeichnungen der Funktionen `GetNext` und `GetPrev` zu Mißverständnissen führen. Die Funktionen geben das aktuelle Element zurück, auf das der `POSITION`-Parameter verweist. Gleichzeitig setzen sie den Parameter auf das nächste respektive vorherige Element.

Ein `POSITION`-Wert kann außerdem in den Aufrufen der Elementfunktionen `GetAt`, `SetAt` und `RemoveAt` verwendet werden. Jede dieser Funktionen benötigt einen Parameter vom Typ `POSITION`. `GetAt` ermittelt einen `CObject`-Zeiger, der der in `POSITION` gespeicherten Position entspricht. `SetAt` setzt das Element an der angegebenen Position auf einen neuen Wert. `RemoveAt` entfernt das Element aus der Liste.

Elemente löschen

Das Entfernen eines Elements während der Iteration kann zu Problemen führen. Mit einem der folgenden Methode ähnlichen Verfahren gewährleisten Sie, daß `GetNext` immer ein zulässiger `POSITION`-Wert übergeben wird:

```
POSITION pos = myList.GetHeadPosition();
while (pos != NULL)
{
    POSITION pos2 = pos;
    CObject *pObject = GetNext(pos);
    if ( /* pObject wird entfernt */ )
    {
        myList.RemoveAt(pos2);
    }
}
```

Weitere Funktionen, die zum Entfernen von Elementen verwendet werden können, sind `RemoveHead` und `RemoveTail`. Die `RemoveAll`-Funktion entfernt alle Elemente (löscht die Liste).

Entfernen Sie ein Element, wird es nicht automatisch zerstört. Das Programm, in dem das Element erstellt wurde, muß das Element wie folgt zerstören:

```
CObject *pObject;
CobList myList;
...
pObject = new CObject;
myList.AddTail(pObject);
...
// später
```

```
...
pObject = myList.RemoveTail();
delete pObject;
```

Verwenden Sie die `IsEmpty`-Elementfunktion, um zu ermitteln, ob eine Liste leer ist. `GetCount` gibt die Anzahl der Elemente in der Liste zurück.

Elemente suchen

Sie können außerdem nach einem bestimmten Element in der Liste suchen. Die `Find`-Elementfunktion prüft, ob ein bestimmter `CObject`-Zeiger in der Liste vorhanden ist. In diesem Fall gibt die Funktion einen Wert vom Typ `POSITION` zurück, der die Position des ersten Zeigers angibt. Die `FindIndex`-Elementfunktion gibt einen `POSITION`-Wert zurück, der der Position des angegebenen numerischen Indexes entspricht. Beachten Sie bitte, daß die Ausführung dieser Funktion für eine umfangreiche `CObList`-Klasse sehr viel Zeit beanspruchen kann, da `CObList`-Container keinen Index speichern.

Serialisierung

Der `CObList`-Typ wird von `CObject` abgeleitet und unterstützt die Serialisierung. Wird die `Serialize`-Elementfunktion des `CObList`-Typs aufgerufen, führt dies zu einer Serialisierung aller `CObject`-Elemente in der Liste. Dies geschieht mit Hilfe der Operatoren `<<` und `>>`. Um zu gewährleisten, daß die Liste korrekt serialisiert wurde, sollten die der Liste hinzugefügten Elemente vom Typ `CObject` sein. Dieser wird mit den Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` deklariert und implementiert.

`CObList`-Objekte können zusammen mit der `CArchive`-Klasse und den Operatoren `<<` und `>>` wie nachfolgend dargestellt verwendet werden:

```
class CMyDocument : public CDocument
{
    CObList m_List;
    // Ab hier die verbleibende Klassendeklaration
    ...
}
void CMyDocument::Serialize(CArchive &ar)
{
    if (ar.IsStoring())
    {
        ar << m_List;
    }
    else
    {
        ar >> m_List;
    }
}
```

Da `CObList` ebenfalls von `CObject` abgeleitet ist, können Sie einen `CObList`-Container verwenden, der auf Elemente vom Typ `CObList` verweist. Sie erzeugen somit eine Liste, die aus Listen besteht.

26.1.2 Die CObArray-Klasse

CObArray-Container sind Arrays, die CObject-Zeiger enthalten. Diese Arrays sind in ihrem Verhalten und in ihrer Funktion den C-Arrays ähnlich. Ein wesentlicher Unterschied besteht jedoch darin, daß ein CObArray dynamisch erweitert oder verkleinert werden kann.

CObArray-Container erstellen

Das Verwenden eines CObArray-Containers bedingt das Erstellen eines CObArray-Objekts. Dem Array werden anschließend CObject-Zeigerelemente hinzugefügt, die beispielsweise mit dem überladenen Operator [] ermittelt werden können.

Sie erstellen ein CObArray wie jede andere Variable entweder auf dem Stack, als eine automatische Variable, oder mit Hilfe des new-Operators.

Elemente in Container einfügen

Die CObArray-Klasse deklariert die Elementfunktionen SetAt, GetAt und SetAtGrow. SetAt und GetAt setzen und ermitteln Elemente an den angegebenen Positionen. SetAtGrow setzt ein Element ebenfalls an die angegebene Position. Diese Funktion führt jedoch dazu, daß das Array erweitert wird, wenn die Position außerhalb des Arrays liegt.

Weder SetAt noch GetAt melden einen Fehler, wenn ein unzulässiger Index angegeben wird. In der Debug-Version der MFC-Bibliothek generieren die Funktionen in diesem Fall jedoch eine Assertion.

Elemente können einem Array ebenfalls mit der Add-Elementfunktion hinzugefügt werden. Diese Funktion erweitert das Array automatisch, sofern erforderlich.

Die Größe des Containers

GetSize Die Anzahl der aktuell in dem Array vorhandenen Elemente wird mit einem Aufruf von GetSize ermittelt. Der größte zulässige Array-Index (der der Anzahl der Array-Elemente abzüglich einem Element entspricht, da die Array-Indizes mit dem Wert Null beginnen) kann mit Hilfe der Funktion GetUpperBound ermittelt werden.

SetSize Die SetSize-Funktion bestimmt die maximale Anzahl der Elemente in dem Array und reserviert, sofern erforderlich, zusätzlichen Speicher. Wird SetSize zum Verringern der Größe des Arrays verwendet, wird der nicht mehr benötigte Speicher freigegeben.

Wird ein Array aufgrund eines Aufrufs von SetAtGrow, Add oder SetSize erweitert, kann ein Speicherreservierungsfehler auftreten. Diese Fehler werden durch eine Ausnahme vom Typ CMemoryException angezeigt.

Die `SetSize`-Funktion wird ebenfalls dazu verwendet, den Wert anzugeben, um den der für `CObArray` reservierte Speicher erweitert werden soll. Immer dann, wenn neuer Speicher reserviert wird, nimmt dieser die Anzahl der `CObject`-Zeiger auf, die in dem zweiten Parameter von `SetSize` angegeben wurde. Wird dieser Parameter nicht gesetzt, versucht die `CObArray`-Klasse den optimalen Speicherbedarf zu ermitteln, der reserviert werden muß, um eine Heap-Fragmentierung zu vermeiden.

Jeder nicht benötigte Speicher, der während der Erweiterung des Arrays reserviert wurde, kann mit `FreeExtra` freigegeben werden. Das gesamte Array wird mit einem Aufruf von `RemoveAll` geleert. Der für das Array benötigte Speicher wird nach diesem Aufruf freigegeben.

Der []-Operator

Die `CObArray`-Klasse stellt eine überladene Version des []-Operators zur Verfügung. Sie greifen über diesen Operator auf die Elemente des Arrays zu. Der Operator kann in Situationen eingesetzt werden, in denen ein L-Wert benötigt wird (beispielsweise auf der linken Seite einer Zuweisung). Dieses Verhalten wird mit Hilfe der `ElementAt`-Elementfunktion implementiert, die einen Verweis auf den `CObject`-Zeiger an der angegebenen Position zurückgibt. Die Zeile

```
myArray[10] = &myObject;
```

hat somit die gleiche Funktion wie die folgende Zeile:

```
myArray.ElementAt(10) = &myObject;
```

Die beiden Elementfunktionen `InsertAt` und `RemoveAt` fügen Elemente in das Array ein oder entfernen das Element mit dem angegebenen Index. Diese Funktionen arbeiten sehr langsam, da in großen Arrays entsprechend große Datenblöcke verschoben werden müssen.

Elemente löschen

Die `CObArray`-Klasse zerstört ihre Elemente nicht, wenn diese aus dem Array entfernt werden. Sie müssen diese Elemente wie folgt freigeben:

```
CObject *pObject;
CObArray myList;
...
pObject = new CObject;
myArray.Add(pObject);
...
// später
...
pObject = myList[myArray.GetUpperBound()];
myArray.SetSize(myArray.GetUpperBound());
delete pObject;
```


Serialisierung

Das `CObArray` wird von `CObject` abgeleitet. Ein Vorteil, der sich daraus ergibt, besteht darin, daß `CObArray`-Container serialisiert werden können. Die `CObArray::Serialize-Elementfunktion` serialisiert die Array-Elemente mit Hilfe der Operatoren `<<` und `>>`. Die dem Array hinzugefügten Elemente müssen vom Typ `CObject` sein, damit sie serialisiert werden können. Dieser Typ wird mit den Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` deklariert und implementiert.

`CObArray`-Container können mit der `CArchive`-Klasse und den Operatoren `<<` sowie `>>` verwendet werden.

26.2 Weitere Listen-Container

Die Features und das Verhalten anderer Listen-Container entsprechen den Features sowie dem Verhalten des `CObList`-Containers.

26.2.1 Die `CPtrList`-Klasse

Die `CPtrList`-Klasse entspricht weitgehend der `CObList`-Klasse. Sie nimmt jedoch Zeiger vom Typ `void` auf.

`CPtrList` unterstützt nicht die Serialisierung.

26.2.2 Die `CStringList`-Klasse

Das Verhalten und die Implementierung der `CStringList`-Klasse gleichen ebenfalls denen der `CObList`-Klasse. Der einzige Unterschied besteht darin, daß nicht die Zeiger auf Elemente vom Typ `CString`, sondern Kopien des `CString`-Objekts selbst in dem `CStringList`-Container gespeichert werden.

Die Konsequenz, die sich aus diesem Umstand ergibt, besteht darin, daß eine Anwendung nicht wie bisher ein `CString`-Objekt dynamisch erzeugen oder explizit löschen muß, nachdem es aus einem `CStringList`-Container entfernt wurde.

```
void MyAddElement(CStringList *pList)
{
    CString myString;
    pList->AddTail(myString);
}
```

`CStringList`-Container unterstützen die Serialisierung.

26.3 Weitere Array-Container

Die MFC-Bibliothek bietet zusätzlich zur `CObArray`-Klasse weitere Array-Container an.

26.3.1 Die `CPtrArray`-Klasse

Die `CPtrArray`-Klasse implementiert das gleiche Verhalten wie `CObArray`, nimmt jedoch `void`-Zeiger auf. Die Elementfunktionen sowie die Features der Klasse entsprechen denen der `CObArray`-Klasse.

`CPtrArray` unterstützt nicht die Serialisierung.

26.3.2 Integrale Array-Klassen

Einige Klassen der MFC speichern Integralelemente. Diese Klassen sind in Tabelle 26.1 aufgeführt.

Klassenname	Elementtyp
<code>CByteArray</code>	<code>BYTE</code>
<code>CWordArray</code>	<code>WORD</code>
<code>CDWordArray</code>	<code>DWORD</code>
<code>CUIntArray</code>	<code>UINT</code>

Tabelle 26.1:
Integral-Array-
Container-Klassen

Abgesehen davon, daß `CUIntArray` keine Serialisierung unterstützt, entsprechen die Features und das Verhalten dieser Klassen den Features sowie dem Verhalten der `CObArray`-Klasse.

26.3.3 Die `CStringArray`-Klasse

Die `CStringArray`-Klasse repräsentiert ein Array aus `CString`-Objekten. Die Klasse speichert keine Zeiger, sondern Kopien der Objekte. Der Anwendungsprogrammierer ist somit nicht für das Zerstören der `CString`-Objekte verantwortlich, die aus dem Array entfernt wurden.

Die Features und das Verhalten von `CStringArray` entsprechen denen von `CObArray`. `CStringArray` unterstützt die Serialisierung.

26.4 Zuordnungen

Zuordnungen repräsentieren einen Containertyp, der sich von Listen und Arrays unterscheidet. Listen und Arrays sind geordnete Container, während Zuordnungen nicht geordnete Zuweisungen von Schlüsselobjekten zu Wertobjekten sind. Zuordnungen werden bisweilen auch als Wörterbücher bezeichnet (das Implementieren der Funktionalität eines Wörterbuchs ist nebenbei bemerkt mit einer Zuordnung, wie zum Beispiel dem Container `CMapStringToString`, eine einfache Aufgabe).

Zuordnungs-Container können anhand eines Schlüsselwertes sehr schnell durchsucht werden. Die Schlüsselwerte einer Zuordnungsklasse müssen eindeutig sein. Versuchen Sie, einem Wert einen bereits bestehenden Schlüssel zuzuweisen, wird der entsprechende Eintrag in dem Zuordnungs-Container überschrieben.

Die MFC-Bibliothek bietet verschiedene Zuordnungs-Container an. Schlüssel, die Zeiger sind, Zeichenfolgen oder 16-Bit-Werte vom Typ `WORD` werden zur Indizierung von Elementen verwendet, die Zeiger auf `CObject`-Objekte, `void`-Elemente oder 16-Bit-Werte sind. Da nicht alle Kombinationen dieser Typen in Form eines Zuordnungs-Containers implementiert sind und Unterschiede in den Verhaltensweisen dieser Klassen bestehen, führen die folgenden Abschnitte die Klassen einzeln auf.

26.4.1 Die `CMapStringToString`-Klasse

Die `CMapStringToString`-Klasse weist Schlüssel vom Typ `CString` den Werten desselben Typs zu.

Container erzeugen

Um einen `CMapStringToString`-Container zu erstellen, deklarieren Sie ein Objekt dieses Typs oder verwenden den `new`-Operator.

Elemente hinzufügen

Nachdem der Container erzeugt wurde, können ihm die Schlüssel-/Wert-Paare mit der `SetAt`-Elementfunktion oder dem überladenen Operator `[]` hinzugefügt werden. Dieser Operator kann lediglich an der Position eines L-Wertes eingesetzt werden. Für die Suche nach Schlüssel ist der Operator ungeeignet.

Nach Elementen suchen

Möchten Sie nach Daten anhand des Schlüssels suchen, verwenden Sie die `Lookup`-Elementfunktion. Der Boolesche Rückgabewert dieser Funktion zeigt an, ob der Schlüssel gefunden wurde.

Elemente löschen

Sie entfernen ein Schlüssel-/Wertpaar aus dem Container mit einem Aufruf der Elementfunktion `RemoveKey`. Mit `RemoveAll` entfernen Sie alle Schlüssel-/Wertpaare und leeren somit den Container.

Die Größe des Containers

Mit Hilfe der `IsEmpty`-Elementfunktion prüfen Sie, ob ein Container leer ist. Die `GetCount`-Elementfunktion gibt die Anzahl der Schlüssel-/Wertpaare in dem Container zurück.

Container durchlaufen

Natürlich können Sie alle Elemente einer Zuordnung nacheinander bearbeiten. Die `GetStartPosition`-Elementfunktion erstellt einen Zähler vom Typ `POSITION`. Dieser wird anschließend in den Aufrufen von `GetNextAssoc` verwendet, um ein Schlüssel-/Wert-Paar zu ermitteln. Beachten Sie bitte, daß die Reihenfolge, in der die Elemente zurückgegeben werden, willkürlich ist. Die Funktionen geben die Elemente daher möglicherweise nicht in aufsteigender Schlüsselreihenfolge zurück.

Serialisierung

`CMapStringToString`-Container können serialisiert werden. Sie können in Verbindung mit der `CArchive`-Klasse und den Operatoren `<<` sowie `>>` verwendet werden.

Listing 26.1 zeigt ein Programm, das ein Lexikon implementiert. Die Anwendung wird mit der folgenden Kommandozeilenanweisung kompiliert:

```
CL /MT VOCAB.CPP.
```

```
#include <afx.h>
#include <iostream.h>
#include <stdio.h>

void main(void)
{
    CString wrd, def;
    CMapStringToString map;
    CStdioFile inf(stdin);

    cout << "Vokabeln einlesen\n";
    while (TRUE)
    {
        cout << "Vokabel eingeben: ";
```

*Listing 26.1:
Verwenden von
CMapString-
ToString in einer
Konsolen-
anwendung*

```

        cout.flush();
        inf.ReadString(wrd);
        if (wrd == "Q") break;
        cout << "Definition: ";
        cout.flush();
        inf.ReadString(def);
        map[wrd] = def;
    }
    if (map.IsEmpty())
    {
        cout << "Wörterbuch ist leer!\n";
        exit(0);
    }
    cout << "Wörterbuch enthält ";
    cout << map.GetCount() << " Einträge.\n";
    cout << "Nach Vokabeln suchen\n";
    while (TRUE)
    {
        cout << "Vokabel eingeben: ";
        cout.flush();
        inf.ReadString(wrd);
        if (wrd == "Q") break;
        if (map.Lookup(wrd, def))
            cout << def << '\n';
        else
            cout << "nicht gefunden!\n";
    }
}

```

Das Programm legt einen `CMapStringToString`-Container an und führt im Anschluß daran eine Eingabeschleife aus. Der Anwender gibt in dieser Schleife einen Begriff und die Definition des Begriffs ein. Diese beiden Eingaben werden dem Container hinzugefügt. Die Schleifenausführung wird beendet, wenn der Anwender ein großes Q für das zu definierende Wort eingibt. Daraufhin wird die Größe des Containers angezeigt und eine zweite Schleife ausgeführt. Dort gibt der Anwender Ausdrücke ein, nach denen im Lexikon gesucht werden soll. Nachfolgend finden Sie einige Beispieleingaben aufgeführt:

```

Vokabeln einlesen
Vokabel eingeben: Maus
Definition: mouse
Vokabel eingeben: Katze
Definition: cat
Vokabel eingeben: Hund
Definition: dog
Vokabel eingeben: Guten Morgen
Definition: good morning
Vokabel eingeben: Q
Wörterbuch enthält 4 Einträge.
Nach Vokabeln suchen
Vokabel eingeben: Katze
cat
Vokabel eingeben: Tiger
nicht gefunden!
Vokabel eingeben:

```

26.4.2 Die CMapStringToOb-Klasse

Die CMapStringToOb-Klasse verbindet CObject-Zeiger (Werte) mit Objekten vom Typ CString (Schlüssel).

Die Features und das Verhalten dieser Klasse gleichen denen der CMapStringToString-Klasse. Ein Unterschied besteht jedoch darin, daß der Programmierer die aus dem Container entfernten CObject-Elemente selbst zerstören muß, da diese Klasse CObject-Zeiger speichert. Dazu das folgende Beispiel:

```
CObject *pObject;
CMapStringToOb myMap;
...
pObject = new CObject;
myMap["myObject"] = pObject;
...
// später
...
if (myMap.Lookup("myObject", pObject))
{
    myMap.RemoveKey("myObject");
    delete pObject;
};
```

CMapStringToOb unterstützt die Serialisierung und kann mit der CArchive-Klasse und den Operatoren << sowie >> verwendet werden.

26.4.3 Die CMapStringToPtr-Klasse

Die CMapStringToPtr-Klasse verbindet void-Zeiger (Werte) mit Objekten vom Typ CString (Schlüssel).

Die Klasse speichert Zeiger auf Elemente und gibt diese nicht frei, wenn die Zeiger aus dem Container entfernt werden. Sie müssen daher die Elemente zerstören, auf die diese Zeiger verwiesen haben.

CMapStringToPtr-Container können nicht serialisiert werden.

In jeder anderen Hinsicht entsprechen die Features und das Verhalten der CMapStringToPtr-Klasse denen von CMapStringToOb.

26.4.4 Die CMapPtrToPtr-Klasse

Die CMapPtrToPtr-Klasse verbindet void-Zeiger (Werte) mit anderen void-Zeigern (Schlüssel). Beachten Sie bitte, daß der Zeigerwert und nicht das Element, auf das er verweist, als Schlüssel dient. Zwei identische Zeiger, die auf zwei verschiedene Objekte verweisen, werden daher von CMapPtrToPtr wie unterschiedliche Schlüssel behandelt. Betrachten Sie dazu auch den folgenden Programmcode:

```
CMapPtrToPtr myMap;  
int a, b, x, y;  
a = b = 123;  
myMap[&a] = &x;  
myMap[&b] = &y;
```

Wenngleich `a` und `b` gleich sind, sind `&a` und `&b` nicht gleich. Dieser Programmcode fügt dem Container somit zwei unterschiedliche Schlüssel-/Wert-Paare hinzu.

Wird ein Schlüssel-/Wert-Paar aus einem `CMapPtrToPtr`-Container entfernt, muß die Anwendung die beiden Elemente zerstören, auf die die Zeiger verwiesen haben.

26.4.5 Die `CMapPtrToWord`-Klasse

Die `CMapPtrToWord`-Klasse verbindet Werte vom Typ `WORD` mit `void`-Zeigern (Schlüssel). Der Zeigerwert und nicht das Element, auf das der Zeiger verweist, bildet den Schlüssel zu diesem Container.

Wird ein Schlüssel-/Wert-Paar aus einem `CMapPtrToWord`-Container entfernt, muß die Anwendung die beiden Elemente zerstören, auf die die Zeiger verwiesen haben.

26.4.6 Die `CMapWordToOb`-Klasse

Die `CMapWordToOb`-Klasse verbindet `CObject`-Zeiger (Werte) mit Indizes vom Typ `WORD` (Schlüssel).

Doch worin besteht der Unterschied zwischen dieser Klasse und einem `CObArray`? In einem Array-Container beginnen die Indizes mit dem Wert Null und sind fortlaufend. Die `WORD`-Indizes in einer `CMapWordToOb`-Auflistung können willkürlich sein. Möchten Sie beispielsweise in einer `CObArray` die Indizes 1 und 100 verwenden, müssen Sie Speicher für einhundertsteins Elemente reservieren. In einem `CMapWordToOb`-Container müssen zu diesem Zweck lediglich zwei Elemente reserviert werden.

`CMapWordToOb` unterstützt die Serialisierung und kann mit der `CArchive`-Klasse und den Operatoren `<<` und `>>` verwendet werden.

26.4.7 Die `CMapWordToPtr`-Klasse

Die `CMapWordToPtr`-Klasse verbindet `void`-Zeiger (Werte) mit Indizes vom Typ `WORD` (Schlüssel). Die Features und das Verhalten dieser Klasse entsprechen denen von `CMapWordToOb`.

Der einzige Unterschied besteht darin, daß `CMapWordToPtr` keine Serialisierung unterstützt.

26.5 Auf Templates basierende Objekt-Container

Die Container-Klassen, die bisher besprochen wurden, sind nicht typensicher. Stellen Sie sich beispielsweise vor, wie ein Container mit CWnd-Objekten in der CObList-Klasse implementiert wird. Der Liste würden CWnd-Zeiger wie folgt hinzugefügt:

```
CWnd *pWnd;  
CObList myList;  
...  
myList.AddTail((CObject *)pWnd);  
...  
pWnd = (CWnd *)myList.GetHead();
```

Der Container kann aufgrund der Typenkonvertierung in dem Aufruf von AddTail nicht überprüfen, ob das erhaltene Objekt vom korrekten Typ ist. Dementsprechend ist ein Objekt, das aus dem Container entnommen wird, immer ein Zeiger auf den CObject-Typ. Wird dem Container wegen eines Programmierfehlers ein Zeiger eines anderen von CObject abgeleiteten Typs übergeben, erhalten Sie keine Fehlermeldung und auch keine Compiler-Warnung. Die Anwendung wird jedoch nicht korrekt ausgeführt.

Sie können dem Container beispielsweise einen Zeiger vom Typ CDocument hinzufügen:

```
CDocument *pDoc;  
...  
myList.AddTail((CObject *)pDoc);
```

Sie bemerken zunächst keine Besonderheiten. Erst später, wenn Sie diesen Zeiger ermitteln und davon ausgehen, daß dieser auf ein CWnd-Objekt verweist, wird sich Ihr Programm ungewöhnlich verhalten.

Typensichere Container-Templates bieten eine Lösung dieses Problems. Durch die folgende Deklaration der Container

```
CTypedPtrList<CObList, CWnd *> myList;
```

vermeiden Sie die Notwendigkeit zur Typenkonvertierung, so daß der Compiler einen Fehler anzeigt, wenn dem Container ein Zeiger hinzugefügt wird, der kein CWnd-Zeiger ist.

Sie können typensichere Versionen ebenfalls von Container-Klassen ableiten, die nicht auf Templates basieren. Fügen Sie der Klasse dazu die entsprechenden Mantelfunktionen hinzu.



Es gibt zwei Arten von Container-Templates.

- Die erste Kategorie enthält einfache Arrays, Listen und Zuordnungen.
- Die zweite Kategorie enthält typisierte Zeiger auf diese Objekte.

Zu den Elementen der ersten Kategorie zählen die Vorlagen `CList`, `CArray` und `CMap`. Die zweite Kategorie besteht aus den Vorlagen `CTypedPtrList`, `CTypedPtrArray` und `CTypedPtrMap`.

26.5.1 Die Helferfunktionen der Container-Klassen

Die einfachen Containervorlagen `CList`, `CArray` und `CMap` verwenden sieben Container-Klassen-Helferfunktionen. Diese Funktionen müssen möglicherweise implementiert werden, um das gewünschte Verhalten zu erzielen.

- Die `ConstructElements`-Helferfunktion wird von Container-Klassen zur Erstellung von Elementen verwendet. Diese Funktion wird aufgerufen, nachdem der Speicher für die neuen Elemente reserviert wurde. Die Standardimplementierung nutzt einen Konstruktor vom Typ `TYPE`, um die Elemente zu erzeugen.
- Die Funktion `DestructElements` wird aufgerufen, bevor der für die Elemente in der Container reservierte Speicher freigegeben wird. Die Standardimplementierung dieser Funktion nutzt einen Konstruktor vom Typ `TYPE`, um die Container-Elemente zu zerstören.
- Die `CompareElements`-Funktion vergleicht zwei Elemente miteinander. Die Standardimplementierung verwendet zu diesem Zweck den Operator `==`. Die Funktion wird von `CList::Find` und von Containern benutzt, die auf `CMap` basieren.
- Die `CopyElements`-Funktion kopiert Elemente. Die Standardimplementierung führt ein bitweises Kopieren aus. Die Funktion wird von `CArray::Append` und `CArray::Copy` verwendet.
- Die Helferfunktion `SerializeElements` serialisiert die in dem Container enthaltenen Elemente. Die Standardimplementierung führt die Serialisierung bitweise aus. Überschreiben Sie diese Funktion, wenn Sie statt dessen beispielsweise die `Serialize-Elementfunktion` Ihrer Container-Elemente aufrufen möchten.
- Die `HashKey`-Helferfunktion wird von Containern verwendet, die auf `CMap` basieren, um einen Hash-Schlüssel zu erstellen. Die Standardimplementierung erstellt einen Hash-Schlüssel, indem der Schlüsselwert vier Bits nach rechts verschoben wird. Überschreiben Sie diese Elementfunktion, wenn Sie einen Hash-Schlüssel nutzen möchten, der Ihrer Anwendung entsprechen soll.

- ➡ Die `DumpElements`-Elementfunktion erstellt eine Diagnose der Container-Elemente. Die Standardimplementierung dieser Funktion führt keine Aufgabe aus. Überschreiben Sie die Funktion, wenn Sie statt dessen beispielsweise die `Dump`-Elementfunktion der Container-Elemente aufrufen möchten.

26.5.2 Das `CList`-Template

Das `CList`-Template wird zur Erstellung von Listen verwendet, die Elemente eines gegebenen Typs aufnehmen. Eine Liste ist ein geordneter Container von Elementen. Der Zugriff auf die Elemente geschieht mit einem Zähler.

Container deklarieren

Das `CList`-Template verlangt zwei Template-Parameter und wird wie folgt deklariert:

```
template<class TYPE, class ARG_TYPE> class CList : public CObject
{
    ...
};
```

Der Parameter `TYPE` gibt den Typ der Elemente an, aus der die Liste besteht. `ARG_TYPE` bestimmt den Typ, der in Funktionsargumenten verwendet wird. `ARG_TYPE` verweist häufig auf `TYPE`. Eine Liste aus `CString`-Objekten könnte beispielsweise wie folgt deklariert werden:

```
CList<CString, CString&> myList;
```

Obwohl das Verhalten von `CList` dem von `CObList` ähnlich ist, besteht ein wesentlicher Unterschied zwischen den Containern: Ein `CList`-Container speichert Objekte vom Typ `TYPE` und keine Zeiger auf diese Objekte. In dem vorherigen Beispiel wurde von jedem der Liste hinzugefügten `CString`-Objekt eine Kopie erstellt. *CList versus CObList*

Ein `CList`-Container wird während seiner Deklaration erzeugt.

Elemente hinzufügen

Elemente vom Typ `TYPE` werden dem Container mit den Elementfunktion `AddHead` oder `AddTail` hinzugefügt. Sie können dem Container auch Elemente an einer bestimmten Position hinzufügen, die mit einem `POSITION`-Wert angegeben wird. Rufen Sie dazu die Funktion `InsertBefore` oder `InsertAfter` auf.

Ein Zähler vom Typ `POSITION` wird mit `GetHeadPosition` oder `GetTailPosition` ermittelt.

Container durchlaufen

Das Bearbeiten der Elemente in einem Container geschieht mit wiederholten Aufrufen von `GetNext` oder `GetPrev` in einer Schleife, wie nachfolgend demonstriert:

```
CList<CString, CString&> myList;
...
// Elemente der Liste hinzufügen
...
POSITION pos = myList.GetHeadPosition();
while (pos != NULL)
{
    CString str = GetNext(pos);
    ...
    // str bearbeiten
    ...
}
```

Das erste oder letzte Element der Liste wird mit der Elementfunktion `GetHead` respektive `GetTail` ermittelt.

In den Aufrufen von `GetAt`, `SetAt` und `RemoveAt` kann ebenfalls ein `POSITION`-Wert verwendet werden. Diese Elementfunktionen ermitteln das Element an der angegebenen Position, setzen das Element an dieser Position auf einen neuen Wert oder löschen das Element.

Elemente löschen

Sie entfernen das erste oder letzte Element der Liste mit `RemoveHead` oder `RemoveTail`. Die vollständige Liste wird mit `RemoveAll` geleert.

Größe des Containers

Die Elementfunktion `IsEmpty` ermittelt, ob die Liste leer ist. `GetCount` kann dazu verwendet werden, die Anzahl der Elemente in der Liste zu ermitteln.

Elemente suchen

Die Suche nach Elementen geschieht mittels eines numerischen Indexes. Verwenden Sie dazu die `FindIndex`-Funktion. Mit der `Find`-Funktion suchen Sie nach einem bestimmten Wert. Beachten Sie bitte, daß Sie möglicherweise eine überschriebene Version von `CompareElements` zur Verfügung stellen müssen, damit die `Find`-Elementfunktion korrekt ausgeführt wird.

Serialisierung

Die `CList`-Template unterstützt die Serialisierung. Sie müssen eventuell eine überschriebene Version der `SerializeElements`-Helferfunktion verwenden, damit die Serialisierung fehlerfrei arbeitet.

26.5.3 Das CArray-Template

Das `CArray`-Template erstellt ein dynamisch reserviertes Array, das Elemente eines spezifizierten Typs aufnimmt. Ein Array ist ein Container, auf dessen Elemente über einen Integer-Index zugegriffen wird, der mit dem Wert Null beginnt. Die Funktion und das Verhalten von `CArray` entspricht der Funktion und dem Verhalten von C-Arrays. Der einzige Unterschied besteht darin, daß ein `CArray`-Container dynamisch erweitert oder verkleinert werden kann.

Container-Deklaration

Das `CArray`-Template verlangt zwei Parameter und wird wie folgt deklariert:

```
template<class TYPE, class ARG_TYPE> class CArray : public CObject
{
    ...
};
```

Der Parameter `TYPE` gibt den Typ der Elemente an, die in der Liste enthalten sind. `ARG_TYPE` bestimmt den Typ, der in Funktionsargumenten verwendet wird. `ARG_TYPE` verweist häufig auf `TYPE`. Nachfolgend ein Beispiel:

```
CArray<CString, CString&> myArray;
```

Abgesehen von den Gemeinsamkeiten, besteht ein wesentlicher Unterschied zwischen dem Verhalten von `CArray` und dem des Array-Containers `CObArray`, der nicht auf einem Template basiert. `CArray` speichert Kopien der Elemente und nicht Zeiger darauf.

*CArray versus
CObArray*

Nach der Deklaration und der daraus resultierenden Erstellung eines `CArray`-Objekts können Sie mit der `SetSize`-Elementfunktion die Größe des Arrays bestimmen.

Elemente einfügen

Mit `SetAt` setzen Sie ein Element mit dem angegebenen Index. `GetAt` ermittelt das Element mit dem übergebenen Index. `SetAt` erweitert das Array nicht, wenn ein Index angegeben wird, der außerhalb des Arrays liegt. Sie können zu diesem Zweck jedoch `SetAtGrow` oder `Add` verwenden.

Anstelle der Elementfunktionen `SetAt` und `GetAt` können Sie den Operator `[]` verwenden. Dieser kann auf beiden Seiten einer Zuweisung angeordnet werden. Als L-Wert nutzt der Operator die `ElementAt`-Elementfunktion, die einen Verweis auf das angegebene Element zurückgibt.

Größe des Containers

Die `SetSize`-Funktion definiert ebenfalls den Wert, um den die Speicherreservierung für das Array vergrößert werden muß, wenn diesem weitere Elemente hinzugefügt werden. Die Standardimplementierung versucht einen optimalen Wert zu ermitteln, um die Heap-Fragmentierung zu minimieren. Jede zusätzliche Reservierung kann mit `FreeExtra` freigegeben werden.

Sie ermitteln die gegenwärtige Größe des Arrays mit `GetSize`. Die Funktion `GetUpperBound` gibt den größtmöglichen Index innerhalb des Arrays zurück.

Einfügen, Löschen, Kopieren

Mit `InsertAt` und `RemoveAt` fügen Sie dem Array Elemente an der angegebenen Position hinzu oder entfernen diese. Die Ausführung der Funktionen kann sehr viel Zeit beanspruchen, da große Datenblöcke verschoben werden müssen.

Die Elemente eines anderen Arrays vom gleichen Typ können mit der `Copy-Elementfunktion` an die angegebene Position in das Array kopiert werden. Eine weitere Möglichkeit besteht darin, diese Elemente an das Array anzuhängen. Verwenden Sie dazu die `Append-Elementfunktion`. Damit diese Funktionen korrekt ausgeführt werden, müssen Sie möglicherweise eine überschriebene Version der `CopyElements-Helferfunktion` zur Verfügung stellen.

Serialisierung

Die `CArray`-Klasse unterstützt die Serialisierung. Damit die Serialisierung fehlerfrei arbeitet, müssen Sie eventuell eine überladene Implementierung der `SerializeElements`-Funktion verwenden.

26.5.4 Das CMap-Template

Das `CMap`-Container-Template bietet einen indizierten Container an, der Schlüssel-/Wert-Paare aufnimmt.

Container-Deklaration

`CMap` wird wie folgt deklariert:

```
template<class KEY, class ARG_KEY, class VALUE, class ARG_VALUE>
    class CMap : public CObject
    {
        ...
    };
```

KEY und VALUE repräsentieren die Typen der Schlüssel und Werte. ARG_KEY und ARG_VALUE repräsentieren die Typen, die als Funktionsargumente übergeben werden. ARG_KEY verweist häufig auf KEY und ARG_TYPE auf TYPE, wie nachfolgend dargestellt:

```
CMap<CString, CString&, CString, CString&> myMap;
```

Eine effiziente Implementierung eines Containers, der auf CMap basiert, erfordert von Ihnen möglicherweise die Bereitstellung einer überladenen HashKey-Funktion für Ihren KEY-Typ.

Elemente hinzufügen

Schlüssel-/Wert-Paare werden dem Container mit der SetAt-Elementfunktion hinzugefügt. Der []-Operator kann ebenfalls dazu verwendet werden. Da nicht jeder Schlüsselwert in dem Container enthalten sein muß, kann der Operator nicht auf der rechten Seite einer Zuweisung verwendet werden.

Elemente suchen

Sie suchen mit der Lookup-Elementfunktion nach Elementen in dem Container.

Elemente löschen

Das dem angegebenen Schlüssel zugewiesene Element wird mit RemoveKey entfernt. RemoveAll entfernt alle Elemente aus dem Container.

Container durchlaufen

Die Elemente des Containers können in einer Schleife bearbeitet werden. Ermitteln Sie dazu mit einem Aufruf von GetStartPosition einen Zähler vom Typ POSITION. Ein wiederholter Aufruf der Funktion GetNextAssoc ermittelt die Elemente sukzessiv. Die Reihenfolge der Elemente ist willkürlich und muß nicht der Reihenfolge der Schlüssel entsprechen.

Größe des Containers

GetCount ermittelt die Anzahl der Elemente. IsEmpty prüft, ob in dem Container Elemente enthalten sind.

InitHashTable und GetHashTable initialisieren die Hash-Tabelle des Containers, so daß die Größe dieser Tabelle gesetzt und ermittelt werden kann.

26.5.5 Die CTypedPtrList-Template

Das CTypedPtrList-Template stellt eine typensichere Liste mit Zeigern zur Verfügung. Dazu wird ein Template-Mantel für die Klassen CObList und CPtrList implementiert. Diese Klassen basieren nicht auf Templates. CTypedPtrList wird wie folgt deklariert:

```
template<class BASE_CLASS, class TYPE>
    CTypedPtrList : public BASE_CLASS
{
    ...
};
```

BASE_CLASS sollte vom Typ CObList oder CPtrList sein. Verwenden Sie CObList, muß TYPE einen Zeiger auf eine von CObject abgeleitete Klasse repräsentieren. Entscheiden Sie sich für CPtrList, kann TYPE jeder Zeigertyp zugewiesen werden.

CTypedPtrList stellt Mantelfunktionen für alle Elementfunktionen der CObList- oder CPtrList-Klasse zur Verfügung, die sich auf die Typen der Container-Elemente beziehen. Die Mantelfunktionen führen die erforderlichen Typenkonvertierungen aus.

Das Verhalten von CTypedPtrList entspricht dem von CObList oder CPtrList.

CTypedPtrList unterstützt die Serialisierung, wenn der Container mit der CObList-Klasse verwendet wird. Wird statt dessen CPtrList benutzt, ist eine Serialisierung nicht möglich.

26.5.6 Die CTypedPtrArray-Template

Das CTypedPtrArray-Container-Template ist ein typensicheres Array, das Zeiger aufnimmt. Dieses Template ist ein Mantel für die Container CObArray und CPtrArray, die nicht auf Templates basieren. Die Deklaration geschieht wie folgt:

```
template<class BASE_CLASS, class TYPE>
    CTypedPtrArray : public BASE_CLASS
{
    ...
};
```

BASE_CLASS sollte entweder vom Typ CObArray oder CPtrArray sein. TYPE repräsentiert einen Zeigertyp. Dieser muß ein Zeiger auf ein von CObject abgeleiteter Typ sein, sofern CObArray als Basisklasse verwendet wird. Weisen Sie BASE_CLASS die Klasse CPtrArray zu, kann jeder Zeigertyp genutzt werden.

`CTypedPtrArray` stellt eine Mantelfunktion für jede `CObArray`- oder `CPtrArray`-Funktion zur Verfügung, die auf die Typen der Container-Elemente verweisen. Die Mantelfunktionen führen außerdem die erforderlichen Typenkonvertierungen aus.

Basieren die von `CTypedPtrArray` abgeleiteten Klassen auf `CObArray`, wird die Serialisierung unterstützt.

26.5.7 Das `CTypedPtrMap`-Template

Das `CTypedPtrMap`-Template dient der Erstellung typensicherer Zuordnungen. Es ist ein Mantel-Template für die Zuordnungs-Container-Klassen `CMapPtrToPtr`, `CMapPtrToWord`, `CMapWordToPtr` und `CMapStringToPtr`.

Ein typensichereres Verhalten wird durch die Implementierung von Mantelfunktionen für die Elementfunktionen der Basisklassen erreicht, die auf die Typen der Elemente verweisen.

`CTypedPtrMap` unterstützt nicht die Serialisierung.

26.6 Zusammenfassung

Die MFC-Bibliothek stellt Ihnen eine Auswahl verschiedener Container-Klassen zur Verfügung. In der Bibliothek sind sowohl Container, die nicht auf Templates basieren, als auch typensichere Container-Templates enthalten.

Die wohl überwiegend verwendeten Container-Klassen nehmen `CObject`-Zeiger auf. Der `CObList`-Container repräsentiert eine geordnete Liste, die `CObject`-Zeiger aufnimmt. Dieser Container speichert beispielsweise Fensterlisten in der MFC. Der Zugriff auf die Elemente eines Containers geschieht mit einem Zähler vom Typ `POSITION`. `CObArray` ist ebenfalls ein Container, der `CObject`-Zeiger aufnimmt. Die Funktion und das Verhalten dieses Containertyps entsprechen denen der C-Arrays. Ein Unterschied besteht jedoch: Ein `CObArray`-Container kann dynamisch erweitert und verkleinert werden. Sowohl `CObArray` als auch `CObList`-Container können serialisiert werden.

Andere Listen-Container sind `CPtrList` (eine Liste, die `void`-Zeiger aufnimmt) und `CStringList` (nimmt `CString`-Elemente auf). `CPtrList` unterstützt nicht die Serialisierung.

Weitere Array-Container sind `CPtrArray`, `CStringArray` und einige integrale Array-Typen. Ein `CPtrArray`-Container kann nicht serialisiert werden.

Die MFC-Bibliothek unterstützt ebenfalls Zuordnungen. Zuordnungen sind nicht geordnete Container, die durch Schlüssel indizierte Schlüssel-/Wert-Paare aufnehmen. Einige Zuordnungsklassen verwenden Schlüssel vom Typ `CString`, `WORD` und `void`-Zeiger. Die entsprechenden Werte können somit vom Typ `CString` sowie `WORD` und `void`- sowie `CObject`-Zeiger sein. Mit Ausnahme der Zuordnungen, in denen entweder der Schlüssel oder der Wert ein `void`-Zeiger ist, können Zuordnungs-Container serialisiert werden.

Seitdem Container-Templates in Visual C++ unterstützt werden, bietet die MFC-Bibliothek typensichere Container-Templates an. Einfache Templates sind Container eines bestimmten Typs, während typisierte Zeiger-Templates typensichere Container sind, die Zeiger aufnehmen.

Die einfachen Container-Templates verwenden mehrere überschreibbare Helferfunktionen. Dazu zählen `SerializeElements`, die der Serialisierung der Container-Elemente dient, und `CompareElements`, die anhand des Wertes nach Container-Elementen sucht. Weitere Helferfunktionen dienen der Erstellung und Zerstörung sowie dem Kopieren von Elementen, der Diagnose und dem Hashing.

Zu den einfachen Container-Templates zählen `CList` (gebundene Listen-Container), `CArray` (dynamisch reserviertes Array) und `CMap` (Schlüssel-/Wert-Zuordnung). Zu den Container-Templates, die auf Zeigern basieren, zählen `CTypedPtrList`, `CTypedPtrArray` und `CTypedPtrMap`.

Einfache Container-Templates können mit jedem Typ verwendet werden. Sie unterstützen die Serialisierung über die Helferfunktion `SerializeElements`.

Die Zeiger-Container-Templates verhalten sich wie die Container, die nicht auf Templates basieren. Sie wurden auf der Basis der Klassen `CObList`, `CPtrList`, `CObArray`, `CPtrArray` und der Zeiger-Zuordnungsklassen erstellt (`CMapPtrToPtr`, `CMapPtrToWord`, `CMapWordToPtr` und `CMapStringToPtr`). Die Serialisierung wird von `CTypedPtrList` und `CTypedPtrArray` unterstützt, wenn diese mit der Basisklasse `CObList` oder `CObArray` verwendet werden. Die auf Zeigern basierenden Zuordnungs-Templates unterstützen die Serialisierung nicht.

Ausnahmen, Multithreading und andere MFC-Klassen

Kapitel 27

Eine große Anzahl von MFC-Funktionen verwenden C++-Ausnahmen zur Fehlermeldung. Dieses Kapitel beginnt mit einer Übersicht über die Ausnahmearbeitung in der MFC. Unser besonderes Interesse gilt dabei der C++-Ausnahmearbeitung und der `CException`-Klasse.

Die MFC-Bibliothek unterstützt auch Multithreading. Eine spezifische Unterstützung für das Win32-Multithreading besteht in Form von Synchronisierungsklassen, aus denen Win32-Synchronisierungsobjekte erstellt werden. Das MFC-Multithreading sowie die `CSyncObject`-Klasse werden in der zweiten Hälfte des Kapitels besprochen.

Daß die MFC Thread-sicher ist, bedeutet nicht, daß Sie alle Features in Multithread-Anwendungen verwenden können. Zugrundeliegende Bibliotheken, die von der MFC genutzte Dienste anbieten, sind möglicherweise nicht Thread-sicher. Ein Beispiel hierfür ist die DAO-Bibliothek (Data Access Objects). MFC-DAO-Klassen können daher ausschließlich in dem primären Thread einer Anwendung verwendet werden. Der Grund hierfür besteht nicht in einer Einschränkung der MFC, sondern in einer Einschränkung der Dienste, die in der Bibliothek implementiert sind.



Schließlich erhalten Sie einen Einblick in verschiedene MFC-Klassen, wie zum Beispiel einfache Datentypen, Unterstützungsklassen für OLE und Datenbanken sowie weitere Klassen und Strukturen.

27.1 Verwenden von Ausnahmen in MFC-Anwendungen

Die MFC-Bibliothek bietet zwei Möglichkeiten der Ausnahmehandlung an. Unterstützt werden C++-Ausnahmen und die Ausnahmehandlung über konventionelle MFC-Makros.

27.1.1 Ausnahmebehandlung mit Makros

Obsolet! Neue Anwendungen sollten MFC-Ausnahmehandlungsmakros nicht mehr verwenden. Da jedoch sehr viele konventionelle Anwendungen auf diese Makros zurückgreifen, bieten die folgenden Absätze eine kurze Zusammenfassung, die beschreibt, wie diese Makros entsprechend der C++-Ausnahmesyntax in Programmcode konvertiert werden.

Umwandlung der Makros in C++-Ausnahmen

Dazu müssen Sie zunächst die Makrobezeichnungen durch C++-Schlüsselwörter ersetzen. Die Makros `TRY`, `CATCH`, `AND_CATCH`, `THROW` und `THROW_LAST` sollten durch die C++-Schlüsselwörter `try`, `catch` und `throw` ersetzt werden. Das `END_CATCH`-Makro verfügt über kein C++-Korrelat und sollte daher gelöscht werden.

CATCH Die Syntax des `CATCH`-Makros unterscheidet sich von der des C++-Schlüsselwortes `catch`.

```
CATCH(CException, e)
```

sollte daher durch

```
catch(CException *e)
```

ersetzt werden.

Ausnahmen löschen Ein wesentlicher Unterschied zwischen den beiden Möglichkeiten der Ausnahmehandlung besteht darin, daß Sie das Ausnahmeobjekt löschen müssen, wenn Sie den C++-Ausnahmehandlungsmechanismus verwenden. Rufen Sie dazu die `Delete-Elementfunktion` des Objekts vom Typ `CException` auf. Fangen Sie beispielsweise die folgende Ausnahme ab,

```
TRY
{
    // Hier den verursachenden Fehler einfügen
}
CATCH(CException, e)
{
    // Hier die Ausnahme bearbeiten
}
END_CATCH
```

sollten Sie den Programmcode wie folgt konvertieren:

```
try
{
    // Hier den verursachenden Fehler einfügen
}
catch (CException *e)
{
    // Hier die Ausnahme bearbeiten
    e->Delete();
}
```

Versuchen Sie nicht, ein Ausnahmeobjekt in einem catch-Block mit dem delete-Operator zu löschen. Dieser Versuch mißlingt, wenn das Ausnahmeobjekt nicht auf dem Heap reserviert wurde.



27.1.2 C++-Ausnahmen und die CException-Klassen

Die C++-Programmiersprache meldet und ermittelt ungewöhnliche Fehler über typisierte Ausnahmen. Die MFC verwendet dazu verschiedene Ausnahmetypen, die von der Klasse CException abgeleitet werden.

Die MFC-Bibliothek unterstützt strukturierte Win32-Ausnahmen nicht direkt. Möchten Sie strukturierte Ausnahmen bearbeiten, verwenden Sie entweder die strukturierte C-Ausnahmebearbeitung oder schreiben eine eigene Funktion, die strukturierte Ausnahmen in C++-Ausnahmen konvertiert.



Die primäre Aufgabe der CException-Klasse besteht darin, als Basisklasse für die Ausnahmen der MFC-Bibliothek zur Verfügung zu stehen. Sogar als leere Klasse erfüllt CException diese Aufgabe.

Sie bietet jedoch verschiedene Elementfunktionen an, die während der Bearbeitung einer Ausnahme genutzt werden können.

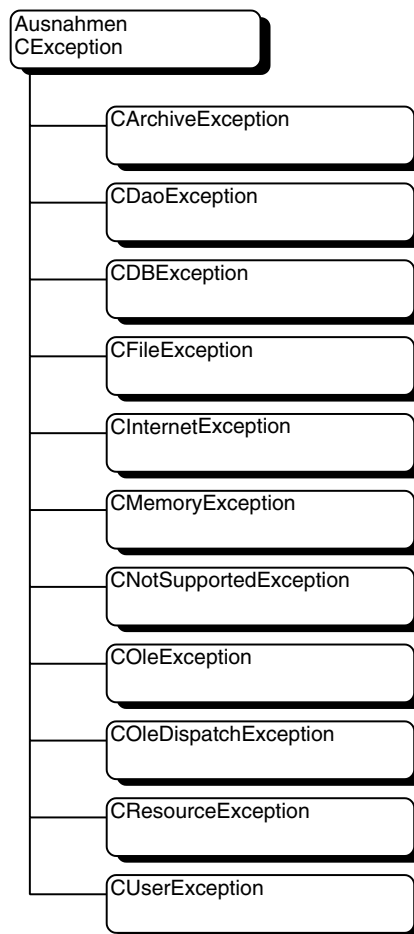
Die Elementfunktion GetErrorMessage gibt eine textliche Beschreibung des Fehlers zurück. Die Elementfunktion ReportError ermittelt die Fehlermeldung und zeigt diese in einem Standardnachrichtenfeld an. Beachten Sie bitte, daß nicht allen Ausnahmen, die von CException abgefangen werden, zulässige Fehlermeldungen zugewiesen sind.

Die dritte wichtige Elementfunktion ist Delete. Verwenden Sie die Funktion, um eine Ausnahme in einem catch-Block zu löschen, wenn Sie die Ausnahme bearbeiten. (Löschen Sie die Ausnahme nicht, wenn Sie diese mit throw an einen anderen Ausnahmebearbeiter weiterleiten.)

Diese Implementierung entspricht nicht dem ANSI-C++-Standard

Weiterhin bestehen einige von `CException` abgeleitete Klassen (Abbildung 27.1). Diese Klassen zeigen Fehler und ungewöhnliche Fehlerursachen an, die sich auf den Speicher, die Dateiverwaltung, die Serialisierung, die Ressourcenverwaltung, die Datenzugriffsobjekte, Datenbankfunktionen, Internet-Klassen und OLE beziehen. Die folgenden Abschnitte beschreiben diese Ausnahmeklassen.

Abbildung 27.1:
Ausnahmeklassen
in der MFC



Die `CMemoryException`-Klasse

Ausnahmen vom Typ `CMemoryException` zeigen einen Speicherreservierungsfehler an. Diese Ausnahmen werden automatisch von dem `new`-Operator der MFC ausgelöst. Schreiben Sie Ihre eigenen Funktionen zur Speicherreservierung, müssen Sie diese Ausnahmen selbst erzeugen:

```
char *p = malloc(1000);
if (p == NULL) AfxThrowMemoryException();
else
{
    // p mit Daten auffüllen
}
```

Die CFileException-Klasse

Ausnahmen vom Typ `CFileException` zeigen Fehler an, die sich auf Dateien beziehen. Die Ursache der Fehler wird mit Hilfe der `m_cause`-Elementvariable ermittelt:

```
try
{
    CFile myFile("myfile.txt", CFile::modeRead);
    // Ab hier den Inhalt der Datei lesen
}
catch(CFileException *e)
{
    if (e->m_cause == CFileException::fileNotFound)
        cout << "File not found!\n";
    else
        cout << "A disk error has occurred.\n";
    e->Delete();
}
```

Tabelle 27.1 führt die Werte auf, die `m_cause` zugewiesen sein können.

Wert	Beschreibung
none	Kein Fehler.
generic	Allgemeiner Fehler.
fileNotFound	Datei konnte nicht gefunden werden.
badPath	Die Pfadangabe ist unzulässig.
tooManyOpenFiles	Die maximale Anzahl der geöffneten Dateien wurde erreicht.
accessDenied	Es wurde versucht, eine Datei zu öffnen, für die die erforderlichen Rechte fehlen.
invalidFile	Ein unzulässiger Datei-Handle wurde verwendet.
removeCurrentDir	Es wurde versucht, das aktuelle Verzeichnis zu löschen.
directoryFull	Die maximale Anzahl der Verzeichniseinträge wurde erreicht.
badSeek	Der Dateizeiger konnte nicht auf die angegebene Position gesetzt werden.
hardIO	Hardware-Fehler.
sharingViolation	Es wurde versucht, auf einen gesperrten Bereich zuzugreifen.

Tabelle 27.1:
CFileException:
m_cause-Werte

Wert	Beschreibung
lockViolation	Es wurde versucht, einen Bereich zu sperren, der bereits gesperrt ist.
diskFull	Der Datenträger ist voll.
endOfFile	Das Ende der Datei wurde erreicht.



Diese `m_cause`-Werte sind unabhängig von dem verwendeten Betriebssystem. Möchten Sie einen spezifischen Betriebssystem-Fehlercode ermitteln, lesen Sie die Elementvariable `m_IOsError` aus.

Die beiden Elementfunktionen `OsErrorToException` und `ErrnoToException` konvertieren spezifische Betriebssystem-Fehlercodes und C-Laufzeitbibliothek-Fehlercodes in Ausnahmecodes. Mit Hilfe der beiden Helferfunktionen `ThrowOsError` und `ThrowErrno` werden die Ausnahmen mit diesen Fehlercodes ausgelöst.

Die CArchiveException-Klasse

Die `CArchiveException`-Klasse wird für Ausnahmen verwendet, die die Serialisierung betreffen. Diese Ausnahmen werden mit den Elementfunktionen der `CArchive`-Klasse ausgelöst.

Lesen Sie die `m_cause`-Elementvariable aus, um die Ursache der Ausnahme zu ermitteln:

```
CMyDocument::Serialize(CArchive &ar)
{
    try
    {
        if (ar.IsLoading())
        {
            // Hier aus dem Archiv lesen
        }
        else
        {
            // Hier in dem Archiv speichern
        }
    }
    catch (CArchiveException *e)
    {
        if (e->m_cause == CArchiveException::badSchema)
        {
            AfxMessageBox("Invalid file version");
            e->Delete();
        }
        else
        {
            throw;
        }
    }
}
```

Tabelle 27.2 führt die Werte auf, die `m_cause` zugewiesen sein können.

Wert	Beschreibung
<code>none</code>	Kein Fehler.
<code>generic</code>	Allgemeiner Fehler.
<code>readOnly</code>	Es wurde versucht, in ein Archiv zu speichern, das zum Lesen geöffnet wurde.
<code>endOfFile</code>	Das Ende der Datei wurde erreicht.
<code>writeOnly</code>	Es wurde versucht, aus einem Archiv zu lesen, das zum Speichern geöffnet wurde.
<code>badIndex</code>	Unzulässiges Dateiformat.
<code>badClass</code>	Es wurde versucht, ein Objekt eines unzulässigen Typs einzulesen.
<code>badSchema</code>	Inkompatible Schemanummer in der Klasse.

Tabelle 27.2:
CArchiveException: : *m_cause*-
Werte

Die CNotSupportedException-Klasse

Ausnahmen vom Typ `CNotSupportedException` werden ausgelöst, wenn ein nicht unterstütztes Feature angefordert wird. Weitere Informationen können nicht zu diesem Fehler ermittelt werden.

Die Ausnahme wird gewöhnlich in überschriebenen Versionen von Elementfunktionen abgeleiteter Klassen verwendet, wenn die abgeleitete Klasse nicht ein Feature der Basisklasse unterstützt. Die Klasse `CStdioFile` unterstützt beispielsweise nicht das `LockRange`-Feature der Basisklasse:

```
try
{
    CStdioFile myFile(stdin);
    myFile.LockRange(0, 1024);
    ...
}
catch (CNotSupportedException *e)
{
    cout << "Unsupported feature requested.\n";
    e->Delete();
}
```

Die CResourceException-Klasse

Ausnahmen vom Typ `CResourceException` werden ausgelöst, wenn die Speicherreservierung für eine Ressource fehlschlägt oder wenn eine Ressource nicht gefunden wird. Weitere Informationen werden nicht zu diesem Fehler angeboten.

Ausnahmen dieses Typs werden in Verbindung mit GDI-Ressourcen verwendet:

```
try
{
    CPen myPen(PS_SOLID, 0, RGB(255, 0, 0));
}
catch (CResourceException *e)
{
    AfxMessageBox("Failed to create GDI pen resource\n");
    e->Delete();
}
```

Die CDaoException-Klasse

CDaoException-Ausnahmen zeigen Fehler an, die auftreten, wenn MFC-Datenbankklassen zusammen mit Datenzugriffsobjekten verwendet werden (DAO). Alle DAO-Fehler sind DAO-Ausnahmen vom Typ CDaoException.

Detaillierte Informationen über einen Fehler werden über die Elemente der CDaoErrorInfo-Struktur ermittelt, auf die m_pErrorInfo verweist. Weitere OLE- und MFC-Fehlercodes werden mit Hilfe der Elementvariablen m_scode und m_nAfxDaoError ausgewertet.

Die GetErrorInfo-Elementfunktion gibt spezifische Informationen über einen DAO-Fehlercode zurück. Die Anzahl der Fehlercodes, für die Informationen erhältlich sind, wird mit GetErrorCount ermittelt.

Die CDBException-Klasse

Ausnahmen vom Typ CDBException zeigen Fehler an, die auftreten, wenn MFC-ODBC-Datenbankklassen verwendet werden.

Die m_nRetCode-Elementvariable enthält Informationen über den Fehler in Form eines ODBC-Fehlercodes. In der m_strError-Elementvariable ist eine textliche Beschreibung des Fehlers gespeichert. Detaillierte Informationen entnehmen Sie der Elementvariable m_strStateNativeOrigin, die eine textliche Erläuterung des Fehlers im folgenden Format aufnimmt:

```
"State: %s,Native: %ld,Origin: %s"
```

Die Formatcodes werden in dieser Zeichenfolge wie folgt ersetzt. Das erste Format (%s) wird durch einen aus fünf Zeichen bestehenden ODBC-Fehlercode ersetzt, der dem szSqlState-Parameter der ::SQL-Error-Funktion entspricht. Der zweite Formatcode entspricht dem pfNativeError-Parameter von ::SQL-Error und repräsentiert einen nativen Fehlercode, der sich auf die Datenquelle bezieht. Das dritte Format wird durch den Fehlertext ersetzt, der in dem szErrorMsg-Parameter der ::SQL-Error-Funktion zurückgegeben wird.

Die CInternetException-Klasse

Die CInternetException-Klasse wird von den MFC-Internet-Klassen zur Anzeige von Fehlern verwendet. Dazu zählen Verbindungs-Timeouts, DNS-Suchfehler und andere Internet-Probleme.

Die Klasse verfügt über die beiden Elementvariablen `m_dwContext` und `m_dwError`. Der Kontextwert wird dem Konstruktor von MFC-Klassen, wie zum Beispiel CInternetSession, übergeben. Er bezeichnet bestimmte Operationen in einer Anwendung, die mehrere Internet-Verbindungen gleichzeitig geöffnet hält.

Der Fehlerwert gibt Aufschluß über den Fehler selbst. Dieser kann ein allgemeiner Windows-Fehler (siehe `GetLastError`-Funktion) oder ein spezifischer Internet-Fehlercode sein. Internet-Fehlercodes sind in der Datei *WININET.H* definiert. Zu den vorwiegend auftretenden Fehlern zählen `ERROR_INTERNET_TIMEOUT`, `ERROR_INTERNET_NAME_NOT_RESOLVED` und `ERROR_INTERNET_CANNOT_CONNECT`.

Die COleException-Klasse

Die COleException-Klasse wird für Ausnahmen verwendet, die allgemeine OLE-Fehler anzeigen. Die `m_sc`-Elementvariable enthält Informationen über den Fehler in Form eines OLE-Statuscodes.

Die statische Elementfunktion `Process` wandelt eine abgefangene Ausnahme in einen OLE-Statuscode um. Wird der Funktion beispielsweise ein Objekt vom Typ CMemoryException übergeben, gibt sie den OLE-Statuscode `E_OUTOFMEMORY` zurück.

Die COleDispatchException-Klasse

Ausnahmen vom Typ COleDispatchException zeigen OLE-Fehler an, die sich auf die OLE-Schnittstelle IDispatch beziehen. Diese Schnittstelle wird zusammen mit der OLE-Automation und den OLE-Steuer-elementen verwendet.

Ein spezifischer IDispatch-Fehlercode ist in der Elementvariable `m_wCode` enthalten. Die Elementvariable `m_strDescription` nimmt eine textliche Beschreibung des Fehlers auf.

Weitere Elementvariablen enthalten einen Hilfekontext (`m_dwHelpContext`), die Bezeichnung der Hilfedatei der Anwendung (`m_strHelpFile`) und den Namen der Anwendung, die den Fehler ausgelöst hat (`m_strSource`).

Die CUserException-Klasse

Ausnahmen vom Typ `CUserException` zeigen einen durch den Anwender verursachten Fehler an. Diese Ausnahmen werden gewöhnlich ausgelöst, nachdem der Anwender in einem Meldungsfenster über die Ursache des Fehlers informiert wurde.

27.1.3 Auslösen einer MFC-Ausnahme

Möchten Sie eine MFC-Ausnahme aus Ihrem Programm heraus auslösen, verwenden Sie dazu eine der Helferfunktionen, die in der MFC zu diesem Zweck angeboten werden. Die Helferfunktionen sind in Tabelle 27.3 aufgeführt.

Tabelle 27.3:
Ausnahme-Helferfunktionen

Funktionsname	Beschreibung
<code>AfxThrowArchiveException</code>	Löst eine <code>CArchiveException</code> -Ausnahme aus.
<code>AfxThrowDaoException</code>	Löst eine <code>CDaoException</code> -Ausnahme aus.
<code>AfxThrowDBException</code>	Löst eine <code>CDBException</code> -Ausnahme aus.
<code>AfxThrowFileException</code>	Löst eine <code>CFileException</code> -Ausnahme aus.
<code>AfxThrowMemoryException</code>	Löst eine <code>CMemoryException</code> -Ausnahme aus.
<code>AfxThrowNotSupportedException</code>	Löst eine <code>CNotSupportedException</code> -Ausnahme aus.
<code>AfxThrowOLEDispatchException</code>	Löst eine <code>COLEDispatchException</code> -Ausnahme aus.
<code>AfxThrowOLEException</code>	Löst eine <code>COLEException</code> -Ausnahme aus.
<code>AfxThrowResourceException</code>	Löst eine <code>CResourceException</code> -Ausnahme aus.
<code>AfxThrowUserException</code>	Löst eine <code>CUserException</code> -Ausnahme aus.

Diese Funktionen verlangen abhängig von der ausgelösten Ausnahme unterschiedlich viele Parameter.

Die Funktionen erstellen ein Ausnahmeobjekt des angegebenen Typs, initialisieren es mit den übergebenen Parametern und lösen die Ausnahme anschließend aus.

Natürlich können Sie ein Ausnahmeobjekt auch manuell erstellen und auslösen. Diese Vorgehensweise ist möglicherweise erforderlich, wenn Sie eine Klasse von `CException` ableiten.

27.2 MFC und Multithreading

Die Multithreading-Unterstützung in der MFC umfaßt zwei Aspekte.

- Die MFC-Bibliothek ist Thread-sicher. Sie kann in Multithreading-Anwendungen somit verwendet werden.
- Außerdem verfügt die Bibliothek über Klassen, die die auf das Multithreading bezogenen Win32-Synchronisierungsobjekte explizit unterstützen.

27.2.1 Die Thread-sichere MFC

Eine kuriose, häufig wiederholte Aussage in der MFC-Dokumentation lautet: »MFC-Objekte sind lediglich auf der Klassenebene, nicht auf der Objektebene sicher.« Diese Aussage ist nicht in jedem Fall zutreffend.

Würden wir diesem Satz Glauben schenken, wäre es nicht sicher, separate Threads in einer Anwendung zu verwenden, um beispielsweise zwei Elementvariablen in einer von `CDocument` abgeleiteten Klasse dieser Anwendung zu bearbeiten. Der Gebrauch von Multithreading wäre sehr eingeschränkt und würde zu einer Inkompatibilität von Multithreading und der MFC führen. Glücklicherweise ist die erwähnte Aussage falsch.

Wenn Sie eigene Elementvariablen in einer abgeleiteten MFC-Klasse definieren, müssen Sie selbst dafür sorgen, daß diese Thread-sicher sind. Dazu können Sie beispielsweise Mantelfunktionen zur Verfügung stellen, die den Zugriff auf die Variablen einschränken. Synchronisierungstechniken sollten innerhalb dieser Mantelfunktionen nur mit Bedacht verwendet werden.

Zugriff auf Elementvariablen

Die zuvor aufgeführte Aussage ist insofern richtig, daß aus Gründen der Performance und einfachen Verwendung dieser besonnene Einsatz von Synchronisierungstechniken nicht in der MFC-Bibliothek realisiert wurde. Aus diesem Grund kann der Zugriff auf dasselbe Objekt durch zwei unterschiedliche Threads mißlingen, wenn kein Synchronisierungsmechanismus verwendet wurde.

Stellen Sie sich beispielsweise ein `CString`-Objekt vor. Wenn Sie diesem Objekt einen Wert zuweisen, gibt das Objekt den zuvor reservierten Speicher frei und reserviert den für die neuen Zeichenfolgendaten

erforderlichen Speicher. Anschließend werden die Daten in diesen Speicherblock kopiert. Diese Operationen sind nicht durch Synchronisierungstechniken geschützt. Versucht somit ein anderer Thread, den Wert von `CString` während dessen Aktualisierung zu ermitteln, befindet sich das Objekt in einem unbeständigen Zustand. Lediglich Abschnitte der Zeichenfolge und Garbage-Informationen würden in diesem Fall zurückgegeben. Außerdem könnte eine Zugriffsverletzung auftreten.

Wenn Sie Ihrer von `CDocument` abgeleiteten Klasse jedoch eine `CString`-Elementvariable hinzufügen, können Sie diese als `protected` oder `private` deklarieren und den Zugriff darauf durch Mantelfunktionen einschränken. In der Mantelfunktion können Sie beispielsweise Mutex-Objekte verwenden, um einen exklusiven Zugriff auf `CString` zur Verfügung zu stellen. Auf diese Weise verfügen die unterschiedlichen Threads in Ihrer Anwendung über die Möglichkeit, sicher auf dasselbe Objekt zuzugreifen.

Entgegen der erwähnten Aussage in der MFC-Dokumentation, ist ein sicherer Zugriff auf MFC-Objekte über mehrere Threads möglich, wenn Sie wissen, was Sie tun. Sie können beispielsweise in mehreren Threads über den Operator `operator_LPCSTR` auf ein `CString`-Objekt zugreifen. Versuchen Sie jedoch nicht, dieses `CString`-Objekt in zwei unterschiedlichen Threads gleichzeitig zu modifizieren.

Wissen Sie nicht, ob die zu realisierende Aufgabe sicher ist, betrachten Sie diese als unsicher. Gewöhnlich ist der gleichzeitige Zugriff auf ein Objekt in zwei unterschiedlichen Threads so lange unsicher, bis Sie wissen, daß der von Ihnen gewählte Zugriffsmechanismus zu keinen unbeabsichtigten Konsequenzen führt.

27.2.2 Erstellen von Threads in der MFC

Die MFC-Bibliothek unterscheidet zwischen zwei Thread-Typen.

- ➔ **BENUTZEROBERFLÄCHEN-THREADS** verfügen über eine Nachrichtenschleife und bearbeiten Nachrichten.
- ➔ **WORKER-THREADS** sind alle verbleibenden Threads.

Benutzeroberflächen-Threads

Benutzeroberflächen-Threads werden gewöhnlich zur Erstellung einer Nachrichtenschleife verwendet, die unabhängig von der Hauptschleife der Anwendung ausgeführt wird.

Das Erstellen eines Benutzeroberflächen-Threads bedingt das Ableiten einer Klasse von `CWinThread` und einen Aufruf von `AfxBeginThread`, woraufhin das Thread-Objekt erzeugt wird. In der von `CWinThread` abgelei-

teten Klasse müssen Sie einige `CWinThread`-Elementfunktionen überschreiben. Die `InitInstance`-Elementfunktion, die während der Initialisierung des Threads ausgeführt wird, muß in jedem Fall überschrieben werden.

Worker-Threads

Worker-Threads werden zur Ausführung einiger Hintergrundarbeiten verwendet (zum Beispiel für den Ausdruck im Hintergrund).

Das Erzeugen von Worker-Threads ist einfacher. Diese Threads erfordern keine von `CWinThread` abgeleitete Klasse. Statt dessen wird `AfxBeginThread` mit der Adresse der Thread-Funktion aufgerufen.

Nachfolgend finden Sie ein Beispiel zur Erstellung eines Worker-Threads aufgeführt:

```
UINT MyWorkerThread(LPVOID pParam)
{
    // Eine zeitintensive Aufgabe mit pParam ausführen
    return 0; // Thread beenden
}

...
// Irgendwo
AfxBeginThread(MyWorkerThread, myParam);
```

Beachten Sie bitte, daß MFC-Objekte nicht in Verbindung mit Threads eingesetzt werden sollten, die nicht mit `AfxBeginThread` erstellt wurden.



27.2.3 Thread-Synchronisierung

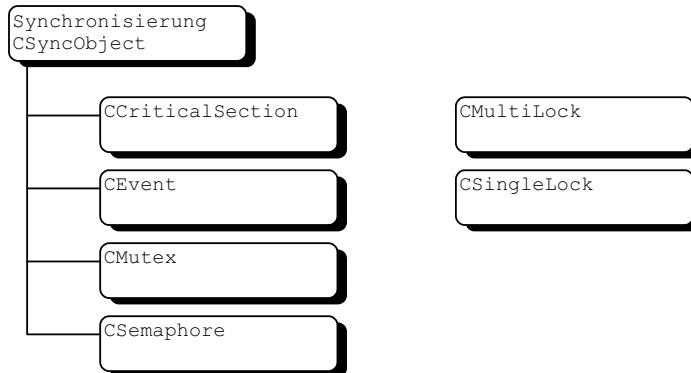
Die Win32-API bietet Synchronisierungsobjekte an, die der Synchronisierung gleichzeitig ausgeführter Threads dienen. Dazu zählen

- Ereignisse,
- Mutex-Objekte,
- kritische Abschnitte und
- Semaphoren,

die durch MFC-Mantelklassen unterstützt werden.

Die virtuelle Basisklasse aller MFC-Synchronisierungsklassen ist `CSyncObject`. Die Hierarchie der MFC-Synchronisierungsklassen ist in Abbildung 27.2 dargestellt.

Abbildung 27.2:
Synchronisie-
rungsklassen



Die `CSyncObject`-Klasse dient der Erstellung eines Synchronisierungsobjekts. Dazu wird der Name des Objekts dem Konstruktor übergeben. Die Elementfunktionen `Lock` und `Unlock` sperren den Zugriff auf das Synchronisierungsobjekt und geben dieses wieder frei. Die spezifische Bedeutung der Funktionen ist von der verwendeten Synchronisierungsklasse abhängig.

Die von `CSyncObject` abgeleiteten Objekte können in Verbindung mit den Klassen `CSingleLock` oder `CMultiLock` verwendet werden. Diese Klassen stellen einen Steuerungsmechanismus für den Zugriff auf die Objekte zur Verfügung.

27.2.4 Die `CEvent`-Klasse

Die `CEvent`-Klasse bietet die Funktionalität eines Win32-Ereignisses. Der Status eines Ereignisses wird mit einem Aufruf der `CEvent::SetEvent`-Funktion auf signalisierend gesetzt.

Ein Ereignis kann entweder ein manuell zurückgesetztes oder ein automatisches Ereignis sein. Ein manuell zurückgesetztes Ereignis muß explizit auf den nicht signalisierenden Status zurückgesetzt werden. Ein automatisches Ereignis wird auf den nicht signalisierenden Status zurückgesetzt, wenn ein wartender Thread freigegeben wird.

Das Warten auf die Signalisierung eines Ereignisses geschieht mit der `Lock`-Elementfunktion. Die `Unlock`-Elementfunktion wird für `CEvent`-Objekte nicht verwendet.

Die `PulseEvent`-Elementfunktion setzt den Status eines Ereignisses auf signalisierend, gibt wartende Threads frei (sofern vorhanden) und setzt den Status des Ereignisses zurück auf nicht signalisierend. Verwenden Sie die Funktion für ein manuell zurückzusetzendes Ereignis, werden alle wartenden Threads freigegeben. Nutzen Sie automatische Ereignisse, wird lediglich ein einzelner Thread freigegeben.

27.2.5 Die CMutex-Klasse

Durch die `CMutex`-Klasse repräsentierte Mutex-Objekte werden für einen exklusiven Zugriff auf eine Ressource verwendet. Während ein Thread ein Mutex besitzt, können andere Threads nicht darauf zugreifen.

Wenn Sie ein `CMutex`-Objekt erstellen, können Sie in dem Aufruf des Konstruktors bestimmen, ob Sie zuerst das Mutex-Objekt besitzen möchten. In diesem Fall wird die Konstruktorfunktion erst dann beendet, wenn der Besitz an dem Mutex-Objekt übernommen wurde.

Möchten Sie erst später den Besitz an einem Mutex-Objekt übernehmen, rufen Sie dazu die `Lock`-Elementfunktion auf. Mit `Unlock` geben Sie das Mutex-Objekt wieder frei.

27.2.6 Die CCriticalSection-Klasse

Kritische Abschnitte sind Objekte, deren Funktionalität der von Mutex-Objekten gleicht. Kritische Abschnitte sind jedoch ein wenig effizienter, können aber nicht außerhalb der Prozeßbegrenzungen verwendet werden. Kritische Abschnitte hindern mehrere gleichzeitig ausgeführte Threads daran, denselben Programmabschnitt auszuführen.

Ein kritischer Abschnitt wird von dem `CCriticalSection`-Konstruktor initialisiert. Mit den `Lock`- und `Unlock`-Elementfunktionen greifen Sie anschließend auf den kritischen Abschnitt zu. Um auf das zugrundeliegende Objekt zuzugreifen, verwenden Sie den Operator `operator_CRITICAL_SECTION*`.

Beachten Sie bitte, daß Objekte vom Typ `CCriticalSection` nicht zusammen mit den Klassen `CSingleLock` und `CMultiLock` verwendet werden können.

27.2.7 Die CSemaphore-Klasse

Semaphoren begrenzen die Anzahl der Zugriffe auf eine Ressource. Semaphor-Objekte werden durch die `CSemaphore`-Klasse repräsentiert.

Während der Erstellung eines Semaphor-Objekts durch den `CSemaphore`-Konstruktor, können Sie die anfängliche sowie die maximale Anzahl der Zugriffe angeben. Der Anfangswert kann mit einem Aufruf von `CSemaphore::Lock` erhöht werden. Wird die maximale Anzahl der Zugriffe erreicht, wartet die Funktion, bis das Semaphor-Objekt wieder verwendet werden kann. Die Anzahl der Zugriffe wird mit `Unlock` verringert.

27.2.8 Synchronisierung mit CSingleLock und CMultiLock

Auf Synchronisierungsobjekte vom Typ `CEvent`, `CMutex` und `CSemaphore` wird über die Synchronisierungszugriffsklassen `CSingleLock` und `CMultiLock` zugegriffen.

CSingleLock

Um ein Zugriffsobjekt vom Typ `CSingleLock` zu erstellen, müssen Sie zunächst das Synchronisierungsobjekt erzeugen. Anschließend übergeben Sie dem `CSingleLock`-Konstruktor einen Zeiger auf dieses Objekt. Der Zugriff auf das Objekt geschieht mit einem Aufruf von `CSingleLock::Lock`. Das Objekt wird mit `CSingleLock::Unlock` wieder freigegeben. Mit der `CSingleLock::IsLocked`-Elementfunktion ermitteln Sie, ob ein Objekt gesperrt ist.

CMultiLock

Die Funktionalität der `CMultiLock`-Klasse gleicht der von `CSingleLock`. `CMultiLock` ermöglicht jedoch das Warten auf mehrere Synchronisierungsobjekte.

Um ein `CMultiLock`-Objekt zu erstellen, müssen Sie dem entsprechenden Konstruktor ein Array übergeben, das aus von `CSyncObject` abgeleiteten Objekten besteht. Sie warten anschließend mit einem Aufruf der `Lock`-Elementfunktion darauf, daß einem oder allen dieser Objekte der signalisierende Status zugewiesen wird. Der Rückgabewert der Funktion bezeichnet das Objekt, dem dieser Status zugewiesen wurde. Sie geben das Objekt mit einem Aufruf von `CMultiLock::Unlock` wieder frei. Die `CMultiLock::IsLocked`-Funktion ermittelt den Zugriffsstatus des angegebenen Synchronisierungsobjekts.

Beachten Sie bitte, daß Objekte vom Typ `CCriticalSection` nicht in Verbindung mit `CSingleLock` und `CMultiLock` verwendet werden können.

27.3 Weitere MFC-Klassen

Die MFC-Bibliothek stellt weitere Klassen zur Verfügung, die allgemeinen Zwecken dienen (zum Beispiel `CString`), während andere Klassen in einem spezifischen Kontext verwendet werden. Die verbleibenden Abschnitte bieten eine Übersicht über diese Klassen.

27.3.1 Einfache Datentypen

In der MFC-Bibliothek sind einige Klassen enthalten, die einfache Datentypen repräsentieren.

CPoint	Die CPoint-Klasse kapselt die Win32-POINT-Struktur. Ein Zeiger auf ein CPoint-Objekt kann immer dann verwendet werden, wenn ein Zeiger auf eine POINT-Struktur verlangt wird. Die CPoint-Klasse unterstützt einige Operatoren zur Addition und Subtraktion und für Vergleichsoperationen sowie die Operatoren += und -=. Die Offset-Elementfunktion wird dazu verwendet, ein CPoint-Objekt anhand eines gegebenen Wertepaares in der horizontalen und vertikalen Richtung zu verschieben.
CRect	Die CRect-Klasse kapselt die Funktionalität der Win32-RECT-Struktur. Zeiger auf Objekte dieser Klasse und Zeiger auf Strukturen vom Typ RECT sind austauschbar. CRect unterstützt einige Elementfunktionen und überladene Operatoren, die für den Vergleich, zum Kopieren und Verschieben sowie zur Vergrößerung und Verkleinerung von Rechtecken verwendet werden können. Außerdem berechnen die Elementfunktionen die Verbindung und Schnittpunkte zweier Rechtecke.
CSize	Die CSize-Klasse kapselt die Win32-SIZE-Struktur. Zeiger auf CSize und Zeiger auf die SIZE-Strukturen sind austauschbar. Die CSize-Klasse definiert Operatoren für den Vergleich, zum Hinzufügen sowie zum Entfernen von CSize-Objekten. Die Operatoren zur Addition und Subtraktion können ebenfalls mit unterschiedlichen Typen verwendet werden. Objekte vom Typ CPoint und CRect können von einem Objekt vom Typ CSize oder CPoint mit Hilfe des Additions- oder Subtraktionsoperators verschoben werden.
CString	CString repräsentiert eine Zeichenfolge mit einer variablen Länge. Der Speicher für eine derartige Zeichenfolge wird in dem CString-Objekt dynamisch reserviert und freigegeben. Objekte vom Typ CString speichern ANSI- und OEM-Zeichenfolgen. Auf Systemen, die Unicode unterstützen

(wie zum Beispiel unter Windows NT), speichern `CString`-Objekte ebenfalls Unicode-Zeichenfolgen. Die `CString`-Klasse verfügt über sehr viele Funktionen und Operatoren, die zur Bearbeitung der Zeichenfolge verwendet werden können.

`CString`-Objekte können mit dem Additionsoperator miteinander verkettet werden. Der Vergleich von `CString`-Objekten geschieht mit Hilfe der Operatoren `==`, `<=` und `>=`. Die Elementfunktionen `Mid`, `Left` und `Right` führen ähnliche Aufgaben aus wie ihre BASIC-Pendants. Andere Funktionen extrahieren Bereiche der Zeichenfolge, ändern die Groß- und Kleinschreibung, suchen Zeichenketten innerhalb der Zeichenfolge und vergleichen Zeichenfolgen miteinander.

Die `CString`-Klasse unterstützt das Laden einer Zeichenfolge aus einer Windows-Ressourcdatei. Verwenden Sie dazu die `LoadString`-Funktion.

Die Serialisierung ist ebenfalls mit der `CString`-Klasse möglich. Zu diesem Zweck müssen die Operatoren `<<` und `>>` des `CString`-Objekts zusammen mit der `CArchive`-Klasse verwendet werden.

`CString`-Objekte können wegen des `operator_LPCSTR`-Operators häufig anstelle von Zeigern auf Elemente vom Typ `char` genutzt werden.

`CTime`

Die `CTime`-Klasse repräsentiert eine absolute Zeit. `CTimeSpan` bildet die Differenz zwischen zwei Zeitwerten. Beide Klassen verfügen über Elementfunktionen zum Setzen, Vergleichen und Bearbeiten von Zeitwerten. Außerdem extrahieren diese Funktionen verschiedene Elemente (zum Beispiel Sekunden, Minuten und Stunden) aus den Zeitwerten. Die `CTime`-Klasse unterstützt ebenfalls Zeitzonen und die Konvertierung eines `CTime`-Wertes in eine formatierte Zeichenfolge, die das Datum und die Zeit aufnimmt. Sowohl `CTime` als auch `CTimeSpan` unterstützen die Serialisierung und die Verwendung der Operatoren `<<` sowie `>>` in Verbindung mit der `CArchive`-Klasse.

Sehr viele Funktionen dieser Klassen wurden durch die der Klasse `COleDateTime` ersetzt.

27.3.2 Strukturen und Unterstützungsklassen

Verschiedene Strukturen und Klassen in der MFC unterstützen spezifische Funktionsbereiche.

CCommandLineInfo	CCommandLineInfo implementiert Kommandozeileninformationen in einer MFC-Anwendung. Ein Objekt vom Typ CCommandLineInfo oder ein davon abgeleitetes Objekt kann mit CWinApp::ParseCommandLine zur Auswertung der Kommandozeile verwendet werden. Die Standardimplementierung von CCommandLineInfo unterstützt einen Dateinamen in der Kommandozeile und einige Flags, die den Ausdruck, DDE, OLE-Automation und das Bearbeiten eines eingebetteten OLE-Objekts spezifizieren. Benötigen Sie andere Kommandozeilen-Flags, leiten Sie eine Klasse von CCommandLineInfo ab und überschreiben dessen ParseParam-Elementfunktion.
CCreateContext	Die CCreateContext-Klasse wird verwendet, wenn der Anwendungsrahmen Rahmenfenster und Ansichten erstellt, die mit einem Dokument in einer MFC-Applikationsrahmen-Anwendung verknüpft sind. Die Elementvariablen von CCreateContext sind Zeiger auf die Ansichtsklasse, das Dokument sowie die Ansicht- und Rahmenfenster.
CFileStatus	Die CFileStatus-Struktur wird von den Funktionen CFile::GetStatus und CFile::SetStatus verwendet, um die Attribute einer Datei zu setzen und zu ermitteln (Erstellungsdatum, Dateiname, Zugriffsrechte usw.).
CMemoryStatus	Die CMemoryState-Klasse ermittelt Speicherverluste. Indem Sie ein CMemoryState-Objekt erstellen und dessen Checkpoint-Elementfunktion zu verschiedenen Zeitpunkten der Programmausführung aufrufen, können Sie überprüfen, ob der reservierte Speicher korrekt freigegeben und der Inhalt nicht freigegebener Objekte gelöscht wurde.
CPrintInfo	CPrintInfo speichert Informationen zu einem Druckauftrag. Objekte dieser Klasse werden von dem Anwendungsrahmen während des Aufrufs von Elementfunktionen der CView-Klasse verwendet, die sich auf den Druckvorgang beziehen.

CCmdUI	Die CCmdUI-Klasse wird in ON_UPDATE_COMMAND_UI-Bearbeiterfunktionen der Klassen verwendet, die von CCmdTarget abgeleitet sind. Über Objekte dieses Typs können Anwendungen die Elemente der Benutzeroberfläche freigeben, sperren oder bearbeiten. Aktualisierungs-Bearbeiterfunktionen werden mit Hilfe des Klassen-Assistenten definiert.
CDataExchange	Die CDataExchange-Klasse ermöglicht den Dialog-Datenaustausch. Objekte dieser Klasse speichern Kontextinformationen, die von DDX- (Dialog Data Exchange) und DDV-Funktionen (Dialog Data Validation) verwendet werden. Klassen mit einer ähnlichen Funktionalität sind CPropExchange (Datenaustausch mit den Eigenschaften von OLE-Steuerelementen), CFieldExchange (Datenaustausch zwischen ODBC-Datensätzen und Dialogfeld-Steuerelementen) und CDaoFieldExchange (Datenaustausch zwischen DAO-Datensätzen und Dialogfeld-Steuerelementen).
CRectTracker	Die CRectTracker-Klasse implementiert ein Überwachungsrechteck für Objekte auf dem Bildschirm. Diese Klasse wird von dem Anwendungsrahmen zusammen mit eingebetteten OLE-Objekten verwendet. Sie kann jedoch ebenfalls von Anwendungen für spezifische Anwendungsobjekte genutzt werden.
CWaitCursor	Die CWaitCursor-Klasse stellt einen Sanduhr-Mauszeiger dar. Der Mauszeiger wird angezeigt, wenn ein Objekt von dieser Klasse erstellt wird. Nachdem das Objekt zerstört wurde, wird wieder der originale Mauszeiger angezeigt.

Weitere Unterstützungsklassen und Strukturen unterstützen OLE, die OLE-Automation, ODBC und DAO sowie den Microsoft-Internet-Informationen-Server.

27.4 Zusammenfassung

Die MFC-Bibliothek verwendet C++-Ausnahmen für die Ausgabe von Fehlermeldungen. Ausnahmen, die von `CException` abgeleitet sind, werden mit Helferfunktionen ausgelöst und von der Anwendung abgefangen.

Ältere MFC-Anwendungen verwenden Makros zu diesem Zweck. Diese Makros können einfach in die C++-Schlüsselworte `try`, `throw` und `catch` konvertiert werden.

Für die meisten dieser Ausnahmen bestehen Helferfunktionen (zum Beispiel `AfxThrowArchiveException`). Sie können außerdem ein von `CException` abgeleitetes Objekt erstellen und die Ausnahme manuell auslösen.

Sie sind dafür verantwortlich, daß das von `CException` abgeleitete Objekt in dem Ausnahmebearbeiter mit einem Aufruf der `Delete-Element-funktion` gelöscht wird.

Sie können ebenfalls selbst eine Klasse von `CException` ableiten.

Die Unterstützung der MFC von Multithreading berücksichtigt zwei Aspekte.

- Die MFC-Bibliothek ist auf der Klassenebene Thread-sicher.
- Außerdem wird Multithreading in Form der `CWinThread`-Klasse und einigen Synchronisierungsklassen angeboten, die sich von `CSyncObject` ableiten.

Die MFC unterscheidet zwischen Threads, die eine Nachrichtenschleife enthalten (Benutzeroberflächen-Threads) und gewöhnlichen Threads (Worker-Threads). Benutzeroberflächen-Threads werden durch das Ableiten einer Klasse von `CWinThread` erstellt, während Worker-Threads lediglich eine Worker-Thread-Funktion erfordern. Beide Threads werden mit einem Aufruf von `AfxBeginThread` erzeugt.

Zu den von `CSyncObject` abgeleiteten Synchronisierungsklassen zählen

- `CEvent`,
- `CMutex`,
- `CCriticalSection` und
- `CSemaphore`.

Jede dieser Klassen, abgesehen von `CCriticalSection`, kann mit den Klassen `CSingleLock` und `CMultiLock` verwendet werden.

Die MFC-Bibliothek definiert Unterstützungsklassen, die verschiedene Win32-Strukturen enthalten oder unterschiedliche Operationen ausführen. Einfache Datentypen sind `CPoint`, `CSize`, `CRect`, `CString`, `CTime` und `CTimeSpan`.

Andere Unterstützungsklassen und Strukturen sind `CCommandLineInfo`, `CCreateContext`, `CFileStatus`, `CMemoryState`, `CPrintInfo` und `CCmdUI`. Der Dialog-Datenaustausch geschieht mit `CDataExchange` und den spezifischen Klassen `CPropExchange`, `CFieldExchange` und `CDaoFieldExchange`. Die `CRectTracker`-Klasse implementiert ein Überwachungsrechteck. Die `CWaitCursor`-Klasse stellt einen Sanduhr-Mauszeiger dar. Weitere Unterstützungsklassen und Strukturen unterstützen OLE, die OLE-Automation, ODBC und DAO sowie den Microsoft-Internet-Informationen-Server.

Die Daten

Teil IV

- 
- 28. OLE, ActiveX und das Komponentenobjektmodell
 - 29. OLE-Server
 - 30. OLE-Container
 - 31. OLE-Drag&Drop
 - 32. Automatisierung
 - 33. Erstellen von ActiveX-
Steuerelementen mit
der MFC
 - 34. Verwenden der ActiveX-
Templatebibliothek
 - 35. ActiveX-Steuerelemente
verwenden
- 

OLE, ActiveX und das Komponentenobjekt- modell

Kapitel 28

OLE (Object Linking and Embedding) bildet den Kern vieler aktueller Windows-Anwendungen. Diese komplexe Technologie könnte nur sehr schwierig ohne die Hilfe der MFC angewendet werden.

Damit die MFC-Bibliothek effizient für OLE-Anwendungen verwendet werden kann, ist ein fundiertes Wissen über die OLE-Grundlagen erforderlich. Dieses Wissen ist für die Implementierung der OLE-Standard-Features (durch den Anwendungsassistenten) in Ihrer Anwendung nicht unbedingt notwendig. Möchten Sie jedoch auf erweiterte Features zurückgreifen, wie zum Beispiel OLE-Drag&Drop, eingebundene OLE-Objekte oder OLE-Zwischenablageoperationen, bildet das Verständnis dieser Technologie eine wesentliche Voraussetzung für Ihre Absicht.

Microsoft führte vor einiger Zeit einen neuen Begriff für eine der OLE-Techniken ein. Benutzerdefinierte OLE-Steuerelemente (früher OCX) sind nun ActiveX-Steuerelemente. ActiveX umfaßt jedoch noch weitere Verfahren, die auf derselben Grundlage basieren, wie OLE.

28.1 OLE-Grundlagen und das Komponentenobjektmodell

Die Basis von OLE und ActiveX ist eine Technologie, die als *Komponentenobjektmodell* bezeichnet wird (COM – Component Object Model). COM ist ein binärer Standard, der definiert, wie OLE- und ActiveX-Komponenten oder Objekte miteinander interagieren. Beachten Sie bitte, daß COM ein von Programmiersprachen unabhängiger Standard ist. Die einzige Anforderung, die eine Programmiersprache erfüllen muß, besteht darin, daß sie das Konzept der Zeiger und der Funkti-

onsaufrufe über Zeiger unterstützt. Das Entwickeln von COM-Anwendungen gestaltet sich daher in objektorientierten Umgebungen einfacher.

28.1.1 Schnittstellen und Methoden

Der exklusive Zugriff auf ein COM-Objekt geschieht über eine oder mehrere Schnittstelle(n). Eine SCHNITTSTELLE ist eine Gruppe verschiedener Funktionen, die auch als METHODEN bezeichnet werden.

Der COM-Standard spezifiziert nicht ausschließlich den binären Objektstandard, sondern ebenfalls einige Standardschnittstellen, die die grundlegende Funktionalität zur Verfügung stellen.

METHODDATA In anderen Worten beschrieben, ist eine COM-Schnittstelle eine Tabelle, die Funktionszeiger und Informationen zu diesen Zeigern aufnimmt. Die Informationen definieren die Parameter und Rückgabewerte der Funktionen. Die Methoden eines Automatisierungsobjekts sind beispielsweise in einer METHODDATA-Struktur aufgeführt, die wie folgt definiert ist:

```
typedef struct FARSTRUCT tagMETHODDATA {
    OLECHAR FAR* szName;
    PARAMDATA FAR* ppdata;
    DISPID dispid;
    UINT iMeth;
    CALLCONV cc;
    UINT cArgs;
    WORD wFlags;
    VARTYPE vtReturn;
} METHODDATA, FAR* LPMETHODDATA;
```

Das *iMeth*-Element dieser Struktur ist besonders interessant. Das Element ist ein Index in eine Tabelle von Funktionszeigern. In C++-Implementierungen wird *iMeth* mit der virtuellen Funktionstabelle einer C++-Klasse verwendet.

Virtuelle Tabellen Virtuelle Funktionstabellen sind nicht sehr bekannt. Programmierer nutzen die Vorteile virtueller Funktionen. Sie wissen jedoch häufig nicht, wie diese implementiert sind. Die nächsten Abschnitte widmen sich daher diesem Thema.

Die virtuellen Funktionen wurden in C++ eingeführt, um ein allgemeines Problem zu beheben: Wie können die Elementfunktionen einer abgeleiteten Klasse verwendet werden, wenn lediglich ein Zeiger auf eine Basisklasse zur Verfügung steht? Indem über eine mit dem entsprechenden Objekt verknüpfte Tabelle mit Funktionszeigern auf abgeleitete Funktionen verwiesen wird, gewährleistet der Compiler, daß die Objektfunktion auch dann aufgerufen wird, wenn Typeninformationen zu dem Objekt fehlen.

Ein Hinweis in Stroustrups THE ANNOTATED C++ REFERENCE MANUAL empfiehlt, eine Tabelle mit Funktionszeigern den Objektdaten voranzustellen. Diese Tabelle wird über Indizes in der METHODDATA-Struktur des Komponentenobjektmodells genutzt.

Da in der virtuellen Tabelle der C++-Objekte nur virtuelle Elementfunktionen eingetragen werden, müssen alle als C++-Elementfunktionen definierten COM-Methoden mit dem `virtual`-Schlüsselwort deklariert werden. Dies geschieht mit einigen Standardmakros, die später in diesem Kapitel beschrieben werden.

Implementieren Sie das Komponentenobjektmodell in C oder einer anderen Sprache, werden nicht automatisch virtuelle Funktionstabellen erzeugt. Sie müssen diese Tabellen möglicherweise manuell erstellen.



Beachten Sie bitte, daß eine COM-Schnittstelle weder eine C++-Klasse, ein C++-Objekt noch eine C-Struktur ist, die zur Implementierung der Schnittstelle verwendet wird.

Der COM-Standard bestimmt, wie Schnittstellen aufgebaut sein müssen!

Die Implementierung der Methoden wird nicht vorgeschrieben!

Der COM-Standard gibt somit zu verstehen, wie Tabellen, die auf Funktionsadressen verweisen, interpretiert werden. Informationen darüber, wie diese Funktionen das erwartete Verhalten implementieren, sind jedoch nicht erhältlich.

Schnittstellen sind in einem hohen Maße typisiert. Eine Schnittstelle kann nicht verändert oder gewechselt werden. Methoden können einer Schnittstelle nicht hinzugefügt und auch nicht daraus entfernt werden. Dies würde zu einer neuen Schnittstelle führen. (Natürlich können COM-Objekte mehrere Schnittstellen implementieren.)

28.1.2 Methoden und Speicherreservierung

Für die Implementierung von Methoden ist die Speicherreservierung von besonderem Interesse. Das Komponentenobjektmodell definiert bestimmte Regeln für Situationen, in denen der Aufrufer der aufgerufenen Methode einen Zeiger übergeben muß, oder in denen die Methode Daten in Form eines Zeigers an den Aufrufer zurückgeben muß.

- ➔ Wird Speicher von dem Aufrufer reserviert, muß dieser auch wieder von dem Aufrufer freigegeben werden.
- ➔ Wird einer Methode ein Zeiger übergeben, gibt diese den Speicher eventuell frei und reserviert ihn erneut. Der Aufrufer muß die letzte Reservierung wieder freigeben. Tritt ein Fehler auf, ist die Methode dafür verantwortlich.
- ➔ Reserviert die Methode Speicher, muß ebenfalls der Aufrufer diesen Speicher wieder freigeben.



Die Ausnahme bilden Fehler. Tritt ein Fehler auf, muß die Methode den von ihr reservierten Speicher freigeben und alle zurückgegebenen Zeiger auf `NULL` setzen.

Das Komponentenobjektmodell stellt eine Schnittstelle zur Speicherreservierung zur Verfügung (die `IMalloc`-Schnittstelle), die über Thread-sichere Speicherreservierungsmethoden verfügt. Ein Zeiger auf diese Schnittstelle wird mit der COM-Funktion `CoGetMalloc` ermittelt.

28.1.3 Vererbung und Wiederverwenden von Objekten

Vererbung und Wiederverwendung sind Begriffe mit einer besonderen Bedeutung für den Entwickler von objektorientiertem Programmcode. Diese Ausdrücke implizieren die Möglichkeiten, eigene Klassen von Basisklassen abzuleiten, Methoden durch angepaßte Versionen zu ersetzen und der abgeleiteten Klasse Methoden hinzuzufügen. Keine dieser Möglichkeiten wird für COM-Objekte unterstützt. Wenngleich eine Schnittstelle vererbt werden kann, gilt dies nicht für die Funktionalität der Schnittstelle. Die Schnittstelle enthält keine Implementierung.

Statt dessen werden COM-Objekte als Black Boxes behandelt. Detaillierte Informationen über die Implementierung der Schnittstelle werden nicht benötigt, lediglich die Spezifikationen der Schnittstelle selbst ist bekannt. Wir wissen somit, wie sich das Objekt verhält. Wie dieses Verhalten implementiert wird, erfahren wir jedoch nicht.

OLE bietet zwei Mechanismen zur Wiederverwendung an.

- ➔ *Containment/Delegation* ist ein Mechanismus, bei dem »äußere« Objekte als Clients der »inneren« Objekte agieren, die wiederum die Server bilden. Das klingt vertraut. Denken Sie an eine OLE-Zeichnung, die in ein Word-Dokument eingebettet ist.
- ➔ Der als *Aggregation* bezeichnete Mechanismus ermöglicht den äußeren Objekten, die Schnittstellen der inneren Objekte zu nutzen.

28.1.4 Schnittstellenbezeichner

Schnittstellen werden über global eindeutige Bezeichner identifiziert, die auch als GUIDs bezeichnet werden (Global Unique Identifiers).

GUIDs sind 128-Bit-Bezeichner. Jeder dieser Bezeichner ist überall auf der Welt eindeutig. Ein Programmierer, der einer Schnittstelle einen GUID zuweist, kann daher davon ausgehen, daß keine andere Schnittstelle über den selben Bezeichner verfügt. GUID

Das Visual-C++-Entwicklungssystem stellt zwei Programme zur Verfügung, die Ihnen bei der Erstellung global eindeutiger Bezeichner helfen:

➡ das Kommandozeilen-Hilfsmittel *UUIDGEN.EXE* und

➡ die Windows-Anwendung *GUIDGEN.EXE*.

Das Programm *GUIDGEN.EXE* erstellt Bezeichner, die über die Windows-Zwischenablage in den Quellcode eingefügt werden können.

Diese Programme nutzen die COM-API-Funktion *CoCreateGuid*, die wiederum die RPC-Funktion *UuidCreate* verwendet, um einen Bezeichner zu erstellen, der mit hoher Wahrscheinlichkeit global eindeutig ist.

28.1.5 Schnittstellendefinition über IUnknown

Alle COM-Objekte müssen die *IUnknown*-Schnittstelle implementieren, die drei Methoden definiert:

➡ *QueryInterface*,

➡ *AddRef* und

➡ *Release*.

AddRef und *Release* verwalten die Lebensdauer eines Objekts. Sie werden gewöhnlich als Funktionen implementiert, die einen Referenzzähler inkrementieren oder dekrementieren. Erreicht der Referenzzähler in *Release* den Wert Null, sollte das Objekt zerstört werden.

QueryInterface ermittelt bestimmte Schnittstellen in einem Objekt. Der Methode wird der eindeutige Bezeichner der zu ermittelnden Schnittstelle übergeben. Die Methode gibt entweder einen indirekten Zeiger auf die Schnittstelle oder einen Fehler zurück, wenn die Schnittstelle nicht von dem Objekt unterstützt wird.

Möchten Sie die *IUnknown*-Schnittstelle in einer C++-Klasse verwenden, leiten Sie eine Klasse von der *IUnknown*-Klasse ab, die in *UNKNWN.H* deklariert ist. Die Elementfunktionen *QueryInterface*, *AddRef* und *Release* sind als virtuelle Funktionen deklariert. Sie müssen Ihre eigene Implementierung vornehmen.

28.1.6 Klassenobjekte und Registrierung

Ein Klassenobjekt sollte nicht mit dem Konzept einer Klasse in den objektorientierten Sprachen verwechselt werden.

Klassenobjekte sind COM-Objekte Ein Klassenobjekt ist ein COM-Objekt, das die `IClassFactory`-Schnittstelle implementiert. Diese Schnittstelle ist der Schlüssel zu einem grundlegenden COM-Feature. Über diese Schnittstelle können Anwendungen Objekte für diese Klasse erstellen. Dies geschieht, indem die Klasse registriert wird und einige Methoden der Klasse aufgerufen werden.

Eine Klasse ist mit einem `CLSID` bezeichnet, der einem GUID ähnlich ist. Das Betriebssystem unterhält eine Datenbank, die alle registrierten `CLSIDs` des Systems aufnimmt. In der Windows-Umgebung wird dazu die Windows-Registrierung verwendet. Registrierungseinträge werden unter dem Schlüssel `HKEY_CLASSES_ROOT\CLSID` gespeichert. Der `CLSID` ist dort als Zeichenfolge vorhanden.

Anwendungen können einen `CLSID` und die COM-API-Funktionen `CoGetClassObject` und `CoCreateInstance` dazu verwenden, ein Klassenobjekt oder ein nicht initialisiertes Objekt zu erstellen, das dem `CLSID` entspricht.

28.1.7 Übergreifende Objektkommunikation

Nachdem Sie einen Zeiger auf eine Schnittstelle ermittelt haben, können Sie die in der Schnittstelle enthaltenen Methoden aufrufen.

- ➔ Befindet sich die Schnittstelle in demselben Prozeß wie der Aufrufer, wird der Aufruf direkt, also ohne einen Eingriff des Betriebssystemcodes, an die Funktionsimplementierung der Schnittstellenmethoden weitergegeben.
- ➔ Befindet sich die Schnittstelle außerhalb des aktuellen Prozesses, greift der Betriebssystemcode ein. (Denken Sie daran, daß Win32-Prozesse in separaten Speicherbereichen ausgeführt werden und keinen Einfluß auf andere Prozesse oder Daten haben.)

Marshaling für prozeßexterne Server Damit der Aufruf einen Server außerhalb des Prozesses erreicht, müssen die Aufrufparameter auf der Client-Seite gepackt und auf der Server-Seite entpackt werden. Das Packen von Parametern für den Transfer zwischen Prozessen wird als `MARSHALING` bezeichnet. Das Entpacken auf der Server-Seite wird `UNMARSHALING` genannt. COM bietet einen nützlichen und effizienten Marshaling-Mechanismus an (Standard-Marshaling). Anwender können jedoch angepaßte Marshaling-Techniken implementieren (benutzerdefiniertes Marshaling).

Das Marshaling wird immer von einem PROXY-OBJEKT ausgeführt. Die Tabelle mit den Funktionszeigern, die die Objektmethoden repräsentieren, verweisen auf Stub- und nicht auf aktuelle Implementierungen. Die Stub-Implementierung konvertiert mit Hilfe eines Kommunikationsmechanismus, wie zum Beispiel RPC (Remote Procedure Call) einen Aufruf auf der Client-Seite in einen Aufruf auf der Server-Seite. Weder der Client noch der Server erkennen den Unterschied zwischen Aufrufen in und außerhalb von Prozessen.

28.1.8 Moniker

Moniker sind COM-Objekte, die die `IMoniker`-Schnittstelle implementieren. Anwendungen können über diese Schnittstelle einen Zeiger auf ein Objekt ermitteln, das den Moniker bezeichnet. Dies geschieht mit einem Aufruf der `IMoniker`-Methode `BindToObject`.

Das Komponentenobjektmodell verfügt über verschiedene Moniker-Typen.

- ➔ **DATEI-MONIKER** bezeichnen Objekte, die in Dateien gespeichert sind.
- ➔ **OBJEKT-MONIKER** bezeichnen Objekte, die in anderen Objekten enthalten sind. Dazu zählen beispielsweise ein in ein OLE-Container-Dokument eingebettetes Objekt und die Auswahl eines Zellenbereichs in einer Tabelle durch den Anwender.
- ➔ **ZUSAMMENGESETZTE MONIKER** sind miteinander verkettete Moniker. Dazu zählen beispielsweise Bereiche eines Dateipfades, die zusammen den vollständigen Pfad ergeben.
- ➔ **ANTI-MONIKER** entfernen Bereiche eines zusammengesetzten Monikers, so wie das Symbol »..<« in Pfadangaben Bereiche des Pfades entfernt.

Moniker werden zum Bezeichnen von COM-Objekten verwendet. Moniker können gespeichert werden. Der Name eines COM-Objekts ist somit beständig.

Ein selten benutzter Moniker ist der Zeiger-Moniker. Dieser Moniker enthält Moniker-ähnliche Schnittstellenzeiger, die übergeben werden können, wenn ein Moniker verlangt wird. Zeiger-Moniker sind nicht beständig. Sie können nicht gespeichert werden.

28.1.9 OLE und Threads

OLE verwendet eine spezifische Thread-sichere Implementierung. Dieses APARTMENT-MODELL definiert einige Regeln, die Anwendungen befolgen müssen, wenn sie Objekte aus separaten Threads desselben Prozesses erstellen und nutzen möchten.

Das Apartment-Modell kategorisiert Objekte nach den Threads, die diese Objekte besitzen. Objekte können lediglich in einem einzelnen Thread (einem APARTMENT) bestehen. Innerhalb desselben Threads können Methoden direkt aufgerufen werden. Werden Methoden jedoch außerhalb des Prozesses aufgerufen, muß die Marshaling-Technik verwendet werden. Die COM-Bibliothek bietet dazu einige Helferfunktionen an.

28.2 OLE und Verbunddokumente

Die OLE-Technologie wird überwiegend in Form von OLE-Containern und OLE-Servern verwendet. Container- und Server-Anwendungen ermöglichen den Anwendern, innerhalb einer einzigen Anwendung die Daten von unterschiedlichen Quellen und Programmen zu bearbeiten.

Die Verbunddokument-Technologie basiert, zusätzlich zu den Grundlagen des Komponentenobjektmodells, auf dem strukturierten Speichern und dem einheitlichen Datentransfer.

28.2.1 Strukturiertes Speichern

Das strukturierte Speichern geschieht mit zwei Schnittstellen, die traditionelle Funktionen der meisten Dateisysteme anbieten.

- Die *IStorage*-Schnittstelle bietet eine Funktionalität, die der von Dateisystemen gleicht (Verzeichnisse). Ein Speicherobjekt kann, wie Dateiverzeichnisse, hierarchische Verweise auf andere Speicherobjekte enthalten. Es überwacht außerdem die Position und die Größe der in ihm gespeicherten Objekte.
- Die *IStream*-Schnittstelle entspricht einer Datei. Ein Stream-Objekt enthält Daten in Form einer Byte-Sequenz.

IRootStorage Verbunddateien bestehen aus einem Basis-Speicherobjekt mit mindestens einem Stream-Objekt, das die nativen Daten repräsentiert. Weitere Speicherobjekte können verknüpfte oder eingebettete Elemente repräsentieren. Das auf Dateien basierende Speichern wird mit Hilfe der *IRootStorage*-Schnittstelle implementiert.

Objekte, die in Dokumente von Container-Anwendungen eingebettet werden können, müssen die `IPersistStorage`-Schnittstelle implementieren. Das Objekt kann daraufhin in einem Speicherobjekt gespeichert werden. Weitere beständige Speicherschnittstellen sind `IPersistStream` und `IPersistFile`.

IPersistStorage

Das strukturierte Speichern bietet weitaus mehr Vorteile als die hierarchische Darstellung von Objekten in einer Datei.

*Vorteile des
strukturierten
Speicherns*

- Das Ersetzen eines einzelnen Objekts erfordert nicht das erneute Schreiben der gesamten Verbunddatei.
- Auf Objekte kann einzeln zugegriffen werden, ohne die vollständige Datei laden zu müssen.
- Das strukturierte Speichern bietet außerdem mehreren Prozessen die Möglichkeit des gleichzeitigen Zugriffs.
- Des weiteren ist eine Transaktionsbearbeitung möglich (Austauschen der Funktionalität).

Die Verbunddatei-Implementierung ist von Betriebssystemen und Dateisystemen unabhängig. Eine Verbunddatei, die beispielsweise auf dem FAT-Dateisystem unter Windows 95/98 erstellt wurde, kann auf dem NTFS-Dateisystem oder auf dem Macintosh-Dateisystem wiederverwendet werden.

Die Namensvergabe für Speicher- und Stream-Objekte unterliegt einigen Regeln.

*Regeln für die
Namensvergabe*

- Das Basis-Speicherobjekt erhält die gleiche Bezeichnung wie die zugrundeliegende Datei.
- Für das Objekt gelten dabei die Dateinamenseinschränkungen des Dateisystems.
- Die Bezeichnungen von eingebetteten Objekten, die aus mehr als 32 Zeichen bestehen (einschließlich der abschließenden Null-Zeichenfolge), müssen von Implementierungen unterstützt werden.
- Das Umwandeln der Dateinamen in Großbuchstaben ist von der jeweiligen Implementierung abhängig.

28.2.2 Datentransfer

Der Datentransfer zwischen Anwendungen geschieht mit Hilfe der `IDataObject`-Schnittstelle. Diese Schnittstelle stellt einen Mechanismus zum Übertragen von Daten und für die Benachrichtigung über Änderungen in den Daten zur Verfügung.

Der Datentransfer wird von zwei Strukturen unterstützt: `FORMATETC` und `STGMEDIUM`.

FORMATETC Die `FORMATETC`-Struktur ist wie folgt definiert:

```
typedef struct tagFORMATETC
{
    CLIPFORMAT cfFormat;
    DVTARGETDEVICE *ptd;
    DWORD dwAspect;
    LONG lindex;
    DWORD tymed;
} FORMATETC; *LPFORMATETC;
```

Diese Struktur setzt die Idee der Zwischenablageformate um und stellt zusätzlich zu dem `cfFormat`-Parameter weitere Parameter zur Verfügung. Diese Parameter bezeichnen das Zielgerät, an das die Daten übermittelt werden sollen. Außerdem geben Sie Aufschluß darüber, wie der Transfer geschehen soll.

STGMEDIUM Die `STGMEDIUM`-Struktur realisiert die Idee der globalen Speicherbearbeiter, die in herkömmlichen Windows-Zwischenablageoperationen verwendet wurden. Die Struktur ist wie folgt definiert:

```
typedef struct tagSTGMEDIUM
{
    DWORD tymed;
    union {
        HBITMAP hBitmap;
        HMETAFILEPICT hMetafilePict;
        HENHMETAFILE hEnhMetaFile;
        HGLOBAL hGlobal;
        LPOLESTR lpszFileName;
        IStream *pstm;
        IStorage *pstg;
    };
    IUnknown *pUnkForRelease;
} STGMEDIUM;
```

Mit Hilfe dieser Strukturen können Daten mit einem effizienten Speicher-Mechanismus übermittelt werden.

28.2.3 Verbunddokumente

Verbunddokumente können zusätzlich zu nativen Daten zwei Elementtypen aufnehmen: `VERKNÜPFTE OBJEKTE` und `EINGEBETTETE OBJEKTE`.

- ➔ Verknüpfte Objekte werden nicht von ihrer ursprünglichen Position entfernt (zum Beispiel aus einer Datei). Verbunddokumente enthalten einen Verweis auf diese Elemente (eine `VERKNÜPFUNG`) und Informationen darüber, wie das Objekt dargestellt werden soll. Der Container kann das verknüpfte Objekt darstellen, ohne die Verknüpfung zu aktivieren. Er kann das Element sogar darstellen, wenn die Anwendung, mit der das Objekt erstellt wurde, nicht auf dem System vorhanden ist.

Mit dem Aktivieren der Verknüpfung ist der Aufruf der Server-Anwendung zur Bearbeitung der verknüpften Daten gemeint.

Das Verwenden von Verknüpfungen führt zu Container-Dokumenten mit einer geringen Größe und ist außerdem vorteilhaft, wenn das verknüpfte Objekt im Besitz eines anderen Anwenders ist, der nicht über das Container-Dokument verfügt.

- ➡ Eingebettete Objekte werden in dem Container-Dokument angeordnet. Der Vorteil dieser Objekte besteht darin, daß Dokumente als einzelne Dateien bearbeitet werden können. Verwenden Sie hingegen verknüpfte Objekte, müssen mehrere Dateien zwischen den Anwendern ausgetauscht werden. Außerdem sind Verknüpfungen nicht mehr gültig, wenn Objekte entfernt werden. (Windows implementiert keinen Überwachungsmechanismus für verknüpfte Objekte.)

Server

Server für Objekte in einem Container-Dokument werden entweder als PROZESS-SERVER oder als LOKALE SERVER implementiert.

- ➡ Ein Prozeß-Server ist eine DLL, die im Prozeßspeicher der Container-Anwendung ausgeführt wird. Der Vorteil eines Prozeß-Servers ist die Performance. Die Methoden werden in solch einem Server direkt aufgerufen.
- ➡ Lokale Server bieten ebenfalls einige Vorzüge. Sie unterstützen Verknüpfungen (Prozeß-Server nicht) und sind kompatibel mit OLE1. Außerdem können sie in einem separaten Prozeßspeicher ausgeführt werden (wodurch die Stabilität vergrößert wird und mehrere konkurrierende Clients bedient werden können).

Verbunddokumente unterstützen außerdem die VORORTAKTIVIERUNG. Dieser Mechanismus ermöglicht die Bearbeitung eingebetteter Objekte innerhalb des Fensters der Container-Anwendung. *Vorort-aktivierung*

Die grundlegende Unterstützung für Verbunddokument-Container und -Server stellen die Schnittstellen `IOleClientSite` und `IOleObject` zur Verfügung. Server implementieren außerdem die Schnittstellen `IDataObject` und `IPersistStorage`. Prozeß-Server verwenden `IViewObject2` und `IOleCache2`. Die Vorortaktivierung geschieht mit `IOleInPlaceSite`, `IOleInPlaceObject`, `IOleInPlaceFrame` und `IOleInPlaceActiveObject`.

28.3 Anwendung von COM und OLE

Wie die bisherigen Erläuterungen zeigen, ist COM eine Gruppe mit Spezifizierungen, die weitaus mehr als das Verknüpfen und Einbetten von Objekten definieren. Verbunddokumente bilden eine Anwendungsmöglichkeit des Komponentenobjektmodells. Andere Möglichkeiten sind die COM-Automatisierung, das OLE-Drag&Drop und ActiveX-Steuerelemente, die in den folgenden Abschnitten beschrieben werden. Auch Bereiche der MAPI (Messaging Application Programming Interface) basieren auf dem Komponentenobjektmodell.

Die meisten COM-Anwendungen erfordern bestimmte Registrierungseinträge. Diese werden unter den Schlüsseln `HKEY_CLASSES_ROOT\CLSID` und `HKEY_CLASSES_ROOT\Interfaces` vorgenommen.

28.3.1 OLE-Dokument-Container und OLE-Server

OLE-Container und OLE-Server implementieren zusammen die Verbunddokument-Technologie. OLE-Container nehmen Verbunddokumente auf, die aus verknüpften und eingebetteten Objekten bestehen. OLE-Server stellen die verknüpften und eingebetteten Elemente sowie die Funktionalität zur Verfügung, die für die Aktivierung der Objekte benötigt wird.

28.3.2 Automatisierung

Die COM-Automatisierung ermöglicht einer Automatisierungs-Server-Anwendung, Automatisierungsobjekte in Form ausgesuchter Eigenschaften und Methoden zur Verfügung zu stellen. Informationen über die Eigenschaften und Methoden werden von der `IDispatch`-Schnittstelle bereitgestellt. Automatisierungs-Clients können diese Schnittstelle abfragen.

Automatisierungsobjekte müssen nicht sichtbar sein. Ein Automatisierungs-Server kann beispielsweise wissenschaftliche Berechnungen durchführen, eine Rechtschreibprüfung vornehmen oder durch Namen bezeichnete physikalische Konstanten zur Verfügung stellen, ohne eine sichtbare Oberfläche darzustellen.

Automatisierungs-Clients sind Anwendungen, die Automatisierungsobjekte bearbeiten. Diese Clients können generisch sein (zum Beispiel in einer Entwicklungsumgebung wie Visual Basic) oder spezifische Automationsobjekte steuern.

28.3.3 OLE-Drag&Drop

OLE-Drag&Drop ist ein leistungsfähiger Mechanismus zum Implementieren der Drag&Drop-Funktionalität.

OLE-Drag&Drop wird über die Schnittstellen `IDropSource` und `IDropTarget` sowie über die Funktion `DoDragDrop` implementiert. Nachdem das Objekt der Drag&Drop-Operation und ein Zeiger auf eine `IDropSource`-Schnittstelle entgegengenommen wurden, führt `DoDragDrop` eine Schleife aus, in der die Mausereignisse überwacht werden. Befindet sich die Maus über einem Fenster, prüft `DoDragDrop`, ob das Fenster als zulässiges Drag&Drop-Ziel registriert ist. `DoDragDrop` ruft verschiedene Methoden von `IDropTarget` und `IDropSource` auf.

Die Drag&Drop-Funktionalität kann mit dem Ausschneiden und Einfügen über die Zwischenablage verglichen werden. Häufig ist eine gemeinsame Implementierung dieser Techniken vorteilhaft, da auf diese Weise die Wiederverwendbarkeit von Programmcode optimiert wird.

28.3.4 ActiveX-Steuerelemente

ActiveX-Steuerelemente repräsentieren eine Technologie, die ein Substitut der 32-Bit-Technologie für Visual-Basic-Steuerelemente ist und für Microsoft zur Schlüsseltechnologie für das Internet wurde.

ActiveX-Steuerelemente sind COM-Objekte, die eine erweiterte Schnittstelle zur Verfügung stellen. Diese Schnittstelle implementiert das Verhalten eines Windows-Steuerelements. ActiveX-Steuerelement-Server werden gewöhnlich als Prozeß-Server implementiert (eine OCX-Datei ist lediglich eine DLL mit einer besonderen Dateinamenserweiterung). ActiveX-Steuerelemente sind daher für das Internet geeignet, da die Steuerelement-DLL von einem Browser heruntergeladen und anschließend in dessen Prozeßspeicher ausgeführt werden kann.

ActiveX-Steuerelement-Container sind Anwendungen, die ActiveX-Steuerelemente in ihren Fenstern oder Dialogen darstellen.

28.3.5 Benutzerdefinierte Schnittstellen

Sie können für spezielle Anwendungen Ihre eigenen benutzerdefinierten COM-Schnittstellen entwickeln. Verwenden Sie dazu die Hilfsmittel des SDK (Software Development Kit), einschließlich des MIDL-Compilers (Microsoft Interface Definition Language).

28.4 Ein einfaches Beispiel

Bevor ich mit diesem Kapitel begann, dachte ich, kein passendes Beispiel finden zu können. Zwar können Programme, die OLE/COM unterstützen, sehr einfach mit Hilfe der MFC-Bibliothek geschrieben werden, doch läßt die Anwendung der MFC die zugrundeliegende Technologie unverständlich werden. Erklärte ich Ihnen lediglich, welche Schaltflächen im Visual Studio betätigt werden müssen, würden Sie die Konzepte und Prinzipien von OLE/COM nicht verstehen. Auch ein umfangreiches Listing eines vom Anwendungsassistenten generierten Programmcodes würde Ihnen nicht weiterhelfen.

Ich habe jedoch eine Möglichkeit gefunden, ein COM-Programm mit einer geringen Größe zu schreiben, das einige der COM-Grundlagen demonstriert. Diese Anwendung, die in diesem Kapitel aufgeführt ist und aus ein wenig mehr als dreihundert Programmzeilen besteht, ist in einer einzelnen Datei enthalten und bietet alle Features eines Automatisierungs-Servers.

Doch wieso ein Automatisierungs-Server? Nun, die Automatisierung verwendet den zugrundeliegenden Mechanismus in einer puristischen Weise. Würde ich beispielsweise einen OLE-Container implementieren, müßten verschiedene Formate berücksichtigt, Daten visuell dargestellt und Fenster verwaltet werden. Dasselbe gilt für OLE-Drag&Drop. Die Komplexität dieser Anwendungen würde die Erläuterung der grundlegenden Konzepte erschweren.

Ein Automatisierungs-Server hingegen kann mit einem minimalen Aufwand implementiert werden. Seine Anwendung kann einfach mit wenigen Skriptzeilen in einem Automatisierungs-Client, wie zum Beispiel Visual Basic, demonstriert werden.

28.4.1 Funktionelle Beschreibung

Der Automatisierungs-Server gibt die Zeichenfolge »Hello, World!« aus. Die entsprechende Zeichenfolge wird zentriert in seinem Fenster dargestellt. Gleichzeitig wird die Zeichenfolge als OLE-Automatisierungseigenschaft über die Methoden `get` und `put` verwendet.

Wurde der Server korrekt installiert, kann er als Automatisierungs-Client verwendet werden. Um den Server mit Visual Basic zu prüfen, erstellen Sie ein Formular mit einer einzigen Schaltfläche und weisen dieser den Programmcode in Listing 28.1 zu. Der Code aktiviert den Server und ändert den Standardtext gemäß den Vorgaben des Visual-Basic-Codes.

```

Sub Command1_Click ()
Dim hello As object
Set hello = CreateObject("HELLO.Application")
hello.text = "Hallo von Visual Basic!"
End Sub

```

Die Anwendung setzt eine Eigenschaft des Anwendungsobjekts. Diese `text`-Eigenschaft bestimmt, welcher Text in der Mitte des Anwendungsfensters dargestellt werden soll. Sie kann ausgelesen und gesetzt werden. Abbildung 28.1 zeigt die Hello-Anwendung, nachdem der Text von Visual Basic verändert wurde.

*Listing 28.1:
Verwenden des
HELLO-Automatisierungs-Servers
in Visual Basic*



*Abbildung 28.1:
Manipulation
des Hello-Servers
in Visual
Basic*

28.4.2 Die Hello-Server-Anwendung

Listing 28.2 führt die vollständige Hello-Server-Anwendung auf. Die Erläuterung des Programms beginnt mit der `WinMain`-Funktion. Beachten Sie jedoch, daß die Anwendung keine Fehlerprüfcodes enthält. Eingabefehler können die Programmausführung beeinträchtigen.

```

#include <windows.h>
#include <initguid.h>

#ifdef INITGUID
#define INITGUID
#endif

DEFINE_GUID(CLSID_CHello, 0xfeb8c280, 0xfd2d, 0x11ce, 0x87, 0xc3,
            0x0, 0x40, 0x33, 0x21, 0xbf, 0xac);

static PARAMDATA rgpDataTEXT = { OLESTR("TEXT"), VT_BSTR };

enum IMETH_CTEXT
{
    IMETH_SET = 0,
    IMETH_GET,
};

enum IDMEMBER_CTEXT
{

```

*Listing 28.2:
Ein Automatisierungs-Server*


```

        IDMEMBER_TEXT = DISPID_VALUE
    };

static METHODDATA rgmdataCHello[] =
{
    { OLESTR("TEXT"), &rgpDataTEXT, IDMEMBER_TEXT, IMETH_SET,
      CC_CDECL, 1, DISPATCH_PROPERTYPUT, VT_EMPTY },
    { OLESTR("TEXT"), NULL, IDMEMBER_TEXT, IMETH_GET,
      CC_CDECL, 0, DISPATCH_PROPERTYGET, VT_BSTR }
};

INTERFACEDATA idataCHello =
{
    rgmdataCHello, 2
};

class CHello;

class CText
{
public:
    STDMETHOD_(void, Set)(BSTR text);
    STDMETHOD_(BSTR, Get)(void);
    CText(CHello *pHello, char *p = NULL);
    ~CText();
    void Paint();
    HWND m_hwnd;
private:
    char *m_text;
    CHello *m_pHello;
};

class CHello : public IUnknown
{
public:
    static CHello *Create(char *p);
    STDMETHOD(QueryInterface)(REFIID riid, void **ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);
    CHello(char *p = NULL);
    CText m_text;
private:
    IUnknown *m_punkStdDisp;
    unsigned long m_refs;
};

class CHelloCF : public IClassFactory
{
public:
    static IClassFactory *Create();
    STDMETHOD(QueryInterface)(REFIID riid, void **ppv);
    STDMETHOD_(unsigned long, AddRef)(void);
    STDMETHOD_(unsigned long, Release)(void);
    STDMETHOD(CreateInstance)(IUnknown *punkOuter,
                              REFIID riid, void **ppv);
    STDMETHOD(LockServer)(BOOL fLock);
    CHelloCF() { m_refs = 1; }
private:
    unsigned long m_refs;
};

CHello *pHello;

CText::CText(CHello *pHello, char *p)
{

```

```

    m_pHello = pHello;
    if (p != NULL)
    {
        m_text = new char[strlen(p) + 1];
        strcpy(m_text, p);
    }
    else m_text = NULL;
    m_hwnd = NULL;
}

CText::~CText()
{
    delete[] m_text;
}

STDMETHODIMP_(void) CText::Set(BSTR p)
{
    char *bf;
    int size;
    size =
        WideCharToMultiByte(CP_ACP, NULL, p, -1, NULL, 0, NULL, NULL);
    bf = new char[size];
    WideCharToMultiByte(CP_ACP, NULL, p, -1, bf, size, NULL, NULL);

    delete[] m_text;
    if (p != NULL)
    {
        m_text = new char[strlen(bf) + 1];
        strcpy(m_text, bf);
    }
    else m_text = NULL;
    if (m_hwnd != NULL) InvalidateRect(m_hwnd, NULL, TRUE);
}

STDMETHODIMP_(BSTR) CText::Get()
{
    static WCHAR *wbf;
    BSTR bbf;
    int size;
    size = MultiByteToWideChar(CP_ACP, 0, m_text, -1, NULL, 0);
    wbf = new WCHAR[size];
    MultiByteToWideChar(CP_ACP, 0, m_text, -1, wbf, size);
    bbf = SysAllocString(wbf);
    delete[] wbf;
    return bbf;
}

void CText::Paint()
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT clientRect;

    if (m_text != NULL)
    {
        hDC = BeginPaint(m_hwnd, &paintStruct);
        if (hDC != NULL)
        {
            GetClientRect(m_hwnd, &clientRect);
            DPTOLP(hDC, (LPPPOINT)&clientRect, 2);
            DrawText(hDC, m_text, -1, &clientRect,
                DT_CENTER | DT_VCENTER | DT_SINGLELINE);
            EndPaint(m_hwnd, &paintStruct);
        }
    }
}

```

```

}

CHello *CHello::Create(char *p)
{
    ITypeInfo *pTI;
    IUnknown *pUnk;
    CHello *pHello = new CHello(p);
    pHello->AddRef();
    CreateDispTypeInfo(&iddataCHello, LOCALE_SYSTEM_DEFAULT, &pTI);
    CreateStdDispatch(pHello, &(pHello->m_text), pTI, &pUnk);
    pTI->Release();
    pHello->m_punkStdDisp = pUnk;
    return pHello;
}

STDMETHODIMP CHello::QueryInterface(REFIID riid, void **ppv)
{
    if (IsEqualIID(riid, IID_IUnknown))
        *ppv = this;
    else
    if (IsEqualIID(riid, IID_IDispatch))
        return m_punkStdDisp->QueryInterface(riid, ppv);
    else
    {
        *ppv = NULL;
        return ResultFromScode(E_NOINTERFACE);
    }
    AddRef();
    return NOERROR;
}

STDMETHODIMP_(unsigned long) CHello::AddRef()
{
    return ++m_refs;
}

STDMETHODIMP_(unsigned long) CHello::Release()
{
    if (--m_refs == 0)
    {
        if(m_punkStdDisp != NULL) m_punkStdDisp->Release();
        PostQuitMessage(0);
        delete this;
        return 0;
    }
    return m_refs;
}

#pragma warning(disable:4355)
CHello::CHello(char *p) : m_text(this, p)
{
    m_refs = 0;
}

IClassFactory *CHelloCF::Create()
{
    return new CHelloCF;
}

STDMETHODIMP CHelloCF::QueryInterface(REFIID riid, void **ppv)
{
    if(IsEqualIID(riid, IID_IUnknown) ||
        IsEqualIID(riid, IID_IClassFactory))
    {
        AddRef();
    }

```

```

        *ppv = this;
        return NOERROR;
    }
    *ppv = NULL;
    return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP_(unsigned long) CHelloCF::AddRef()
{
    return m_refs++;
}

STDMETHODIMP_(unsigned long) CHelloCF::Release()
{
    if (--m_refs == 0)
    {
        delete this;
        return 0;
    }
    return m_refs;
}

STDMETHODIMP CHelloCF::CreateInstance(IUnknown *punkOuter,
                                     REFIID riid, void **ppv)
{
    if(punkOuter != NULL)
        return ResultFromScode(CLASS_E_NOAGGREGATION);
    return pHello->QueryInterface(riid, ppv);
}

STDMETHODIMP CHelloCF::LockServer(BOOL fLock)
{
    return NOERROR;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                        WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_PAINT:
            pHello->m_text.Paint();
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;
    IClassFactory *pHelloCF;
    unsigned long dwHelloCF = 0;
    unsigned long dwRegHello = 0;

    OleInitialize(NULL);
    pHello = CHello::Create("Hello, World!");
    pHelloCF = CHelloCF::Create();

```

```

CoRegisterClassObject(CLSID_CHello, pHelloCF,
    CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE, &dwHelloCF);
RegisterActiveObject(pHello, CLSID_CHello, NULL, &dwRegHello);
pHelloCF->Release();

if (hPrevInstance == NULL)
{
    memset(&wndClass, 0, sizeof(wndClass));
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wndClass.lpszClassName = "HELLO";
    if (!RegisterClass(&wndClass)) return FALSE;
}
hwnd = CreateWindow("HELLO", "HELLO",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
    NULL, NULL, hInstance, NULL);
pHello->m_text.m_hwnd = hwnd;
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
RevokeActiveObject(dwRegHello, NULL);
CoRevokeClassObject(dwHelloCF);
pHello->Release();
OleUninitialize();
return msg.wParam;
}

```

WinMain WinMain beginnt mit einem Aufruf von `OleInitialize`. Diese Funktion initialisiert die OLE/COM-Bibliotheken.

Anschließend werden zwei Objekte vom Typ `CHello` und `CHelloCF` erstellt. Das erste Objekt repräsentiert die Schnittstelle des Anwendungsobjekts. Das zweite stellt eine `IClassFactory`-Schnittstelle zur Verfügung. Dazu gleich mehr.

Der Zugriff auf das Klassenobjekt wird mit `CoRegisterClassObject` freigegeben. Das aktive Objekt, das die Klasse repräsentiert, wird mit `RegisterActiveObject` registriert. Das Klassenobjekt kann anschließend freigegeben werden.

Die verbleibenden Zeilen von `WinMain` implementieren eine Standardnachrichtenschleife und ein einfaches Fenster, in dem der Anwendungstext dargestellt wird. Die COM-Registrierungen werden mit dem Beenden der Anwendung gelöscht. Außerdem wird das Anwendungsobjekt zerstört. Die Initialisierung der OLE/COM-Bibliothek wird ebenfalls gelöscht.

WndProc Bevor wir die Klassen `CHello` und `CHelloCF` untersuchen, sollten wir einen Blick auf die `WndProc`-Funktion werfen. Diese bearbeitet zwei Nachrichten. Die `WM_PAINT`-Nachricht führt dazu, daß der Inhalt des Anwendungsfensters mit den Daten des `CHello`-Objekts aktualisiert wird. Eine `WM_DESTROY`-Nachricht beendet die Anwendung.

Den Kern der Automatisierungs-Implementierung bilden die Klassen *CHello* und *CHelloCF*. Beide Klassen verwenden die *IUnknown*-Schnittstelle. Die Implementierungen der Funktionen *AddRef* und *Release* bedürfen keiner Erklärung. In *QueryInterface* reagiert *CHello* auf Anfragen für *IUnknown* und *IDispatch*. *CHelloCF* reagiert auf Anfragen für *IUnknown* und *IClassFactory*.

CHelloCF verwendet eine einfache Implementierung von *CreateInstance*, die einen Zeiger auf die entsprechende von *CHello* bereitgestellte Schnittstelle zurückgibt.

Die Objekte *CHello* und *CHelloCF* werden mit der statischen Elementfunktion *Create* erstellt. Das Generieren von *CHelloCF* ist sehr einfach. In *CHello::Create* müssen einige Aufgaben ausgeführt werden. *CHello* verwendet eine Standardimplementierung, um eine *IDispatch*-Schnittstelle nutzen zu können. Diese Implementierung wird mit *CreateDispTypeInfo* und *CreateStdDispatch* erzeugt. Die von der Standardimplementierung verwendeten Informationen befinden sich in Form von Strukturen zu Beginn der Datei.

Die Methoden *get* und *set*, die die *text*-Eigenschaft zur Verfügung stellen, befinden sich in der dritten Klasse mit der Bezeichnung *CText*. Die ersten beiden Elementfunktionen dieser Klasse ermitteln den Wert der *m_text*-Elementvariable oder setzen diesen. Eine weitere Elementfunktion, *Paint*, zeigt den Text in dem durch die *m_hwnd*-Elementvariable bezeichneten Fenster an. Die *Set*-Elementfunktion ruft *InvalidateRect* auf, um das Fenster erneut zeichnen zu lassen (und somit zu gewährleisten, daß der modifizierte Text angezeigt wird).

Beachten Sie bitte, wie *CText::Get* und *CText::Set* Zeichenfolgen bearbeiten. Sie konvertieren OLE-Zeichenfolgen in darstellbare ASCII-Zeichenfolgen und umgekehrt. Beachten Sie auch, wie *CText::Set*, gemäß den Regeln der Speicherreservierung, nicht den für den *BSTR*-Parameter reservierten Speicher freigibt. Auch *CText::Get* gibt nicht den Speicher für den Rückgabewert frei. Dafür ist der Aufrufer (die OLE/COM-Bibliothek) verantwortlich.

Der CLSID für den Server wird mit *GUIDGEN.EXE* ermittelt.

Die Anwendung kann über die Kommandozeile kompiliert werden. Geben Sie dazu die folgende Anweisung ein:

```
CL HELLO.CPP USER32.LIB GDI32.LIB OLE32.LIB OLEAUT32.LIB
UUID.LIB
```

Natürlich können Sie auch ein Visual-C++-Projekt für dieses Programm erstellen, um die Debug-Features des Visual Studios zu nutzen.

28.4.3 Registrieren und Starten des Servers

Nachdem das Programm kompiliert wurde, kann es gestartet werden. Soll es jedoch als Automatisierungs-Server verwendet werden, müssen einige Einträge in der Registrierung vorgenommen werden. Diese sind in Listing 28.3 aufgeführt.

Listing 28.3:
Registrierungs-
einträge

```
HKEY_CLASSES_ROOT\  
HELLO.Application = HELLO Application  
HKEY_CLASSES_ROOT\  
HELLO.Application\CLSID = {FEB8C280-FD2D-11ce-87C3-00403321BFAC}  
  
HKEY_CLASSES_ROOT\  
CLSID\{FEB8C280-FD2D-11ce-87C3-00403321BFAC} = HELLO Application  
HKEY_CLASSES_ROOT\  
CLSID\{FEB8C280-FD2D-11ce-87C3-00403321BFAC}\LocalServer32 =  
HELLO.EXE /Automation  
HKEY_CLASSES_ROOT\  
CLSID\{FEB8C280-FD2D-11ce-87C3-00403321BFAC}\ProgId =  
HELLO.Application
```

Befindet sich die ausführbare Datei *HELLO.EXE* nicht in Ihrem Pfad, müssen Sie möglicherweise den *LocalServer32*-Schlüssel ändern, um den vollständigen Pfadnamen der Datei anzugeben.

Das manuelle Hinzufügen der Registrierungseinträge kann zu Fehlern führen. Sie haben daher die Möglichkeit, die Einträge mit Hilfe des Registrierungseditors zu importieren. Kopieren Sie Listing 28.3 dazu in eine ASCII-Datei (zum Beispiel *HELLO.REG*). Beachten Sie bitte, daß kein Zeilenwechsel die Einträge wie in dem Listing trennen darf. Jeder der fünf Einträge muß in eine Zeile geschrieben werden. Haben Sie die Zeilen übernommen, importieren Sie die Datei, indem Sie in dem Registrierungseditor den Eintrag *REGISTRIERUNGSDATEI IMPORTIEREN* aus dem Menü *REGISTRIERUNG* auswählen.

28.5 Zusammenfassung

Die komplexe Technologie von OLE und ActiveX bildet den Kern vieler moderner Windows-Anwendungen.

OLE und ActiveX verwenden das Komponentenobjektmodell (COM). Der Zugriff auf COM-Objekte geschieht über Schnittstellen, die aus Methoden bestehen. Der Standard definiert die Schnittstelle, aber nicht deren Implementierung. Diese muß der Programmierer zur Verfügung stellen.

COM-Objekte sind Black Boxes. Obwohl sie wiederverwendbare Komponenten repräsentieren, ist die Wiederverwendung wie in objektorientierten Sprachen nicht möglich. COM-Objekte können andere COM-Objekte enthalten und die Implementierung bestimmter Schnitt-

stellen an andere COM-Objekte delegieren. Sie können jedoch nicht ein Objekt von einem bereits bestehenden Objekt ableiten, so wie Sie unter C++ eine Klasse von einer Basisklasse ableiten.

COM-Objekte sind mit GUIDs bezeichnet. GUIDs sind 128 Bit breite global eindeutige Schnittstellenbezeichner.

Alle COM-Objekte verwenden die Schnittstelle `IUnknown`. Über diese Schnittstelle können Informationen über Schnittstellen ermittelt werden, die andere COM-Objekte möglicherweise nutzen.

Über die von COM-Klassenobjekten implementierte `IClassFactory`-Schnittstelle können Anwendungen Objekte erstellen. Klassenobjekte werden mit ihren CLSIDs in einer Systemdatenbank registriert (unter Windows die Registrierung). Ein CLSID entspricht einem GUID.

Objekte verfügen über zwei Möglichkeiten der Kommunikation. Wird eine Methode innerhalb des Prozeßspeichers des Aufrufers aufgerufen, wird die Funktion, die diese Methode implementiert, direkt aufgerufen. Befindet sich die Methode außerhalb des Prozeßspeichers des aufrufenden Prozesses, greift der Betriebssystemcode ein. Die Schnittstelle verwendet Proxy-Objekte und das Marshaling und Unmarshaling. Diese Techniken packen und entpacken die Parameter einer Methode.

COM-Objekte können ebenfalls anhand ihres Namens über Moniker identifiziert werden. Verschiedene Moniker-Typen identifizieren in Dateien gespeicherte Objekte oder Objekte, die in anderen Objekten gespeichert sind. Moniker können außerdem miteinander verkettet und zusammen mit Anti-Monikern verwendet werden, um neue Moniker zu erstellen.

OLE verwendet die Verbunddokument-Technologie. Diese Technologie basiert zusätzlich zu dem Komponentenobjektmodell auf dem strukturierten Speichern und dem einheitlichen Datentransfer. Verbunddaten werden wie die Verzeichnisse und Dateien eines Dateisystems hierarchisch in einer Datei gespeichert. Der einheitliche Datentransfer bietet einen Mechanismus für die Kommunikation zwischen Objekten und Anwendungen.

Weitere Anwendungsmöglichkeiten des Komponentenobjektmodells sind Automatisierung, ActiveX-Steuerelemente und OLE-Drag&Drop. Spezielle Anwendungen können benutzerdefinierte COM-Schnittstellen implementieren, die mit dem SDK erzeugt werden.

OLE-Server, die auch als OLE-Komponenten-Anwendungen bezeichnet werden, sind Programme, die OLE-Komponenten für die Verwendung in einer OLE-Container-Anwendung zur Verfügung stellen. Die MFC und Visual C++ ermöglichen die Entwicklung von Servern, Containern und Server-Containern.

29.1 Server-Konzepte

Dieser Abschnitt bietet eine Übersicht über einige Server-Konzepte. Anschließend erfahren Sie, wie die MFC das Erstellen von OLE-Servern unterstützt.

29.1.1 Voll-Server und Mini-Server

Zwischen Voll-Servern und Mini-Servern besteht ein Unterschied.

- ➡ Ein Voll-Server ist eine OLE-Komponenten-Anwendung die selbstständig ausgeführt werden kann.
- ➡ Ein Mini-Server kann lediglich aus einer Container-Anwendung aufgerufen werden.

Verwechseln Sie einen Mini-Server bitte nicht mit einem Prozeß-Server. Ein Prozeß-Server wird in dem Prozeßspeicher einer Client-Anwendung ausgeführt. Ein Beispiel für einen OLE-Prozeß-Server ist ein OLE-Steuerelement. Mini-Server sind ausführbare Programme. Sie bieten lediglich keine Features an, die eine selbständige



Ausführung rechtfertigen würden. Ein Mini-Server offeriert beispielsweise keine Funktionen, um eine Datei auf einem Datenträger zu speichern oder ein Dokument auszudrucken.

29.1.2 Vorortbearbeitung

Vorortbearbeitung repräsentiert die Möglichkeit, ein Serverobjekt innerhalb des Fensters einer Container-Anwendung darzustellen. Dazu sind komplexe Interaktionen zwischen dem Server und der Container-Anwendung erforderlich.

Zusätzlich zu der Darstellung des Serverobjekts in einem rechteckigen Bereich, übernehmen die Server ebenfalls die Symbolleisten und bestimmte Menüabschnitte der Container-Anwendungen.

29.1.3 Server-Aktivierung

Server werden über BEFEHLSVERBEN aktiviert. Ein Befehlsverb wird ausgeführt, indem die Methode `DoVerb` der `IOleObject`-Schnittstelle ausgeführt wird. Vordefinierte Werte der Befehlsverben korrespondieren mit der Vorortbearbeitung eines Objekts, der Bearbeitung eines Objekts in dessen Fenster sowie der Aktivierung und dem Verbergen eines Objekts.

29.2 Erstellen einer Server-Anwendung mit der MFC

OLE-Server-Komponenten-Anwendungen werden mit Hilfe der Visual-C++-Anwendungsassistenten erstellt. Zum Hinzufügen und Bearbeiten von Server-Funktionen verwenden Sie den Klassen-Assistenten.

29.2.1 Erstellen eines Anwendungsgerüsts mit dem Anwendungsassistenten

Möchten Sie eine OLE-Server-Anwendung mit dem Anwendungsassistenten generieren, müssen Sie zunächst bestimmen, ob Ihre Anwendung aus einem einzelnen (SDI) oder mehreren (MDI) Dokumenten bestehen soll.

1. Entscheiden Sie sich für eine MDI-Anwendung.

Für dialogfeldbasierende Anwendungen werden keine Server-Funktionen unterstützt. Möchten Sie einen Server erzeugen, der einem Dialog ähnlich ist (solch ein Server würde gewöhnlich nicht die visuelle Bearbeitung und die Vorortbearbeitung unterstützen), können Sie eine Ansichtsklasse vom Typ `CFormView` verwenden.



Unabhängig davon, ob Ihr neues Programm eine SDI- oder MDI-Anwendung ist,

2. wählen Sie im dritten Schritt des Anwendungsassistenten (Abbildung 29.1) einen Servertyp aus.

Drei Server werden dort angeboten. Sie können eine der Optionen

- ➡ MINI-SERVER,
- ➡ VOLL-SERVER oder
- ➡ CONTAINER und SERVER

auswählen.

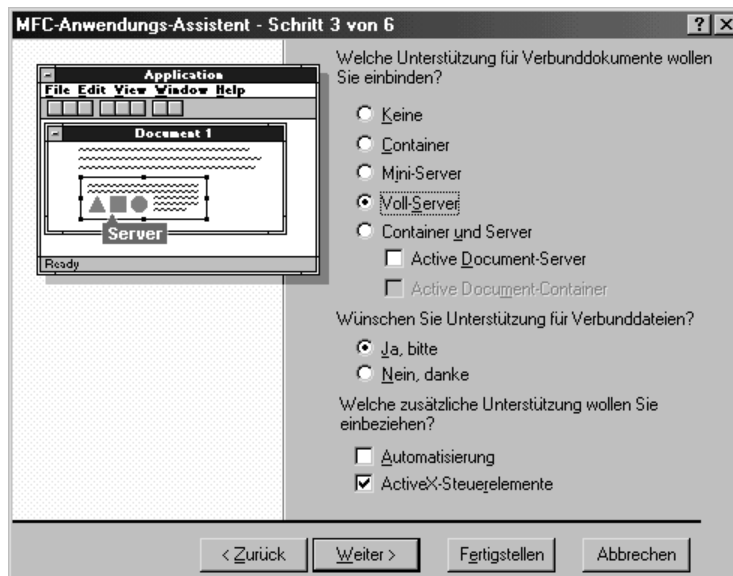


Abbildung 29.1:
Erstellen eines
OLE-Servers mit
dem Anwen-
dungsassisten-
ten

Auf dieser Dialogseite bestimmen Sie außerdem, ob Ihre Anwendung OLE-Verbunddateien verwenden soll. Diese erfordern zusätzlichen Festplattenspeicher, bieten jedoch eine verbesserte Performance und eine Standard-Dateistruktur.

Des Weiteren befinden sich Kontrollkästchen für die OLE-Automation und OLE-Steuerelemente auf der Dialogseite. Diese Optionen beziehen sich nicht auf OLE-Container und OLE-Komponenten-Server. Wir beachten sie daher zunächst nicht.

29.2.2 Das Anwendungsgerüst des OLE-Servers

Dieser Abschnitt erörtert das Anwendungsgerüst eines Voll-Servers, der von dem Anwendungsassistenten erstellt wurde.

Die OSRV-Anwendung wurde mit den Standardeinstellungen des Anwendungsassistenten als eine aus mehreren Dokumenten bestehende Voll-Server-Anwendung erstellt. Abbildung 29.2 zeigt die Klassen, die von dem Anwendungsassistenten für die Anwendung erzeugt wurden.

Abbildung 29.2:
Die von dem
Anwendungsas-
sistenten
erzeugten Klas-
sen für einen
OLE-Server



Worin unterscheiden sich diese Klassen von denen, die der Anwendungsassistent für Anwendungen erstellt, die keine OLE-Anwendungen sind?

- ➔ Ein Unterschied besteht darin, daß die beiden neuen Klassen `CInPlaceFrame` und `COSRVServerItem` angelegt wurden.
- ➔ Eine weitere Differenz ist weniger offensichtlich. Wenn Sie einen Doppelklick auf der Klassenbezeichnung `COSRVDoc` ausführen, werden Sie feststellen, daß diese Klasse nicht, wie für eine gewöhnliche Anwendung, von `CDocument` abgeleitet ist. Statt dessen bildet `COleServerDoc` die Basisklasse.

COleServerDoc Der Grund für diese Änderungen besteht darin, daß die Basisklasse `COleServerDoc` einige Dienste zur Verfügung stellt, die das Anwendungsdokument in einer Server-Umgebung implementieren. Diese Klasse ermöglicht außerdem die Interaktion mit einem Vorortrahmen,

bietet Container-Benachrichtigungsfunktionen an und verfügt über Helferfunktionen, die Objekte während der Vorortbearbeitung innerhalb des Fensters der Container-Anwendung positionieren.

Die neue von `COleIPFrameWnd` abgeleitete Klasse `CInPlaceFrame` repräsentiert das Vorort-Rahmenfenster. Während einer Vorortbearbeitung dient dieses Fenster als Child-Rahmenfenster (in einer SDI-Anwendung als Anwendungsrahmenfenster). Abbildung 29.3 vergleicht die Beziehung zwischen Rahmenfenstern und Ansichten während der Vorortbearbeitung und der Bearbeitung innerhalb einer selbständig ausgeführten Anwendung.

Das Vorort-Rahmenfenster

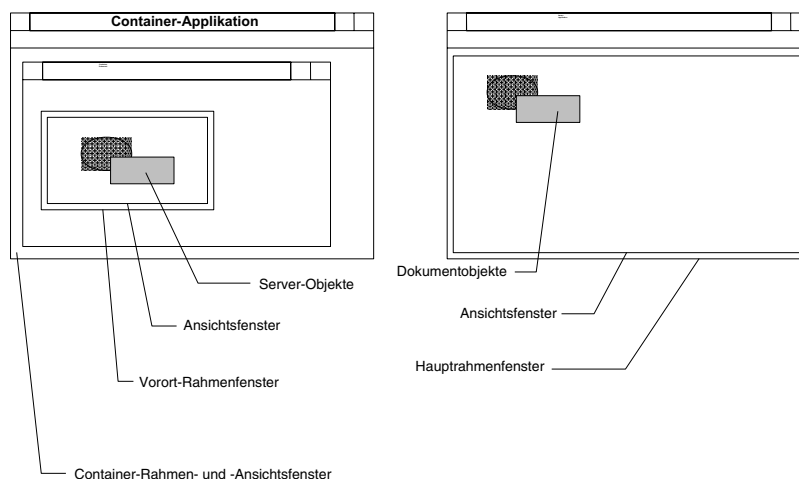


*Abbildung 29.3:
Vergleich der
Vorortbearbei-
tung mit der
Bearbeitung in
einer selbstän-
dig ausgeführ-
ten Anwendung*

Die neue Klasse `COSRVsrvrItem` repräsentiert die OLE-Oberfläche in der Container-Anwendung.

29.2.3 Das Serverobjekt (COleServerItem)

Die von `COleServerItem` abgeleitete Klasse implementiert die COM-Schnittstellen `IOleObject` und `IDataObject`, die die Darstellung eines OLE-Serverobjekts unterstützen.

Die `COleServerItem`-Serverobjekte sind den von `COleDocument` abgeleiteten Dokumenten sehr ähnlich. Sie repräsentieren das gesamte Dokument oder einen Bereich des Dokuments in einem OLE-Datentransferkontext. Ihre Möglichkeiten sind jedoch nicht auf das Verknüpfen und Einbetten von Objekten beschränkt. Sie erleichtern außerdem den Zwischenablagetransfer und das OLE-Drag&Drop.

Serialisierung

Die überschreibbaren Elementfunktionen von `COleServerItem` sind `OnDraw` und `Serialize`. `COleServerItem::Serialize` serialisiert das in der Container-Anwendung eingebettete Objekt zum Speichern. `COleServerItem::OnDraw` zeichnet das Objekt. Gewöhnlich wird diese Funktion aufgerufen, um das eingebettete Objekt in einen Metadatei-Gerätekontext zu zeichnen. Das Metadatei-Objekt wird anschließend von der Container-Anwendung gespeichert. Diese Vorgehensweise führt dazu, daß der Server nicht aktiviert werden muß, wenn ein erneutes Zeichnen des Objekts erforderlich ist.

Die von dem Anwendungsassistenten generierte Standardimplementierung von `COleServerItem::Serialize` (Listing 29.1) verwendet die `Serialize`-Elementfunktion der Dokumentklasse zur Serialisierung. Einfachen Situationen wird diese Funktion gerecht. Die Implementierung muß jedoch überarbeitet werden, wenn ein von `COleServerItem` abgeleitetes Objekt verwendet wird, um lediglich Bereiche des Dokuments darzustellen. Dies ist der Fall, wenn der Server OLE-Verknüpfungen ermöglicht und `COleServerItem` genutzt wird, um Bereiche des Dokuments (beispielsweise die aktuelle Selektion) in die Zwischenablage zu kopieren.

*Listing 29.1:
Die Anwendungsassistenten-Implementierung der `Serialize`-Elementfunktion einer Serverobjekt-Klasse*

```
void COSRVsrItem::Serialize(CArchive& ar)
{
    // COSRVsrItem::Serialize wird automatisch aufgerufen, wenn
    // das Element in die Zwischenablage kopiert wird. Dies kann
    // automatisch über die OLE-Rückruffunktion
    // OnGetClipboardData geschehen. Ein Standardwert für
    // das eingebundene Element dient einfach zur Delegierung der
    // Serialisierungsfunktion des Dokuments. Wenn Sie Verweise
    // unterstützen, möchten Sie vielleicht nur einen Teil des
    // Dokuments serialisieren.

    if (!IsLinkedItem())
    {
        COSRVDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->Serialize(ar);
    }
}
```

Zeichnen

Die Standardimplementierung von `COleServerItem::OnDraw` (Listing 29.2) zeichnet keine Objekte. Sie müssen Ihre eigene Implementierung dieser Funktion vornehmen. Für einfache Situationen genügt eine Kopie der `OnDraw`-Elementfunktion Ihrer Ansichts-Klasse.

```

BOOL COSRVsrItem::OnDraw(CDC* pDC, CSize& rSize)
{
    // Entfernen Sie dies, wenn Sie rSize verwenden
    UNREFERENCED_PARAMETER(rSize);

    COSRVDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // ZU ERLEDIGEN: Setzen Sie Mapping-Modus und Extent
    // (Das Extent stimmt üblicherweise mit der von OnGetExtent
    // zurückgelieferten Größe überein)
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);

    // ZU ERLEDIGEN: Hier Code zum Zeichnen einfügen. Füllen Sie
    // wahlweise das HIMETRIC-Extent aus. Alle Zeichenoperationen
    // finden innerhalb des Metadatei-Gerätekontexts (pDC) statt.

    return TRUE;
}

```

*Listing 29.2:
Die Anwen-
dungsassisten-
ten-Implementie-
rung der
OnDraw-Ele-
mentfunktion ei-
ner Serverobjekt-
Klasse*

29.2.4 COleDocument und Dokumentobjekte

Die Klasse `COleDocument` ist die Basisklasse von `COleServerDoc`. Ein `COleDocument`-Dokument kann eine Liste von Dokumentobjekten vom Typ `CDocItem` enthalten. `CDocItem` ist die Basisklasse von `COleServerItem` und kann außerdem dazu verwendet werden, spezifische Dokumentobjekte zu implementieren. Es ist daher möglich, eine einfache OLE-Server-Anwendung zu generieren, ohne der von `COleDocument` abgeleiteten Klasse des Programms Programmcode hinzuzufügen. Der von dem Anwendungsassistenten erstellte Programmcode verwaltet die Liste der `CDocItem`-Objekte.

29.2.5 Das Vorort-Rahmenfenster

Das Vorort-Rahmenfenster verfügt über die gleiche Funktionalität wie das Rahmenfenster in SDI-Anwendungen. Es ist das Parent-Fenster der aktuellen Ansicht. Außerdem verwaltet das Vorort-Rahmenfenster Menüs und Symbolleisten. Dazu ersetzt es in Kooperation mit dem Rahmenfenster der Container-Anwendung die Menüs und Symbolleisten dieses Rahmenfensters für die Dauer der Vorortbearbeitung.

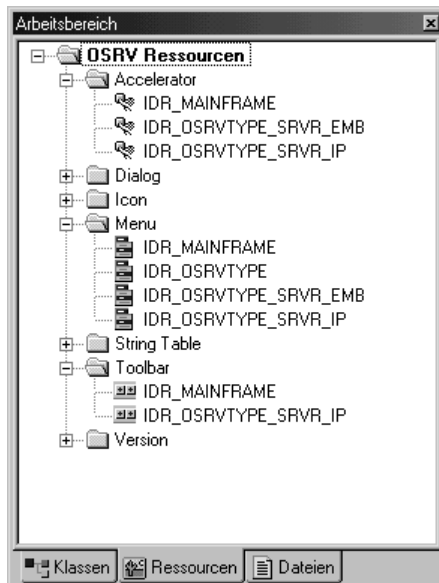
29.2.6 Operationsmodi und Ressourcen

Wenn Sie mit einem OLE-Server arbeiten, dürfen Sie die duale Funktion der Anwendung niemals vergessen. Eigentlich verfügt ein OLE-Server über drei grundlegende Operationsmodi.

- Er kann selbständig ausgeführt sowie
- zur Bearbeitung eines OLE-Objekts in seinem Fenster und
- zur Vorortbearbeitung verwendet werden.

Diesen Umstand gibt die vom Anwendungsassistenten generierte Ressourcendatei wieder (Abbildung 29.4). Wie Sie sehen, verwendet die Anwendung drei Zugriffstasten, vier Menüs und zwei unterschiedliche Symbolleisten.

Abbildung 29.4:
Ressourcen
einer OLE-Server-Anwendung



Zwei der vier Menüs werden in jeder Anwendung mit mehreren Dokumenten angelegt.

Ein Menü wird angezeigt, wenn keine Dokumente in der Anwendung geöffnet sind.

Andernfalls wird das zweite Menü dargestellt.

Ein OLE-Server muß ein drittes Menü anbieten, das den Status während der Bearbeitung eines eingebetteten Objekts wiedergibt. Der Unterschied zwischen diesem und dem regulären Menü besteht darin, daß in dem Menü DATEI die Einträge AKTUALISIEREN und KOPIE SPEICHERN UNTER anstelle der Einträge SPEICHERN und SPEICHERN UNTER enthalten sind. Das »Speichern« eines eingebetteten Objekts impliziert somit dessen Serialisierung in der Container-Anwendung. Das Speichern unter einem anderen Dateinamen erzeugt eine Kopie des eingebetteten Objekts.

Das vierte Menü (Abbildung 29.5) wird angezeigt, wenn der Server für die Vorortbearbeitung genutzt wird. Das Menü ist unvollständig. Die Positionen der beiden fehlenden Menüs sind durch zwei Trennlinien gekennzeichnet.

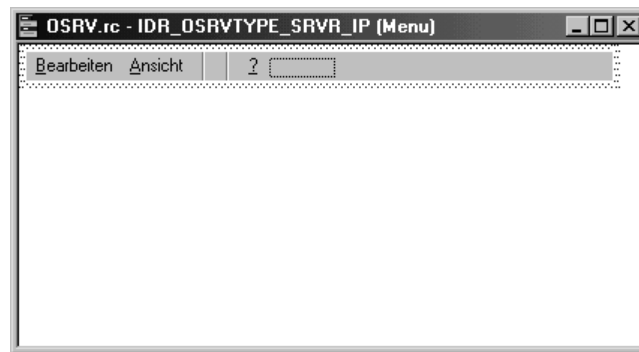


Abbildung 29.5:
Das unvollständige Menü, das während der Vorortbearbeitung verwendet wird

Während der Vorortbearbeitung werden dieses Menü und das Hauptmenü der Container-Anwendung miteinander kombiniert. Dieser Vorgang zeigt, daß während der Vorortbearbeitung verschiedene Funktionen von der Server-Anwendung und andere Funktionen (zum Beispiel die Fensterverwaltung) von der Container-Anwendung ausgeführt werden. Abbildung 29.6 verdeutlicht, wie sich das vollständige Menü aus den Menüs des Servers und des Containers zusammensetzt.

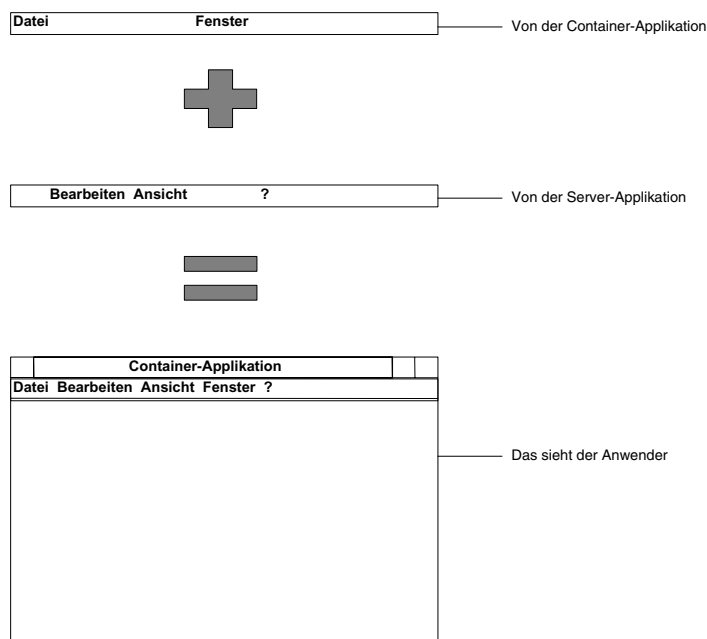


Abbildung 29.6:
Kombinieren von Server- und Container-Menüs während der Vorortbearbeitung

Da separate Menüs verwendet werden, bestehen in der Ressourcendatei ebenfalls separate Zugriffstastentabellen. Ein kurzer Blick auf die Symbolleisten (Abbildung 29.7) verdeutlicht, daß die während der Vorortbearbeitung verwendete Symbolleiste keine Dateifunktionen anbietet.

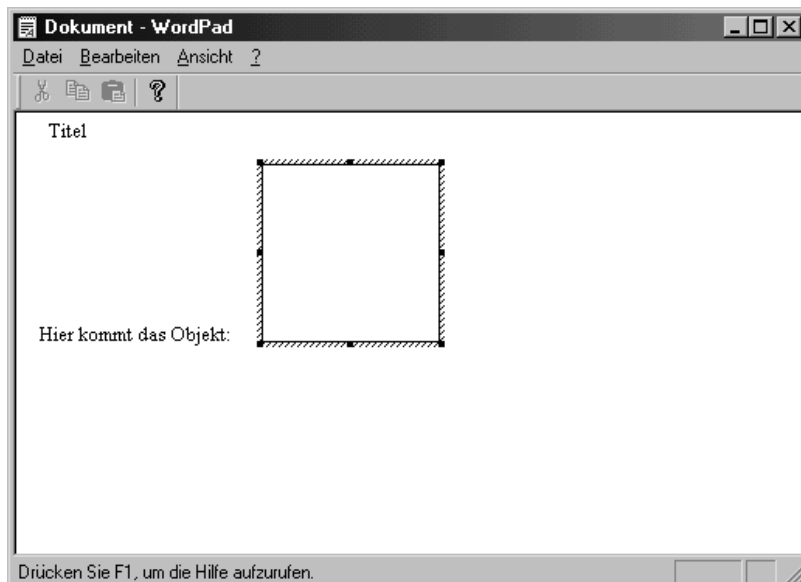
Abbildung 29.7:
Symbolleiste für
die Vorortbe-
arbeitung



29.2.7 Ausführen des Server-Gerüsts

Das von dem Anwendungsassistenten generierte Server-Gerüst kann ohne Änderungen kompiliert und gestartet werden. Obwohl das Programm keine nützlichen Funktionen ausführt, ist eine Demonstration der Interaktion zwischen dem Server und Container möglich. Sie können beispielsweise mit der Windows-95/98-Anwendung WordPad den neuen Server aufrufen. Beachten Sie bitte, daß Sie dazu den Server zunächst selbständig ausführen lassen müssen, damit dieser die erforderlichen Einträge in der Registrierung vornimmt. Anschließend können Sie WordPad starten und ein neues Objekt vom Typ OSRV DOKUMENT einfügen. (Befehl EINFÜGEN/OBJEKT). Abbildung 29.8 zeigt eine Vorortbearbeitung unter Verwendung des OSRV-Servers mit WordPad.

Abbildung 29.8:
Vorortbearbei-
tung



Beachten Sie bitte, wie die Menüs BEARBEITEN und ANSICHT, die Symbolleisten und die Statuszeile von der Server-Anwendung übernommen wurden. Die Symbolleiste, die Lineale und die Menüs von WordPad werden während der Bearbeitung nicht angezeigt.

29.3 Bearbeiten eines Server-Gerüsts

Wir werden dem Server-Gerüst nun einige Funktionen hinzufügen. Der neue OSRV-Server führt eine einfache Aufgabe aus. Er zeigt eine Zeichenfolge in der Mitte des Ansichtsfensters an. Die Zeichenfolge wird in einer Elementvariablen der Dokumentklasse des Servers gespeichert und von dieser Klasse serialisiert.

Damit die Zeichenfolge von dem Anwender geändert werden kann, müssen wir der Anwendung einen Dialog hinzufügen. Der Dialog wird aufgerufen, wenn der Anwender innerhalb einer Ansicht auf die linke Maustaste klickt.

Wenngleich dieses Beispiel sehr einfach ist, demonstriert es dennoch die wesentlichen Aspekte, die während der Entwicklung eines Servers berücksichtigt werden müssen. Eine echte OLE-Server-Anwendung ist natürlich weitaus komplexer. Die grundlegende Vorgehensweise unterscheidet sich jedoch nicht von der unseres Beispiels.

29.3.1 Modifizieren des Dokuments

Fügen Sie der Dokumentklasse zunächst die neue Elementvariable `COSRVDoc` hinzu. Die Elementvariable vom Typ `CString` sollte im Attribut-Abschnitt der Deklaration von `COSRVDoc` definiert werden:

```
// Attribute
public:
    COSRVsrItem* GetEmbeddedItem()
    { return (COSRVsrItem*)COleServerDoc::GetEmbeddedItem(); }
    CString m_sData;
```

Dieser Variable muß nun ein anfänglicher Wert zugewiesen werden. Modifizieren Sie `COSRVDoc::OnNewDocument` in der Implementierungsdatei von `COSRVDoc` wie folgt:

```
BOOL COSRVDoc::OnNewDocument()
{
    if (!COleServerDoc::OnNewDocument())
        return FALSE;

    // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
    // (SDI-Dokumente verwenden dieses Dokument)
```

```

        m_sData = _T("Hello, Wordl!");
        return TRUE;
    }

```

29.3.2 Hinzufügen von Zeichencode

Für den Fall, daß die Anwendung nicht als Server ausgeführt wird, muß der Zeichencode der entsprechenden Ansichtsklasse hinzugefügt werden. Der Zeichencode einer Server-Anwendung wird der `OnDraw`-Elementfunktion in der OLE-Serverobjekt-Klasse hinzugefügt.

Modifizieren Sie dazu die Implementierung von `COSRVView::OnDraw` wie folgt:

```

void COSRVView::OnDraw(CDC* pDC)
{
    COSRVDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen
    // Daten hinzufügen

    CRect rect;
    CFont font, *pOldFont;
    GetClientRect(&rect);
    font.CreateStockObject(SYSTEM_FONT);
    pOldFont = pDC->SelectObject(&font);
    pDC->DPtoLP(&rect);
    pDC->TextOut((rect.left + rect.right) / 2,
                (rect.top + rect.bottom) / 2, pDoc->m_sData);
    pDC->SelectObject(pOldFont);
}

```

Eine ähnliche Veränderung muß in `COSRVServerItem::OnDraw` vorgenommen werden, für den Fall, daß die Anwendung als Server fungiert:

```

BOOL COSRVServerItem::OnDraw(CDC* pDC, CSize& rSize)
{
    // Entfernen Sie dies, wenn Sie rSize verwenden
    UNREFERENCED_PARAMETER(rSize);

    COSRVDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // ZU ERLEDIGEN: Setzen Sie Mapping-Modus und Extent
    // (Das Extent stimmt üblicherweise mit der von OnGetExtent
    // zurückgelieferten Größe überein)
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);

    // ZU ERLEDIGEN: Hier Code zum Zeichnen einfügen. Füllen Sie
    // wahlweise das HIMETRIC-Extent aus.
    // Alle Zeichenoperationen finden innerhalb des Metadatei-
    // Gerätekontexts (pDC) statt.

    CRect rect;
    CFont font, *pOldFont;
    rect.TopLeft() = pDC->GetWindowOrg();
    rect.BottomRight() = rect.TopLeft() + pDC->GetWindowExt();
    font.CreateStockObject(SYSTEM_FONT);
}

```

```

pOldFont = pDC->SelectObject(&font);
pDC->TextOut((rect.left + rect.right) / 2,
             (rect.top + rect.bottom) / 2, pDoc->m_sData);
pDC->SelectObject(pOldFont);

return TRUE;
}

```

In einer komplexeren Anwendung würde der erste Abschnitt dieser Funktion ebenfalls modifiziert und die tatsächliche Größe des Objekts wiedergegeben (im Gegensatz zu der willkürlich gewählten Größe unseres Beispiels). Die `COSRVServerItem::OnGetExtent`-Funktion würde ebenfalls einen höherentwickelten Mechanismus zur Bestimmung der Größe des eingebetteten Objekts verwenden.

Sie können die Anwendung nun erneut kompilieren und ausführen lassen. Sie testen das Programm, indem Sie eine OLE-Container-Anwendung starten (beispielsweise WordPad) und dort aus dem Menü EINFÜGEN den Eintrag OBJEKT auswählen. Selektieren Sie in dem anschließend dargestellten Dialog den Listeneintrag OSRV DOKUMENT. Abbildung 29.9 zeigt die neue OSRV-Anwendung während einer Vorortbearbeitung mit WordPad.

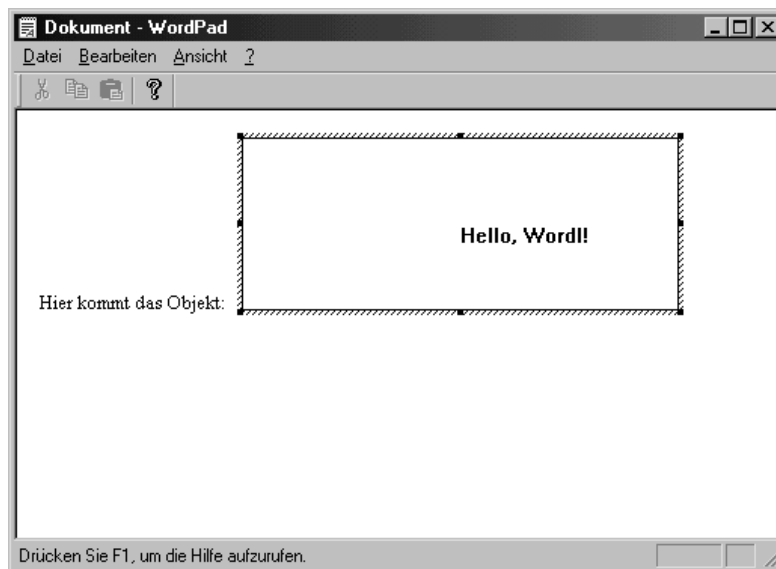


Abbildung 29.9:
OSRV und
WordPad

Wenn Sie OSRV starten, werden Sie feststellen, daß der Text nicht zentriert dargestellt wird. Nicht der Text selbst, sondern die linke obere Ecke des Textbereichs wurde in der Mitte des Fensters angeordnet. Sollten wir diesen Mangel beheben, indem wir beispielsweise die Größe des Textes berechnen oder den Aufruf von `OutText` durch den Aufruf der umfassenderen Funktion `DrawText` ersetzen?

Textzentrierung

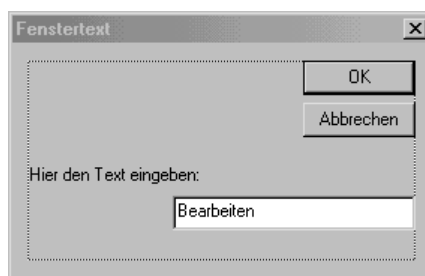
Leider ist die Antwort auf diese Frage nicht einfach. Die Funktion `COSRVView::OnDraw` kann sehr leicht modifiziert werden. Sie werden jedoch feststellen, daß Ihre Änderungen in `COSRVServerItem::OnDraw` nicht das erwartete Ergebnis zur Folge haben, wenn Sie `DrawText` mit dem Attribut `DT_CENTER` verwenden oder `GetTextExtent` aufrufen und versuchen, den Text manuell zu positionieren. Der Grund hierfür besteht darin, daß `COSRVServerItem::OnDraw`, im Gegensatz zu der `COSRVView::OnDraw`-Funktion mit einem »echten« Gerätekontext (der den Client-Bereich eines Fensters repräsentiert), in eine Metadatei zeichnet. Bestimmte Konzepte ergeben in bezug auf eine Metadatei keinen Sinn. Das Berechnen der Textgröße führt beispielsweise nicht zu den gewünschten Resultaten, da die reale Größe erst dann ermittelt werden kann, wenn die Metadatei (mit der eigenen Schriftartkonfiguration) auf einem Zielgerät ausgegeben wird. Wir werden diese Einschränkung zunächst außer acht lassen (betrachten Sie den Mangel einfach als Dokumentfehler). Dieses Beispiel sollte Ihnen jedoch als Warnung dienen, daß die beiden `OnDraw`-Funktionen (in der Ansichtsklasse und in der OLE-Serverobjekt-Klasse) unter verschiedenen Umständen aufgerufen werden, und daher eine separate Überprüfung erforderlich ist.

29.3.3 Hinzufügen eines Dialogs

Wir werden der Anwendung einen Dialog hinzufügen, in dem der Anwender den von OSRV angezeigten Text ändern kann.

1. Verwenden Sie zur Erstellung des Dialogs den Dialog-Editor. Der in Abbildung 29.10 dargestellte Dialog sollte ein einzelnes Eingabefeld mit der Bezeichnung `IDC_EDIT1` enthalten, in das der neue Text eingegeben werden kann.

Abbildung 29.10:
Der Text-Dialog
in OSRV



2. Rufen Sie zum Abschluß der Dialogerstellung den Klassen-Assistenten auf. Lassen Sie diesen eine neue Klasse (`CTextDlg`) erzeugen, die den Dialog repräsentiert. Fügen Sie der Klasse die Elementvariable `m_sText` hinzu, die das Eingabefeld `IDC_EDIT1` repräsentieren soll.

3. Wie wird dieser Dialog aufgerufen? Um uns die Arbeit zu ersparen, die durch das Definieren eines zusätzlichen Menüeintrags entstehen würde, öffnen wir den Dialog, wenn ein Ansichtsfenster die Nachricht `WM_LBUTTONDOWN` erhält. Diese Nachricht wird ausgelöst, wenn der Anwender innerhalb der Ansicht auf die linke Maustaste klickt.
4. Wechseln Sie zur Seite Nachrichtenzuordnungstabellen und richten Sie für die Klasse `OSRVView` eine `WM_LBUTTONDOWN`-Bearbeiterfunktion ein.

Die Bearbeiterfunktion ist in Listing 29.3 aufgeführt. Die Funktion erzeugt den Dialog, ruft ihn auf und transferiert Daten zwischen dem Dialog und dem Dokument.

Damit diese Funktion korrekt kompiliert wird, müssen Sie die Header-Datei `TEXTDLG.H` zu Beginn der Datei `OSRVVIEW.CPP` einbinden (`#include`-Anweisung).

```
void COSRVView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Nachrichtebearbeiter hier einfügen

    // CView::OnLButtonDown(nFlags, point);
    CTextDlg dlg;
    COSRVDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    dlg.m_sText = pDoc->m_sData;
    if ((dlg.DoModal() == IDOK)
        && (dlg.m_sText != pDoc->m_sData))
    {
        pDoc->m_sData = dlg.m_sText;
        pDoc->UpdateAllViews(NULL);
        pDoc->UpdateAllItems(NULL);
        pDoc->SetModifiedFlag();
    }
}
```

*Listing 29.3:
Implementierung von COSRV-
View: : OnLBut-
tonDown*

Die Funktion ruft zusätzlich zu den Elementfunktionen `UpdateAllViews` (aktualisiert die Eingabe des Anwenders in allen Ansichten des Dokuments) und `SetModifiedFlag` (benachrichtigt die Dokumentklasse, daß der Inhalt geändert wurde und gespeichert werden muß) `UpdateAllItems` auf. Über diese Funktion werden die Container informiert, daß sich der Inhalt des eingebetteten Objekts verändert hat und daher neu gezeichnet werden muß.

29.3.4 Serialisierung

Unsere Anwendung ist erst dann vollständig, wenn ihre Daten gespeichert werden. Dazu wird die `m_sData`-Elementvariable der Dokumentklasse in `COSRVDoc::Serialize` serialisiert:


```

void COSRVDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
        ar << m_sData;
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
        ar >> m_sData;
    }
}

```

Die Server-Anwendung ist nun vollständig und kann erneut kompiliert werden.

29.3.5 Registrieren der neuen Anwendung

Sie werden sicherlich bereits bemerkt haben, daß wir die OSRV-Anwendung ausführen, ohne zuvor Informationen in die Registrierung einzutragen. Wie aber erfahren Clients, wie zum Beispiel WordPad, von unserer Anwendung?

Wir müssen deshalb keine Einträge in der Registrierung vornehmen, da der Anwendungsassistent Anwendungen generiert, die sich selbst registrieren. Immer dann, wenn solch eine Anwendung selbständig ausgeführt wird, erstellt oder aktualisiert sie die entsprechenden Einträge in der Registrierung. Dies gilt ebenfalls für Mini-Server, obwohl diese gewöhnlich nicht selbständig ausgeführt werden.

Der Anwendungsassistent erzeugt außerdem eine Datei, die alle relevanten Registrierungseinträge aufnimmt. Die entsprechende Datei unseres Beispiels trägt den Dateinamen *OSRV.REG*. Sie enthält die folgenden Einträge, die unter dem Schlüssel *HKEY_CLASSES_ROOT* gespeichert werden:

```

OSRV.Document = OSRV Document
OSRV.Document\protocol\StdFileEditing\server = OSRV.EXE
OSRV.Document\protocol\StdFileEditing\verb\0 = &Edit
OSRV.Document\Insertable =
OSRV.Document\CLSID = {494CDF20-FE4B-11CE-87C3-00403321BFAC}

```

Über diese Einträge identifiziert ein OLE-Container die OSRV-Dokumente als einfügbare Objekte. Der letzte Eintrag bezeichnet die CLSID-Kennung, unter der zusätzliche Informationen über den Server abgerufen werden können. (Beachten Sie bitte, daß Ihre CLSID von meiner abweichen kann.) Weitere Registrierungseinträge, die diesen CLSID nutzen, befinden sich unter dem Schlüssel *HKEY_CLASSES_ROOT\CLSID*:

```

{494CDF20-FE4B-11CE-87C3-00403321BFAC} = OSRV Document
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\DefaultIcon = OSRV.EXE,1
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\LocalServer32 = OSRV.EXE
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\ProgId = OSRV.Document
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\MiscStatus = 32

```

```
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\AuxUserType\3 = OSRV  
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\AuxUserType\2 = OSRV  
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\Insertable =  
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\verb\1 = &Öffnen,0,2  
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\verb\0 = &Bearbeiten,0,2  
{494CDF20-FE4B-11CE-87C3-00403321BFAC}\InprocHandler32 = ole32.dll
```

29.4 Zusammenfassung

OLE-Server, die auch als OLE-Komponenten-Anwendungen bezeichnet werden, sind Programme, die OLE-Komponenten für die Verwendung in einer OLE-Container-Anwendung zur Verfügung stellen.

Ein MFC-OLE-Server wird mit dem Visual-C++-Anwendungsassistenten erstellt. Selektieren Sie dazu die Server-Unterstützung im dritten Schritt des Assistenten.

OLE-Server werden lediglich von Anwendungen unterstützt, die ein oder mehrere Dokument(e) verwenden. Dialogfeldbasierende Anwendungen können keinen OLE-Server zur Verfügung stellen. Sie können jedoch einen OLE-Server erzeugen, der einem Dialog ähnlich ist. Verwenden Sie dazu die Klasse `CFormView`.

Die MFC unterscheidet zwischen

- ➡ Mini-Servern,
- ➡ Voll-Servern und
- ➡ Container-Servern.

Ein Mini-Server ist eine Anwendung, die nur in Verbindung mit eingebetteten Objekten verwendet werden kann. Ein Voll-Server kann ebenfalls selbständig ausgeführt werden und die Daten in einer Datei speichern. Ein Container-Server ist ein OLE-Server, der Container-Funktionalität bietet.

Möchten Sie einen OLE-Server mit Hilfe des Anwendungsassistenten erstellen, wählen Sie zunächst den gewünschten Server im dritten Schritt des Assistenten aus. Ein OLE-Server verfügt über zwei neue Klassen. Die von `COleServerItem` abgeleitete Serverobjekt-Klasse repräsentiert ein eingebettetes Objekt in der Container-Anwendung. Diese Klasse offeriert eine `IOleObject`-COM-Schnittstelle, über die die Container-Anwendung mit dem Server kommunizieren kann. Die von `COleIPFrameWnd` abgeleitete Vorort-Rahmenfensterklasse repräsentiert das Rahmenfenster während der Vorortbearbeitung und verwaltet die Interaktion zwischen dem Server und dem Rahmenfenster des Containers.

Das Vorort-Rahmenfenster verfügt über die gleiche Funktionalität wie das Rahmenfenster in SDI-Anwendungen. Es ersetzt in Kooperation mit dem Rahmenfenster der Container-Anwendung die Menüs und Symbolleisten dieses Rahmenfensters für die Dauer der Vorortbearbeitung. Während einer Vorortbearbeitung übernimmt die Server-Anwendung die Symbolleiste und Statusleiste. Die während der Bearbeitung angezeigte Menüleiste ist eine Kombination der Menüs des Servers und des Containers.

Die Serverobjekt-Klasse verfügt über zwei wesentliche Elementfunktionen. Die `OnDraw`-Elementfunktion zeichnet das OLE-Objekt in einen Metadatei-Gerätekontext. Das Objekt wird später ohne einen Aufruf des Servers von dem Container angezeigt. Die `Serialize`-Elementfunktion serialisiert das in dem Container-Dokument enthaltene Objekt.

Die nachfolgend aufgeführten Schritte beschreiben, wie ein Server-Gerüst bearbeitet wird:

- ➔ Fügen Sie dem zuvor erstellten Server-Gerüst die grundlegende Funktionalität hinzu. Definieren Sie dazu in der Dokumentklasse und in der Ansichtsklasse Elementvariablen und Programmcode. Verwenden Sie weitere Elemente (Ressourcen, Dialoge), sofern erforderlich.
- ➔ Modifizieren Sie die `OnDraw`-Elementfunktion des Serverobjekts, um das Objekt in einen Metadatei-Gerätekontext zu zeichnen. Dieses Objekt wird in dem Container angezeigt, wenn der Server nicht aktiviert ist.
- ➔ Modifizieren Sie die `Serialize`-Elementfunktion des Serverobjekts, um das Objekt über die Serialisierung einzubetten. Unterstützt Ihre Anwendung lediglich eingebettete Objekte, können Sie die Standardimplementierung dieser Funktion verwenden. Arbeiten Sie mit Verknüpfungen oder benutzen Sie die Serverobjekt-Klasse, um den Zwischenablagetransfer zu erleichtern, sollten Sie die Funktion erweitern, um die selektierten Objekte serialisieren zu können.

Sie müssen Ihre Anwendung nicht registrieren. Die Anwendung registriert sich selbst (oder aktualisiert die entsprechenden Registrierungseinträge), wenn sie selbständig ausgeführt wird. Der Anwendungsassistent erzeugt jedoch ebenfalls eine Datei, die zur manuellen Registrierung der Anwendung in einem Installationsprogramm verwendet werden kann.

OLE-Container

Kapitel

30

LE-Container sind Anwendungen, die zusätzlich zu den gewöhnlichen Daten auch OLE-Serverobjekte bearbeiten können.

Die MFC-Bibliothek unterstützt die Erstellung von OLE-Containern. Mit einem vom Anwendungsassistenten generierten Container-Anwendungsgerüst kann mit nur wenig Programmcode eine funktionsfähige Container-Anwendung erzeugt werden.

Wir werden in diesem Kapitel zunächst ein vom Anwendungsassistenten erstelltes Container-Anwendungsgerüst besprechen. Später fügen wir dem Gerüst den erforderlichen Programmcode zur Auswahl und Bearbeitung mehrerer eingebetteter Objekte hinzu.

30.1 Erstellen einer Container-Anwendung mit dem Anwendungsassistenten

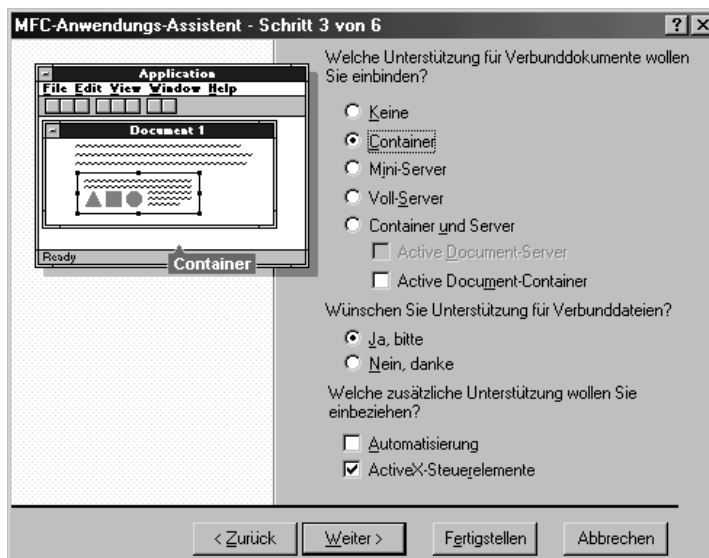
Der Visual-C++-Anwendungsassistent kann die Gerüste für einen einfachen Container und für einen Container-Server erstellen. Container-Server sind Anwendungen, die sowohl die Funktionalität für OLE-Container als auch für OLE-Komponenten-Server bieten. Da sich die beiden Container-Typen voneinander unterscheiden, werden in diesem Kapitel zunächst die einfachen Container beschrieben.

30.1.1 Erstellen des Anwendungsgerüsts

Um eine Container-Anwendung namens OCON mit Hilfe des Anwendungsassistenten zu erstellen:

1. Entscheiden Sie zunächst zwischen einer Anwendung, die ein Dokument verwendet (SDI), und einer Anwendung, die mehrere Dokumente verwendet (MDI). (Dialogfeldbasierende Anwendungen können keinen Container bilden. Sie haben jedoch die Möglichkeit, eine Container-Anwendung zu erstellen, die auf der Ansichtsklasse `CFormView` basiert.)
2. Selektieren Sie im dritten Schritt des Assistenten (Abbildung 30.1) die gewünschte OLE-Container-Unterstützung aus (CONTAINER oder CONTAINER UND SERVER). Entscheiden Sie sich für die Option CONTAINER.
3. Abgesehen von dieser Auswahl sollten Sie die übrigen Voreinstellungen des Anwendungsassistenten beibehalten.

Abbildung 30.1:
Erstellen eines
OLE-Containers
mit dem
Anwendungs-
assistenten



30.1.2 Das OLE-Container-Anwendungsgerüst

Wenn Sie sich die vom Anwendungsassistenten erstellten Klassen (Abbildung 30.2) ansehen, werden Sie feststellen, daß entgegen einer Anwendung ohne OLE-Unterstützung eine zusätzliche Klasse angelegt wurde. Diese Klasse ist mit `COCONtrItem` bezeichnet und repräsentiert die OLE-Komponenten, die das Dokument der Container-Anwendung aufnehmen wird.



Abbildung 30.2:
Die vom Anwendungsassistenten generierten Klassen des OLE-Containers

Natürlich ist die neue Klasse nicht der einzige Unterschied zwischen einem OLE-Container und anderen Anwendungen. Differenzen bestehen außerdem zwischen der Implementierung einiger Klassen.

30.1.3 Ausführen des Container-Gerüsts

Bevor wir mit der Erläuterung des Container-Anwendungsgerüsts fortfahren, sollten wir die Anwendung starten. Dadurch erfahren wir, wie der Container arbeitet und welche Funktionalität der Anwendung hinzugefügt werden muß.

Testen Sie die OCON-Container-Anwendung, indem Sie sie kompilieren und ausführen lassen. Wählen Sie dann aus dem Menü BEARBEITEN den Eintrag NEUES OBJEKT EINFÜGEN aus. Daraufhin wird der in Abbildung 30.3 dargestellte Dialog angezeigt, in dem Sie den Objekttyp selektieren, der in das Dokument eingefügt werden soll.

Fügen Sie ein Bitmap-Objekt ein, das auf allen Windows-95/98-Systemen verfügbar ist, die das Programm *Paint* installiert haben. Abbildung 30.4 zeigt die Bearbeitung eines Bitmap-Objekts innerhalb des OCON-Containers.

Möglicherweise haben Sie einige Besonderheiten bemerkt, die darauf schließen lassen, daß einige Programmabschnitte überprüft werden müssen.

- ➡ Zunächst scheint ein Beenden der Vorortbearbeitung nicht möglich zu sein. Gewöhnlich würden Sie außerhalb des Bereichs der Vorortbearbeitung klicken, um die Sitzung zu beenden. Dies ist jedoch nicht möglich. Der Grund hierfür ist sehr einfach: Das OCON-Anwendungsgerüst verfügt über keine Bearbeiterfunktion für Mausklicks.

Abbildung 30.3:
Der Dialog
Objekt einfügen

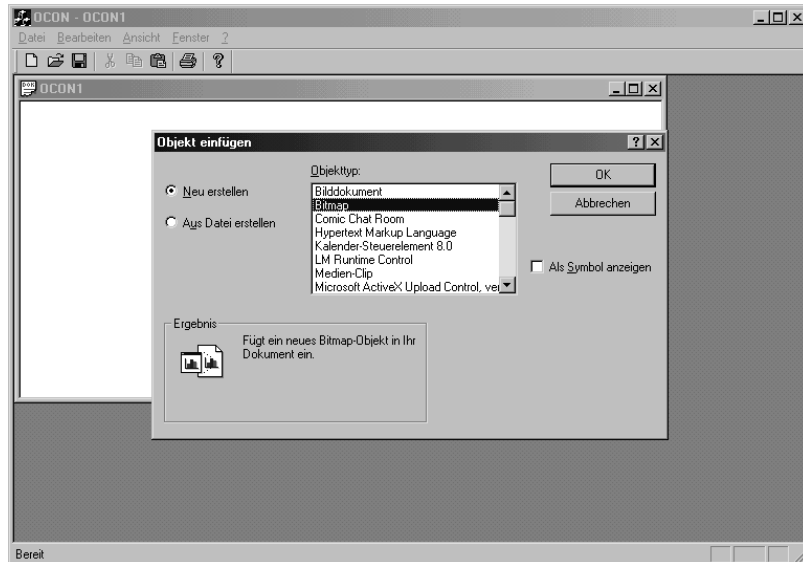
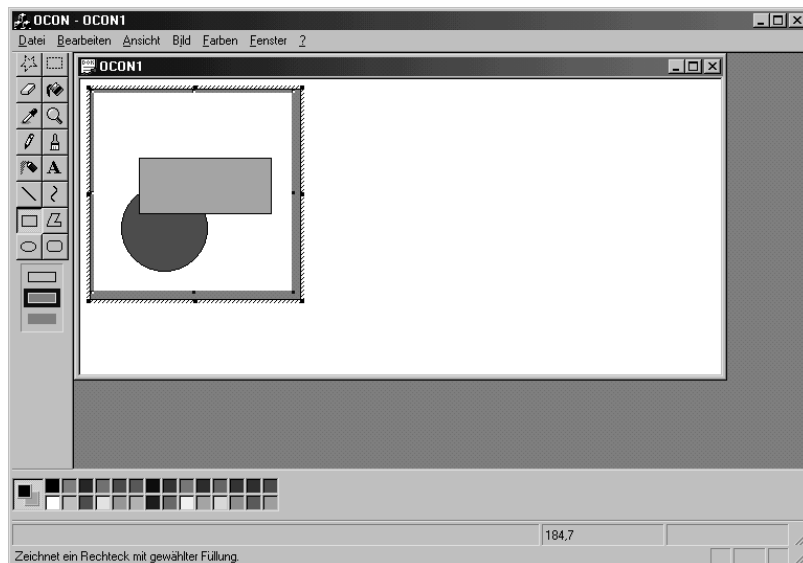


Abbildung 30.4:
Vorortbearbei-
tung eines Bit-
map-Objekts



- ➡ Eine weitere Besonderheit zeigt sich, wenn Sie versuchen, den Vorortbereich zu verschieben oder dessen Größe zu verändern. Sie ändern die Größe, indem Sie einen der acht Haltepunkte auf die gewünschte Größe ziehen. Sie können das Objekt ebenfalls verschieben, indem Sie den Markierungsrahmen an eine neue Position ziehen. Achten Sie jedoch darauf, was anschließend geschieht

(Abbildung 30.5). Nachdem der Vorortbereich verschoben wurde, erscheint ein weiteres Bild des Objekts an der ursprünglichen Position. Die Container-Anwendung zeichnet das Bild dort, da bisher kein Programmcode besteht, der diesen Bereich aktualisiert, nachdem Änderungen während der Vorortbearbeitung vorgenommen wurden.

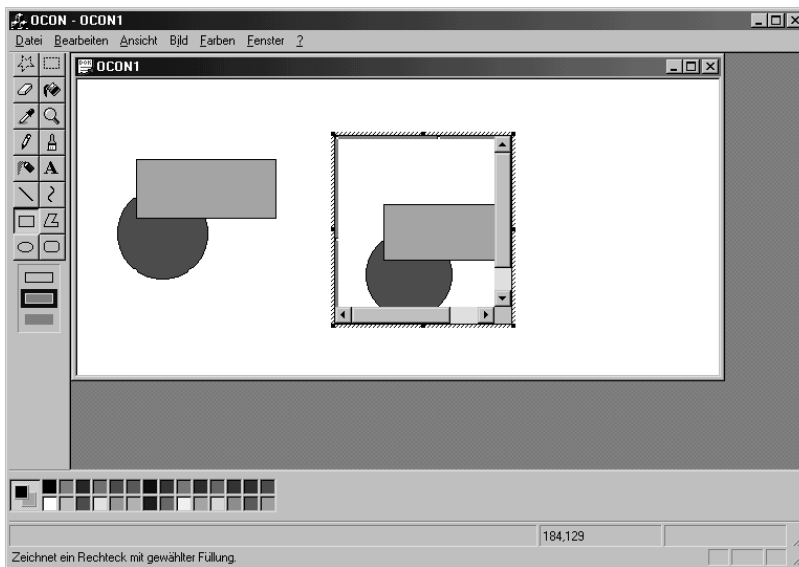


Abbildung 30.5:
Abweichungen
zwischen dem
Vorortrahmen
und den Positio-
nen des einge-
betteten
Objekts

Da keine einfache Möglichkeit besteht, die Vorortbearbeitung zu beenden, können Sie die Container-Datei sogar während einer aktiven Vorortbearbeitung speichern. Nachdem Sie die Datei gespeichert haben, beenden Sie die Vorortbearbeitung, indem Sie das Dokumentfenster schließen. Wenn Sie die Datei erneut öffnen, werden Sie feststellen, daß das eingebettete Objekt korrekt gespeichert wurde und an der ursprünglichen Position dargestellt wird. Sie können das Objekt nun bearbeiten oder ein neues Objekt einfügen. Verwenden Sie dazu die Anweisungen in dem Menü BEARBEITEN.

- ➡ Fügen Sie ein neues Objekt ein, wird dieses über dem ersten Objekt in dem Container angezeigt. Aus diesem Grund erkennen Sie die dritte Besonderheit nicht, die das vom Anwendungsassistenten generierte Anwendungsgerüst betrifft. Die Anwendung stellt lediglich ein Objekt gleichzeitig dar. Dies ist jedoch ausschließlich ein Darstellungsproblem. Die Anwendung kann Container-Dateien mit mehreren Objekten speichern. Wir müssen lediglich die `OnDraw`-Elementfunktion der Ansichts-klasse modifizieren.

30.1.4 Der Programmcode des Container-Gerüsts

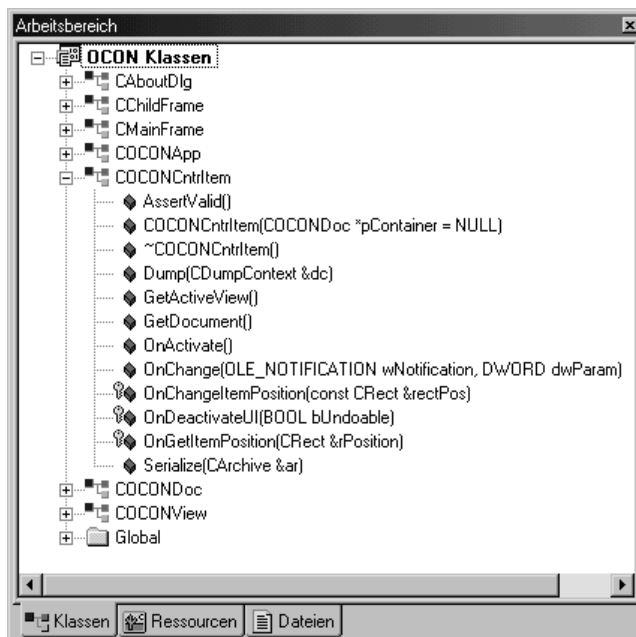
Beenden Sie die Container-Anwendung, und lassen Sie uns einen Blick darauf werfen, wie die Funktionen dieser Anwendung implementiert werden.

Wie werden dem Dokument neue Objekte hinzugefügt? Wie werden neue Objekte in einer Container-Datei gespeichert? Wie werden sie gezeichnet? Diese Fragen werden in den folgenden Abschnitten beantwortet.

Die Container-Objekte (COCONCtrlItem)

Sie erfahren zunächst, wie Objekte in einem Container repräsentiert werden. Der Anwendungsassistent generiert dazu die neue Klasse COCONCtrlItem. Diese Klasse wird von COleClientItem abgeleitet und verfügt über eine Standardimplementierung verschiedener Elementfunktionen (Abbildung 30.6).

Abbildung 30.6:
Die Element-
funktionen der
Container-
Objektklasse



Diese Klasse implementiert die erforderliche OLE-Schnittstelle zur Vorortbearbeitung. Außerdem stellt sie einige Elementfunktionen zur Verfügung (wie zum Beispiel `Serialize`), die den Einsatz der Klasse in einer MFC-Applikationsrahmen-Anwendung ermöglichen.

Die Implementierung der Klasse enthält einige Unzulänglichkeiten. Dies trifft überwiegend für die Elementfunktionen `OnChangeItemPosition` und `OnGetItemPosition` zu (Listing 30.1). Wie Sie der Standardimplementierung entnehmen können, gibt `OnGetItemPosition` immer eine willkürliche Position zurück. `OnChangeItemPosition` erfährt nicht von der neuen Position, obwohl die Funktion die Basisklassenimplementierung aufruft.

*OnChangeItem-
Position
OnGetItem-
Position*

```

BOOL COCONtrItem::OnChangeItemPosition(const CRect& rectPos)
{
    ASSERT_VALID(this);

    // Während einer direkten Aktivierung wird
    // COCONtrItem::OnChangeItemPosition vom Server aufgerufen,
    // um die Position des In-Place-Fensters zu ändern.
    // Üblicherweise ist dies ein Ergebnis von Datenänderungen
    // im Server-Dokument, etwa ein geändertes Extent oder als
    // Ergebnis direkter Größenänderungen.
    //
    // Standardmäßig wird hier die Basisklasse aufgerufen, die
    // wiederum COleClientItem::SetItemRects zum Bewegen des
    // Elements an die neue Position aufruft.

    if (!COleClientItem::OnChangeItemPosition(rectPos))
        return FALSE;

    // ZU ERLEDIGEN: Aktualisieren Sie alle für das Rechteck/das
    // Extent dieses Elements angelegten Caches

    return TRUE;
}

void COCONtrItem::OnGetItemPosition(CRect& rPosition)
{
    ASSERT_VALID(this);

    // Während einer direkten Aktivierung wird
    // COCONtrItem::OnGetItemPosition zur Bestimmung der
    // Position dieses Elements aufgerufen. Die durch
    // den Anwendungsassistenten erstellte
    // Standardimplementierung gibt einfach ein fest
    // einprogrammiertes Rechteck zurück.
    // Normalerweise gibt dieses Rechteck die aktuelle
    // Position des Elements relativ zu der Ansicht an, die zur
    // Aktivierung verwendet wird. Sie erhalten die Ansicht,
    // indem Sie COCONtrItem::GetActiveView aufrufen.

    // ZU ERLEDIGEN: Geben Sie das korrekte Rechteck (in Pixeln)
    // in rPosition zurück

    rPosition.SetRect(10, 10, 210, 210);
}

```

*Listing 30.1:
Die Standardim-
plementierung
der COCONtr-
Item-Element-
funktionen On-
ChangeItem-
Position und On-
GetItemPosition*

Die von dem Anwendungsassistenten generierten Kommentare geben zu verstehen, daß wir diese Klasse bearbeiten müssen, um die Position korrekt zu verwalten.

Die Dokumentklasse (COCONDoc)

Kein Programmcode in dem Anwendungsgerüst repräsentiert die COCONCntrItem-Objekte in der Dokumentklasse. Der einzige Unterschied zu Anwendungen, die keine Container bilden, besteht darin, daß die Dokumentklasse der Anwendung von COleDocument und nicht von CDocument abgeleitet ist. COleDocument verwaltet und speichert eine Liste mit von CDocItem abgeleiteten Objekten. Ein neues aus der Klasse COCONCntrItem (die von COleClientItem abgeleitet ist) erstelltes Objekt, wird automatisch der Dokumentliste des Containers hinzugefügt. Das Container-Dokument kann sich selbst ohne zusätzlichen Programmcode mit Hilfe dieser Liste serialisieren.

Sie können dem Container Ihre eigenen von CDocItem abgeleiteten Objekte hinzufügen. Der Container wird diese korrekt bearbeiten. Sie müssen lediglich spezifische Anwendungsobjekte, Container-Objekte und – wenn die Anwendung ebenfalls als OLE-Server verwendet wird – Serverobjekte mit Bedacht in Ihrem Programmcode bearbeiten. Sie könnten beispielsweise einige Mantelfunktionen schreiben, die einen bestimmten Objekttyp mit Hilfe der MFC-Laufzeittypeninformationen bestimmen.

Die Ansichtsklasse (COCONView)

Der Unterschied zwischen der Implementierung der Dokumentklasse in einem Container und einer gewöhnlichen Anwendung ist relativ gering (wenngleich die Auswirkung dieses Unterschieds auf die Basisklasse, von der die Dokumentklasse abgeleitet wird, sehr wesentlich ist). Im Gegensatz dazu ist der Unterschied zwischen der Implementierung der Ansichtsklasse in diesen Anwendungen sehr groß. Die Implementierungsdatei der Ansichtsklasse der Container-Anwendung ist ungleich umfangreicher und verfügt über eine größere Anzahl verschiedener Elementfunktionen (Abbildung 30.7).

Die Implementierung dieser Elementfunktionen wird unsere Fragen hinsichtlich des sonderbaren Verhaltens des Container-Gerüsts beantworten.

In der Deklaration von COCONView ist die neue Elementvariable m_pSelection enthalten:

```
class COCONView : public CView
{
protected: // Nur aus Serialisierung erzeugen
    COCONView();
    DECLARE_DYNCREATE(COCONView)

// Attribute
public:
    COCONDoc* GetDocument();
```

```
// m_pSelection enthält die Auswahl des aktuellen
// COCONtrItem-Objekts.
// Für viele Anwendungen ist eine derartige Member-Variable
// nicht angemessen, um z.B. eine Mehrfachauswahl oder eine
// Auswahl von Objekten zu repräsentieren,
// die keine COCONtrItem-Objekte sind. Dieser
// Auswahlmechanismus dient nur dazu, Ihnen bei den ersten
// Schritten zu helfen.

// ZU ERLEDIGEN: Ersetzen Sie diesen Auswahlmechanismus durch
// einen für Ihre Anwendung geeigneten.
COCONtrItem* m_pSelection;
```

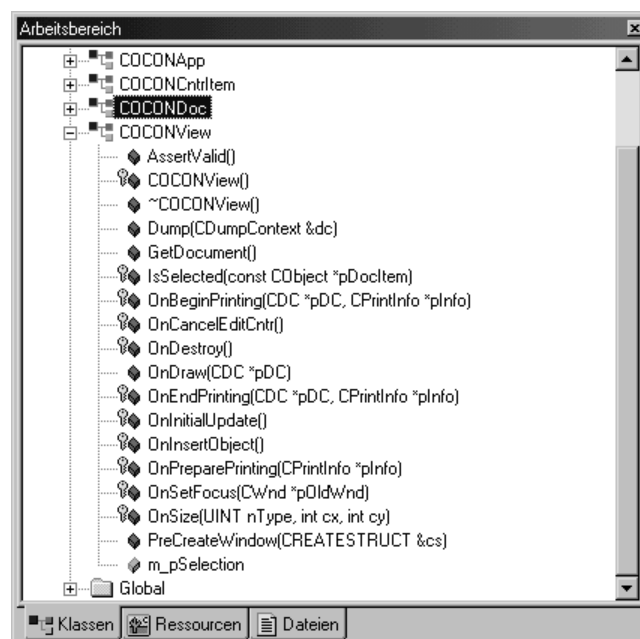


Abbildung 30.7:
Die Element-
funktionen der
Container-
Ansichtsklasse

Diese Elementvariable repräsentiert einen einfachen Mechanismus zur Auswahl eines Objekts. Dieser Mechanismus ermöglicht lediglich die Auswahl eines einzelnen Dokumentobjekts. Während diese Auswahlmöglichkeit für hochentwickelte Anwendungen unangebracht ist, genügt sie unserer einfachen Container-Anwendung.

Die `m_pSelection`-Elementvariable ist an der Implementierung von `OnDraw` beteiligt. Dies führt zur Beantwortung der Frage, wieso lediglich ein Objekt in einem Dokument dargestellt wird, in dem mehrere Objekte eingebettet sind. Diese Funktion (Listing 30.2) zeichnet einfach keine anderen Objekte.

Listing 30.2:
Die Standardimplementierung
von COCON-
View: : OnDraw

```
void COCONView::OnDraw(CDC* pDC)
{
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen
    // Daten hinzufügen ZU ERLEDIGEN: Auch alle OLE-Elemente im
    // Dokument zeichnen Auswahl an beliebiger Position zeichnen.
    // Dieser Code sollte entfernt werden, sobald Ihr
    // tatsächlicher Zeichencode implementiert ist. Diese
    // Position entspricht exakt dem von COCONCntrItem
    // zurückgegebenen Rechteck, um den Effekt der direkten
    // Bearbeitung zu gewährleisten.

    // ZU ERLEDIGEN: Entfernen Sie diesen Code, sobald Ihre
    // richtige Zeichenroutine vollständig ist.

    if (m_pSelection == NULL)
    {
        POSITION pos = pDoc->GetStartPosition();
        m_pSelection =
            (COCONCntrItem*)pDoc->GetNextClientItem(pos);
    }
    if (m_pSelection != NULL)
        m_pSelection->Draw(pDC, CRect(10, 10, 210, 210));
}
```

Die Unzulänglichkeit dieser Standardimplementierung ist in einem der vom Anwendungsassistenten generierten Kommentare beschrieben. Das ausgewählte Objekt wird an einer beliebigen Position gezeichnet. Veränderungen der Position während der Vorortbearbeitung bleiben unberücksichtigt. Die Implementierung muß zusammen mit COCONCntrItem::OnChangeItemPosition und COCONCntrItem::OnGetItemPosition modifiziert werden.

Die vom Anwendungsassistenten erzeugte Implementierung der Ansichtsklasse enthält fünf neue Elementfunktionen (Listing 30.3). Diese verwalten das Einfügen neuer Objekte und die Vorortbearbeitung. Wir werden diese Funktionen nacheinander besprechen.

Listing 30.3:
Neue Element-
funktionen der
Ansichtsklasse

```
BOOL COCONView::IsSelected(const COBJECT* pDocItem) const
{
    // Die nachfolgende Implementierung ist angemessen, wenn sich
    // Ihre Auswahl nur aus COCONCntrItem-Objekten zusammensetzt.
    // Zur Bearbeitung unterschiedlicher
    // Auswahlmechanismen sollte die hier gegebene Implementierung
    // ersetzt werden.

    // ZU ERLEDIGEN: Implementieren Sie diese Funktion, die auf
    // ein ausgewähltes OLE-Client-Element testet

    return pDocItem == m_pSelection;
}

void COCONView::OnInsertObject()
{
    // Standarddialogfeld zum Einfügen von Objekten aufrufen, um
    // Infos abzufragen für neues COCONCntrItem-Objekt.
    COleInsertDialog dlg;
```

```

if (dlg.DoModal() != IDOK)
    return;

BeginWaitCursor();

COCONtrItem* pItem = NULL;
TRY
{
    // Neues, mit diesem Dokument verbundenes Element erzeugen.
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pItem = new COCONtrItem(pDoc);
    ASSERT_VALID(pItem);

    // Element mit Dialogfelddaten initialisieren.
    if (!dlg.CreateItem(pItem))
        AfxThrowMemoryException(); // Beliebige Ausnahme
                                   // erzeugen
    ASSERT_VALID(pItem);

    if (dlg.GetSelectionType() ==
        COleInsertDialog::createNewItem)
        pItem->DoVerb(OLEIVERB_SHOW, this);

    ASSERT_VALID(pItem);

    // Die Größe wird hier willkürlich auf die Größe
    // des zuletzt eingefügten Elements gesetzt.

    // ZU ERLEDIGEN: Implementieren Sie die Auswahl erneut in
    // einer für Ihre Anwendung geeigneten Weise

    m_pSelection = pItem; // die Auswahl auf das zuletzt
                          // eingefügte Element setzen
    pDoc->UpdateAllViews(NULL);
}
CATCH(CException, e)
{
    if (pItem != NULL)
    {
        ASSERT_VALID(pItem);
        pItem->Delete();
    }
    AfxMessageBox(IDP_FAILED_TO_CREATE);
}
END_CATCH

EndWaitCursor();
}

// Der folgende Befehls-Handler stellt die Standardtastatur als
// Benutzerschnittstelle zum Abbruch der direkten
// Bearbeitungssitzung zur Verfügung. Hier
// verursacht der Container (nicht der Server) die Deaktivierung.
void COCONView::OnCancelEditCntr()
{
    // Schließen aller direkt aktiven Elemente dieser Ansicht.
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
    {
        pActiveItem->Close();
    }
    ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}

```

```

// Für einen Container müssen OnSetFocus und OnSize speziell
// gehandhabt werden, wenn ein Objekt direkt bearbeitet wird.
void COCONView::OnSetFocus(CWnd* pOldWnd)
{
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL &&
        pActiveItem->GetItemState() ==
            COleClientItem::activeUIState)
    {
        // dieses Element muss den Fokus erhalten, wenn es sich in
        // der gleichen Ansicht befindet
        CWnd* pWnd = pActiveItem->GetInPlaceWindow();
        if (pWnd != NULL)
        {
            pWnd->SetFocus(); // kein Aufruf der Basisklasse
            return;
        }
    }

    CView::OnSetFocus(pOldWnd);
}

void COCONView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
        pActiveItem->SetItemRects();
}

```

IsSelected Die *IsSelected*-Elementfunktion wird von dem Anwendungsrahmen aufgerufen, um zu ermitteln, ob ein bestimmtes Objekt in der Ansicht selektiert ist.

OnInsertObject Die *OnInsertObject*-Elementfunktion implementiert die Anweisung NEUES OBJEKT EINFÜGEN im Menü BEARBEITEN. Diese Funktion basiert auf der OLE-Standarddialogklasse *COleInsertDialog*. Der Dialog wird nicht nur dazu verwendet, die Liste der auf Ihrem System verfügbaren OLE-Objekte anzuzeigen. Seine Elementfunktion *CreateItem* kann dazu verwendet werden, ein neues Objekt des Typs zu erzeugen, das von dem Anwender ausgewählt wurde. Nachdem das Objekt erstellt wurde, startet *OnInsertObject* die Server-Anwendung. Die aus einer Datei generierten Objekte werden einfach in das Container-Dokument eingefügt. Die Implementierung dieser Funktion nutzt MFC-Ausnahmebehandlungsmakros, um die Fehler abzufangen, die während der Erstellung eines neuen Objekts auftreten können.

OnCancelEditCntr Die Elementfunktion *OnCancelEditCntr* beendet eine Vorortbearbeitung. Diese Funktion wird von dem Anwendungsrahmen aufgerufen, wenn der Anwender die Taste (Esc) betätigt.

OnSetFocus Die *OnSetFocus*-Elementfunktion gewährleistet, daß der Fokus auf das Rahmenfenster des Vorortobjekts gesetzt wird, wenn eine Ansicht mit einer aktiven Vorortsetzung den Fokus erhält.

Die `OnSize`-Elementfunktion zeigt Änderungen der Größe des Ansichtsfensters an. *OnSize*

Wir beenden an dieser Stelle die Übersicht über die Unterschiede zwischen OLE-Containern und gewöhnlichen Anwendungen. Wir werden nun die Ressourcendatei der Container-Anwendung besprechen.

30.1.5 Container-Menüs

Ein Blick auf die vom Anwendungsassistenten generierte Ressourcendatei (Abbildung 30.8) für unsere OCON-Anwendung läßt zwei Unterschiede zu gewöhnlichen Anwendungen erkennen:

- ➡ eine neue Zugriffstastentabelle und
- ➡ ein neues Menü.

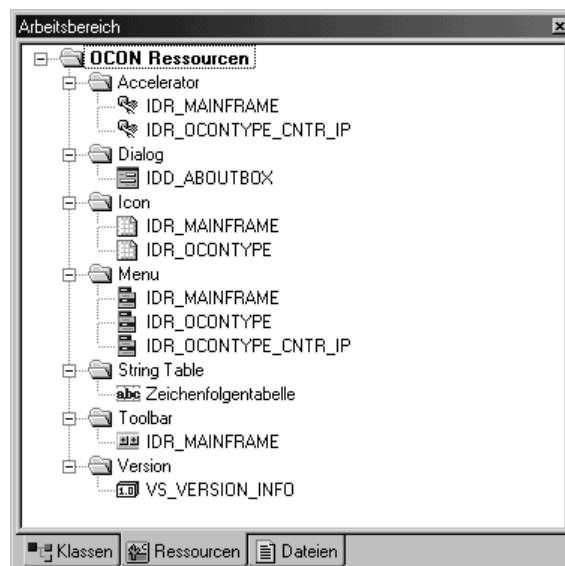


Abbildung 30.8:
Container-
Ressourcen

Das neue Menü `IDR_OCTYPE_CNTR_IP` (Abbildung 30.9) ist unvollständig. Es wird während einer Vorortbearbeitung mit einem ähnlich unvollständigen Menü der OLE-Komponenten-Server-Anwendung kombiniert.

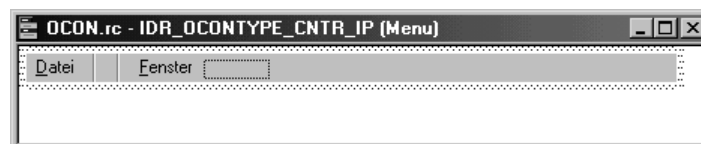
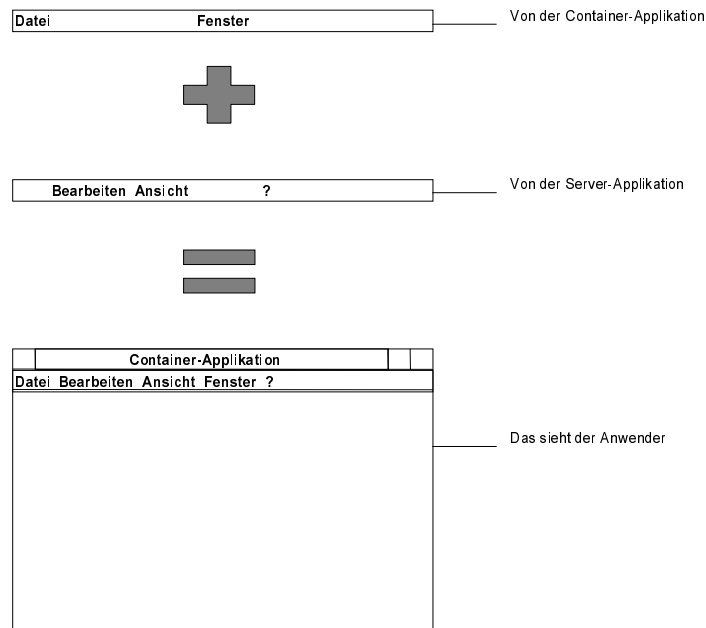


Abbildung 30.9:
Das Container-
Menü für die
Vorortbearbei-
tung

Das in Abbildung 30.10 dargestellte Kombinieren der Container- und Server-Menüs gewährleistet, daß die Anweisungen, für die der Container verantwortlich ist, von der Container-Anwendung ausgeführt werden. Server-Anweisungen werden von dem Server ausgeführt.

Abbildung 30.10:
Kombinieren
von Container-
und Server-
Menüs während
der Vorortbear-
beitung



Aufgrund der unterschiedlichen Menüs müssen ebenfalls verschiedene Zugriffstastentabellen verwendet werden.

Der nächste Abschnitt zeigt Ihnen, in welcher Weise das Anwendungsgerüst modifiziert werden muß, um einige nützliche Container-Funktionen zu implementieren.

30.2 Bearbeiten der Anwendung

Bevor wir mit der Programmierung beginnen, sollten wir wissen, welche Änderungen an dem OLE-Container-Gerüst vorgenommen werden müssen. Unsere OCON-Anwendung soll

- das Verändern der Größe und Position während der Vorortbearbeitung berücksichtigen;
- Informationen über die Größe und Position in Dokumentdateien speichern;
- alle Container-Objekte in einem Dokument anzeigen;

- ➡ einen einfachen Mechanismus für die Auswahl mit der Maus implementieren und
- ➡ die gegenwärtige Auswahl anzeigen.

Weitere Container-Features, die wir unserer Anwendung noch nicht hinzufügen, sind

- ➡ die Auswahl mehrerer Objekte,
- ➡ das Ändern der Größe und das Positionieren nicht aktiver Objekte mit der Maus sowie
- ➡ spezifische benutzerdefinierte Anwendungsobjekte.

30.2.1 Objektpositionen

Wenn Sie die Implementierung von `COCONCntrItem::OnChangeItemPosition` betrachten, werden Sie feststellen, daß eine Elementvariable definiert werden muß, die die Position des Objekts aufnimmt. Dazu fügen wir der `COCONCntrItem`-Klasse eine Elementvariable vom Typ `CRect` hinzu:

```
class COCONCntrItem : public COleClientItem
{
    ...
    // Attribute
public:
    ...
    CRect m_rect;
```

Diese Elementvariable muß initialisiert werden. Dazu weisen wir dem Element in dem Konstruktor von `COCONCntrItem` einen fixen Begrenzungsrahmen zu (Listing 30.4).

```
COCONCntrItem::COCONCntrItem(COCONDoc* pContainer)
    : COleClientItem(pContainer)
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion einfügen
    m_rect = CRect(10, 10, 210, 210);
}
```

*Listing 30.4:
Initialisierung
von m_rect
in dem
Konstruktor von
COCONCntrItem*

Eine komplexe Anwendung könnte versuchen, `m_rect` derart zu initialisieren, daß diese Variable eine vom Server zur Verfügung gestellte Größe aufnimmt. Dazu dürfte `m_rect` kein Begrenzungsrahmen zugewiesen werden (0, 0, 0, 0). Der Server würde die Variable während des ersten Aufrufs von `COCONCntrItem::OnGetItemPosition` aktualisieren. Wir geben uns jedoch zunächst mit einem fixen Wert zufrieden.

Die Funktion `COCONCntrItem::OnChangeItemPosition` muß modifiziert werden, um `m_rect` zu aktualisieren. `COCONCntrItem::OnGetItemPosition` muß ebenfalls verändert werden, damit die aktualisierte Position ermittelt werden kann.

OnChangeItemPosition Die `OnChangeItemPosition`-Funktion erfordert lediglich zwei Zeilen neuen Programmcode (Listing 30.5). Der Begrenzungsrahmen wird zunächst aktualisiert. Da sich dadurch die Position des Objekts verändert, muß anschließend die Ansicht aktualisiert werden, damit diese die Änderung anzeigt.

Listing 30.5:
Die modifizierte Version von `COCONtrItem::OnChangeItemPosition`

```

BOOL COCONtrItem::OnChangeItemPosition(const CRect& rectPos)
{
    ASSERT_VALID(this);
    if (!COleClientItem::OnChangeItemPosition(rectPos))
        return FALSE;
    // ZU ERLEDIGEN: Aktualisieren Sie alle für das RECHTECK/das
    // Extent dieses Elements angelegten Caches
    m_rect = rectPos;
    GetDocument()->UpdateAllViews(NULL);
    return TRUE;
}

```

OnGetItemPosition In `COCONtrItem::OnGetItemPosition` muß die Anweisung, die den `rPosition`-Parameter auf einen konstanten Begrenzungsrahmen setzt, durch eine Programmzeile ersetzt werden, die dem Parameter den Inhalt der Variablen `m_rect` zuweist. Die modifizierte Funktion ist in Listing 30.6 aufgeführt.

Listing 30.6:
Die modifizierte Version von `COCONtrItem::OnGetItemPosition`

```

void COCONtrItem::OnGetItemPosition(CRect& rPosition)
{
    ASSERT_VALID(this);
    // ZU ERLEDIGEN: Geben Sie das korrekte Rechteck (in Pixeln)
    // in rPosition zurück
    rPosition = m_rect;
}

```

Serialize Ändern Sie nun die `Serialize`-Elementfunktion von `COCONtrItem`. Da wiederholt auf die Objektpositionen zugegriffen wird, sollten diese serialisiert werden. Dies geschieht mit zwei zusätzlichen Programmzeilen in `COCONtrItem::Serialize`, wie in Listing 30.7 aufgeführt:

Listing 30.7:
Die modifizierte Version von `COCONtrItem::Serialize`

```

void COCONtrItem::Serialize(CArchive& ar)
{
    ASSERT_VALID(this);
    COleClientItem::Serialize(ar);
    // jetzt die speziellen Daten für COCONtrItem
    // einlesen/speichern
    if (ar.IsStoring())
    {
        // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
        ar << m_rect;
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
        ar >> m_rect;
    }
}

```

Nun, da wir die Informationen über die Größe und die Position der Objekte in dem Container implementiert haben, sollten wir uns der Ansichtsklasse zuwenden. Dort muß die neue Position während des Zeichnens der Objekte berücksichtigt werden.

30.2.2 Zeichnen aller Objekte

Wir haben zwei Unzulänglichkeiten der Standardimplementierung der Ansichtsklasse `COCONView` festgestellt. Zunächst wird lediglich das aktuelle Objekt gezeichnet. Außerdem wird das Objekt an einer fixen Position gezeichnet und berücksichtigt nicht die während der Vorortbearbeitung vorgenommenen Änderungen der Größe und Position.

Die Lösung dieses Problems ist in Listing 30.8 aufgeführt. Diese Version von `COCONView::OnDraw` zeichnet nicht ausschließlich ein Objekt, sondern bearbeitet die vollständige Liste aller Objekte des Dokuments. Die einzelnen Objekte werden an Positionen gezeichnet, die in ihrer `m_rect`-Elementvariable gespeichert sind. Die Größe des Objekts ist ebenfalls in dieser Variablen enthalten.

```
void COCONView::OnDraw(CDC* pDC)
{
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // ZU ERLEDIGEN: Entfernen Sie diesen Code, sobald Ihre
    // richtige Zeichenroutine vollständig ist.

    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        COCONCntrItem *pItem =
            (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        pItem->Draw(pDC, pItem->m_rect);
    }
}
```

*Listing 30.8:
Die modifizierte
Version von CO-
CONView: :
OnDraw*

Beachten Sie bitte, daß dieser Programmcode auch dann verändert werden muß, wenn Ihre Anwendung gewöhnliche Objekte bearbeitet. Möglicherweise möchten Sie anstelle von `COleDocument::GetNextClientItem` die Funktion `COleDocument::GetNextItem` verwenden, um die Liste der Objekte zu bearbeiten und Laufzeittypeninformationen zu ermitteln. Diese bestimmen, wie die unterschiedlichen Objekttypen gezeichnet werden müssen.



30.2.3 Objektauswahl

Um einen einfachen Mechanismus zur Auswahl eines Objekts zu implementieren, definieren Sie zunächst einen Bearbeiter für das Mauseignis `WM_LBUTTONDOWN`. Die aktuelle Selektion muß in `COCONView` berücksichtigt werden, wenn das Objekt gezeichnet wird.

Erstellen Sie mit dem Klassen-Assistenten einen Bearbeiter für das Mauseignis. Die Bearbeiterfunktion sollte der `COCONView`-Klasse hinzugefügt werden, da die Selektion eines Objekts in der aktuellen Ansicht geschieht (unterschiedliche Ansichten können verschiedene Selektionen enthalten).

Die Bearbeiterfunktion ist in Listing 30.9 aufgeführt. Sie schließt zunächst ein aktives Vorortobjekt. Anschließend wird `InvalidateRect` aufgerufen, um den Begrenzungsrahmen zu löschen. Die Bedeutung dieser Vorgehensweise wird später offensichtlich, wenn wir den Programmcode besprechen, der den Rahmen zeichnet.

Listing 30.9:
Bearbeiter-
funktion für
`WM_LBUTTONDOWN`-Nachrichten

```
void COCONView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Nachrichtebearbeiter hier einfügen

    // CView::OnLButtonDown(nFlags, point);
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COCONCntrItem *pItem =
        (COCONCntrItem *)pDoc->GetInPlaceActiveItem(this);
    if (pItem != NULL) pItem->Close();
    if (m_pSelection != NULL)
    {
        CRect rect = m_pSelection->m_rect;
        rect.InflateRect(1, 1);
        InvalidateRect(rect);
        m_pSelection = NULL;
    }
    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        pItem = (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        if (pItem->m_rect.PtInRect(point))
            m_pSelection = pItem;
    }
    if (m_pSelection != NULL)
    {
        CRect rect = m_pSelection->m_rect;
        rect.InflateRect(1, 1);
        InvalidateRect(rect);
    }
}
```

In der zweiten Hälfte dieser Funktion wird eine Schleife ausgeführt, um das Objekt zu identifizieren, das der Anwender mit der Maus auswählt. Wird solch ein Objekt gefunden, wird es zur aktuellen Selektion. Die Schleife wird in diesem Fall weiterhin durchlaufen, um die Möglichkeit

zu berücksichtigen, daß die aktuelle Selektion das oberste Objekt ist. Dies ist für die Situationen wichtig, in denen die Maustaste an einer Position betätigt wird, an der mehrere Objekt übereinander angeordnet sind. Nachdem eine neue Selektion ermittelt wurde, wird deren Begrenzungsrahmen gelöscht.

Da dieser Programmcode die Auswahl einzelner Objekte mit der Maus ermöglicht, modifizieren wir `COCONView::OnDraw`, damit die neue Selektion angezeigt wird. Listing 30.10 zeigt die endgültige Version von `COCONView::OnDraw`.

```
void COCONView::OnDraw(CDC* pDC)
{
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        COCONCntrItem *pItem =
            (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        pItem->Draw(pDC, pItem->m_rect);
        if (pItem == m_pSelection)
        {
            CRectTracker tracker;
            tracker.m_rect = pItem->m_rect;
            tracker.m_nStyle = CRectTracker::resizeInside |
                              CRectTracker::solidLine;
            if (pItem->GetItemState() ==
                COleClientItem::openState
                || pItem->GetItemState() ==
                COleClientItem::activeUIState)
                tracker.m_nStyle |=
                    CRectTracker::hatchInside;
            tracker.Draw(pDC);
        }
    }
}
```

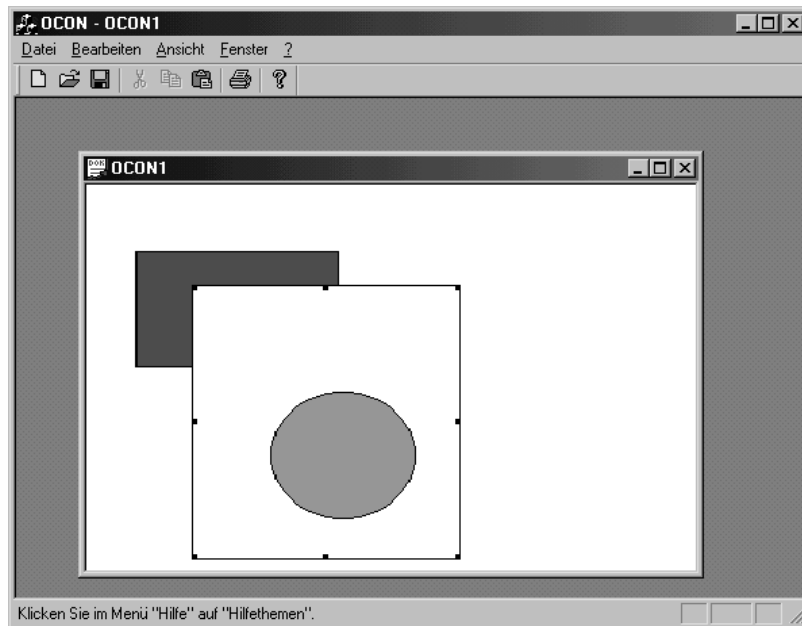
*Listing 30.10:
Die endgültige
Version von
COCONView::
OnDraw*

In dieser Version der `OnDraw`-Elementfunktion verwenden wir die Klasse `CRectTracker`, um einen RAHMEN zu erstellen, der die Selektion umgibt. Wir nutzen nicht alle Features dieser Klasse. Sie kann ebenfalls dazu eingesetzt werden, das Objekt zu verschieben und dessen Größe zu ändern. Wir verwenden die Klasse, um ein visuelles Feedback zur Verfügung zu stellen.

Ein Nachteil dieser Implementierung besteht darin, daß der Überwachungsrahmen kontinuierlich auch dann von der `OnDraw`-Elementfunktion gezeichnet wird, wenn der Anwendungsrahmen `OnDraw` für den Ausdruck und die Druckvorschau verwendet. In einer umfassenden Anwendung würden wir die Ausführung dieses Programmabschnitts davon abhängig machen, ob eine »gewöhnliche« Zeichensituation vorliegt. Läge diese nicht vor, würden wir das Zeichnen des Überwachungsrahmens einstellen.

Die Änderungen an der OCON-Anwendung sind nun abgeschlossen. Nach dem Kompilieren und Starten des Programms können Sie testen, wie mehrere eingebettete Objekte verwaltet und Dokumente geladen sowie gespeichert werden (Abbildung 30.11).

Abbildung 30.11:
Die finale Version
der OCON-
Anwendung



30.2.4 Andere Features

Es bestehen weitere Container-Anwendungsfeatures, die der OCON-Anwendung einfach hinzugefügt werden können.

Aktivierung per Doppelklick

In der aktuellen Version der Anwendung müssen Sie ein eingebettetes Objekt selektieren und anschließend das entsprechende Untermenü aus dem Menü BEARBEITEN auswählen, um das Objekt bearbeiten zu können. Möchten Sie eine einfache Möglichkeit zur Aktivierung eines eingebetteten Objekts anbieten, implementieren Sie einen Bearbeiter für die `WM_LBUTTONDOWNBLCLK`-Funktion. In der Funktion verwenden Sie die `DoVerb`-Elementfunktion in der `COCONCntrItem`-Klasse, um ein Objekt für die Vorortbearbeitung zu aktivieren.

Objekte verschieben

In dem Bearbeiter für `WM_LBUTTONDOWN` können Sie die `CRectTracker`-Klasse nutzen, um ein Objekt zu verschieben oder dessen Größe zu ändern.

Mehrere Objekte auswählen

Die Selektion ist nicht auf ein Objekt beschränkt. Anstelle eines einzelnen Zeigers können Sie die Selektion als Liste mit `COCONtrItem`-Objekten in einer Auflistungsklasse, wie zum Beispiel `COBList`, implementieren. Die mehrfache Auswahl wird realisiert, indem Sie die Tasten (`⌘`) und (`Strg`) in dem Bearbeiter für `WM_LBUTTONDOWN` kontrollieren oder ein Verfahren zum Aufziehen eines Markierungsrahmens um die zu selektierenden Objekte implementieren (Gummibandtechnik).

Diese Techniken sind in echten Anwendungen unerlässlich. Zur Demonstration der grundlegenden Container-Funktionen ist unsere `OCON`-Anwendung jedoch völlig ausreichend.

30.3 Zusammenfassung

OLE-Container sind Anwendungen, die eingebettete oder verknüpfte Objekte bearbeiten. Container-Anwendungen können mit dem Anwendungsassistenten erstellt werden. Sie müssen lediglich den gewünschten Container-Typ im dritten Schritt des Assistenten auswählen. OLE-Container müssen Anwendungen sein, die ein oder mehrere Dokumente verwenden. Dialogfeldbasierende Container-Anwendungen werden nicht unterstützt.

Ein vom Anwendungsassistenten generiertes Anwendungsgerüst unterstützt das Einfügen eines einzelnen Komponentenobjekts an einer fixen Position. Das Objekt wird über das Menü `BEARBEITEN` und dem Eintrag `NEUES OBJEKT EINFÜGEN` in das Dokument eingefügt. Die Anwendung unterstützt nicht das Verschieben oder das Ändern der Größe des Objekts während der Vorortbearbeitung.

Die Container-Anwendung repräsentiert Komponentenobjekte mit einer von `COleClientItem` abgeleiteten Klasse. Weitere Unterschiede zwischen einem Container und einer gewöhnlichen Anwendung sind der neue Programmcode in der Ansichtsklasse der Container-Anwendung, der das Einfügen und Selektieren von Objekten ermöglicht, sowie ein unvollständiges Menü, das die während der Vorortbearbeitung angebotenen Container-Funktionen enthält.

Das Bearbeiten des Anwendungsgerüsts geschieht wie folgt:

- ➔ Modifizieren Sie die Elementfunktionen `OnChangeItemPosition` und `OnGetItemPosition` der Container-Objektklasse, um die Container-Objekte korrekt zu positionieren. Fügen Sie der Klasse eine Elementvariable hinzu, die die aktuelle Größe und Position des jeweiligen Objekts aufnimmt. Stellen Sie sicher, daß diese Elementvariable serialisiert wird.
- ➔ Modifizieren Sie die `OnDraw`-Elementfunktion der Ansichtsklasse, damit die Objekte an der korrekten Position gezeichnet werden.
- ➔ Modifizieren Sie diese Klasse ebenfalls, um andere Objekte als die aktuelle Selektion zu zeichnen und um ein visuelles Feedback zur Verfügung zu stellen, das die aktuelle Selektion berücksichtigt.
- ➔ Nutzen Sie die `CRectTracker`-Klasse, um einen Rahmen um die Selektion zu zeichnen.
- ➔ Fügen Sie der Ansichtsklasse einen Bearbeiter für `WM_LBUTTONDOWN`-Ereignisse hinzu, um die Selektion mit Hilfe der Maus zu ermöglichen.
- ➔ Um ein Objekt mit einem Doppelklick für die Vorortbearbeitung zu aktivieren, fügen Sie der Ansichtsklasse einen Bearbeiter für `WM_LBUTTONDBLCLK`-Ereignisse hinzu.
- ➔ Ersetzen Sie die `m_pSelection`-Variable Ihrer Ansichtsklasse, um die Auswahl mehrerer Objekte zu implementieren. Modifizieren Sie dazu außerdem den `WM_LBUTTONDOWN`-Bearbeiter.
- ➔ Nutzen Sie in dem `WM_LBUTTONDOWN`-Bearbeiter die `CRectTracker`-Klasse, um die Größenänderung und Positionierung von Objekten zu implementieren.
- ➔ Möchten Sie in dem Dokument gewöhnliche Objekte bearbeiten, sollten Sie eine Klasse von `CDocItem` ableiten, die diese Objekte repräsentiert. Nutzen Sie die Funktionen von `COleDocument` für die Bearbeitung und Serialisierung der `CDocItem`-Objekte. Überarbeiten Sie die `OnDraw`-Elementfunktion der Ansichtsklasse, um Objekte zu zeichnen, die keine Container-Objekte sind. Modifizieren Sie ebenfalls den Bearbeiter für die Mausereignisse.

OLE-Drag&Drop

Kapitel

31

OLE bietet eine einfache, einheitliche Möglichkeit der Implementierung von Drag&Drop in Anwendungen.

Das OLE-Drag&Drop wird in diesem Kapitel anhand einer OLE-Container-Anwendung beschrieben, die wir erstellen werden. Diese Anwendung wird mehrere Container-Objekte aufnehmen können. Mit Hilfe des OLE-Drag&Drop-Mechanismus können die Objekte dieser Anwendung zwischen

- ➡ dem Programm und anderen Anwendungen,
- ➡ zwischen den eigenen Ansichtsfenstern des Programms sowie
- ➡ innerhalb eines einzelnen Ansichtsfensters verschoben werden.

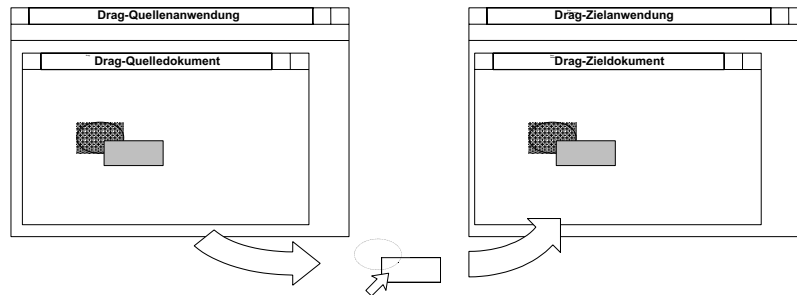
31.1 Drag&Drop-Grundlagen

Drag&Drop repräsentiert eine Technik der gemeinsamen Datenverwendung von Anwendungen, die als DRAG-QUELLEN und DROP-ZIELE agieren.

Drag-Quellen sind Anwendungen, die das Verschieben ihrer Objekte in andere Anwendungen ermöglichen.

Drop-Ziele sind Programme, die diese Objekte von den Drag-Quellen entgegennehmen und in ihren Fenstern anordnen (Abbildung 31.1).

Das Implementieren einer Drag-Quelle ist mit Hilfe der MFC sehr einfach. Das Erstellen eines Drop-Ziels gestaltet sich ein wenig umständlicher.

Abbildung 31.1:
Drag&Drop

COleClientItem Die in diesem Kapitel vorgestellte einfache Implementierung basiert auf der Drag&Drop-Unterstützung in der Klasse `COleClientItem`. In Anwendungen, die das Drag&Drop der ursprünglichen Daten unterstützen und mehrere Objekte eines Drag&Drop-Vorgangs entgegennehmen können, sollten Sie statt dessen die Drag&Drop-Möglichkeiten der Klasse `COleServerItem` verwenden.

Die Funktionalität von Drag&Drop und die der Zwischenablage sind sich sehr ähnlich. Möchten Sie das Ausschneiden, Kopieren und Einfügen über die Zwischenablage unter Verwendung von OLE implementieren, können Sie den Programmcode für die Unterstützung der Zwischenablage und den OLE-Code gemeinsam verwenden.

31.2 Erstellen einer Container-Anwendung

Die Container-Anwendung, die wir erstellen werden, basiert auf einer vom Anwendungsassistenten generierten Standard-Container-Anwendung. Lediglich einige Änderungen müssen an der Anwendung vorgenommen werden, damit Objekte mit der Maus selektiert und die Positionen der Objekte gespeichert werden können.

31.2.1 Erstellen der Anwendung

1. Zum Erstellen der Anwendung verwenden Sie die Voreinstellungen des Anwendungsassistenten.
2. Wählen Sie im dritten Schritt des Assistenten die Option CONTAINER aus.
3. Nennen Sie die Anwendung OCON, da sie auf dem in Kapitel 30 verwendeten Beispiel basiert.

Die folgenden Abschnitte beschreiben in einer kurzen Übersicht, welche Änderungen an dem vom Anwendungsassistenten generierten Anwendungsgerüst vorgenommen werden müssen, damit die Objektpositionen gespeichert und die Auswahl mit der Maus unterstützt werden.

31.2.2 Speichern der Objektpositionen

Die Standardimplementierung einer Container-Anwendung positioniert eingefügte Objekte in einem fixen Rahmen. Um die Objekte an anderen Positionen der Zeichenoberfläche anordnen zu können, muß der Klasse `COCONCntrItem` eine Elementvariable vom Typ `CRect` hinzugefügt werden. Die Variable muß serialisiert werden können. Die `Draw`-Elementfunktion der Ansichtsklasse nutzt diese Variable zum Zeichnen der Objekte.

Die Variable sollte außerdem in den `COCONCntrItem`-Elementfunktionen `OnChangeItemPosition` und `OnGetItemPosition` verwendet werden. `COCONCntrItem` muß außerdem eine neue Helferfunktion mit der Bezeichnung `Invalidate` hinzugefügt werden. Diese Funktion nimmt einen `CWnd`-Zeiger als Parameter entgegen und löscht den Rahmen eines Objekts in dem angegebenen Fenster.

Um die Elementvariable der Klasse `COCONCntrItem` hinzuzufügen und die neue Helferfunktion `Invalidate` zu deklarieren, verwenden Sie den folgenden Programmcode:

```
class COCONCntrItem : public COleClientItem
{
    ...
    // Attribute
public:
    ...
    CRect m_rect;
    ...
    // Vom Klassenassistenten generierte Überladungen virtueller
    // Funktionen
    ///{AFX_VIRTUAL(COCONCntrItem)public:
    virtual void Invalidate(CWnd *pWnd);

// Implementierung
    ...
}
```

Die neue Elementvariable wird in dem `COCONCntrItem`-Konstruktor initialisiert (Listing 31.1).

```
COCONCntrItem::COCONCntrItem(COCONDoc* pContainer)
: COleClientItem(pContainer)
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion einfügen
    m_rect = CRect(0, 0, 0, 0);
}
```

*Listing 31.1:
Initialisierung
von m_rect in
dem COCON-
CntrItem-Kon-
struktor*

Die modifizierten Versionen von `COCONCntrlItem::OnChangeItemPosition` und `COCONCntrlItem::OnGetItemPosition` (Listing 31.2) berücksichtigen die während der Vorortbearbeitung vorgenommenen Änderungen an der Objektgröße und Objektposition. Sie verwenden die `GetCacheExtent`-Elementfunktion, um die Größe und Position eines Objekts zu aktualisieren, wenn dieses nicht initialisiert wurde.

Listing 31.2:
COCONCntrlItem:
: OnChange-
ItemPosition und
COCONCntrlItem:
: OnGetItem-
Position

```

BOOL COCONCntrlItem::OnChangeItemPosition(const CRect& rectPos)
{
    ASSERT_VALID(this);

    if (!COleClientItem::OnChangeItemPosition(rectPos))
        return FALSE;

    // ZU ERLEDIGEN: Aktualisieren Sie alle für das Rechteck/das
    // Extent dieses Elements angelegten Caches
    m_rect = rectPos;
    GetDocument()->UpdateAllViews(NULL);

    return TRUE;
}

void COCONCntrlItem::OnGetItemPosition(CRect& rPosition)
{
    ASSERT_VALID(this);

    if (m_rect.IsRectNull())
    {
        CSize size;
        CClientDC dc(NULL);
        GetCachedExtent(&size, GetDrawAspect());
        dc.HIMETRICtoDP(&size);
        m_rect = CRect(CPoint(10, 10), size);
    }
    rPosition = m_rect;
}

```

Listing 31.3 zeigt, wie `COCONCntrlItem::Serialize` modifiziert werden muß, um die neue Elementvariable zu serialisieren.

Listing 31.3:
Modifizierte Ver-
sion von COCON-
CntrlItem: : Seria-
lize

```

void COCONCntrlItem::Serialize(CArchive& ar)
{
    ASSERT_VALID(this);

    COleClientItem::Serialize(ar);

    if (ar.IsStoring())
    {
        // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
        ar << m_rect;
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
        ar >> m_rect;
    }
}

```

Schließlich beenden wir die Änderungen an `COCONCntrItem`, indem wir der Klasse die neue Helferfunktion `Invalidate` hinzufügen (Listing 31.4).

```
virtual void COCONCntrItem::Invalidate(CWnd *pWnd)
{
    CRect rect = m_rect;
    rect.InflateRect(1, 1);
    pWnd->InvalidateRect(rect);
}
```

*Listing 31.4:
Die Helferfunktion
COCONCntr-
Item::Invalidate*

Listing 31.5 zeigt die Änderungen an `COCONView::OnDraw`. Die neue Version berücksichtigt die gespeicherten Positionen der Objekte und zeichnet die Objekte gemäß dieser Positionen. Außerdem zeichnet die Funktion alle Objekte (und nicht nur die aktuelle Selektion). Ein `CRectTracker`-Objekt wird verwendet, um einen Markierungsrahmen um die gegenwärtige Selektion darzustellen.

```
void COCONView::OnDraw(CDC* pDC)
{
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        COCONCntrItem *pItem =
            (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        pItem->Draw(pDC, pItem->m_rect);
        if (pItem == m_pSelection)
        {
            CRectTracker tracker;
            tracker.m_rect = pItem->m_rect;
            tracker.m_nStyle = CRectTracker::resizeInside |
                CRectTracker::solidLine;
            if (pItem->GetItemState() ==
                COleClientItem::openState
                || pItem->GetItemState() ==
                COleClientItem::activeUIState)
                tracker.m_nStyle |=
                    CRectTracker::hatchInside;
            tracker.Draw(pDC);
        }
    }
}
```

*Listing 31.5:
Die modifizierte
Version von CO-
CONView::
OnDraw*

31.2.3 Objekt selektieren

In den vorherigen Abschnitten wurde häufig die gegenwärtige Selektion erwähnt. Wir werden die Auswahl von Objekten mit der Maus unterstützen, indem wir der Ansichtsklasse einen Bearbeiter für `WM_LBUTTONDOWN`-Ereignisse hinzufügen. Dieser in Listing 31.6 aufgeführte Bearbeiter kann mit Hilfe des Klassen-Assistenten generiert werden.

Listing 31.6:
 Bearbeiter-
 funktion für
 WM_LBUTTONDOWN-
 Nachrichten

```
void COCONView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Nachrichtebearbeiter hier einfügen

    // CView::OnLButtonDown(nFlags, point);
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COCONtrItem *pSelection = NULL;
    COCONtrItem *pItem =
        (COCONtrItem *)pDoc->GetInPlaceActiveItem(this);

    if (pItem != NULL) pItem->Close();
    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        pItem = (COCONtrItem*)pDoc->GetNextClientItem(pos);
        if (pItem->m_rect.PtInRect(point))
            pSelection = pItem;
    }
    if (pSelection != m_pSelection)
    {
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
        m_pSelection = pSelection;
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
    }
}
```

31.3 Drag&Drop-Unterstützung hinzufügen

Obwohl das Ziehen (Drag) und das Fallenlassen (Drop) gewöhnlich in Anwendungen gemeinsam verwendet werden, unterscheiden sich die Anforderungen an eine Drag-Quelle von den Anforderungen an ein Drop-Ziel. Die Implementierung einer Drag-Quelle ist die leichtere Aufgabe. Die beiden Techniken werden jedoch umfassend von der OLE-Architektur in der MFC unterstützt.

- ➔ Eine Drag-Quelle wird über die `DoDragDrop`-Elementfunktion verschiedener Klassen, wie `COleClientItem`, `COleServerItem` und `COleDataSource`, implementiert.
- ➔ Die Drop-Ziel-Funktionalität wird über verschiedene Elementfunktionen der `CView`-Klasse realisiert. Dazu zählen `OnDrop`, `OnDragEnter`, `OnDragOver` und `OnDragLeave`. Im Gegensatz zu der `DoDragDrop`-Funktion, müssen diese Funktionen in Ihrer Anwendung modifiziert werden.

31.3.1 Implementieren einer Drag-Quelle

Sie fügen dem OLE-Container mit Hilfe der `COleClientItem::DoDragDrop`-Elementfunktion Drag-Quellen-Funktionalität hinzu.

Modifizieren Sie dazu den `COCONView::OnLButtonDown`-Bearbeiter für `WM_LBUTTONDOWN`-Ereignisse.

- ➡ Selektiert der Anwender ein Objekt mit der Maus, rufen wir die `DoDragDrop`-Elementfunktion des Objekts auf, um die Drag&Drop-Operation auszuführen.
- ➡ Wird das Objekt von einer Anwendung in eine andere Anwendung verschoben, löschen wir es aus der von der Dokumentklasse geführten Objektliste. Für von `CDocItem` abgeleitete Objekte ist dies eine sehr einfache Aufgabe. Das Objekt wird mit dem `delete`-Operator gelöscht. Die Destruktorfunktion `CDocItem` gewährleistet, daß das Objekt korrekt aus der Objektliste entfernt wird.

Die modifizierte Version von `COCONView::OnLButtonDown` ist in Listing 31.7 aufgeführt.

```
void COCONView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Nachrichtensbearbeiter hier einfügen

    // CView::OnLButtonDown(nFlags, point);
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COCONCntrItem *pSelection = NULL;
    COCONCntrItem *pItem =
        (COCONCntrItem *)pDoc->GetInPlaceActiveItem(this);

    if (pItem != NULL) pItem->Close();
    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        pItem = (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        if (pItem->m_rect.PtInRect(point))
            pSelection = pItem;
    }
    if (pSelection != m_pSelection)
    {
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
        m_pSelection = pSelection;
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
    }

    if (m_pSelection != NULL)
    {
        m_dragRect = m_pSelection->m_rect;
        if (m_pSelection->DoDragDrop(m_pSelection->m_rect,
            (CPoint)(point - m_pSelection->m_rect.TopLeft())) ==
            DROPEFFECT_MOVE)
        {

```

Listing 31.7:
COCON View::
OnLButtonDown
mit Drag&Drop-
Unterstützung


```

        m_pSelection->Invalidate(this);
        delete m_pSelection;
        m_pSelection = NULL;
    }
}

```

Bevor wir die Drop-Ziel-Funktionalität implementieren, möchte ich Ihnen noch einige Hinweise zu dieser Drag-Quellenimplementierung geben.

Die Funktionalität der meisten Anwendungen übersteigt die eines gewöhnlichen OLE-Containers (sofern Container-Funktionalität implementiert ist). Doch wie weisen Sie solchen Anwendungen Drag-Quellen-Features zu?

COleServerItem Eine Möglichkeit besteht darin, die *DoDragDrop*-Elementfunktion von *COleServerItem* zu nutzen. Ist Ihre Anwendung ein OLE-Server, verfügt sie bereits über eine von *COleServerItem* abgeleitete Klasse, die das Dokument Ihrer Anwendung in einem eingebetteten Objekt repräsentiert. Modifizieren Sie diese Klasse, um lediglich die aktuelle Selektion darzustellen (und nicht das gesamte Dokument). Ändern Sie dazu die Elementfunktionen *Serialize* und *OnDraw*. Erstellen Sie anschließend einen Konstruktor, der einen Parameter entgegennimmt, der die aktuelle Selektion repräsentiert. Die Klasse kann ebenfalls dazu verwendet werden, verknüpfte Objekte darzustellen und erleichtert den Transfer der gegenwärtigen Selektion in die Zwischenablage.

Nachdem Sie die von *COleServerItem* abgeleitete Klasse bearbeitet haben, erstellen Sie ein Objekt dieses Typs in Ihrem *WM_LBUTTONDOWN*-Bearbeiter (oder dort, wo Sie die Drag-Quellenfunktionalität implementieren möchten). Sie können die Elementfunktion *COleServerItem::DoDragDrop* im Anschluß daran nutzen.

COleDataSource Möchten Sie keine von *COleServerItem* abgeleitete Klasse zu diesem Zweck verwenden (wenn Ihre Anwendung beispielsweise kein OLE-Server ist), setzen Sie die *COleDataSource*-Klasse ein. Diese repräsentiert eine Selektion, die in die Zwischenablage oder über Drag&Drop verschoben werden soll. *COleDataSource* verfügt über die *DoDragDrop*-Elementfunktion. Das Implementieren der Drag-Quellenfunktionalität ist daher mit Hilfe dieser Klasse sehr einfach.

31.3.2 Implementieren eines Drop-Ziels

Das Implementieren eines Drop-Ziels erfordert eine komplexere Vorgehensweise als die Implementierung einer Drag-Quelle. Verschiedene Elementfunktionen Ihrer Ansichtsklasse müssen überschrieben werden. Die Ansichtsklasse selbst muß als Drop-Ziel registriert werden. Bestimmte Umstände müssen berücksichtigt werden, um zu gewährlei-

sten, daß die ursprünglichen Anwendungsobjekte korrekt und effizient bearbeitet werden. Wird eine Drag&Drop-Operation beispielsweise auf das Verschieben eines Objekts innerhalb desselben Fensters beschränkt, muß dies in dem Programmcode der Elementfunktionen beachtet werden.

Die zu überschreibenden CView-Elementfunktionen sind in Tabelle 31.1 aufgeführt.

Elementfunktion	Beschreibung
OnDragEnter	Wird aufgerufen, wenn ein Objekt in das Fenster gezogen wird.
OnDragLeave	Wird aufgerufen, wenn ein in das Fenster gezogenes Objekt aus dem Fenster herausgezogen wird.
OnDragOver	Wird aufgerufen, wenn ein Fenster innerhalb desselben Fensters verschoben wird.
OnDrop	Wird aufgerufen, wenn ein Objekt in dem Fenster abgelegt wird.

*Tabelle 31.1:
Drag&Drop-
Überschreibungen
in CView*

Bevor wir den entsprechenden Programmcode schreiben, bestimmen wir zunächst, wie unsere Drop-Ziel-Anwendung aufgebaut sein soll.

- ➡ Wird ein Objekt über dem Ansichtsfenster der Anwendung abgelegt, soll es an dieser Position und in der originalen Größe angeordnet werden.
- ➡ Außerdem soll ein Markierungsrahmen dargestellt werden, während sich die Maus innerhalb des Ansichtsfensters befindet. Der Rahmen zeigt die Größe des Objekts an, das in dem Fenster abgelegt werden soll.
- ➡ Schließlich soll es möglich sein, ein Objekt per Drag&Drop innerhalb desselben Fensters zu verschieben.

*Anforderungs-
katalog*

Die Implementierung der Funktionen OnDragEnter, OnDragLeave und OnDragOver erfordert einige Elementvariablen. Diese Variablen nehmen die Größe des Rahmens, dessen Position und andere Merkmale während der Drag-Operation auf. Weiterhin wird eine Elementvariable vom Typ COleDropTarget erzeugt, die das Ansichtsfenster als Drop-Ziel registriert. Die Deklaration der Variablen wird der Deklaration der Ansichtsklasse hinzugefügt:

```
class COCONView : public CView
{
    ...
    // Attribute
```

```
public:
    ...
    COCONtrItem* m_pSelection;
    COleDropTarget m_dropTarget;
    BOOL m_bInDrag;
    DROPEFFECT m_prevDropEffect;
    CRect m_dragRect;
    CPoint m_dragPoint;
    CSize m_dragSize;
    CSize m_dragOffset;
    ...
```

Die Deklaration der Funktionen `OnDragEnter`, `OnDragLeave`, `OnDragOver` und `OnDrop` sollten mit dem Klassen-Assistenten erzeugt werden.

Die Anwendung muß zunächst als Drop-Ziel registriert werden. Dazu fügen wir der Anwendung eine Elementvariable vom Typ `COleDropTarget` hinzu und rufen die Register-Elementfunktion auf. Ein dafür geeigneter Ort ist die `OnCreate`-Elementfunktion der Ansichtsklasse. Erstellen Sie für diese Registrierung einen `WM_CREATE`-Nachrichtenbearbeiter mit Hilfe des Klassen-Assistenten, und fügen Sie der Funktion den in Listing 31.8 aufgeführten Programmcode hinzu.

Listing 31.8:
Die `COCONView::OnCreate`-Elementfunktion

```
int COCONView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Speziellen Erstellungscode hier einfügen
    m_dropTarget.Register(this);

    return 0;
}
```

Die Elementfunktionen `OnDragEnter`, `OnDragLeave` und `OnDragOver` verwalten das visuelle Feedback während der Drag-Operation.

OnDragEnter `OnDragEnter` (Listing 31.9) versucht die Objektgröße mit einer Abfrage des Objekts vom Zwischenablagetyp `ObjectDescriptor` zu ermitteln. Dieser Datentyp enthält Informationen über das zu transferierende Objekt einschließlich der Größe und der relativen Verschiebung des Mauszeigers zur linken oberen Ecke des Objekts.

Listing 31.9:
Die `COCONView::OnDragEnter`-Elementfunktion

```
DROPEFFECT COCONView::OnDragEnter(COleDataObject* pDataObject,
                                   DWORD dwKeyState, CPoint point)
{
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse
    // aufrufen

    // return CView::OnDragEnter(pDataObject, dwKeyState, point);
    ASSERT(m_prevDropEffect == DROPEFFECT_NONE);

    m_dragSize = CSize(0, 0);
    m_dragOffset = CSize(0, 0);
    HGLOBAL hObjDesc =
        pDataObject->GetGlobalData(cfObjectDescriptor);
    if (hObjDesc != NULL)
    {
```

```

        LPOBJECTDESCRIPTOR pObjDesc =
            (LPOBJECTDESCRIPTOR)GlobalLock(hObjDesc);
        ASSERT(pObjDesc != NULL);
        m_dragSize.cx = (int)pObjDesc->size.cx;
        m_dragSize.cy = (int)pObjDesc->size.cy;
        m_dragOffset.cx = (int)pObjDesc->point.x;
        m_dragOffset.cy = (int)pObjDesc->point.y;
        GlobalUnlock(hObjDesc);
        GlobalFree(hObjDesc);
    }
    CClientDC dc(NULL);
    dc.HIMETRICtoDP(&m_dragSize);
    dc.HIMETRICtoDP(&m_dragOffset);
    m_dragPoint = point - CSize(1, 1);
    return OnDragOver(pDataObject, dwKeyState, point);
}

```

Diese Funktion verwendet die globale Variable `cfObjectDescriptor`. Deklarieren Sie diese Variable zu Beginn der Ansichtsklassendatei:

```

static cfObjectDescriptor =
    (CLIPFORMAT)::RegisterClipboardFormat(_
        T("Object Descriptor"));

```

Die Elementfunktion `OnDragOver` (Listing 31.10) wird immer dann aufgerufen, wenn sich die Maus innerhalb des Client-Bereichs des Ansichtsfensters befindet. Sie bestimmt die gegenwärtig ausgeführte Drop-Funktion. Abhängig von dem Status der Tasten (`[Strg]`), (`[⇧]`) und (`[Alt]`) bestimmt die Funktion außerdem, ob das Objekt mit dem Fenster verknüpft oder in das Fenster verschoben respektive kopiert werden soll, sofern es zuvor dort abgelegt wurde.

OnDragOver

```

DROPEFFECT COCONView::OnDragOver(COLEDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse
    // aufrufen

    // return CView::OnDragOver(pDataObject, dwKeyState, point);
    DROPEFFECT de = DROPEFFECT_NONE;

    point -= m_dragOffset;

    if ((dwKeyState & (MK_CONTROL|MK_SHIFT)) ==
        (MK_CONTROL|MK_SHIFT))
        de = DROPEFFECT_LINK;
    else if ((dwKeyState & MK_CONTROL) == MK_CONTROL)
        de = DROPEFFECT_COPY;
    else if ((dwKeyState & MK_ALT) == MK_ALT)
        de = DROPEFFECT_MOVE;
    else
        de = DROPEFFECT_MOVE;

    if (point == m_dragPoint)
        return de;

    CClientDC dc(this);
    if (m_prevDropEffect != DROPEFFECT_NONE)
        dc.DrawFocusRect(CRect(m_dragPoint, m_dragSize));
    m_prevDropEffect = de;
    if (m_prevDropEffect != DROPEFFECT_NONE)

```

*Listing 31.10:
Die COCONView:
: OnDragOver-
Elementfunktion*

```

    {
        m_dragPoint = point;
        dc.DrawFocusRect(CRect(point, m_dragSize));
    }
    return de;
}

```

Die Funktion unterstützt ebenfalls das visuelle Feedback. Dazu wird ein Rahmen mit der `CDC::DrawFocusRect`-Funktion gezeichnet.

OnDragLeave `OnDragLeave` (Listing 31.11) markiert das Ende einer Drag-Operation.

Listing 31.11:
Die COCONView:
: OnDragLeave-
Elementfunktion

```

void COCONView::OnDragLeave()
{
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse
    // aufrufen

    // CView::OnDragLeave();
    CClientDC dc(this);
    if (m_prevDropEffect != DROPEFFECT_NONE)
    {
        dc.DrawFocusRect(CRect(m_dragPoint, m_dragSize));
        m_prevDropEffect = DROPEFFECT_NONE;
    }
}

```

OnDrop Richten Sie Ihre Aufmerksamkeit nun auf die `OnDrop`-Elementfunktion (Listing 31.12). `OnDrop` ist die wichtigste Funktion in der Implementierung der Drop-Ziel-Funktionalität. Sie ermöglicht das Einfügen eines abgelegten Objekts.

Listing 31.12:
Die
COCONView:
OnDrop-Ele-
mentfunktion

```

BOOL COCONView::OnDrop(COleDataObject* pDataObject,
                       DROPEFFECT dropEffect, CPoint point)
{
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse

    // return CView::OnDrop(pDataObject, dropEffect, point);
    ASSERT_VALID(this);
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CSize size;

    OnDragLeave();
    CClientDC dc(NULL);
    point -= m_dragOffset;

    pDoc->SetModifiedFlag(TRUE);
    if ((dropEffect & DROPEFFECT_MOVE) && m_bInDrag)
    {
        ASSERT(m_pSelection != NULL);
        m_pSelection->Invalidate(this);
        m_pSelection->m_rect =
            m_dragRect + point - m_dragRect.TopLeft();
        m_bInDrag = FALSE;
        return TRUE;
    }

    COCONCntrItem* pItem = NULL;
    TRY
    {
        pItem = new COCONCntrItem(pDoc);
        ASSERT_VALID(pItem);
    }
}

```

```

        if (dropEffect & DROPEFFECT_LINK)
        {
            if (!pItem->CreateLinkFromData(pDataObject))
                AfxThrowMemoryException();
        }
        else
        {
            if (!pItem->CreateFromData(pDataObject))
                AfxThrowMemoryException();
        }
        ASSERT_VALID(pItem);
        pItem->GetExtent(&size, pItem->GetDrawAspect());
        dc.HIMETRICtoDP(&size);
        pItem->m_rect = CRect(point, size);
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
        m_pSelection = pItem;
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
    }
    CATCH(CException, e)
    {
        if( pItem != NULL ) delete pItem;
        AfxMessageBox(IDP_FAILED_TO_CREATE);
    }
    END_CATCH

    return pItem != NULL;
}

```

Diese Funktion beendet die Drag-Operation, indem sie `OnDragLeave` aufruft. Danach benachrichtigt sie das Dokument mit einem Aufruf von `CDocument::SetModifiedFlag` darüber, daß sich der Inhalt verändert hat.

Im Anschluß daran ermittelt die Funktion, ob das Objekt innerhalb desselben Fensters per Drag&Drop verschoben wurde. Dazu verwendet sie die `m_bInDrag`-Elementvariable. Diese wird in `COCONView::OnLButtonDown` gesetzt, wenn eine Drag-Operation beginnt. Wurde das Objekt verschoben, aktualisiert die Funktion den Objektrahmen und wird beendet.

Bei einer fensterübergreifenden Drop-Operation versucht die Funktion, ein neues Objekt vom Typ `COCONCntrItem` anhand der Drop-Daten zu erstellen. Ist dieser Versuch erfolgreich, wird die Größe des Objekts ermittelt und dessen Rahmen aktualisiert.

Wie bereits erwähnt wurde, wird mit Hilfe der Elementvariable `m_bInDrag` ermittelt, ob eine Drag&Drop-Operation innerhalb desselben Fensters ausgeführt wurde. Um diese Variable setzen zu können, müssen wir eine Änderung an `COCONView::OnLButtonDown` vornehmen. Die endgültige Version dieser Funktion ist in Listing 31.13 aufgeführt.

OnLButtonDown

Listing 31.13:
Finale Version
von COCON-
View::OnLBut-
tonDown

```
void COCONView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Nachrichtensarbeiter hier einfügen

    // CView::OnLButtonDown(nFlags, point);
    COCONDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COCONCntrItem *pSelection = NULL;
    COCONCntrItem *pItem =
        (COCONCntrItem *)pDoc->GetInPlaceActiveItem(this);

    if (pItem != NULL) pItem->Close();
    POSITION pos = pDoc->GetStartPosition();
    while (pos)
    {
        pItem = (COCONCntrItem*)pDoc->GetNextClientItem(pos);
        if (pItem->m_rect.PtInRect(point))
            pSelection = pItem;
    }
    if (pSelection != m_pSelection)
    {
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
        m_pSelection = pSelection;
        if (m_pSelection != NULL)
            m_pSelection->Invalidate(this);
    }

    if (m_pSelection != NULL)
    {
        m_bInDrag = TRUE;
        m_dragRect = m_pSelection->m_rect;
        DROPEFFECT dropEffect =
            m_pSelection->DoDragDrop(m_pSelection->m_rect,
                (CPoint)(point -
                    m_pSelection->m_rect.TopLeft()));
        m_pSelection->Invalidate(this);
        if (m_bInDrag == TRUE && dropEffect == DROPEFFECT_MOVE)
        {
            delete m_pSelection;
            m_pSelection = NULL;
        }
        m_bInDrag = FALSE;
    }
}
```

Die `m_bInDrag`-Elementvariable wird auf `TRUE` gesetzt, bevor `COCONCntrItem::DoDragDrop` aufgerufen wird. Wird während des Ziehens des Objekts ein Rückruf an das Ansichtsobjekt vorgenommen, können wir feststellen, daß die Quelle der Drag-Operation dasselbe Ansichtsfenster ist. Dazu überprüfen wir den Wert von `m_bInDrag`.

Die `OnDrop`-Elementfunktion setzt `m_bInDrag` auf `FALSE` zurück, wenn ein Objekt lediglich innerhalb eines Ansichtsfensters verschoben wird. In diesem Fall löscht `OnLButtonDown` das selektierte Objekt nicht. Das Entfernen des Objekts ist nur dann erforderlich, wenn dieses in ein anderes Fenster verschoben wird.

```
COCONView::COCONView()
{
    m_pSelection = NULL;
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen,
    m_bInDrag = FALSE;
    m_prevDropEffect = DROPEFFECT_NONE;
}
```

*Listing 31.14:
Die Initialisierung der Elementvariablen in dem Konstruktor von COCONView*



*Abbildung 31.2:
Die OCON-Anwendung*

31.4 Zusammenfassung

OLE stellt eine umfassende Drag&Drop-Unterstützung zur Verfügung. Diese ist in Form von Drag&Drop-Klassen und Funktionen in der MFC-Bibliothek enthalten.

Die Implementierung einer Drag-Quelle ist relativ einfach. Verwenden Sie dazu die Elementfunktionen von `COleClientItem`, `COleServerItem` und `COleDataSource`.

Der Einsatz von `COleClientItem::DoDragDrop` ist empfehlenswert, wenn das Drag-Objekt ein eingebettetes oder verknüpftes OLE-Objekt ist, das durch eine von `COleClientItem` abgeleitete Klasse repräsentiert wird. Rufen Sie in diesem Fall `DoDragDrop` für das Objekt auf, das der Anwender für den Drag-Vorgang selektiert hat. Der Anwendungsrahmen übernimmt den Rest. Denken Sie daran, das Objekt zu löschen, wenn der Rückgabewert von `DoDragDrop` anzeigt, daß die Selektion in eine andere Anwendung verschoben (und nicht kopiert oder mit dieser verknüpft) wurde.

`COleServerItem::DoDragDrop` sollte für Anwendungen verwendet werden, die OLE-Server sind. Nutzen Sie diese Funktion, wenn Ihre von `COleServerItem` abgeleitete Klasse eine Selektion repräsentieren kann (und nicht den gesamten Inhalt eines Dokuments). Erstellen Sie ein von `COleServerItem` abgeleitetes Objekt, das die Selektion repräsentiert, und verwenden Sie die `DoDragDrop`-Elementfunktion, um die Drag-Operation auszuführen.

In Anwendungen, die keine OLE-Server sind, können Sie mit `COleDataSource` eine Drag-Quelle implementieren.

Das Erstellen eines Drop-Ziels ist ein komplexer Vorgang. Zusätzlich zu der überschriebenen Version der `OnDrop`-Elementfunktion, müssen Sie die Elementfunktionen `OnDragEnter`, `OnDragLeave` und `OnDragOver` überschreiben. Diese Funktionen werden von dem Anwendungsrahmen aufgerufen, wenn sich die Maus während der Drag&Drop-Operation über dem Ansichtsfenster befindet. Sie generieren das visuelle Feedback während des Drag-Vorgangs und informieren den Anwendungsrahmen über die möglichen Drop-Operationen.

In einer einfachen Implementierung von `OnDrop` können Sie ein von `COleClientItem` abgeleitetes Objekt erstellen, das das Drop-Objekt repräsentiert. In komplexen Anwendungen überprüfen Sie das Objekt und unterscheiden die nativen Daten Ihrer Anwendung von den Daten, die in Formaten vorliegen, die von Ihrer Anwendung bearbeitet werden können.

In einer Anwendung, die sowohl Drag-Quelle als auch Drop-Ziel ist, sollten Sie die Effizienz der Anwendung verbessern, indem Sie das Verschieben eines Objekts innerhalb desselben Fensters berücksichtigen. Anstatt das Drag-Objekt zu löschen und ein neues Objekt zu erstellen, können Sie Anweisungen implementieren, die lediglich die Position des Objekts verändern.

Automatisierung

Kapitel

32

Automatisierung (bisher OLE-Automatisierung) ist ein Mechanismus zur Kommunikation zwischen kooperierenden Anwendungen. Anwendungen, die Automatisierungs-Server sind, erstellen ihre Objekte mit EIGENSCHAFTEN und METHODEN. Automatisierungs-Clients, die auch als Automatisierungsg-Controller bezeichnet werden, greifen über die OLE-Schnittstelle `IDispatch` auf diese Eigenschaften und Methoden zu.

Die Automatisierung wird gewöhnlich verwendet, um eine Anwendung derart zu erweitern, daß sie von einem Automatisierungs-Client, wie zum Beispiel Microsoft-Visual-Basic, gesteuert werden kann. Auf diese Weise kann Visual Basic für eine Anwendung als leistungsfähige Makrosprache agieren.

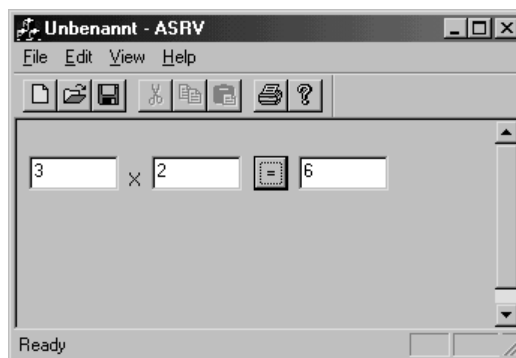
Die MFC-Bibliothek und das Visual-C++-Developer-Studio unterstützen die Entwicklung von Automatisierungs-Servern. In diesem Kapitel werden wir zunächst einen einfachen Server erstellen. Später besprechen wir weitere Themen zur Automatisierungs-Entwicklung.

32.1 Erstellen eines Automatisierungs-Servers

Der Automatisierungs-Server, den wir in diesem Kapitel erstellen, führt lediglich eine Aufgabe aus. Er multipliziert zwei Werte. Das Programm wird selbständig ausgeführt und verwendet einen einfachen Dialog als Benutzeroberfläche. Außerdem kann es aus einem Automatisierungs-Controller aufgerufen werden.

Abbildung 32.1 zeigt die Benutzeroberfläche der Anwendung, die wir ASRV nennen. Sie verwenden dieses Programm, indem Sie zunächst die beiden zu multiplizierenden Werte in die Textfelder auf der linken Seite eingeben und anschließend die Schaltfläche = betätigen, um das Ergebnis berechnen zu lassen.

Abbildung 32.1:
Die Benutzeroberfläche des
ASRV-Server



Über die Methoden und Eigenschaften der Automatisierungs-Anwendung werden diese Benutzeroberflächenelemente gesteuert.

- Die Eigenschaften `Multi1` und `Multi2` entsprechen den beiden Multiplikatoren. Diese Eigenschaften können gesetzt und ermittelt werden.
- Die `Result`-Eigenschaft repräsentiert das Ergebnis der Multiplikation und kann lediglich ausgelesen werden.
- Die `Set`-Methode korrespondiert mit der Funktion der Schaltfläche `=`. Sie führt die eigentliche Multiplikation durch.

Wir erstellen zunächst das Anwendungsgerüst der Anwendung. Anschließend bearbeiten wir die Benutzeroberfläche und schreiben die Funktionen, die die Berechnungen ausführen. Schließlich implementieren wir die Automatisierung.

32.1.1 Erstellen des ASRV-Anwendungsgerüsts

1. Das Anwendungsgerüst der ASRV-Anwendung wird mit dem Anwendungsassistenten generiert. Die Anwendung sollte ein einzelnes Dokument (SDI) verwenden.
2. Die Unterstützung der Automatisierung wählen Sie im dritten Schritt des Anwendungsassistenten aus (Abbildung 32.2). Selektieren Sie dort das Optionsfeld AUTOMATISIERUNG.

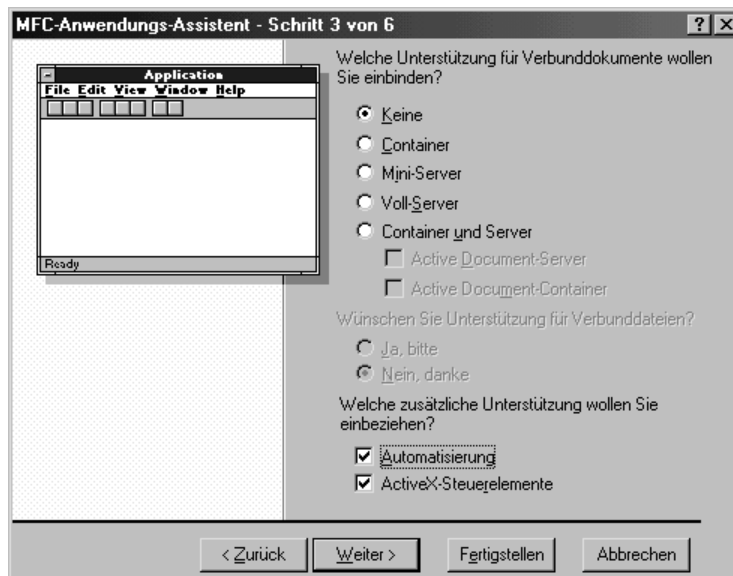


Abbildung 32.2:
Auswahl der
Automatisie-
rung

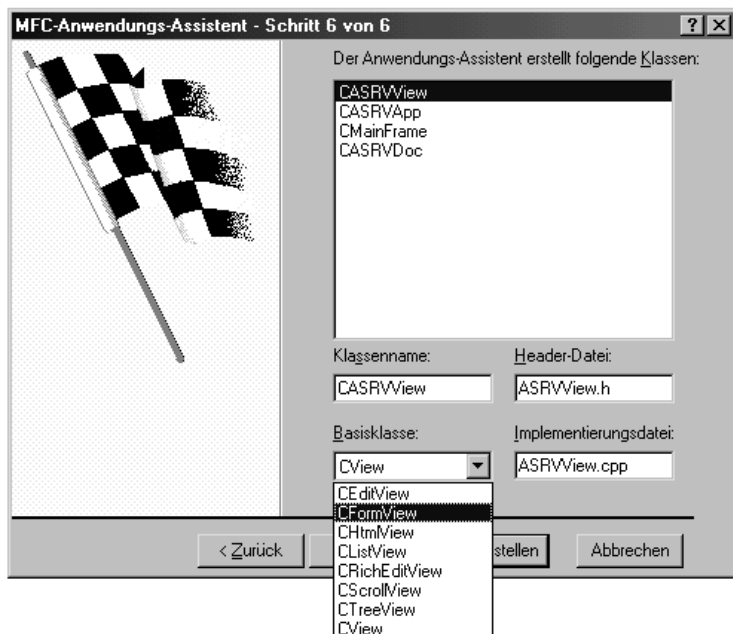


Abbildung 32.3:
Ändern der
Basisklasse der
Ansicht

3. Ändern Sie außerdem die Basisklasse der neuen Anwendungsansichtsklasse. Wählen Sie dazu im sechsten Schritt des Assistenten die Klasse `CASRVView` aus dem großen Listenfeld aus, und selektieren Sie anschließend unter Basisklasse den Eintrag `CFormView`. Als eine von `CFormView` abgeleitete Klasse verwendet unsere neue Ansichtsklasse eine Dialogvorlage für die visuelle Benutzeroberfläche.
4. Betätigen Sie die Schaltfläche FERTIGSTELLEN, um das Anwendungsgerüst erzeugen zu lassen.

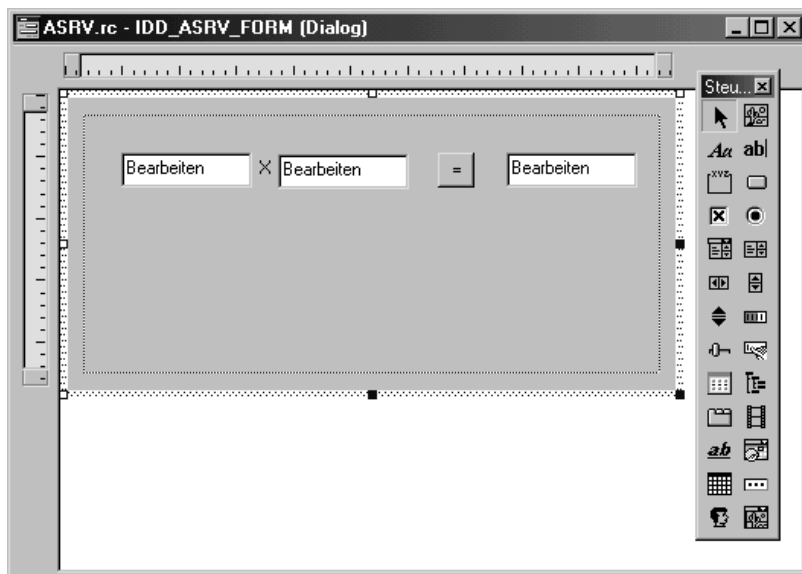
32.1.2 Implementierung der Berechnung

Bevor Sie den Programmcode für die Berechnung schreiben, erstellen Sie zunächst die Benutzeroberfläche. Der Anwendungsassistent hat einen Dialog generiert, der auf der von `CFormView` abgeleiteten Klasse basiert. Dieser Dialog enthält noch keine Elemente (Abbildung 32.4).

Dialog zusammenstellen

1. Bearbeiten Sie ihn, indem Sie zunächst das TODO-Standardfeld entfernen und drei Eingabefelder, eine Schaltfläche und ein statisches Textfeld anordnen, wie in Abbildung 32.4 angezeigt.

Abbildung 32.4:
Der Dialog



- Die beiden Eingabefelder auf der linken Seite sollen die Namen IDC_MULTI1 und IDC_MULTI2 erhalten.
- Nennen Sie das Eingabefeld auf der rechten Seite IDC_RESULT. Aktivieren Sie den Schreibschutz für dieses Steuerelement.
- Beschriften Sie die Schaltfläche mit einem Gleichheitszeichen.
- Weisen Sie der Schaltfläche die Bezeichnung IDC_CALCULATE zu. Aktivieren Sie die Option STANDARDSCHALTFLÄCHE im Register FORMATE, um mit Hilfe der Taste (=) die Berechnung ausführen zu können.

Berechnung ausführen

- Rufen Sie den Klassen-Assistenten auf, während der Dialog zur Bearbeitung geöffnet ist. Fügen Sie der Ansichtsklasse mit Hilfe des Assistenten eine Elementfunktion hinzu, die einen Mausklick auf die Schaltfläche IDC_CALCULATE bearbeitet (Abbildung 32.5).

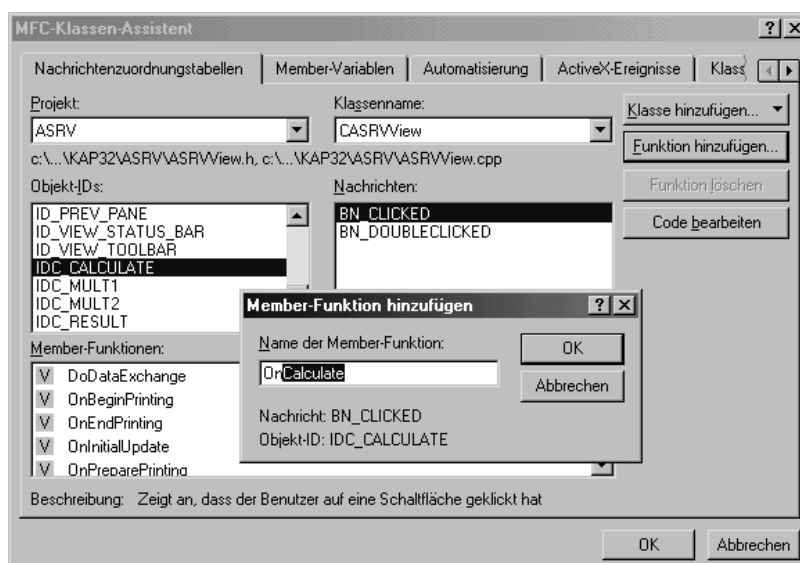


Abbildung 32.5:
Hinzufügen
einer Element-
funktion, die
das Betätigen
der Schaltfläche
bearbeitet

Sie erwarten nun wahrscheinlich, daß für die Eingabefelder Elementvariablen definiert werden müssen. Die Variablen, die die Multiplikatoren und das Ergebnis repräsentieren, werden jedoch der Dokumentklasse hinzugefügt. Wir müssen daher die DoDataExchange-Elementfunktion der Ansichtsklasse manuell modifizieren.

- Schließen Sie bitte den Klassen-Assistenten, und öffnen Sie die Header-Datei der Dokumentklasse.

3. Geben Sie in dem Attribute-Abschnitt der `CASRVDoc`-Klassendeklaration die Deklarationen der drei neuen Elementvariablen wie folgt ein:

```
class CASRVDoc : public CDocument
{
protected: // Nur aus Serialisierung erzeugen
    CASRVDoc();
    DECLARE_DYNCREATE(CASRVDoc)

// Attribute
public:
    long m_lMulti1;
    long m_lMulti2;
    long m_lResult;
    ...
}
```

4. Die Variablen müssen in dem Konstruktor initialisiert werden (Listing 32.1).

Listing 32.1:
Variablen-initialisierung in dem `CASRVDoc`-Konstruktor

```
CASRVDoc::CASRVDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion einfügen
    m_lMulti1 = 0;
    m_lMulti2 = 0;
    m_lResult = 0;

    EnableAutomatisierung();

    AfxOleLockApp();
}
```

5. Um die Berechnung ausführen zu können, benötigen wir eine neue Elementfunktion in `CASRVDoc`. Die Elementfunktion `DoCalculate` wird wie folgt deklariert:

```
class CASRVDoc : public CDocument
{
    ...
// Operationen
public:
    void DoCalculate();
    ...
}
```

Die Implementierung von `DoCalculate` ist in Listing 32.2 aufgeführt.

Listing 32.2:
Implementierung von `CASRVDoc::DoCalculate`

```
void CASRVDoc::DoCalculate()
{
    m_lResult = m_lMulti1 * m_lMulti2;
}
```

Zwei Aufgaben stehen uns noch bevor. Zunächst müssen die Elementvariablen in `CASRVDoc` mit dem auf `CFormView` basierenden Dialog verbunden werden. Schließlich rufen wir `CASRVDoc::DoCalculate` aus der Bearbeiterfunktion für die Betätigung der Schaltfläche `IDC_CALCULATE` auf.

6. Die modifizierte Version von `CASRVView::DoDataExchange` ist in Listing 32.3 aufgeführt. Wir ändern die Funktion derart, wie sie der Klassen-Assistent modifiziert hätte, wenn die Variablen lokal für diese Klasse deklariert worden wären.

```
void CASRVView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CASRVView)
    //}}AFX_DATA_MAP
    CASRVDoc *pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    DDX_Text(pDX, IDC_MULT11, pDoc->m_lMulti1);
    DDX_Text(pDX, IDC_MULT12, pDoc->m_lMulti2);
    DDX_Text(pDX, IDC_RESULT, pDoc->m_lResult);
}
```

*Listing 32.3:
Die modifizierte
Version von
CASRVDoc::
Do-Data-
Exchange*

7. In der Implementierung von `CASRVView::OnCalculate` (Listing 32.4) rufen wir die Elementfunktion `DoCalculate` der Dokumentklasse auf, um die eigentliche Berechnung vorzunehmen. Außerdem verwenden wir die `UpdateData`-Funktion, um die Elementvariablen und Dialogfelder vor und nach der Berechnung korrekt zu aktualisieren.

```
void CASRVView::OnCalculate()
{
    // TODO: Code für die Behandlungsroutine der
    // Steuerelement-Benachrichtigung hier einfügen
    CASRVDoc *pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    UpdateData(TRUE);
    pDoc->DoCalculate();
    UpdateData(FALSE);
}
```

*Listing 32.4:
Implementie-
rung von
CASRVView::
OnCalculate*

Die Anwendung kann nun kompiliert und als selbständige Anwendung ausgeführt werden.

32.1.3 Hinzufügen der Automatisierungs-Unterstützung

Zum Hinzufügen der Automatisierungs-Unterstützung verwenden Sie zunächst den Klassen-Assistenten.

1. Rufen Sie den Klassen-Assistenten auf, um Eigenschaften und Methoden für die `CASRVDoc`-Klasse zu definieren.
2. Selektieren Sie in dem Dialog des Assistenten das Register AUTOMATISIERUNG und dort die Klasse `CASRVDoc` (Abbildung 32.6).
3. Fügen Sie der Klasse zunächst zwei Eigenschaften hinzu, über die Sie die beiden Multiplikatoren steuern. Betätigen Sie dazu die Schaltfläche EIGENSCHAFT HINZUFÜGEN, um den gleichlautenden Dialog aufzurufen. Bestimmen Sie in diesem Dialog den externen Namen `Multi1` für die neue Eigenschaft und `long` als Typ.

Abbildung 32.6:
Automatisie-
rungs-Unter-
stützung mit
Hilfe des Klas-
sen-Assistenten

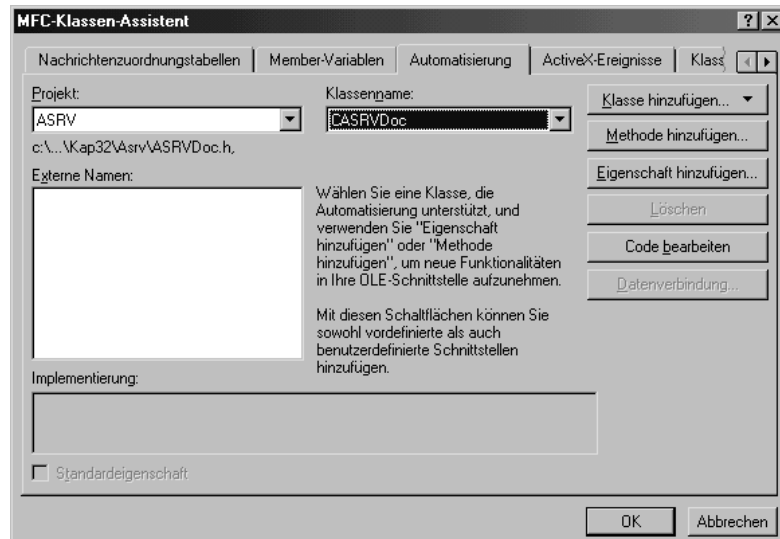
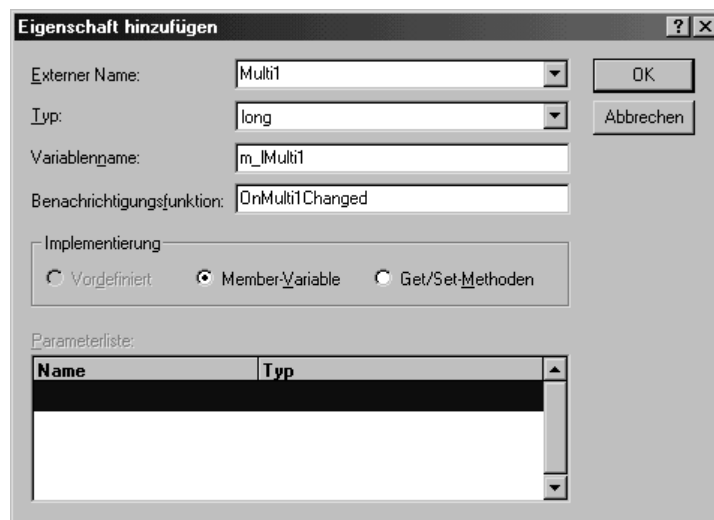


Abbildung 32.7:
Hinzufügen
einer neuen
Eigenschaft



Ändern Sie den vom Klassen-Assistenten vorgeschlagenen Variablennamen in `m_Multi1` (Abbildung 32.7). Betätigen Sie die Schaltfläche OK, um den Dialog zu verlassen.

Fügen Sie der Klasse die zweite Eigenschaft mit der Bezeichnung `Multi2` hinzu.

Eigenschaft hinzufügen

Externer Name: Result

Typ: long

Get-Funktion: GetResult

Set-Funktion:

Implementierung:

☐ Vordefiniert ☐ Member-Variable ☒ Get/Set-Methoden

Parameterliste:

Name	Typ

Abbildung 32.8:
Hinzufügen
einer schreibge-
schützten Eigen-
schaft

Die dritte Eigenschaft, `Result`, unterscheidet sich von den letzten beiden Eigenschaften. Diese Eigenschaft ist schreibgeschützt. Wir implementieren die Eigenschaft mit Hilfe der Get-/Set-Methoden.

4. Löschen Sie im Dialogfeld EIGENSCHAFT HINZUFÜGEN – nachdem Sie die Option GET-/SET-METHODEN aktiviert haben – die vom Klassen-Assistenten generierte Bezeichnung der Set-Methode. Wir verwenden lediglich die Get-Methode, um der Eigenschaft einen Nur-Lese-Status zuzuweisen (Abbildung 32.8).

Methode hinzufügen

Externer Name: Calculate

Interner Name: Calculate

Rückgabebetyp: void

Implementierung:

☐ Vordefiniert ☒ Benutzerdefiniert

Parameterliste:

Name	Typ

Abbildung 32.9:
Hinzufügen
einer neuen
Methode

5. Fügen Sie der Klasse bitte noch die Methode `Calculate` hinzu, die die Multiplikation ausführt. Betätigen Sie dazu die Schaltfläche `METHODE HINZUFÜGEN`. Geben Sie in dem daraufhin angezeigten Dialog den externen und internen Namen `Calculate` an, und selektieren Sie `void` als Rückgabebetyp der Methode (Abbildung 32.9).
6. Der Klassen-Assistent kann nun mit einem Klick auf die Schaltfläche `OK` geschlossen werden.

Betrachten Sie kurz die Header-Datei von `CASRVDoc`. Am Ende der Klassendeklarationen hat der Klassen-Assistent der `DISPATCH`-Tabelle einige Elemente hinzugefügt:

```
class CASRVDoc : public CDocument
{
    ...
protected:
    ...
    // Generierte OLE-Dispatch-Map-Funktionen
    //({AFX_DISPATCH(CASRVDoc)
    long m_lMulti1;
    afx_msg void OnMulti1Changed();
    long m_lMulti2;
    afx_msg void OnMulti2Changed();
    afx_msg long GetResult();
    afx_msg void Calculate();
    //({AFX_DISPATCH
    DECLARE_DISPATCH_MAP()
    DECLARE_INTERFACE_MAP()
};
```

7. Wie Sie sehen, hat der Klassen-Assistent Deklarationen für die Elementvariablen `m_lMulti1` und `m_lMulti2` generiert. Die zuvor von uns vorgenommenen Deklarationen werden daher nicht mehr benötigt und sollten aus dem `Attribute`-Abschnitt gelöscht werden.
8. Der Klassen-Assistent hat außerdem die Methode `Calculate` erzeugt. `DoCalculate` erfüllt somit keine besondere Funktion mehr und muß gelöscht werden.
9. Beachten Sie bitte, daß die Variablen `m_lMulti1` und `m_lMulti2` in dem `protected`-Abschnitt der Klassendeklaration enthalten sind. Wir greifen jedoch aus der Funktion `CASRVView::DoDataExchange` auf diese Variablen zu. Sie kompilieren das Programm daher bestmöglich, indem Sie `CASRVView` als `friend`-Klasse von `CASRVDoc` deklarieren.

Die endgültige Version der `CASRVDoc`-Klassendeklaration ist aufgrund unserer umfangreichen Änderungen vollständig in Listing 32.5 aufgeführt.

```

class CASRVDoc : public CDocument
{
friend class CASRVView;

protected: // Nur aus Serialisierung erzeugen
    CASRVDoc();
    DECLARE_DYNCREATE(CASRVDoc)

// Attribute
public:
    long m_lResult;

// Operationen
public:

// Überladungen
// Vom Klassenassistenten generierte Überladungen virtueller
// Funktionen
//{{AFX_VIRTUAL(CASRVDoc)
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
//}}AFX_VIRTUAL

// Implementierung
public:
    virtual ~CASRVDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generierte Message-Map-Funktionen
protected:
    //{AFX_MSG(CASRVDoc)
    // HINWEIS - An dieser Stelle werden Member-Funktionen vom
    // Klassen-Assistenten eingefügt und entfernt.
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    //{AFX_MSG
    DECLARE_MESSAGE_MAP()

    // Generierte OLE-Dispatch-Map-Funktionen
    //{AFX_DISPATCH(CASRVDoc)
    long m_lMulti1;
    afx_msg void OnMulti1Changed();
    long m_lMulti2;
    afx_msg void OnMulti2Changed();
    afx_msg long GetResult();
    afx_msg void Calculate();
    //{AFX_DISPATCH
    DECLARE_DISPATCH_MAP()
    DECLARE_INTERFACE_MAP()
};

```

*Listing 32.5:
CASRVDoc-Klas-
sendeclaration
mit Auto-
matisierung-
Unterstützung*

Der Klassen-Assistent hat der Implementierung von CASRVDoc zwei Nachrichtenspeicher und zwei Methoden hinzugefügt.

10. Nachdem Sie die `DoCalculate`-Elementfunktion aus der Implementierungsdatei entfernt haben, modifizieren Sie nun `CASRVDoc::GetResult` und `CASRVDoc::Calculate`, wie in Listing 32.6 dargestellt. Der Nachrichtensarbeiter muß nicht verändert werden.

Listing 32.6:
CASRVDoc::Get-
Result und CAS-
RVDoc::Cal-
culate

```
long CASRVDoc::GetResult()
{
    // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
    // Eigenschaften hinzu

    return m_lResult;
}

void CASRVDoc::Calculate()
{
    // ZU ERLEDIGEN: Fügen Sie hier den Code für Ihre
    // Dispatch-Behandlungsroutine ein
    m_lResult = m_lMulti1 * m_lMulti2;
}
```

11. Bevor wir die Anwendung erneut kompilieren, müssen wir noch eine Aufgabe ausführen. In der Funktion `CASRVView::OnCalculate` wurde ein nicht mehr benötigter Verweis auf die `DoCalculate`-Elementfunktion der Dokumentklasse angelegt. Ändern Sie diesen Verweis auf die `Calculate`-Funktion.

12. Kompilieren Sie das Programm anschließend erneut.

32.1.4 Die Typenbibliothek

Betrachten Sie zunächst eine der vom Klassen-Assistenten generierten Dateien, bevor Sie die Anwendung testen. Die Datei `ASRV.ODL` (Listing 32.7) ist eine Skript-Datei, die in der MICROSOFT-OBJEKTDEFINITIONSSPRACHE (MODL) erstellt wurde. Sie wird mit dem MIDL-Compiler kompiliert. Die daraus resultierende Typenbibliotheksdatei (TLB) ist ein Verbunddokument, auf das Automatisierungs-Clients zugreifen können. Die Clients verwenden dazu die COM-Schnittstellen `ITypeComp`, `ITypeInfo` und `ITypeLib`.

Listing 32.7:
Die Typen-
bibliotheksdatei
ASRV.odl

```
// ASRV.odl : Quellcode der Typbibliothek für ASRV.exe
// Diese Datei wird vom MIDL-Compiler bearbeitet, um die
// Typbibliothek zu erzeugen (ASRV.tlb).

[ uuid(80185F62-EC89-11D0-8728-444553540000), version(1.0) ]
library ASRV
{
    importlib("stdole32.tlb");

    // Primäre Dispatch-Schnittstelle für CASRVDoc

    [ uuid(80185F63-EC89-11D0-8728-444553540000) ]
    dispinterface IASRV
    {
        properties:
            // HINWEIS - Der Klassen-Assistent verwaltet hier
```

```

// Informationen über die Eigenschaften.
// Lassen Sie äußerste Vorsicht walten, falls Sie
// diesen Abschnitt bearbeiten.
//{{AFX_ODL_PROP(CASRVDoc)
[id(1)] long Multi1;
[id(2)] long Multi2;
[id(3)] long Result;
//}}AFX_ODL_PROP

methods:
// HINWEIS - Der Klassen-Assistent verwaltet hier
// Informationen über die Methoden.
// Lassen Sie äußerste Vorsicht walten, falls Sie
// diesen Abschnitt bearbeiten.
//{{AFX_ODL_METHOD(CASRVDoc)
[id(4)] void Calculate();
//}}AFX_ODL_METHOD

};

// Klassen-Informationen für CASRVDoc

[ uuid(80185F61-EC89-11D0-8728-444553540000) ]
coclass Document
{
    [default] dispinterface IASRV;
};

//{{AFX_APPEND_ODL}}
//}}AFX_APPEND_ODL}}
};

```

Die Schnittstellen `ICreateTypeInfo` und `ICreateTypeLib` werden von Hilfsmitteln, wie zum Beispiel dem MIDL-Compiler, verwendet.

32.1.5 Testen der Anwendung

Automatisierungs-Server werden mit einem Automatisierungs-Controller getestet. Haben Sie Visual Basic auf Ihrem System installiert, können Sie damit die Anwendung überprüfen. (Sowohl die 16-Bit-Version als auch die 32-Bit-Version arbeiten mit 32-Bit-Automatisierungs-Servern zusammen, die unter Visual C++ erstellt wurden.)

Um die Funktionalität des ASRV-Servers zu prüfen, habe ich das in Abbildung 32.10 dargestellte Formular generiert.

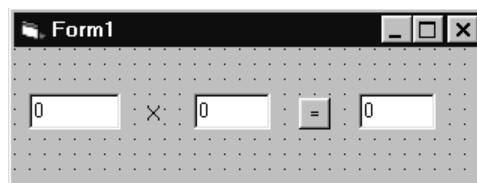


Abbildung 32.10:
Das Visual-
Basic-Formular
zum Testen von
ASRV

Bezeichnen Sie die drei Eingabefelder des Formulars mit `Multi1`, `Multi2` und `Result`. Nennen Sie die Schaltfläche mit dem Gleichheitszeichen `Calculate`.

Um den Test ausführen zu können, muß dem Formular der in Listing 32.8 aufgeführte Programmcode zugewiesen werden. Die Deklaration (Dim-Anweisung) von ASRV wird in dem allgemeinen Abschnitt vorgenommen. Die `Form_Load`-Routine ist dem `Load`-Ereignis des `Form`-Objekts zugeteilt. Die `Calculate_Click`-Unterroutine wird ausgeführt, wenn das `Click`-Ereignis der `Calculate`-Schaltfläche ausgelöst wird. Diese Art der Implementierung unserer Testanwendung führt dazu, daß wir nicht immer dann den ASRV-Server aufrufen müssen, wenn eine Berechnung durchgeführt wird.

Listing 32.8:
Visual-Basic-Programcode zum
Testen von ASRV

```
Dim ASRV As object

Sub Form_Load ()
    Set ASRV = CreateObject("ASRV.Document")
End Sub

Sub Calculate_Click ()
    ASRV.Multi1 = Val(Multi1.Text)
    ASRV.Multi2 = Val(Multi2.Text)
    ASRV.Calculate
    Result.Text = Str(ASRV.Result)
End Sub
```

Beachten Sie bitte, daß die ASRV-Anwendung während des Tests nicht angezeigt wird. Der Grund hierfür besteht darin, daß Visual Basic die Anwendung mit dem Kommandozeilenparameter `/Automatisierung` aufruft. Die vom Anwendungsassistenten generierte Anwendung zeigt daraufhin nicht ihr Hauptfenster an. Wäre ASRV eine Anwendung, die mehrere Dokumente verwalten kann (MDI), würde das Hauptfenster angezeigt. Ein neues Rahmenfenster würde jedoch nicht zur Repräsentation des neuen Dokumentfensters dargestellt. Denken Sie also bitte daran, daß einem aufgerufenen Automatisierungs-Dokument kein Ansichtsfenster zugewiesen ist.

Die Funktionalität und der Aufbau der Visual-Basic-Anwendung gleicht denen der originalen Benutzeroberfläche von ASRV. Der Unterschied besteht darin, daß ASRV als ein Black-Box-Modul ausgeführt wird. Diese einfache Server-Anwendung demonstriert nicht nur die Leistungsfähigkeit der Automatisierung, sondern ebenfalls das Potential von Automatisierungs-Clients, wie zum Beispiel Visual Basic.

32.2 Standardmethoden und Standardeigenschaften

Ich habe für den ASRV-Server einige einfache Methoden und Eigenschaften verwendet, um das Beispielprogramm leichtverständlich darzustellen. Microsoft empfiehlt jedoch die Berücksichtigung einiger Bedingungen während der Implementierung der Eigenschaften und Methoden:

- Jede Anwendung sollte mindestens ein Anwendungsobjekt mit den Standardmethoden und Standardeigenschaften verwenden.
- Die Programme sollten außerdem Dokumentauflistungen, Dokumentobjekte und Objektauflistungen verwenden, sofern möglich.

Automatisierung von Objekten

Diese Empfehlungen führen uns zu einer Besonderheit, die wir während der Erstellung von ASRV nicht berücksichtigen konnten. Die einfachen Eigenschaften der Anwendung sind Integer vom Typ `Long`. Das Verwenden von Zeichenfolgen-Eigenschaften ist ebenfalls relativ einfach. Doch was geschieht, wenn der Wert einer Eigenschaft oder das Ergebnis der Methode ein anderes Objekt ist? Dies ist beispielsweise der Fall, wenn ein Automatisierungs-Client ein bestimmtes Objekt von einer Dokumentauflistung anfordert. Wie wird das Dokumentobjekt dem Client übergeben?

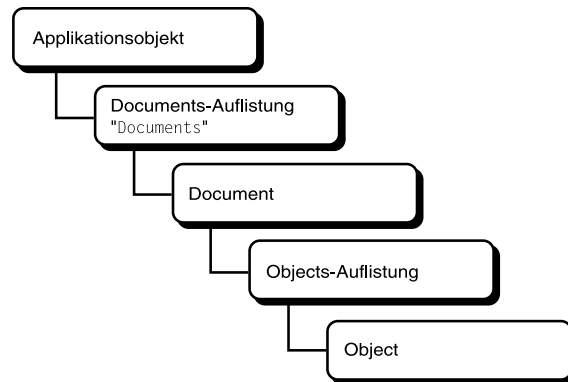
Die Antwort auf diese Frage ist einfach, jedoch nicht offensichtlich. Eine Methode, die ein anderes Objekt zurückgibt, sollte einen Zeiger auf eine *IDispatch*-Schnittstelle zurückgeben. Ein derartiger Zeiger kann für jedes von `CCmdTarget` abgeleitete Objekt ermittelt werden. Rufen Sie dazu die `CCmdTarget`-Elementfunktion `GetIDispatch` auf. Verfügt Ihr Anwendungsobjekt beispielsweise über eine Methode, die das aktive Dokument zurückgibt, könnte diese Methode wie folgt implementiert sein:

```
LPDISPATCH GetActiveDocument()
{
    CFrameWnd *pWnd = (CFrameWnd *)AfxGetMainWnd();
    ASSERT_VALID(pWnd);
    CMyDocument *pDoc = pWnd->GetActiveDocument();
    ASSERT_VALID(pDoc);
    return pDoc->GetIDispatch(TRUE);
}
```

Die Standardobjekte

Die Hierarchie der Standardobjekte, die Automatisierungs-Server unterstützen sollten, ist in Abbildung 32.11 dargestellt.

Abbildung 32.11:
Automatisie-
rungs- Stan-
dardobjekte



Alle Objekte unterstützen zwei Standardeigenschaften, die in Tabelle 32.1 aufgeführt sind.

Tabelle 32.1:
Die Eigenschaf-
ten der Stan-
dardobjekte

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
E: Application	Ja	Nur Lesen	Das Anwendungs- objekt.
E: Parent	Ja	Nur Lesen	Parent des aktuel- len Objekts.

Alle Auflistungsobjekte (beispielsweise die Dokumentauflistung) unterstützen zusätzliche Eigenschaften und Methoden, die in Tabelle 32.2 aufgeführt sind.

Tabelle 32.2:
Eigenschaften
und Methoden
der Auflistungs-
objekte

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
E: Count	Ja	Nur Lesen	Anzahl der Objekte in der Auflistung.
E: _NewEnum	Ja	Nur Lesen	Aufzähler für Schleife.
M: Add	Nein		Fügt der Auflis- tung ein Objekt hinzu.
M: Item	Ja		Ermittelt ein Auflis- tungsobjekt.
M: Remove	Nein		Entfernt ein Auflis- tungsobjekt.

Das Anwendungsobjekt

Das Anwendungsobjekt repräsentiert die Anwendung. Dieses Objekt sollte die in Tabelle 32.3 aufgeführten Eigenschaften und Methoden unterstützen.

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
E: ActiveDocument	Nein	Nur Lesen	Das aktive Dokumentobjekt.
E: Caption	Nein	Lesen/ Schreiben	Fenstertitel der Anwendung.
E: DefaultFilePath	Nein	Lesen/ Schreiben	Standard-Verzeichnispfad.
E: Documents	Nein	Nur Lesen	Die Dokumentauflistung.
E: FullName	Ja	Nur Lesen	Vollständiger Dateiname der ausführbaren Datei.
E: Height	Nein	Lesen/ Schreiben	Höhe des Anwendungsfensters.
E: Interactive	Nein	Lesen/ Schreiben	Anwenderinteraktion.
E: Left	Nein	Lesen/ Schreiben	Linke Seite des Anwendungsfensters.
E: Name	Ja	Nur Lesen	Name der Anwendung.
E: Path	Nein	Nur Lesen	Verzeichnis der ausführbaren Datei.
E: StatusBar	Nein	Lesen/ Schreiben	Statusleistertext.
E: Top	Nein	Lesen/ Schreiben	Oberer Rand des Anwendungsfensters.
E: Visible	Ja	Lesen/ Schreiben	Anwendungsfenster anzeigen.
E: Width	Nein	Lesen/ Schreiben	Breite des Anwendungsfensters.
M: Help	Nein		Online-Hilfe anzeigen.
M: Quit	Ja		Anwendung beenden.

*Tabelle 32.3:
Eigenschaften
und Methoden
des Auflistungs-
objekts*

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
M: Repeat	Nein		Letzte Aktion wiederholen.
M: Undo	Nein		Letzte Aktion rückgängig machen.

Die Dokumentauflistung

Die Dokumentauflistung nimmt die gegenwärtig aktiven Dokumente der Anwendung auf. Dieses Auflistungsobjekt sollte, zusätzlich zu den allgemeinen Objekteigenschaften und -methoden, zwei weitere Methoden aufnehmen, die in Tabelle 32.4 aufgeführt sind.

Tabelle 32.4:
Methoden der
Dokumentauflistung

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
M: Close	Ja		Schließt ein Dokument.
M: Open	Ja		Öffnet ein neues Dokument.

Das Dokumentobjekt

Das Dokumentobjekt repräsentiert ein einzelnes Dokument. Zusätzlich zu den Standardeigenschaften und -methoden sollte dieses Objekt die in Tabelle 32.5 aufgeführten Eigenschaften und Methoden unterstützen.

Tabelle 32.5:
Eigenschaften
und Methoden
des Dokument-
objekts

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
E: Author	Nein	Lesen/ Schreiben	Autor des Dokuments.
E: Comments	Nein	Lesen/ Schreiben	Dokumentkommentare.
E: FullName	Ja	Nur Lesen	Vollständiger Dateiname des Dokuments.
E: Keywords	Nein	Lesen/ Schreiben	Dokument-Schlüsselwörter.
E: Name	Nein	Nur Lesen	Dokumentdateiname.

Eigenschaft/Methode	Optional	Lesen/ Schreiben	Beschreibung
E: Path	Ja	Nur Lesen	Dokumentverzeichnis.
E: ReadOnly	Nein	Nur Lesen	Schreibgeschütztes Dokument.
E: Saved	Ja	Lesen/ Schreiben	Änderungen wurden gespeichert.
E: Subject	Nein	Lesen/ Schreiben	Thema des Dokuments.
E: Title	Nein	Lesen/ Schreiben	Dokumenttitel.
M: Activate	Ja		Aktiviert erstes Dokumentfenster.
M: Close	Ja		Schließt das Dokument.
M: NewWindow	Nein		Erstellt neues Fenster für das Dokument.
M: Print	Ja		Druckt das Dokument.
M: PrintOut	Nein		Siehe Print.
M: PrintPreview	Nein		Aktiviert die Druckvorschau.
M: RevertToSaved	Nein		Lädt die gespeicherte Version erneut.
M: Save	Ja		Speichert das Dokument.
M: SaveAs	Ja		Speichert das Dokument unter einem neuen Dateinamen.

Die Objektauflistung

Die Objektauflistung wird von Dokumentobjekten angeboten, sofern möglich. Diese Auflistung enthält spezifische Anwendungsobjekte. Die Objektauflistung einer Grafikanwendung kann beispielsweise einige Figuren enthalten, die zusammen die Zeichnung ergeben (das Doku-

ment). Die Objektauflistung einer Textverarbeitung nimmt z.B. die Absätze des Textes auf. Die Objektauflistung einer Tabellenkalkulation repräsentiert die Zellen des Dokuments.

Das Verwenden dieser Objektauflistung und der Einsatz einzelner Objekte ist unabhängig von der jeweiligen Implementierung. Möchten Sie diese Objekte nutzen, sollten Sie daran denken, Eigenschaften und Methoden für alle Objekte und Auflistungen zu definieren.

32.3 Zusammenfassung

Automatisierung beschreibt die Kommunikation zwischen Automatisierungs-Servern und Automatisierungs-Controllern mit Hilfe der IDispatch-Schnittstelle. Die MFC-Bibliothek und Visual Studio unterstützen die Entwicklung von Automatisierungs-Servern.

Das Erstellen eines Automatisierungs-Server geschieht mit Hilfe des Anwendungsassistenten. Selektieren Sie im dritten Schritt des Assistenten die Option AUTOMATISIERUNG. Auf diese Weise fügen Sie Ihrer Anwendung und der Dokumentklasse die Unterstützung der Automatisierung hinzu.

Automatisierungs-Eigenschaften und Automatisierungs-Methoden werden mit dem Klassen-Assistenten definiert. Methoden werden durch Elementfunktionen repräsentiert. Eigenschaften werden aus Elementvariablen oder Get-/Set-Methoden erzeugt. Möchten Sie eine schreibgeschützte Eigenschaft implementieren, verwenden Sie die Get-/Set-Methoden als Eigenschaften und löschen die Bezeichnung der nicht benötigten Set-Methode.

Der Klassen-Assistent erstellt außerdem eine Typenbibliothek-Ressourcendatei, die ein in der Objektbeschreibungssprache verfaßtes Skript ist. Wird eine Typenbibliothek mit Hilfe des MIDL-Compilers aus diesem Skript erzeugt, kann es von anderen Anwendungen verwendet werden, um Informationen über die Automatisierungs-Schnittstelle des Server zu ermitteln.

Automatisierungs-Server können von Automatisierungs-Clients geprüft und verwendet werden. Allgemeine Automatisierungs-Controller sind Programmierungsumgebungen wie Visual.

Die Möglichkeit, Automatisierungs-Server als Black-Box zu verwenden, und das Potential von Visual Basic und anderen Entwicklungssystemen, die System-Integrationswerkzeuge sind, zeigen die Leistungsfähigkeit der Automatisierung.

Microsoft empfiehlt, daß Automatisierungs-Anwendungen einige Standardobjekte über deren Eigenschaften und Methoden verwenden sollten. Zu diesen Objekten zählen die Dokumentauflistung, einzelne Dokumentobjekte, die Objektauflistung innerhalb eines Dokuments und, sofern möglich, Objekte, die ein Dokument enthalten. Für diese Objekte bestehen einige Standardeigenschaften und Methoden, die von anderen Anwendungen verwendet werden können.

Erstellen von ActiveX- Steuerelementen mit der MFC

Kapitel 33

Das Erstellen von benutzerdefinierten OLE-Steuerelementen, wie ActiveX-Steuerelemente bisher genannt wurden, erforderte sehr viel Programmierwissen. Mit Visual C++ 4 änderte sich dieser Umstand. Die vierte Version von C++ unterstützte bereits die Entwicklung benutzerdefinierter Steuerelemente über den Anwendungsassistenten.

ActiveX-Steuerelemente können mittlerweile mit einem relativ geringen Aufwand generiert werden. Visual C++ bietet dazu zwei Mechanismen an: Sie können

- ➡ ein MFC-ActiveX-Steuerelement oder
- ➡ ein ActiveX-Steuerelement, das auf der neuen ActiveX-Vorlagenbibliothek (ATL – ActiveX Template Library) basiert,

erstellen.

Dieses Kapitel beschreibt das Erstellen von ActiveX-Steuerelementen mit der MFC. Im nächsten Kapitel wird die ATL erläutert.

Ein ActiveX-Steuerelement besteht aus dem dargestellten Element, Eigenschaften, Methoden und Ereignissen. Die Eigenschaften, Methoden und Ereignisse des Steuerelements werden mit Hilfe des Klassen-Assistenten definiert.

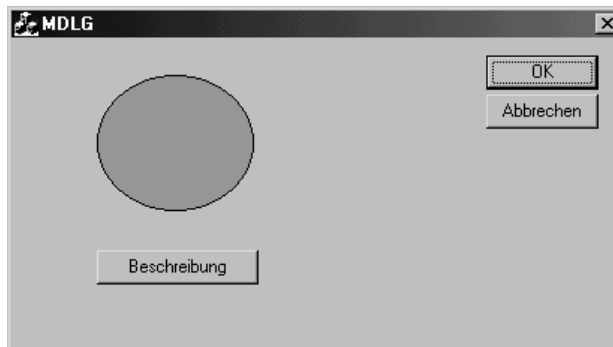
Um Ihnen die Leistungsfähigkeit eines ActiveX-Steuerelements zu demonstrieren, habe ich ein einfaches Steuerelement entwickelt, das mit MCTL bezeichnet ist. Es besteht aus einer einzelnen Schaltfläche, die drei unterschiedliche Figuren darstellen kann (Ellipse, Rechteck und Dreieck). Wird die Schaltfläche betätigt, ändert sie ihren Status von nicht selektiert in selektiert und umgekehrt.

*Das Beispiel-
Steuerelement*

Ist das Steuerelement selektiert, erscheint es in der Farbe Rot. Andernfalls wird es Grün dargestellt. Das Steuerelement generiert immer Ereignisse, wenn sich sein Status ändert.

Abbildung 33.1 zeigt das benutzerdefinierte MCTL-Steuerelement.

Abbildung 33.1:
Das MCTL-Steuerelement



Wie jedes andere Steuerelement, kann auch MCTL in Dialoge eingebettet und in HTML-Dokumenten sowie von anderen ActiveX-Steuerelement-Containern verwendet werden. Das Visual Studio unterstützt das Erstellen von Dialogen, die ActiveX-Steuerelemente enthalten. ActiveX-Steuerelemente können außerdem über Mantelklassen in Anwendungen eingesetzt werden.

Um ein ActiveX-Steuerelement zu generieren, gehen Sie wie folgt vor:

1. Erstellen Sie das Gerüst des ActiveX-Steuerelements mit Hilfe des Anwendungsassistenten.
2. Fügen Sie dem Steuerelement die Eigenschaften, Methoden und Ereignisse mit dem Klassen-Assistenten hinzu.
3. Aktualisieren Sie die Bitmap, die das Steuerelement repräsentiert, mit dem Bitmap-Editor des Visual Studio.
4. Schreiben Sie den Zeichencode für das Steuerelement.
5. Fügen Sie den Programmcode für die Methoden und Ereignisse hinzu.
6. Erstellen Sie die Eigenschaftenseite des Steuerelements.
7. Schreiben Sie den Programmcode, der die Eigenschaftenseite bearbeitet.

Dieses Kapitel beschreibt jeden dieser Schritte detailliert. Das MCTL-Steuerelement dient dabei als Beispiel für die Umsetzung der Theorie in die Praxis.

33.1 Erstellen eines Steuerelementgerüsts mit dem Anwendungsassistenten

Erstellen Sie mit dem Anwendungsassistenten zunächst das Gerüst des Steuerelements. Verwenden Sie dazu den MFC-ActiveX-Steuerelement-Assistenten.

Erstellen des Steuerelementgerüsts

1. Wählen Sie aus dem Menü DATEI den Eintrag NEU aus.
2. Öffnen Sie in dem anschließend dargestellten Dialog NEU (Abbildung 33.2) das Register PROJEKTE.

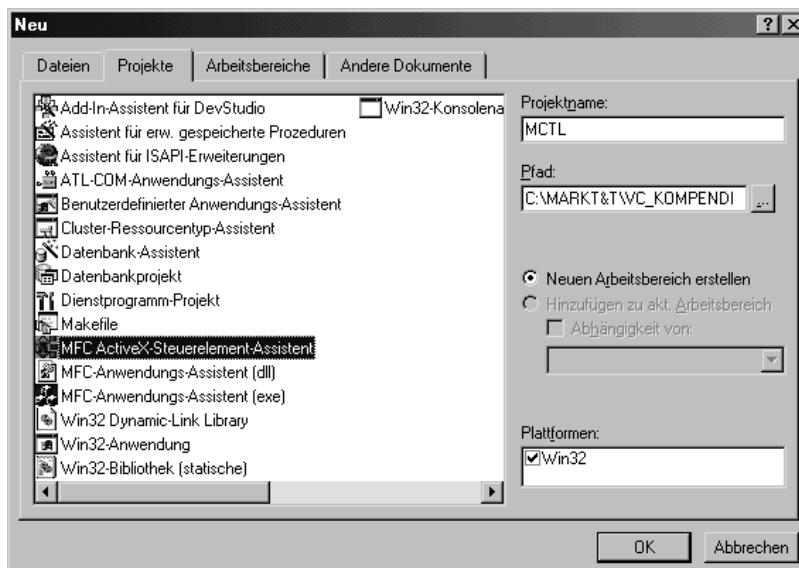
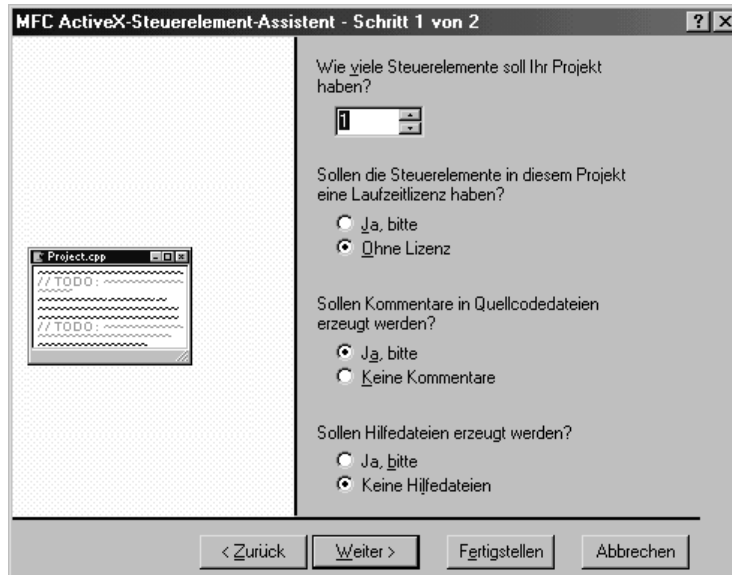


Abbildung 33.2:
Erstellen des
MCTL-Steuerele-
ments

3. Wählen Sie dort den Projekttyp MFC ACTIVE X-STEUELEMENT-ASSISTENT aus, und geben Sie die Bezeichnung des neuen Projekts ein (zum Beispiel MCTL). Bestimmen Sie das Verzeichnis, in dem das Projekt gespeichert werden soll.

Nachdem Sie die Schaltfläche OK betätigt haben, wird die erste Dialogseite des ActiveX-Steuerelement-Assistenten angezeigt (Abbildung 33.3).

Abbildung 33.3:
MFC-ActiveX-
Steuerelement-
Assistent,
Seite 1



4. Hier geben Sie an, wie viele Steuerelemente Ihr Projekt verwenden soll (ein ActiveX-Projekt kann mehrere Steuerelemente enthalten). Sie bestimmen außerdem, ob Ihre Steuerelemente über eine Laufzeitlizenz verfügen sollen und ob Kommentare im Quellcode sowie Hilfedateien erzeugt werden. (Die letzte Option sollten Sie aktivieren, da das Entfernen von Programmcode immer einfacher als das Hinzufügen desselben ist.)



Lizenzierung

Entscheiden Sie sich für die Unterstützung der Lizenzierung, erstellt der Steuerelement-Assistent eine LIC-Standarddatei für Ihr Steuerelement. Der Steuerelementklasse werden außerdem Funktionen hinzugefügt, die der Überprüfung von Lizenzinformationen dienen. Ohne eine zulässige Lizenzdatei (LIC) kann das Steuerelement nicht im Entwurfsmodus verwendet werden. Anwendungsprogrammierer sollten Ihr Steuerelement auf diese Weise an den Endanwender distribuieren (also ohne die Lizenzdatei). Sie wiederum stellen dem Programmierer Ihr Steuerelement mit der Lizenzdatei zur Verfügung, damit dieser mit dem Objekt arbeiten kann.

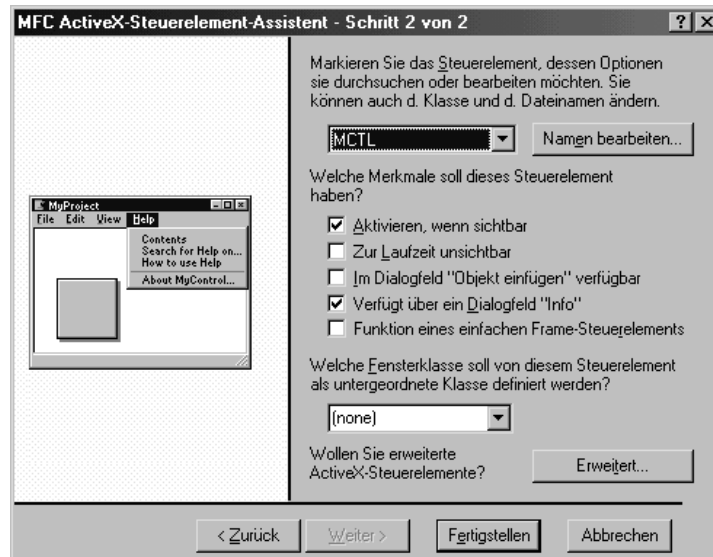


Abbildung 33.4:
Steuerelement-
Assistent,
Seite 2

Auf der zweiten Seite des Steuerelement-Assistenten (Abbildung 33.4) können Sie die vom Assistenten generierten Klassennamen Ihres Steuerelements einsehen und verändern. Der Anwendungsassistent generiert für jedes Steuerelement Ihres Projekts zwei Klassen: eine Steuerelementklasse und eine Klasse für die Eigenschaftenseite des Steuerelements.

Benötigt Ihr Steuerelement mehr als eine Eigenschaftenseite, können Sie diese später erzeugen. Geben Sie zunächst mit Hilfe des Steuerelement-Assistenten die Parameter der ersten Eigenschaftenseite an.

Auf der zweiten Seite des Steuerelement-Assistenten definieren Sie ebenfalls zusätzliche Eigenschaften. Die Option ZUR LAUFZEIT UNSICHTBAR ermöglicht Ihnen das Erstellen von Steuerelementen, die zur Laufzeit über keine sichtbare Oberfläche verfügen. Die Option IM DIALOGFELD »OBJEKT EINFÜGEN« VERFÜGBAR ermöglicht Ihnen, ein Steuerelement zu generieren, das in Container-Dokumente eingefügt werden kann. Diese Option sollte gewöhnlich nicht aktiviert werden. Wenn Sie eine neue Schaltfläche erstellen, die in Dialogen verwendet werden soll, möchten Sie nicht, daß diese in ein Word-für-Windows-Dokument eingefügt wird.

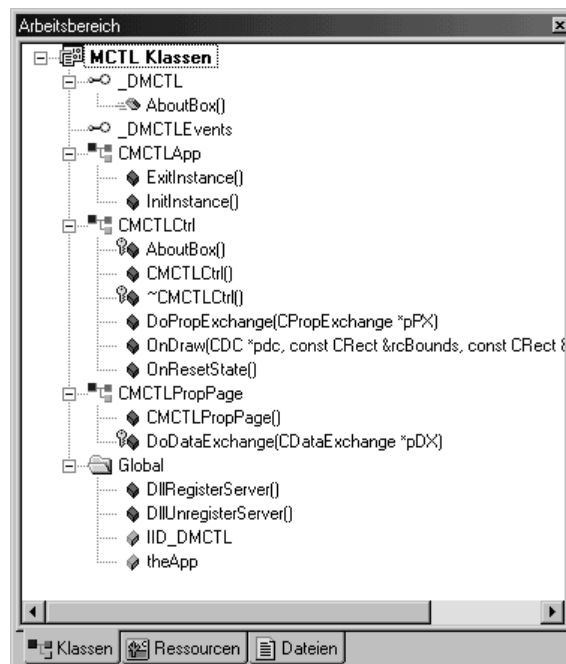
Sie können außerdem eine Steuerelementklasse als Unterklasse des neuen benutzerdefinierten Steuerelements spezifizieren. So haben Sie beispielsweise die Möglichkeit, eine BUTTON-Klasse zu erstellen, um ein benutzerdefiniertes Steuerelement zu erzeugen, das einige Eigenschaften des Schaltflächen-Standardsteuerelements übernimmt.

5. Für das MCTL-Steuerelement behalten Sie die Voreinstellungen des Steuerelement-Assistenten bei.

ActiveX-Steuerelement Programmübersicht

Betrachten Sie den vom Steuerelement-Assistenten generierten Programmcode sowie die neuen Steuerelementklassen und Ressourcen. Bevor wir mit der Bearbeitung des Codes beginnen, werden diese Elemente beschrieben.

Abbildung 33-5:
Die vom Steuerelement-Assistenten generierten Klassen



Wenn Sie Ihr Projekt in der Klassen-Ansicht darstellen lassen, erkennen Sie, daß der Steuerelement-Assistent drei Klassen generiert hat.

Eine dieser Klassen repräsentiert

- das benutzerdefinierte Steuerelement-Bibliotheksojekt.

Die verbleibenden Klassen repräsentieren

- das Steuerelement-Objekt und
- die Eigenschaftenseite des Steuerelements.

(Haben Sie ein Projekt mit mehreren Steuerelementen erzeugen lassen, generiert der Steuerelement-Assistent weitere Steuerelement-Objekte und entsprechende Eigenschaftenseiten.)

CMCTLApp

Die einfachste Klasse des MCTL-Projekts ist mit `CMCTLApp` bezeichnet. Sie wird von `COleControlModule` abgeleitet und repräsentiert das OCX-Objekt, also die Bibliothek, die Ihre ActiveX-Steuerelemente enthält. Lediglich zwei Elementfunktionen werden in der Klasse deklariert (Listing 33.1).

```
class CMCTLApp : public COleControlModule
{
public:
    BOOL InitInstance();
    int ExitInstance();
};
```

*Listing 33.1:
CMCTLApp-Klas-
sendeclaration*

Diese Funktionen sind sehr oberflächlich implementiert (Listing 33.2). In der Datei `MCTL.CPP` befinden sich ebenfalls zwei weitere Funktionen, die mit `DllRegisterServer` und `DllUnregisterServer` bezeichnet sind. Diese Funktionen speichern und entfernen OLE-Informationen über Ihr neues Steuerelement in der Registrierung. Das distribuierbare Programm `REGSVR32.EXE` ruft diese Funktionen auf, um ActiveX-Steuerelemente zu registrieren oder diese Registrierung rückgängig zu machen.

```
////////////////////////////////////
CMCTLApp::InitInstance - DLL-Initialisierung

BOOL CMCTLApp::InitInstance()
{
    BOOL bInit = COleControlModule::InitInstance();

    if (bInit)
    {
        // ZU ERLEDIGEN: Hier den Code für das Initialisieren der
        // eigenen Module einfügen.
    }

    return bInit;
}

////////////////////////////////////
CMCTLApp::ExitInstance - DLL-Beendigung

int CMCTLApp::ExitInstance()
{
    // ZU ERLEDIGEN: Hier den Code für das Beenden der eigenen
    // Module einfügen.

    return COleControlModule::ExitInstance();
}

////////////////////////////////////
DllRegisterServer - Fügt der Systemregistrierung Einträge
// hinzu

STDAPI DllRegisterServer(void)
```

*Listing 33.2:
CMCTLApp-Klas-
senimplementie-
rung und Steuer-
element-
Registrierungs-
funktionen*

```

{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleRegisterTypeLib(AfxGetInstanceHandle(), _tlid))
        return ResultFromCode(SELFREG_E_TYPELIB);

    if (!COleObjectFactoryEx::UpdateRegistryAll(TRUE))
        return ResultFromCode(SELFREG_E_CLASS);

    return NOERROR;
}

////////////////////////////////////
DllUnregisterServer - Entfernt Einträge aus der
// Systemregistrierung

STDAPI DllUnregisterServer(void)
{
    AFX_MANAGE_STATE(_afxModuleAddrThis);

    if (!AfxOleUnregisterTypeLib(_tlid, _wVerMajor, _wVerMinor))
        return ResultFromCode(SELFREG_E_TYPELIB);

    if (!COleObjectFactoryEx::UpdateRegistryAll(FALSE))
        return ResultFromCode(SELFREG_E_CLASS);

    return NOERROR;
}

```

CMCTLPropPage

Die Klasse `CMCTLPropPage` implementiert die Eigenschaftenseite des Steuerelements, die von Entwurfsanwendungen (wie zum Beispiel dem Dialog-Editor des Visual Studio) in einem Eigenschaftendialog angezeigt wird.

Abbildung 33.6:
Die vom Steuer-
element-
Assistenten
generierte Stan-
dardeigenschaf-
tenseite



Der Steuerelement-Assistent generiert eine leere Eigenschaftenseite (Abbildung 33.6). Sie fügen dieser Seite die zum Einrichten der Eigenschaften Ihres Steuerelements benötigten Felder hinzu.

Die Klassendeklaration von `CMCTLPropPage` (Listing 33.3) enthält lediglich die Deklaration eines Konstruktors und eine `DoDataExchange`-Elementfunktion. Beachten Sie bitte, daß die Makros `DECLARE_DYNCREATE` und `DECLARE_OLECREATE_EX` verwendet werden. Das zuletzt genannte Makro ermöglicht die Erstellung der Eigenschaftenseite des ActiveX-Steuerelements.

```
class CMCTLPropPage : public COlePropertyPage
{
    DECLARE_DYNCREATE(CMCTLPropPage)
    DECLARE_OLECREATE_EX(CMCTLPropPage)

    // Konstruktor
public:
    CMCTLPropPage();

    // Dialogfelddaten
    //({AFX_DATA(CMCTLPropPage)
    enum { IDD = IDD_PROPPAGE_MCTL };
        // HINWEIS - Der Klassen-Assistent fügt Datenelemente
        // hier ein. Innerhalb dieser generierten
        // Quelltextabschnitte NICHTS VERÄNDERN!
    //})AFX_DATA

    // Implementierung
protected:
    // DDX/DDV-Unterstützung
    virtual void DoDataExchange(CDataExchange* pDX);

    // Nachrichtenzuordnungstabellen
protected:
    //({AFX_MSG(CMCTLPropPage)
        // HINWEIS - Der Klassen-Assistent fügt Member-Funktionen
        // hier ein und entfernt diese.
        // Innerhalb dieser generierten Quelltextabschnitte
        // NICHTS VERÄNDERN!
    //})AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

*Listing 33.3:
CMCTLPropPage-Klassendeklaration*

Obwohl diese Klasse von `COlePropertyPage` abgeleitet ist, arbeitet sie wie eine von `CPropertyPage` abgeleitete Klasse. Sie können den Klassen-Assistenten verwenden, um der Klasse Elementvariablen hinzuzufügen, die den Steuerelementen auf der Eigenschaftenseite entsprechen. Bisweilen ist es erforderlich, zusätzliche Datenaustauschfunktionen und Makros zu benutzen, wenn beispielsweise eine beständige Steuerelementeigenschaft definiert wird.

Die Implementierungsdatei von `CMCTLPropPage` enthält zusätzlich zu den Gerüsten des Konstruktors und der `DoDataExchange`-Elementfunktion verschiedene OLE-Elemente (Listing 33.4). Eine manuelle Bearbeitung dieser Elemente ist jedoch nur selten erforderlich. Diese Aufgabe übernimmt gewöhnlich der Klassen-Assistent.

Listing 33.4:
CMCTLPropPage-
ge-Klassenimple-
mentierung

```
IMPLEMENT_DYNCREATE(CMCTLPropPage, COlePropertyPage)

////////////////////////////////////
Nachrichtenzuordnungstabelle

BEGIN_MESSAGE_MAP(CMCTLPropPage, COlePropertyPage)
    ///{AFX_MSG_MAP(CMCTLPropPage)
    // HINWEIS - Der Klassen-Assistent fügt
    // Nachrichtenzuordnungseinträge hinzu und entfernt diese
    // Innerhalb dieser generierten Quelltextabschnitte
    // NICHTS VERÄNDERN!
    ///}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////// Klassenerzeugung
und GUID initialisieren

IMPLEMENT_OLECREATE_EX(CMCTLPropPage, "MCTL.MCTLPropPage.1",
    0x45c91f64, 0x6029, 0x11d2, 0xb4, 0xf9, 0xfc, 0xa4, 0x62,
    0x2b, 0xb1, 0x2f)

////////////////////////////////////
CMCTLPropPage::CMCTLPropPageFactory::UpdateRegistry -
// Fügt Einträge der Systemregistrierung für CMCTLPropPage hinzu
// oder entfernt diese

BOOL CMCTLPropPage::CMCTLPropPageFactory::UpdateRegistry(
    BOOL bRegister)
{
    if (bRegister)
        return AfxOleRegisterPropertyPageClass(
            AfxGetInstanceHandle(),
            m_clsid, IDS_MCTL_PPG);
    else
        return AfxOleUnregisterClass(m_clsid, NULL);
}

////////////////////////////////////
CMCTLPropPage::CMCTLPropPage - Konstruktor

CMCTLPropPage::CMCTLPropPage() :
    COlePropertyPage(IDD, IDS_MCTL_PPG_CAPTION)
{
    ///{AFX_DATA_INIT(CMCTLPropPage)
    // HINWEIS: Der Klassen-Assistent fügt die
    // Elementinitialisierung hier ein. Innerhalb dieser
    // generierten Quelltextabschnitte NICHTS VERÄNDERN!
    ///}AFX_DATA_INIT
}

////////////////////////////////////
CMCTLPropPage::DoDataExchange - Verschiebt Daten zwischen Dialogfeld+++ und den
Variablen+++

void CMCTLPropPage::DoDataExchange(CDataExchange* pDX)
{
    ///{AFX_DATA_MAP(CMCTLPropPage)
    // HINWEIS: Der Klassen-Assistent fügt DDP-, DDX- und D
    // DV-Aufrufe hier ein. Innerhalb dieser generierten
    // Quelltextabschnitte NICHTS VERÄNDERN!
}
```

```

    ///AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}

```

Gewöhnlich werden Sie diesen Programmcode kaum bearbeiten müssen. Der Programmcode zur Unterstützung der Elementvariablen, die den Steuerelementen auf der Eigenschaftenseite entsprechen, wird automatisch von dem Klassen-Assistenten eingefügt. Das manuelle Modifizieren ist nur dann erforderlich, wenn ein Steuerelementverhalten erwünscht ist, das nicht dem Standard entspricht. Doch selbst in diesen Fällen sind Änderungen häufig lediglich an der `DoDataExchange`-Elementfunktion notwendig.

CMCTLCtrl

Die Klasse `CMCTLCtrl` enthält die Funktionalität des Steuerelements und ist das zentrale Element des Projekts. Die Deklaration der `CMCTLCtrl`-Klasse ist in Listing 33.5 aufgeführt.

```

class CMCTLCtrl : public CControl
{
    DECLARE_DYNCREATE(CMCTLCtrl)

// Konstruktor
public:
    CMCTLCtrl();

// Überladungen
// Vom Klassenassistenten generierte Überladungen virtueller
// Funktionen
    ///AFX_VIRTUAL(CMCTLCtrl)
    public:
        virtual void OnDraw(CDC* pdc, const CRect& rcBounds,
                           const CRect& rcInvalid);
        virtual void DoPropExchange(CPropExchange* pPX);
        virtual void OnResetState();
    ///AFX_VIRTUAL

// Implementierung
protected:
    ~CMCTLCtrl();

    DECLARE_OLECREATE_EX(CMCTLCtrl) // Klassenerzeugung und GUID
    DECLARE_OLETYPELIB(CMCTLCtrl)   // GetTypeInfo
    DECLARE_PROPPAGEIDS(CMCTLCtrl)  // Eigenschaftenseiten-IDs
    DECLARE_OLECTLTYPE(CMCTLCtrl)   // Typname und versch. Status

// Nachrichtenzuordnungstabellen
    ///AFX_MSG(CMCTLCtrl)
    // HINWEIS - Der Klassen-Assistent fügt Member-Funktionen
    // hier ein und entfernt diese. Innerhalb dieser
    // generierten Quelltextabschnitte NICHTS VERÄNDERN!
    ///AFX_MSG
    DECLARE_MESSAGE_MAP()

// Dispatch-Tabellen
    ///AFX_DISPATCH(CMCTLCtrl)
    // HINWEIS - Der Klassen-Assistent fügt Member-Funktionen
    // hier ein und entfernt diese. Innerhalb dieser
    // generierten Quelltextabschnitte NICHTS VERÄNDERN!

```

*Listing 33.5:
CMCTLCtrl-Klassendeklaration*

```

    ///AFX_DISPATCH
    DECLARE_DISPATCH_MAP()

    afx_msg void AboutBox();

// Ereignistabellen
    ///AFX_EVENT(CMCTLCtrl)
    // HINWEIS - Der Klassen-Assistent fügt Member-Funktionen
    // hier ein und entfernt diese. Innerhalb dieser
    // generierten Quelltextabschnitte NICHTS VERÄNDERN!
    ///AFX_EVENT
    DECLARE_EVENT_MAP()

// Dispatch- und Ereignis-IDs
public:
    enum {
        ///AFX_DISP_ID(CMCTLCtrl)
        // HINWEIS: Der Klassen-Assistent fügt hier
        // Aufzählungselemente hinzu und entfernt diese.
        // Innerhalb dieser generierten Quelltextabschnitte
        // NICHTS VERÄNDERN!
        ///AFX_DISP_ID
    };
};

```

Der Implementierungsabschnitt dieser Deklaration beginnt mit vier Makroaufrufen, die verschiedene OLE-Elemente deklarieren. Andere Makros implementieren diese Elemente in der Implementierungsdatei. Besonders interessant ist der Aufruf von `DECLARE_OLECREATE_EX`. Ein ActiveX-Steuerelement, das die Lizenzierung unterstützt, verwendet an dieser Stelle kein Makro, sondern einen Block mit Funktionsdeklarationen, die sich zwischen den Schlüsselwörtern `BEGIN_OLEFACTORY` und `END_OLEFACTORY` befinden.

Der verbleibende Programmcode deklariert die Nachrichtenzuordnungstabelle, Verteilertabelle, Ereignistabelle sowie die Bezeichner der Verteiler und Ereignisse des Steuerelements. Diese Tabellen leiten Nachrichten weiter und definieren die Eigenschaften, Methoden sowie die Ereignisse der OLE-Schnittstelle des Steuerelements. Der Klassen-Assistent fügt diesem Programmabschnitt automatisch die erforderlichen Einträge hinzu, so daß Sie keine Modifizierungen vornehmen müssen.

Die Implementierungsdatei der Klasse `CMCTLCtrl` (Listing 33.6) enthält verschiedene Elemente, mit denen Sie sich vertraut machen sollten.

Listing 33.6:
CMCTLCtrl-Klassenimplementierung

```

IMPLEMENT_DYNCREATE(CMCTLCtrl, COleControl)

////////////////////////////////////
Nachrichtenzuordnungstabelle

BEGIN_MESSAGE_MAP(CMCTLCtrl, COleControl)
    ///AFX_MSG_MAP(CMCTLCtrl)
    // HINWEIS - Der Klassen-Assistent fügt
    // Nachrichtenzuordnungseinträge hinzu und entfernt diese
    // Innerhalb dieser generierten Quelltextabschnitte

```

```

// NICHTS VERÄNDERN!
//}}AFX_MSG_MAP
ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
END_MESSAGE_MAP()

//////////////////////////////////// Dispatch-Tabelle

BEGIN_DISPATCH_MAP(CMCTLCtrl, CObject)
//{{AFX_DISPATCH_MAP(CMCTLCtrl)
// HINWEIS - Der Klassen-Assistent fügt Einträge in die
// Dispatch-Tabelle ein und entfernt diese
// Innerhalb dieser generierten Quelltextabschnitte
// NICHTS VERÄNDERN!
//}}AFX_DISPATCH_MAP
DISP_FUNCTION_ID(CMCTLCtrl, "AboutBox", DISPID_ABOUTBOX,
    AboutBox, VT_EMPTY, VTS_NONE)
END_DISPATCH_MAP()

//////////////////////////////////// Ereignistabelle

BEGIN_EVENT_MAP(CMCTLCtrl, CObject)
//{{AFX_EVENT_MAP(CMCTLCtrl)
// HINWEIS - Der Klassen-Assistent fügt Einträge in die
// Ereignistabelle ein und entfernt diese
// Innerhalb dieser generierten Quelltextabschnitte
// NICHTS VERÄNDERN!
//}}AFX_EVENT_MAP
END_EVENT_MAP()

////////////////////////////////////
Eigenschaftenseiten

// ZU ERLEDIGEN: Fügen Sie mehr Eigenschaftenseiten ein, als
// erforderlich sind. Denken Sie daran, den Zähler zu erhöhen!
BEGIN_PROPPAGEIDS(CMCTLCtrl, 1)
    PROPPAGEID(CMCTLPropPage::guid)
END_PROPPAGEIDS(CMCTLCtrl)

//////////////////////////////////// Klassenerzeugung
und GUID initialisieren

IMPLEMENT_OLECREATE_EX(CMCTLCtrl, "MCTL.MCTLCtrl.1", 0x20e26da6,
    0x584a, 0x11d2, 0xb4, 0xf9, 0x9b, 0x7, 0x8c, 0x21, 0x32, 0x15)

////////////////////////////////////
// Typbibliothek-ID und Version

IMPLEMENT_OLETYPELIB(CMCTLCtrl, _tlibid, _wVerMajor, _wVerMinor)

//////////////////////////////////// Schnittstellen-
IDs

const IID BASED_CODE IID_DMCTL =
    { 0x45c91f62, 0x6029, 0x11d2, { 0xb4, 0xf9, 0xfc, 0xa4, 0x62,
        0x2b, 0xb1, 0x2f } };
const IID BASED_CODE IID_DMCTLEvents =
    { 0x45c91f63, 0x6029, 0x11d2, { 0xb4, 0xf9, 0xfc, 0xa4, 0x62,
        0x2b, 0xb1, 0x2f } };

```

```

////////////////////////////////////// Steuerelement-
Typinformation

static const DWORD BASED_CODE _dwMCTL01eMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;

IMPLEMENT_OLECTLTYPE(CMCTLCtrl, IDS_MCTL, _dwMCTL01eMisc)

//////////////////////////////////////
CMCTLCtrl::CMCTLCtrlFactory::UpdateRegistry -
// Fügt Einträge der Systemregistrierung für CMCTLCtrl hinzu oder
// entfernt diese

BOOL CMCTLCtrl::CMCTLCtrlFactory::UpdateRegistry(BOOL bRegister)
{
    // ZU ERLEDIGEN: Prüfen Sie, ob Ihr Steuerelement den
    // Thread-Regeln nach dem "Apartment"-Modell entspricht.
    // Weitere Informationen finden Sie unter MFC TechNote 64.
    // Falls Ihr Steuerelement nicht den Regeln nach dem
    // Apartment-Modell entspricht, so müssen Sie den
    // nachfolgenden Code ändern, indem Sie den 6. Parameter von
    // afxRegApartmentThreading auf 0 ändern.

    if (bRegister)
        return AfxOleRegisterControlClass(
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_MCTL,
            IDB_MCTL,
            afxRegApartmentThreading,
            _dwMCTL01eMisc,
            _tlib,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}

//////////////////////////////////////
CMCTLCtrl::CMCTLCtrl - Konstruktor

CMCTLCtrl::CMCTLCtrl()
{
    InitializeIDs(&IID_DMCTL, &IID_DMCTLEvents);

    // ZU ERLEDIGEN: Daten der Steuerelementinstanz hier
    // initialisieren.
}

//////////////////////////////////////
CMCTLCtrl::~CMCTLCtrl - Destruktor

CMCTLCtrl::~CMCTLCtrl()
{
    // ZU ERLEDIGEN: Daten der Steuerelementinstanz hier
    // bereinigen.
}

```

```

////////////////////////////////////
CMCTLCtrl::OnDraw - Zeichenfunktion

void CMCTLCtrl::OnDraw(CDC* pdc, const CRect& rcBounds,
                      const CRect& rcInvalid)
{
    // ZU ERLEDIGEN: Folgenden Code durch eigene Zeichenfunktion
    // ersetzen.
    pdc->FillRect(rcBounds, CBrush::FromHandle(
        (HBRUSH)GetStockObject(WHITE_BRUSH)));
    pdc->Ellipse(rcBounds);
}

////////////////////////////////////
CMCTLCtrl::DoPropExchange - Unterstützung dauerhafter
// Eigenschaften

void CMCTLCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);

    // ZU ERLEDIGEN: PX_ Funktionen für jede dauerhafte
    // benutzerdefinierte Eigenschaft aufrufen.
}

////////////////////////////////////
CMCTLCtrl::OnResetState - Setzt das Steuerelement in den
// Standardzustand zurück

void CMCTLCtrl::OnResetState()
{
    COleControl::OnResetState(); // Setzt die Standards zurück,
                                // die in DoPropExchange
                                // gefunden wurden

    // ZU ERLEDIGEN: Andere Steuerelementzustände hier
    // zurücksetzen.
}

////////////////////////////////////
CMCTLCtrl::AboutBox - Ein Dialogfeld "Info" für den Benutzer anzeigen

void CMCTLCtrl::AboutBox()
{
    CDialog dlgAbout(IDD_ABOUTBOX_MCTL);
    dlgAbout.DoModal();
}

```

Die Programmabschnitte, in denen die Nachrichtenzuordnungs-, Verteiler- und Ereignistabellen implementiert sind, werden gewöhnlich von dem Klassen-Assistenten verwaltet. Möchten Sie jedoch Ihrem Eigenschaftendialog zusätzliche Eigenschaftenseiten zuweisen, müssen Sie den entsprechenden Abschnitt in diesem Programmcode manuell modifizieren. Für ein Steuerelement, das beispielsweise über drei Eigenschaftenseiten verfügt, könnte dieser Abschnitt aus dem folgenden Programmcode bestehen:

```
BEGIN_PROPPAGEIDS(CMCTLCtrl, 3)
    PROPPAGEID(CMCTLPropPage1::guid)
    PROPPAGEID(CMCTLPropPage2::guid)
    PROPPAGEID(CMCTLPropPage3::guid)
END_PROPPAGEIDS(CMCTLCtrl)
```

OnDraw Drei der in der Klassenimplementierung definierten Funktionen sind für den Steuerelementprogrammierer von besonderer Bedeutung. Die Elementfunktion `OnDraw` zeichnet das Steuerelement. Der Steuerelement-Assistent generiert diese Elementfunktion mit einigen Standardanweisungen für den Zeichenvorgang (die Funktion zeichnet entweder eine Ellipse oder ruft die Zeichenfunktion der Superklasse auf, wenn das Steuerelement aus einer Subklasse erzeugt wurde).

DoPropExchange Die Elementfunktion `DoPropExchange` implementiert die Unterstützung beständiger Eigenschaften. Eigenschaften können somit zur Entwurfszeit gespeichert und wieder geladen werden, wenn das Steuerelement zur Laufzeit angezeigt wird. Dieses Thema wird später in dem Kapitel detailliert erläutert.

OnResetState Die Funktion `OnResetState` setzt die Werte der Eigenschaften auf die Voreinstellungen zurück. Wenngleich der Aufruf der Basisklassenimplementierung von `COleControl::OnResetState` alle beständigen Eigenschaften zurücksetzt, müssen andere Eigenschaften in dieser Funktion möglicherweise manuell zurückgesetzt werden.

33.2 Bearbeiten des Steuerelements

Zuerst fügen Sie dem Steuerelement mit Hilfe des Klassen-Assistenten Eigenschaften, Methoden und Ereignisse hinzu.

Eigenschaften	Eine Eigenschaft entspricht auch dann einem Datenelement einer C++-Klasse, wenn sie mit den <code>Get-/Set-</code> Methoden und nicht als Elementvariablen implementiert wird.
Methoden	Eine Methode ist vergleichbar mit einer Elementfunktion und wird in der Steuerelementklasse auch als Elementfunktion implementiert.
Ereignisse	Ereignisse sind Nachrichten, die das Steuerelement an das übergeordnete Fenster senden kann. Wenn Sie der Klasse mit Hilfe des Klassen-Assistenten ein Ereignis hinzufügen, erstellt der Assistent eine Elementfunktion, die das Ereignis auslöst. Sie können diese Elementfunktion aus Ihrem Programmcode heraus aufrufen.

Das MCTL-Steuerelement verfügt über zwei Eigenschaften und ein Ereignis.

- ➔ Die Eigenschaft `Selected` ist ein Boolescher Wert, der den Selektionszustand des Steuerelements wiedergibt. Wird die linke Maustaste über dem Steuerelement betätigt, ändert sich dessen Status von »selektiert« in nicht »selektiert« oder umgekehrt. Die `Selected`-Eigenschaft wird mit den Get-/Set-Methoden implementiert. Diese Art der Implementierung ermöglicht uns das Erstellen einer schreibgeschützten Eigenschaft. Dazu muß lediglich die Set-Methode gelöscht werden.
- ➔ Die Eigenschaft `Shape` ist ein Integer-Wert, der die Figur des Steuerelements bestimmt. Der Wert kann auf 0 (Ellipse), 1 (Rechteck) oder 2 (Dreieck) gesetzt werden. Diese Eigenschaft kann sowohl ausgelesen als auch gesetzt werden. Wir erlauben das Setzen der Eigenschaft jedoch ausschließlich im Entwurfsmodus. Dazu implementieren wir die Eigenschaft ebenfalls mit den Get-/Set-Methoden und prüfen in der Set-Methode, ob das Element im Entwurfsmodus verwendet wird.
- ➔ Das Ereignis `Select` zeigt dem übergeordneten Fenster an, daß sich der Status des Steuerelements geändert hat. Wir lösen dieses Ereignis aus, wenn das Steuerelement über die Betätigung der linken Maustaste benachrichtigt wird. (Dazu fügen wir dem Steuerelement eine Bearbeiterfunktion für dieses Ereignis hinzu.)

33.2.1 Ändern der Steuerelement-Bitmap

Bevor wir den Programmcode des Steuerelements bearbeiten, werden wir die Steuerelement-Bitmap aktualisieren. Das Steuerelement wird mit dieser Bitmap in einer Werkzeugleiste dargestellt, die von den Anwendern benutzt wird, um Ihr Steuerelement in ein Projekt einzufügen. Abbildung 33.7 zeigt die Bitmap, die ich für das MCTL-Steuerelement entworfen habe.

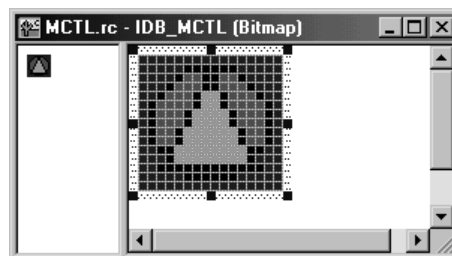


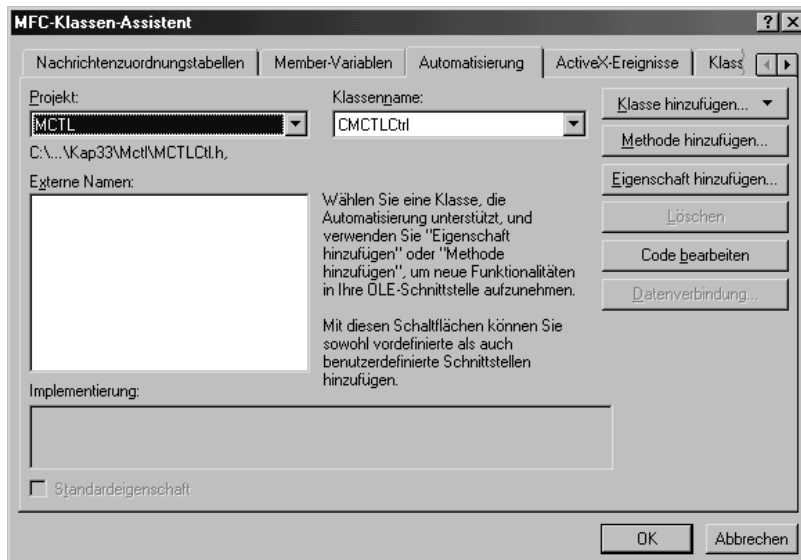
Abbildung 33.7:
Die MCTL-Steuerelement-Bitmap

33.2.2 Hinzufügen von Eigenschaften

Fügen Sie Ihrem Steuerelement nun die Eigenschaften `Selected` und `Shape` hinzu.

1. Rufen Sie dazu den Klassen-Assistenten auf, und selektieren Sie das Register `Automatisierung`.
2. Wählen Sie aus dem Listefeld unter `KLASSENNAME` die Klasse `CMCTLCtrl` aus (Abbildung 33.8).
3. Betätigen Sie die Schaltfläche `EIGENSCHAFT HINZUFÜGEN`.

Abbildung 33.8:
Hinzufügen von
Eigenschaften



4. Geben Sie in dem anschließend dargestellten Dialog in dem Eingabefeld neben `EXTERNER NAME` die Bezeichnung `Selected` ein. Wählen Sie aus dem Listefeld neben `TYP` den Eintrag `BOOL` aus, und aktivieren Sie die Option `Get/Set-Methoden` unter `IMPLEMENTIERUNG`. Der Dialog gibt nun die Namen der `Get-/Set-Funktionen` vor. Löschen Sie die Bezeichnung der `Set-FUNKTION`, damit die Eigenschaft schreibgeschützt ist (Abbildung 33.9). Betätigen Sie anschließend die Schaltfläche `OK`.
5. Das Hinzufügen der `Shape`-Eigenschaft geschieht in einer ähnlichen Weise. Der Typ der Eigenschaft ist jedoch `short` und die Bezeichnung der `Set-Funktion` darf nicht gelöscht werden, da die Eigenschaft sowohl gesetzt als auch ausgelesen werden kann.

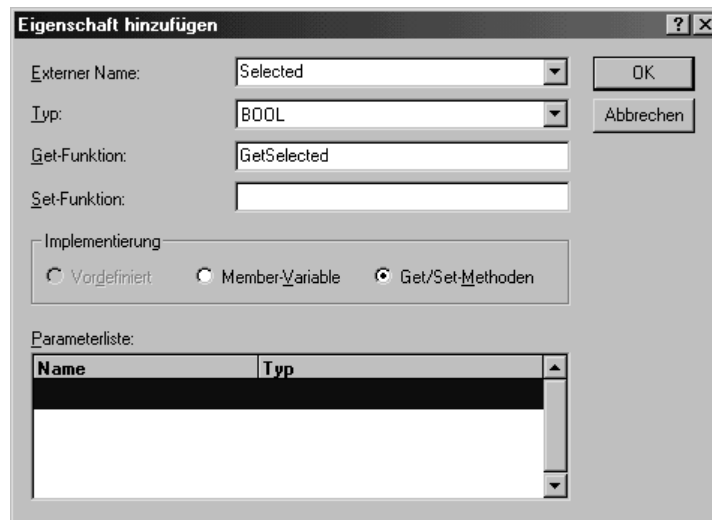


Abbildung 33.9:
Hinzufügen der
Selected-Eigen-
schaft

Wenn Sie die Definition der Eigenschaft mit einem Klick auf die Schaltfläche OK abschließen, generiert der Klassen-Assistent drei Elementfunktionen in der Steuerelementklasse.

6. Um den Status des Steuerelements wiedergeben und die Figur darstellen zu können, werden wir nun der Klassendeklaration des Steuerelements manuell einige `public`-Elementvariablen hinzufügen. Dies geschieht in der Deklaration der Klasse `CMCTLCtrl` in der Header-Datei `MCTLCtrl.h`:

```
BOOL m_bSelected;
short m_nShape;
```

7. Die Elementvariablen müssen außerdem mit angemessenen Voreinstellungen in dem Konstruktor der Steuerelementklasse initialisiert werden. Schreiben Sie dazu die folgenden Programmzeilen in `CMCTLCtrl::CMCTLCtrl`:

```
m_bSelected = FALSE;
m_nShape = 0;
```

8. Mit diesen Elementvariablen können wir die Implementierung der Get-/Set-Funktionen bearbeiten, für die der Assistent ein Gerüst generiert hat. Die vollständigen Funktionen sind in Listing 33.7 aufgeführt.

```
BOOL CMCTLCtrl::GetSelected()
{
    // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
    // Eigenschaften hinzu
    return m_bSelected;
}
short CMCTLCtrl::GetShape()
```

Listing 33.7:
CMCTLCtrl-Get-/
Set-Methoden

```

{
    // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
    // Eigenschaften hinzu
    return m_nShape;
}

void CMCTLCtrl::SetShape(short nNewValue)
{
    // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
    // Eigenschaften hinzu
    if (AmbientUserMode())
        ThrowError(CTL_E_SETNOTSUPPORTEDATRUNTIME);
    else if (nNewValue < 0 || nNewValue > 2)
        ThrowError(CTL_E_INVALIDPROPERTYVALUE);
    else
    {
        m_nShape = nNewValue;
        SetModifiedFlag();
        InvalidateControl();
    }
}

```

Die Funktionen `GetSelected` und `GetShape` geben lediglich die Werte der Elementvariablen `m_bSelected` und `m_nShape` zurück.

Die Funktion `SetSelected` verwendet die `AmbientUserMode`-Funktion, um zu prüfen, ob sich der Steuerelement-Container im Anwendermodus oder Entwurfsmodus befindet. `SetSelected` gewährleistet, daß die Figur des Steuerelements nicht im Anwendermodus verändert wird. Beachten Sie bitte die `ThrowError`-Funktion. Diese Elementfunktion von `COLEControl` zeigt während des Zugriffs auf eine Eigenschaft über die `Get-/Set`-Methoden einen Fehler an.

Der Anwendungsrahmen verfügt über einige STANDARD EIGENSCHAFTEN. Dazu zählen beispielsweise Eigenschaften, über die die Schriftart und die Farbe bestimmt werden. Wenn Sie eine Standardeigenschaft verwenden möchten, können Sie die in `COLEControl` enthaltenen Eigenschaften nutzen und müssen keine eigenen Eigenschaften definieren.

33.2.3 Definieren einer beständigen Eigenschaft

Haben Sie dem Steuerelement eine Eigenschaft hinzugefügt, die Sie während der Entwurfszeit setzen, wird diese nicht automatisch gespeichert. Rufen Sie dazu eine `PX_-`Funktion in der `DoPropExchange`-Elementfunktion der Steuerelementklasse auf.

DoPropExchange Was macht die Funktion `DoPropExchange`? Sie serialisiert die Werte von Eigenschaften in einem Eigenschaftenaustauschobjekt. Dadurch, daß der Wert einer Eigenschaft in solch einem Objekt gespeichert und daraus eingelesen wird, bleibt die Eigenschaft beständig. Der Dialog-Editor des Visual Studio schreibt beispielsweise die Inhalte des Eigenschaftenaustauschobjekts in die Ressourcendatei der Anwendung. Diese wird zur Laufzeit ausgelesen.

Für jede beständige Eigenschaft muß ein entsprechender PX_-Funktionsaufruf in die DoPropExchange-Funktion eingefügt werden. Verschiedene PX_-Funktionen werden für unterschiedliche Eigenschaftstypen aufgerufen. Tabelle 33.1 führt diese Funktionen auf.

Funktionsname	Eigenschaftstyp
PX_Blob	BLOB-Daten (Binary Large Object).
PX_Bool	Boolescher Wert (Typ BOOL).
PX_Color	Farbwert (Typ OLE_COLOR).
PX_Currency	Währungswert.
PX_Double	Wert vom Typ double.
PX_Float	Wert vom Typ float.
PX_Font	Eine Schriftart (Zeiger auf eine FONTDESC-Struktur).
PX_IUnknown	Ein Objekt mit einer von IUnknown abgeleiteten Schnittstelle.
PX_Long	Wert vom Typ long.
PX_Picture	Ein Bild (CPictureHolder-Verweis).
PX_Short	Wert vom Typ short.
PX_String	Ein Zeiger auf eine Zeichenfolge (Typ LPCSTR).
PX_ULong	Wert vom Typ long ohne Vorzeichen (ULONG).
PX_UShort	Wert vom Typ short ohne Vorzeichen.
PX_VBXFontConvert	Eigenschaften, die sich auf die Schriftart eines VBX-Steuerelements beziehen.

Tabelle 33.1:
PX_-Funktionen

In der MCTL-Anwendung soll die Shape-Eigenschaft beständig sein. Diese ist vom Typ short. Der CMCTLCtrl::DoPropExchange-Funktion muß daher die folgende Zeile hinzugefügt werden:

```
PX_Short(pPX, _T("Shape"), m_nShape, 0);
```

33.2.4 Hinzufügen von Methoden

Das Hinzufügen von Methoden zu einem ActiveX-Steuerelement gleicht der Vorgehensweise für Eigenschaften. Auch Methoden werden mit Hilfe des Klassen-Assistenten und dem Register AUTOMATISIERUNG definiert.

Drei Methoden sind bereits in der Steuerelementklasse vorhanden. Dabei handelt es sich um die Get-/Set-Methoden der beiden Eigenschaften.

Einer Methode können mehrere Parameter übergeben werden. Diese werden ebenfalls mit dem Klassen-Assistenten definiert.

Die `ColeControl`-Klasse unterstützt zwei Standardereignisse:

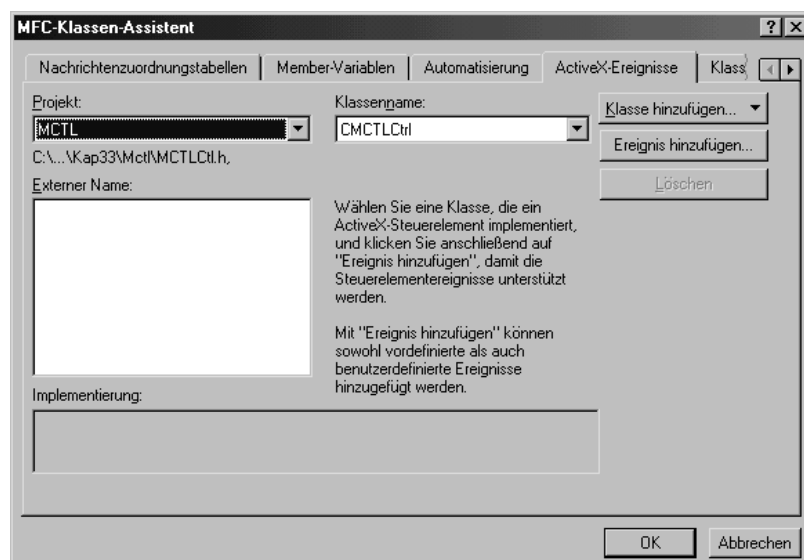
- ➡ `DoClick` (löst ein Click-Ereignis aus) und
- ➡ `Refresh` (aktualisiert die Darstellung des Steuerelements).

33.2.5 Hinzufügen von Ereignissen

Auch Ereignisse werden dem ActiveX-Steuerelement über den Klassen-Assistenten hinzugefügt.

- Öffnen Sie dazu in dem Dialog des Assistenten das Register **ACTIVE-X-EREIGNISSE** (Abbildung 33.10).

Abbildung 33.10:
Hinzufügen von
ActiveX-
Ereignissen



- Betätigen Sie in dem Register die Schaltfläche **EREIGNIS HINZUFÜGEN**, um das `Select`-Ereignis zu definieren.
- Geben Sie in dem anschließend dargestellten gleichlautenden Dialog den externen Ereignisnamen `Select` ein. Der Klassen-Assistent gibt daraufhin automatisch den internen Namen `FireSelect` vor (den Namen der Funktion, die das Ereignis auslöst). Definieren Sie außerdem den Parameter `IsSelected` zu diesem Ereignis (dieser Parameter teilt dem Empfänger des Ereignisses mit, ob das Steuerelement selektiert oder deselektiert wurde). Der Typ des Parameters soll `BOOL` sein (Abbildung 33.11).

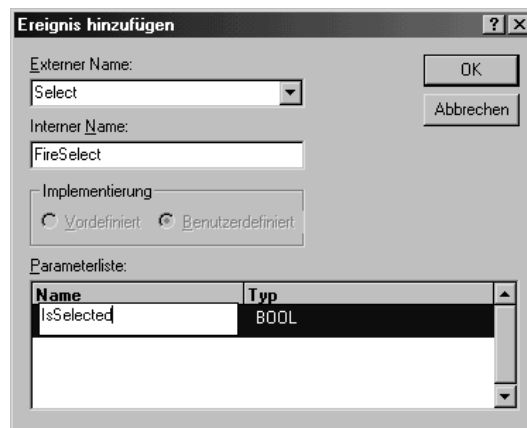


Abbildung 33.11:
Hinzufügen des
Select-Ereignis-
ses

Wie ich bereits erwähnte, wird dieses Ereignis ausgelöst, wenn sich der Status des Steuerelements ändert. Dies geschieht, wenn der Anwender über dem Steuerelement mit der linken Maustaste klickt.

4. Mit dem Klassen-Assistenten fügen Sie der `CMCTLCtrl`-Klasse einen Bearbeiter für die `WM_LBUTTONDOWN`-Nachricht hinzu. In der in Listing 33.8 aufgeführten Bearbeiterfunktion kehren wir den Wert der Booleschen Variable `m_bSelected` um, und lösen für den neuen Wert ein Select-Ereignis aus. Wir gewährleisten außerdem, daß das Steuerelement mit der Farbe gezeichnet wird, die dem Status entspricht. Dazu rufen wir die `InvalidateControl`-Funktion auf.

```
void CMCTLCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten hier
    // einfügen und/oder Standard aufrufen
    CObjectControl::OnLButtonDown(nFlags, point);
    m_bSelected = !m_bSelected;
    InvalidateControl();
    FireSelect(m_bSelected);
}
```

Listing 33.8:
Bearbeiten eines
Mausereignisses
in MCTL

33.2.6 Zeichnen des Steuerelements

Wenn die `OnDraw-Elementfunktion` der Steuerelementklasse aufgerufen wird, nimmt sie einen Zeiger auf ein Gerätekontextobjekt und ein Rechteck entgegen, das die Grenzen des Steuerelements beschreibt. Mit diesen Informationen ist das Zeichnen des Steuerelements sehr einfach.

Gewöhnlich hängt die Darstellung des Steuerelements von den Werten einiger Elementvariablen ab, die den Status des Steuerelements repräsentieren. Diese Aussage gilt ebenfalls für das MCTL-Steuerelement.

Dort bestimmt die `m_nShape`-Elementvariable die darzustellende Figur. Die `m_bSelected`-Elementvariable definiert die Farbe des Steuerelements. Listing 33.9 zeigt die Implementierung der MCTL-Zeichenfunktion.

Listing 33.9: *CMCTLCtrl::OnDraw-Elementfunktion*

```
void CMCTLCtrl::OnDraw(CDC* pdc, const CRect& rcBounds,
                      const CRect& rcInvalid)
{
    // ZU ERLEDIGEN: Folgenden Code durch eigene Zeichenfunktion
    // ersetzen.
    // pdc->FillRect(rcBounds,
    //      CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    // pdc->Ellipse(rcBounds);
    CPen pen;
    CBrush foreBrush, backBrush;
    CPoint points[3];
    pdc->SaveDC();
    pen.CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
    backBrush.CreateSolidBrush(
        TranslateColor(AmbientBackColor()));
    foreBrush.CreateSolidBrush(GetSelected() ? RGB(255, 0, 0) :
        RGB(0, 255, 0));

    pdc->FillRect(rcBounds, &backBrush);
    pdc->SelectObject(&pen);
    pdc->SelectObject(&foreBrush);
    switch (m_nShape)
    {
    case 0:
        pdc->Ellipse(rcBounds);
        break;
    case 1:
        pdc->Rectangle(rcBounds);
        break;
    case 2:
        points[0].x = rcBounds.left;
        points[0].y = rcBounds.bottom - 1;
        points[1].x = (rcBounds.left + rcBounds.right - 1) / 2;
        points[1].y = rcBounds.top;
        points[2].x = rcBounds.right - 1;
        points[2].y = rcBounds.bottom - 1;
        pdc->Polygon(points, 3);
        break;
    }
    pdc->RestoreDC(-1);
}
```

Beachten Sie die Funktion `AmbientBackColor`. Dies ist eine der Funktionen, die die Umgebungseigenschaften des Steuerelement-Containers ermitteln. In unserem Beispiel gewährleistet die Funktion, daß die Hintergrundfarbe des Steuerelements der Hintergrundfarbe des Steuerelement-Containers entspricht. Die `TranslateColor`-Funktion konvertiert eine OLE-Farbe in einen RGB-Wert. Diese Vorgehensweise ist erforderlich, da die OLE-Farbe einen Palettenindex und keinen RGB-Wert repräsentiert.

33.3 Hinzufügen eines Eigenschaftendialogs

Wenngleich die Implementierung unseres neuen Steuerelements abgeschlossen ist, sind wir noch nicht am Ziel angelangt. Wir müssen außerdem eine oder mehrere Eigenschaftenseiten definieren, über die auf die Eigenschaften unseres Steuerelements zugegriffen werden kann. Diese Eigenschaftenseiten werden von Anwendungen (wie z. B. den Dialog-Editor des Visual Studio) gewöhnlich im Entwurfsmodus verwendet.

Die dazu notwendige Vorgehensweise unterscheidet sich nicht von der zur Implementierung eines gewöhnlichen Dialogs. Wir entwerfen zunächst die visuelle Darstellung der Eigenschaftenseite und schreiben im Anschluß daran den erforderlichen Programmcode, der die Steuerelemente auf der Eigenschaftenseite mit den Eigenschaften des Steuerelements verknüpft.

33.3.1 Bearbeiten der Eigenschaftenseite

Öffnen Sie die Dialogvorlage der Eigenschaftenseite, um deren visuelle Darstellung zu modifizieren. Das Bearbeiten des Dialogs unterscheidet sich nicht von der Bearbeitung gewöhnlicher Dialoge.

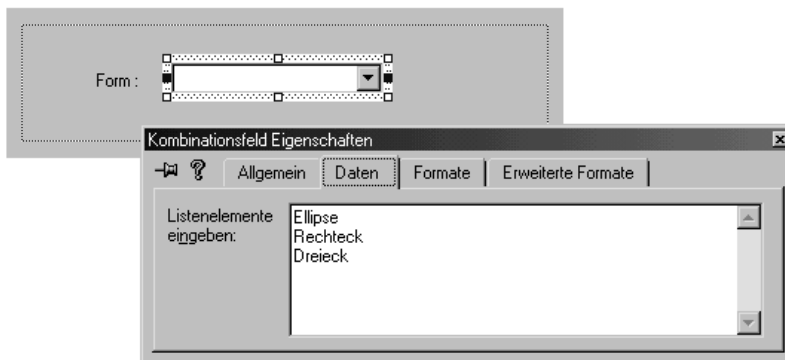


Abbildung 33.12:
Bearbeiten der
MCTL-Eigen-
schaftenseite

Der einzigen Eigenschaftenseite von MCTL sollen lediglich zwei Steuerelemente hinzugefügt werden:

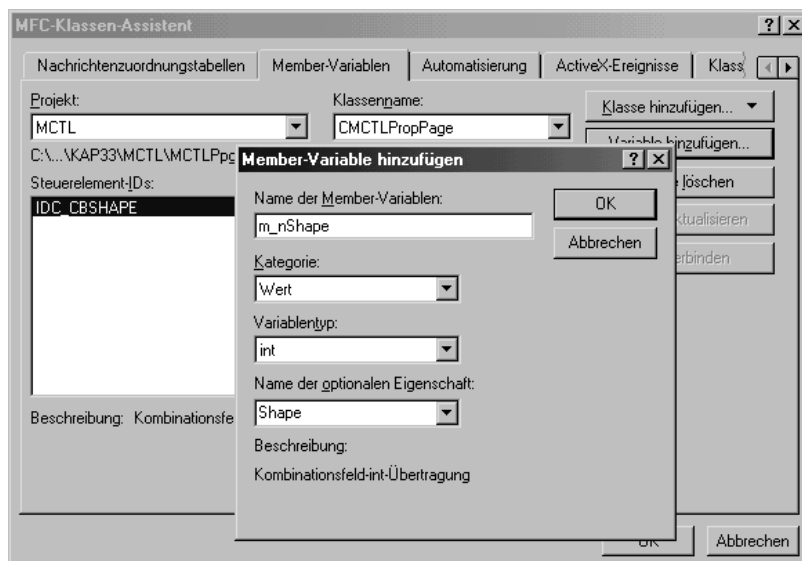
- ➡ ein statisches Textfeld und
- ➡ ein Kombinationsfeld.

Definieren Sie für das Kombinationsfeld den Bezeichner IDC_CBSHAPE. Des weiteren muß das Kombinationsfeld auf der Eigenschaftenseite FORMATE als vom Typ DROPDOWN-LISTENFELD eingestellt werden (andernfalls können Sie diesem Steuerelement keine Elementvariable vom Typ `int` mit Hilfe des Klassen-Assistenten hinzufügen). Deaktivieren Sie die Option SORTIEREN, aktivieren Sie die Option VERT. BILDLAUF. Der Dialog-Editor kann ebenfalls zur Eingabe der Listenelemente verwendet werden (Abbildung 33.12). Möchten Sie mehrere Werte in den Bereich der Listenelemente eingeben, erzeugen Sie mit der Tastenkombination Strg + Enter einen Zeilenumbruch. Betätigen Sie lediglich die Taste Enter, wird der Eigenschaftendialog geschlossen.

33.3.2 Verknüpfen der Eigenschaftenseiten-Steuerelemente mit dem ActiveX-Steuerelement

Fügen Sie mit Hilfe des Klassen-Assistenten dem ActiveX-Steuerelement eine Elementvariable für das neue Eigenschaftenseiten-Steuerelement hinzu.

Abbildung 33.13:
Verknüpfen
eines Steuerelements mit einer
Eigenschaft



Die dazu erforderliche Vorgehensweise unterscheidet sich nicht von der Definition einer gewöhnlichen Elementvariable. Lediglich der Umstand, daß die Elementvariable (die ein Element der Klasse `CMCTLPropPage` ist) mit der entsprechenden Eigenschaft des `MCTL`-Steuerelements verknüpft werden muß, differiert von dieser Vorgehensweise.

Sie erstellen diese Verknüpfung, indem Sie den Namen der ActiveX-Eigenschaft in dem Klassen-Assistenten, Dialog MEMBER-VARIABLE HINZUFÜGEN, angeben (Abbildung 33.13).

Der Klassen-Assistent fügt daraufhin, zusätzlich zu den DDX_- und DDV_- Funktionsaufrufen, einige DDP_-Funktionsaufrufe in die DoDataExchange-Elementfunktion der Eigenschaftenseite ein. Die folgenden Zeilen werden erzeugt:

```
DDP_CBIndex(pDX, IDC_CBSHAPE, m_nShape, _T("Shape"));
DDX_CBIndex(pDX, IDC_CBSHAPE, m_nShape);
```

Verschiedene DDP_-Funktionen entsprechen den unterschiedlichen Steuerelementtypen. Die Funktionen sind in Tabelle 33.2 aufgeführt.

Funktionsname	Beschreibung
DDP_CBIndex	Kombinationsfeld, int-Transfer.
DDP_CBString	Kombinationsfeld, Zeichenfolgentransfer.
DDP_CBStringExact	Kombinationsfeld, Zeichenfolgentransfer.
DDP_Check	Kontrollkästchen.
DDP_LBIndex	Listenfeld, int-Transfer.
DDP_LBString	Listenfeld, Zeichenfolgentransfer.
DDP_LBStringExact	Listenfeld, Zeichenfolgentransfer.
DDP_Radio	Optionsfeld, int-Transfer.
DDP_Text	Textfeldtransfer (Zeichenfolgen, numerisch usw.)

Tabelle 33.2:
DDP_-Funktionen

Die Funktion DDP_PostProcessing sollte aufgerufen werden, nachdem der Transfer von Steuerelementeigenschaften über die DDP_-Funktionen ausgeführt wurde.

Betrachten Sie bitte einmal das Ergebnis. Abbildung 33.14 zeigt die Eigenschaftenseite des MCTL-Steuerelements als Register des Eigenschaftendialogs, der von dem Dialog-Editor des Visual Studio angezeigt wird, wenn einer Dialogvorlage ein MCTL-Steuerelement hinzugefügt wird.

Die Implementierung des MCTL-Steuerelements ist nun abgeschlossen.

Kompilieren Sie das Projekt, um die OCX-Datei zu erstellen. Während des Kompilierens wird das Steuerelement automatisch registriert.

Abbildung 33.14:
Der Eigenschaftendialog zum
MCTL-Steuerelement im Visual
Studio



33.3.3 Zusätzliche Eigenschaftenseiten

Möchten Sie Ihrem ActiveX-Steuerelement weitere Eigenschaftenseiten hinzufügen, generieren Sie diese mit dem Steuerelement-Assistenten. Um die entsprechende Klasse zu erstellen, erzeugen Sie zunächst eine Dialogvorlage und generieren anschließend mit Hilfe des Klassen-Assistenten eine neue von `COlePropertyPage` abgeleitete Klasse.

Nutzen Sie die Standardeigenschaften zum Setzen der Farbe, Schriftart und zum Einrichten des Bildes, können Sie ebenfalls Standard-Eigenschaftenseiten verwenden, über die Sie auf die Standardeigenschaften zugreifen. Bestimmen Sie dazu den Klassenbezeichner der Standard-Eigenschaftenseite, wenn Sie der Implementierungsdatei der Steuerelementklasse Eigenschaftenseiten hinzufügen. Sie müssen eine Klasse für eine Eigenschaftenseite nicht selbst erstellen.

33.3.4 Testen, Distribuieren und Verwenden eines benutzerdefinierten Steuerelements

Das Testen eines ActiveX-Steuerelements ist ein wichtiger Abschnitt des Entwicklungsprozesses. Nicht alle Steuerelementprojekte sind derart einfach, wie das MCTL-Steuerelement.

ActiveX-Steuerelemente werden häufig als wiederverwendbare Komponenten von einem Programmierer erstellt und von anderen Anwendungsprogrammierern genutzt. Sie sollten daher erfahren, wie ein Steuerelement an diese Anwender distribuiert wird. Lassen Sie uns jedoch zunächst das Testen des Steuerelements erörtern.

33.3.5 Testen eines ActiveX-Steuerelements

Sie müssen nicht Ihre eigene Steuerelement-Container-Anwendung schreiben, um ein ActiveX-Steuerelement zu testen. Statt dessen können Sie die ActiveX-Steuerelement-Container-Anwendung verwenden, die mit Visual C++ ausgeliefert wird. Wenn Sie aus dem Menü PROJEKT den Eintrag EINSTELLUNGEN auswählen, können Sie in dem daraufhin angezeigten Dialog (Register DEBUG) die Datei TSTCON32.EXE als

ausführbares Programm für die Debug-Sitzung angeben. Auf diese Weise wird es Ihnen möglich sein, Ihr Steuerelement zu prüfen, Haltepunkte zu setzen und die Ausführung über den Visual-C++-Debugger zu überwachen.

Sie können Ihr ActiveX-Steuerelement außerdem testen, indem Sie es innerhalb des Dialog-Editors verwenden.

33.3.6 Distribution von ActiveX-Steuerelementen

Möchten Sie Ihr ActiveX-Steuerelement distribuieren, sollten Sie sich einige Fragen stellen. Welche Dateien müssen Sie mit dem Steuerelement an die Anwender ausliefern, die Ihr ActiveX-Steuerelement verwenden? Welche Dateien müssen von diesen Anwendern an die Endanwender distribuiert werden? Wie muß das Steuerelement installiert werden, damit es korrekt registriert wird?

Der Endanwender benötigt lediglich die Datei mit der Endung .OCX. Diese Datei ist die DLL, die den Programmcode des Steuerelements enthält. Entwickler benötigen außerdem die TLB-Datei (Typbibliothek), die EXP-Datei und die LIB-Datei, die während des Kompilierens erstellt werden.

Das Steuerelement wird mit dem Server-Registrierungsprogramm von Microsoft registriert:

```
\\windows\\system\\regsvr32.exe /s <steuerelement>.ocx
```

Die Datei *REGSVR32.EXE* kann ohne Einschränkungen weitergegeben werden. Wenn Sie ein Setup-Programm schreiben, das die Installation eines ActiveX-Steuerelements ausführt, möchten Sie diese Datei möglicherweise verwenden, um das Steuerelement zu registrieren. Sie können jedoch ebenfalls die OCX-Datei als DLL laden und deren `DllRegisterServer`-Funktion aufrufen.

Lizenzieren Sie Ihr Steuerelement, sollten Sie dem Anwendungsprogrammierer die Lizenzdatei (LIC) überlassen. Der Endanwender erhält diese Datei nicht.

33.3.7 Verwenden von ActiveX-Steuerelementen in Anwendungen

Nachfolgend finden Sie eine Übersicht aufgeführt, die beschreibt, wie ein ActiveX-Steuerelement in einer Anwendung eingesetzt wird:

- ➡ Rufen Sie während der Initialisierung der Anwendung die Funktion `AfxEnableControlContainer` auf, damit die Anwendung ActiveX-Steuerelemente unterstützt.

- ➔ Fügen Sie das benutzerdefinierte Steuerelement der Dialogvorlage hinzu.
- ➔ Setzen Sie die anfänglichen Eigenschaften in dem Eigenschaftendialog des Steuerelements.
- ➔ Fügen Sie der Anwendungsklasse Elementvariablen hinzu, die das Steuerelement oder dessen Eigenschaften repräsentieren.
- ➔ Definieren Sie Nachrichtenbearbeiter, um Nachrichten von dem Steuerelement zu bearbeiten.

33.4 Zusammenfassung

Das Erstellen eines ActiveX-Steuerelements wurde mit Visual C++ zu einer einfachen Aufgabe.

Das Erstellen eines ActiveX-Steuerelements wird wie folgt ausgeführt:

- ➔ Erstellen des ActiveX-Steuerelementgerüsts mit dem Steuerelement-Assistenten.
- ➔ Hinzufügen der Eigenschaften, Methoden und Ereignisse mit dem Klassen-Assistenten.
- ➔ Überarbeiten der Steuerelement-Bitmap.
- ➔ Hinzufügen des Zeichencodes.
- ➔ Bearbeiten des Programmcodes der Methoden und Ereignisse.
- ➔ Erstellen der Eigenschaftenseite des Steuerelements.
- ➔ Hinzufügen von Programmcode zur Bearbeitung der Eigenschaftenseite.

ActiveX-Steuerelemente können mit dem ActiveX-Container getestet werden, der mit Visual C++ ausgeliefert wird.

Das Distribuieren eines Steuerelements an den Endanwender erfordert die Weitergabe der OCX-Datei. Entwickler benötigen die LIB-, TLB- und EXP-Dateien. Wird das Steuerelement lizenziert, benötigen die Entwickler die Lizenzdatei (LIC).

Das Steuerelement muß registriert werden, bevor es verwendet wird. Dies geschieht mit Hilfe der Datei REGSVR32.EXE, die ohne Einschränkungen distribuiert werden kann. Alternativ dazu können Sie die DllRegisterServer-Funktion aus dem Setup-Programm aufrufen.

Verwenden der ActiveX- Templatebibliothek

Kapitel

34

Microsofts ActiveX-Vorlagenbibliothek (ATL – ActiveX Template Library) ist eine neue Klassenbibliothek für den C++-Programmierer. Die Bibliothek kapselt das Komponentenobjektmodell (COM), das über verschiedene C++-Klassenvorlagen repräsentiert wird. Diese Klassen bilden die COM-Objekttypen. Eine spezielle Unterstützung besteht für COM-Schlüsselschnittstellen, wie zum Beispiel `IUnknown`, `IClassFactory` oder `IDispatch`. ATL bildet daher die ideale Basis für ActiveX-Steuerelemente oder Automatisierungs-Server-Projekte.

ATL ist eine sehr große und komplexe Bibliothek. Eine vollständige Erläuterung würde mehrere Kapitel oder ein vollständiges Buch umfassen. Dieses Kapitel bietet somit lediglich eine kurze Übersicht über die Funktionen der ATL. Außerdem erstellen Sie ein ActiveX-Steuerelement, dessen Funktionalität der des in dem vorherigen Kapitel vorgestellten MFC-ActiveX-Steuerelements entspricht. Auf diese Weise können Sie die beiden Projekte miteinander vergleichen und entscheiden, welche Art der Erstellung für Sie geeignet ist.

34.1 Warum ATL?

Ist die ATL eine Alternative zur MFC? Unter welchen Umständen würden Sie die ATL in Ihren Projekten verwenden? Können die ATL und die MFC koexistieren oder schließen sie sich gegenseitig aus? Welche Funktionalität wird ausschließlich von der ATL angeboten? Diese Fragen stellen sich, wenn die ATL erstmals verwendet wird.

Die ATL ist sicherlich eine Alternative zur MFC, wenn Sie ActiveX-Steuerelemente erstellen. Sie stellt eine weitaus detailliertere und flexiblere Implementierung des Komponentenobjektmodells als die MFC

zur Verfügung. Die ATL ersetzt die MFC jedoch nicht. Die MFC wird in umfassenden und komplexen Windows-Anwendungen eingesetzt, in denen das Komponentenobjektmodell lediglich eines der Features ist. Die ATL ist jedoch auf das Komponentenobjektmodell beschränkt und bietet lediglich einige zusätzliche Dialoge und die Fensterunterstützung an.

ATL-Code ist schlanker, was für die Distribution übers Internet äußerst interessant ist

Ein entscheidender Vorteil der ATL ist der geringe Programmcode, der sich ergibt, wenn diese Bibliothek verwendet wird. Sogar das einfache Beispiel dieses Kapitels demonstriert diesen Umstand sehr anschaulich. Lassen Sie sich von der geringen Größe einer ausführbaren MFC-Datei oder MFC-Bibliothek nicht täuschen. Diese resultiert aus dem Verknüpfen der Dateien mit der MFC-DLL. Der MFC-Code befindet sich somit nicht in der ausführbaren Datei oder Bibliothek, sondern in den zu distribuierenden MFC-Dateien. Dies bedeutet, daß möglicherweise mehrere Megabytes Bibliothekdateien mit einem Steuerelement oder Automatisierungsobjekt ausgeliefert werden müssen. Für den Diskettenvertrieb und die Verwendung auf WWW-Seiten im Internet ist die MFC-Version somit nicht in jedem Fall geeignet.

Natürlich können MFC-Projekte mit der statischen Version der MFC-Bibliothek kompiliert werden. Die ausführbare Datei wird dadurch jedoch sehr viel größer. Das MFC-ActiveX-Steuerelement des letzten Kapitels generiert eine ausführbare Datei, deren Größe ca. 150 Kbyte beträgt. Die ATL-Implementierung des gleichen Steuerelements beansprucht lediglich ein Drittel dieser Größe. Dieser Unterschied spricht für sich selbst.

Ein Grund, der gegen die Verwendung der ATL spricht, ist die mangelnde Integration dieser Bibliothek im Visual Studio. Im Gegensatz zur ATL wird die MFC über die Features des Klassen-Assistenten unterstützt. Die Unterstützung der ATL ist auf den ATL-COM-Anwendungsassistenten und den Eintrag NEUES ATL-OBJEKT im Menü EINFÜGEN begrenzt. Der Assistent generiert ein ATL-Anwendungsgerüst und der Menüeintrag fügt ein ATL-Objekt in Ihr Projekt ein.

Die ATL und die MFC können nebeneinander bestehen. Sie können die beiden Bibliotheken jedoch nicht gemeinsam verwenden. Konflikte entstehen, wenn Sie versuchen, ATL- und MFC-ActiveX-Klassen in demselben Projekt zu verwenden. Sie sollten außerdem darauf achten, die mit der ATL generierten Windows-Objekte (zum Beispiel eine Eigenschaftenseite oder ein Steuerelementfenster) nicht mit MFC-Klassen zu verwalten.

34.2 Erstellen eines ActiveX-Steuerelements mit der ATL

Erfahren Sie nun, wie ein ActiveX-Steuerelement mit der ATL erstellt wird. Nachfolgend finden Sie eine kurze Zusammenfassung der dazu notwendigen Schritte:

- Ein ATL-COM-Projektgerüst wird mit Hilfe des ATL-COM-Anwendungsassistenten erstellt.
- Dem Projekt wird ein Steuerelementobjekt hinzugefügt.
- Die Eigenschaften und Methoden des Steuerelements werden definiert.
- Der Programmcode zum Zeichnen des Steuerelements wird geschrieben.
- Dem Steuerelement werden die gewünschten Ereignisse hinzugefügt.
- Die Eigenschaftenseiten werden definiert.
- Die Bitmap des Steuerelements wird gezeichnet.

Das Steuerelement, das Sie in diesem Kapitel erstellen werden, ist in Abbildung 34.1 dargestellt. Es besteht aus einer Schaltfläche, die ihre Farbe ändert, wenn der Anwender darauf klickt. Das Steuerelement verfügt über drei Figuren (Rechteck, Ellipse und Dreieck). Die Figur wird über eine Eigenschaft bestimmt, die lediglich zur Entwurfszeit gesetzt werden kann.

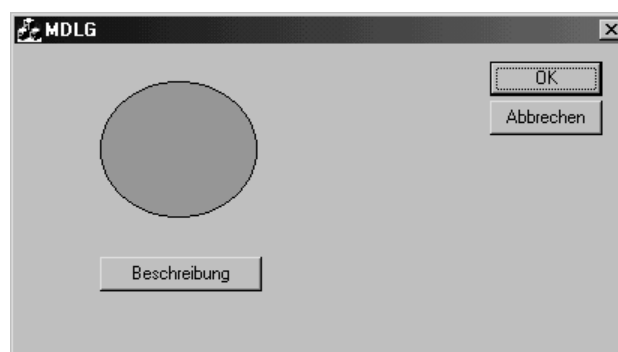
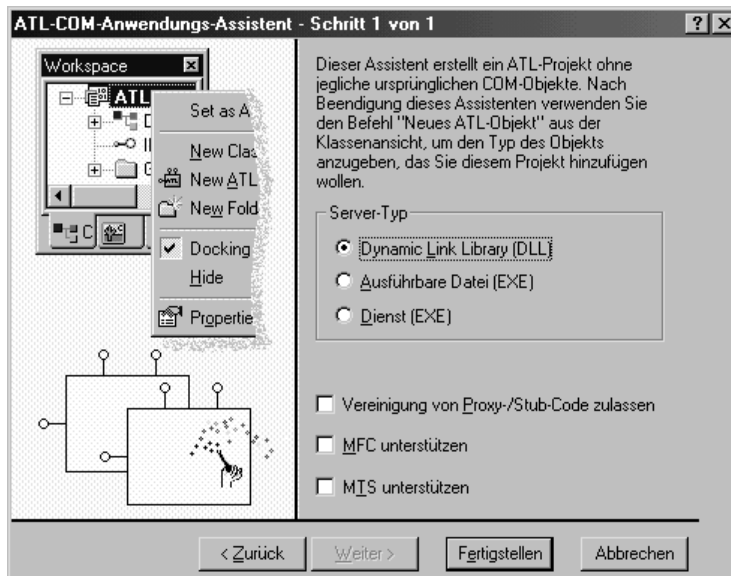


Abbildung 34.1:
Das ATL-Steuerelement im
MDLG-Dialog

34.2.1 Erstellen des ATL-COM-Projektgerüsts

1. Rufen Sie zunächst den ATL-COM-Anwendungsassistenten auf, um das Projektgerüst generieren zu lassen. Öffnen Sie dazu den Dialog NEU aus dem Menü DATEI.
2. Wählen Sie auf der Registerseite PROJEKTE den Projekttyp ATL-COM-ANWENDUNGS-ASSISTENT aus. Geben Sie die Projektbezeichnung AC und das gewünschte Projektverzeichnis ein. Das Steuerelement, das wir erstellen werden, soll den Namen AC erhalten.
3. Betätigen Sie die Schaltfläche OK, um sich den Dialog des ATL-COM-Anwendungsassistenten anzeigen zu lassen, der in Abbildung 34.2 dargestellt ist.

Abbildung 34.2:
Erstellen eines
Projekts mit
dem ATL-COM-
Anwendungsas-
sistenten



4. Übernehmen Sie die Voreinstellungen dieses Dialogs für unser Projekt. Das Steuerelement wird ein Prozeß-Server, also eine DLL sein. Das Vereinigen von Proxy- und Stub-Code ist in dieser DLL nicht erforderlich. Außerdem verzichten wir auf die Unterstützung der MFC. Klicken Sie somit auf die Schaltfläche FERTIGSTELLEN.

Wenn Sie sich das Projekt in der Klassen-Ansicht ansehen (Abbildung 34.3), werden Sie feststellen, daß wesentliche Unterschiede zu einem MFC-Projekt bestehen.

In dem Projektgerüst sind keine Klassen definiert!



Abbildung 34.3:
Das ATL-COM-
Projekt in der
Klassen-Ansicht

Wir verfügen lediglich über einige Funktionen, die DLL-Standardfunktionsaufrufe implementieren. Diese Funktionen befinden sich in der Quelldatei `AC.CPP`.

`AC.IDL` enthält die Typenbibliotheksinformationen zu unserem Projekt.

Der ATL-COM-Anwendungsassistent erzeugt einige weitere Dateien, wie zum Beispiel ein beinahe leeres Ressourcen-Skript und eine Standardmodul-Definitionsdatei, die exportierte Funktionen spezifiziert.

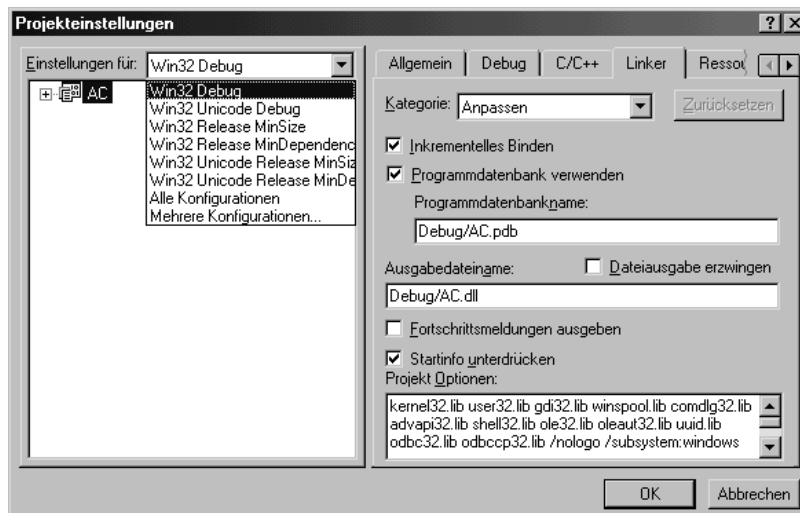


Abbildung 34.4:
ATL-COM-Projekt-
einstellungen

Betrachten Sie als nächstes die Projektkonfiguration, die der ATL-COM-Anwendungsassistent generiert hat (Abbildung 34.4). Wählen Sie dazu den Eintrag `EINSTELLUNGEN` aus dem Menü `PROJEKT` aus.

Wie Sie sehen, sind zwei Debug- und vier Release-Konfigurationen vorhanden. Sowohl die Debug- als auch die Release-Konfigurationen unterstützen Unicode. Selektieren Sie die Release-Konfiguration, können Sie sich zwischen dem Minimieren der Größe des erzeugten Moduls und dem Minimieren der Abhängigkeiten dieses Moduls entscheiden. Ein minimiertes ATL-COM-Modul benötigt die Bibliothek *AC.DLL*, um ausgeführt werden zu können.

In dem in Abbildung 34.4 dargestellten Dialog PROJEKTEINSTELLUNGEN ist das Register LINKER geöffnet und die Kategorie ANPASSEN ausgewählt. Hier sehen Sie, daß die Ausgabedatei des Projekts die Bezeichnung *AC.DLL* trägt. Wenngleich dieser Name technisch korrekt ist, müssen Sie für ActiveX-Steuerelemente die Dateiendung *.OCX* verwenden.



Denken Sie daran, die Endung der Ausgabedatei für alle sechs Konfigurationen zu ändern. Sie sollten außerdem die Moduldefinitionsdatei modifizieren (AC.DEF), so daß diese die Änderung berücksichtigt.

34.2.2 Hinzufügen eines Steuerelements

1. Um dem Projekt ein Steuerelement hinzuzufügen, wählen Sie aus dem Menü EINFÜGEN den Eintrag NEUES ATL-OBJEKT aus.
2. Selektieren Sie in dem anschließend dargestellten Dialog ATL-OBJEKT-ASSISTENT (Abbildung 34.5) den Eintrag STEUERELEMENTE.

Abbildung 34.5:
Einfügen eines
Steuerelements
mit dem ATL-
Objektassistenten



3. Entscheiden Sie sich für ein VOLLSTÄNDIGES STEUERELEMENT und klicken Sie auf WEITER >.

Im Anschluß daran erscheint der Dialog EIGENSCHAFTEN VON ATL-OBJEKT-ASSISTENT (Abbildung 34.6).

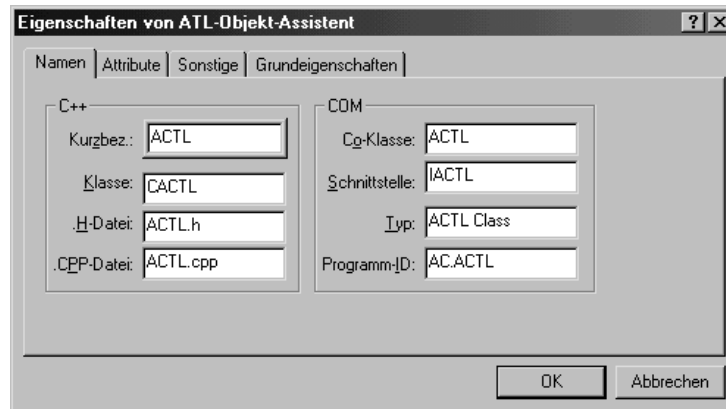


Abbildung 34.6:
Eigenschaften
des ATL-Objekt-
assistenten:
Namen

4. Hier bestimmen Sie die verschiedenen Attribute des neuen Steuerelements. Geben Sie in das Feld neben Kurzbezeichnung den Namen ACTL ein. Den verbleibenden Feldern des Dialogs werden daraufhin automatisch Werte zugewiesen, die dieser Kurzbezeichnung entsprechen.

Bevor wir den Dialog schließen, werden wir noch eine Einstellung auf der Registerseite ATTRIBUTE vornehmen.

5. Damit das Steuerelement die Fehlerbearbeitung unterstützt, aktivieren wir die Optionen ISUPPORTERRORINFO UNTERSTÜTZEN und VERBINDUNGSPUNKTE UNTERSTÜTZEN (Abbildung 34.7). Diese Einstellung ermöglicht unserem Steuerelement, Fehler und Ereignisse korrekt zu bearbeiten.

Die Einstellungen in den Registern SONSTIGE und GRUNDEIGENSCHAFTEN müssen nicht geändert werden.

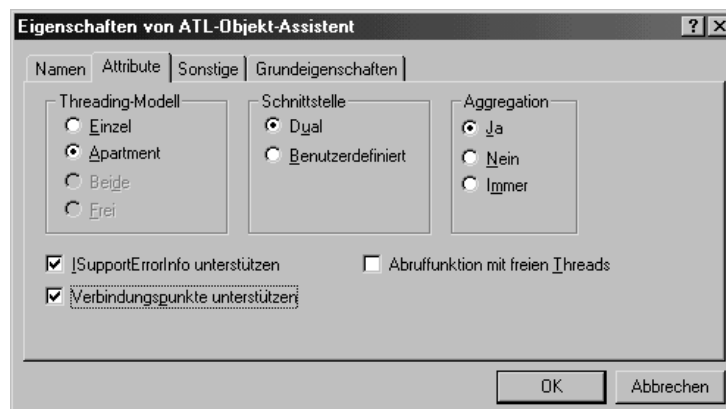


Abbildung 34.7:
Eigenschaften
des ATL-Objekt-
assistenten:
Attribute

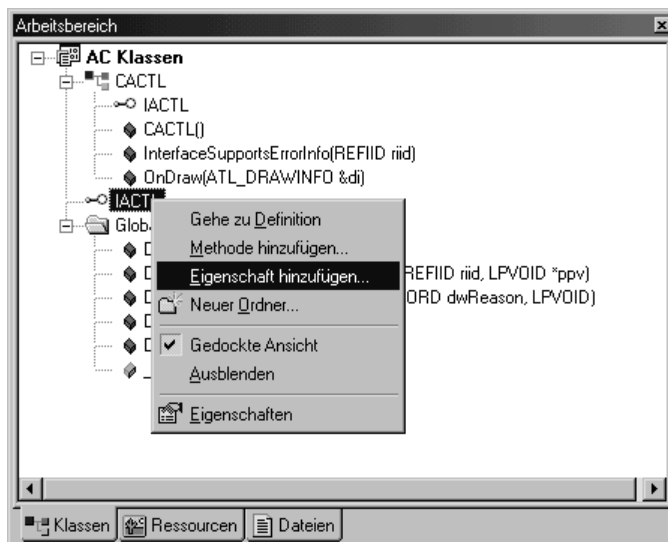
6. Sie beenden das Erstellen des Steuerelements, indem Sie die Schaltfläche OK betätigen.

Ein Blick auf das Projekt in der Klassen-Ansicht offenbart zwei neue Elemente:

1. eine neue Klasse (CACTL) und
2. eine neue COM-Schnittstelle (IACTL).

34.2.3 Hinzufügen von Eigenschaften und Methoden

Abbildung 34.8:
Hinzufügen von
COM-Eigen-
schaften mit
dem Visual
Studio



Sie fügen der COM-Schnittstelle mit Hilfe des Visual Studio Eigenschaften und Methoden hinzu, wie in Abbildung 34.8 dargestellt.

Hinzufügen der Eigenschaften

Das ACTL-Steuerelement soll zwei Eigenschaften erhalten.

- ➡ Die *Selected*-Eigenschaft zeigt an, ob das Objekt mit einem Mausklick darauf selektiert wurde.
 - ➡ Die *Shape*-Eigenschaft bestimmt die Figur des Steuerelements.
1. Wählen Sie aus dem Kontextmenü der IACTL-Schnittstelle den Eintrag EIGENSCHAFT HINZUFÜGEN aus. Es wird der Dialog EIGENSCHAFT ZU SCHNITTSTELLE HINZUFÜGEN aufgerufen.
 2. Setzen Sie die erste Eigenschaft mit der Bezeichnung *Shape* auf den Typ *short*.

- Die Eigenschaft `Selected` soll vom Typ `BOOL` und schreibgeschützt sein, da ihr Wert automatisch gesetzt wird, wenn der Anwender auf das Steuerelement klickt. Um der Eigenschaft dieses Attribut zuzuweisen, deaktivieren Sie die Option PUT-FUNKTION (Abbildung 34.9).

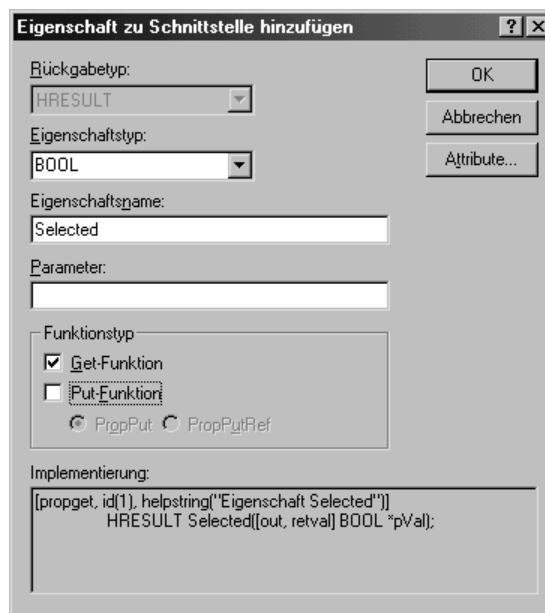


Abbildung 34.9:
Definieren einer
schreibge-
schützten Eigen-
schaft

Visual Studio fügt der `CACTL`-Klasse nun drei neue Elementfunktionen mit den Bezeichnungen `get_Shape`, `put_Shape` und `get_Selected` hinzu. Ihre Implementierung ist in der Datei `ACTL.CPP` enthalten. Unsere nächste Aufgabe besteht darin, diesen Funktionen Programmcode hinzuzufügen, um die gewünschte Funktionalität zu erzielen.

Dazu werden wir die beiden Dateien `ACTL.H` und `ACTL.CPP` modifizieren. `ACTL.H` enthält die Deklaration der Klasse `CACTL`.

- Fügen Sie an das Ende der Klassendeklaration die Deklarationen für die Variablen `m_bSelected` und `m_nShape` ein:

```
// IACTL
public:
    STDMETHOD(get_Selected)(/*[out, retval]*/ BOOL *pVal);
    STDMETHOD(get_Shape)(/*[out, retval]*/ short *pVal);
    STDMETHOD(put_Shape)(/*[in]*/ short newVal);
    HRESULT OnDraw(ATL_DRAWINFO& di);

protected:
    BOOL m_bSelected;
    short m_nShape;
};
```

5. Diese Variablen müssen in dem Klassenkonstruktor initialisiert werden. Der Konstruktor befindet sich in der Datei *ACTL.H*, zu Beginn der Klassendeklaration.

```
public:
    CACTL()
    {
        m_bSelected = FALSE;
        m_nShape = 0;
    }

    DECLARE_REGISTRY_RESOURCEID(IDR_ACTL)
```

Hinzufügen der Methoden

Wir werden den Funktionen `get_Shape`, `put_Shape` und `get_Selected` nun den gewünschten Programmcode hinzufügen. Die Funktionen sind in Listing 34.1 aufgeführt. Die Implementierungen der beiden `Get`-Funktionen sind sehr einfach. Die `put_Shape`-Funktion ist ein wenig komplexer, da die `m_nShape`-Variable auf einen neuen Wert gesetzt und eine Fehlerüberprüfung ausgeführt werden muß. Außerdem benachrichtigen wir das Steuerelement darüber, daß sich seine Darstellung geändert hat und das es daher neu gezeichnet werden muß.

Listing 34.1:
Die `Get`- und `Put`-
Funktionen in
ACTL

```
STDMETHODIMP CACTL::get_Shape(short * pVal)
{
    // ZU ERLEDIGEN: Implementierungscode hier hinzufügen

    *pVal = m_nShape;
    return S_OK;
}

STDMETHODIMP CACTL::put_Shape(short newVal)
{
    // ZU ERLEDIGEN: Implementierungscode hier hinzufügen

    BOOL bUserMode;
    GetAmbientUserMode(bUserMode);

    if (bUserMode)
        return Error(_T("Not supported at run-time"));
    else if (newVal < 0 || newVal > 2)
        return Error(_T("Invalid property value"));
    else
    {
        m_nShape = newVal;
        FireViewChange();
    }
    return S_OK;
}

STDMETHODIMP CACTL::get_Selected(BOOL * pVal)
{
    // ZU ERLEDIGEN: Implementierungscode hier hinzufügen

    *pVal = m_bSelected;
    return S_OK;
}
```

In der Fehlerüberprüfung ermitteln wir, ob der Figurwert zwischen 0 und 2 liegt und ob das Steuerelement im Entwurfsmodus verwendet wird. Der Modus wird mit `CComControl::GetAmbientUserMode` abgefragt.

Fehler werden von der `CCoClass::Error-Elementfunktion` zurückgegeben. Diese Funktion ist in dem Programmcode enthalten, da wir uns für die Unterstützung von `ISupportErrorInfo` entschieden haben, als das Steuerelement erstellt wurde.

34.2.4 Hinzufügen des Zeichencodes

Nachdem wir die Eigenschaften eingerichtet haben, die die Darstellung des Steuerelements definieren, werden wir nun den Programmcode zum Zeichnen des Elements modifizieren. Dazu fügen wir der `CACTL::OnDraw-Elementfunktion` den entsprechenden Programmcode hinzu (Listing 34.2).

```
HRESULT CACTL::OnDraw(ATL_DRAWINFO& di)
{
    RECT& rc = *(RECT*)di.prcBounds;
    HPEN hPen;
    HBRUSH hForeBrush, hBackBrush;
    POINT points[3];

    SaveDC(di.hdcDraw);
    hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
    OLE_COLOR ocBack;
    COLORREF crBack;
    HPALETTE hPalAmb;
    GetAmbientBackColor(ocBack);
    GetAmbientPalette(hPalAmb);
    OleTranslateColor(ocBack, hPalAmb, &crBack);
    hBackBrush = CreateSolidBrush(crBack);
    hForeBrush = CreateSolidBrush(m_bSelected ? RGB(255, 0, 0) :
                                         RGB(0, 255, 0));

    FillRect(di.hdcDraw, &rc, hBackBrush);
    SelectObject(di.hdcDraw, hPen);
    SelectObject(di.hdcDraw, hForeBrush);
    switch (m_nShape)
    {
    case 0:
        Ellipse(di.hdcDraw, rc.left, rc.top,
                rc.right, rc.bottom);
        break;
    case 1:
        Rectangle(di.hdcDraw, rc.left, rc.top,
                  rc.right, rc.bottom);
        break;
    case 2:
        points[0].x = rc.left;
        points[0].y = rc.bottom - 1;
        points[1].x = (rc.left + rc.right - 1) / 2;
        points[1].y = rc.top;
        points[2].x = rc.right - 1;
        points[2].y = rc.bottom - 1;
        Polygon(di.hdcDraw, points, 3);
        break;
    }
}
```

*Listing 34.2:
Die CACTL::
OnDraw-Elementfunktion*


```

RestoreDC(di.hdcDraw, -1);
return S_OK;
}

```

Abhängig von den Eigenschafteneinstellungen, zeichnet `OnDraw` die entsprechende Figur in der entsprechenden Farbe in einem rechteckigen Bereich, den die Funktion mit dem `ATL_DRAWINFO`-Parameter ermittelt.

34.2.5 Ereignisbearbeitung

In MFC-Projekten werden Ereignis-Bearbeiterfunktionen mit Hilfe des Klassen-Assistenten definiert. In ATL-Projekten müssen solche Funktionen manuell definiert und der Klasse hinzugefügt werden. Dies ist keine einfache Aufgabe.

Durch Auswahl der Option VERBINDUNGSPUNKTE UNTERSTÜTZEN beim Erzeugen des Steuerelements wurde bereits eine passende Schnittstelle namens `_IACtlEvents` als Standard-Quellschnittstelle für `COACTL` in der `.idl`-Datei eingetragen:

```

...
dispinterface _IACtlEvents
{
    properties:
    methods:
};

[
    uuid(E4119AF4-6043-11D2-B4F9-FCA4622BB12F),
    helpstring("CACTL Class")
]
coclass CACTL
{
    [default] interface ICACTL;
    [default, source] dispinterface _IACtlEvents;
};

```

Der nächste Schritt ist das Ereignis für die Schnittstelle einzurichten.

1. Klicken Sie in der Klassen-Ansicht mit der rechten Maustaste auf die `_IACtlEvents`-Schnittstelle, und wählen Sie im Kontextmenü den Eintrag **METHODEN HINZUFÜGEN** aus.
2. Wählen Sie im erscheinenden Dialogfeld `void` als Rückgabebetyp, geben Sie als Name `Select` und als Parameter `[in] BOOL IsSelected` ein.
3. Klicken Sie auf **OK**.

Als nächstes wird die Typenbibliothek des Steuerelements (TLB-Datei) erstellt.

4. Sie können diese Datei erzeugen, ohne das gesamte Projekt erneut generieren zu lassen, indem Sie die Datei `AC.IDL` im Visual Studio öffnen und den Eintrag **KOMPILIEREN VON AC.IDL** aus dem Menü **ERSTELLEN** auswählen.

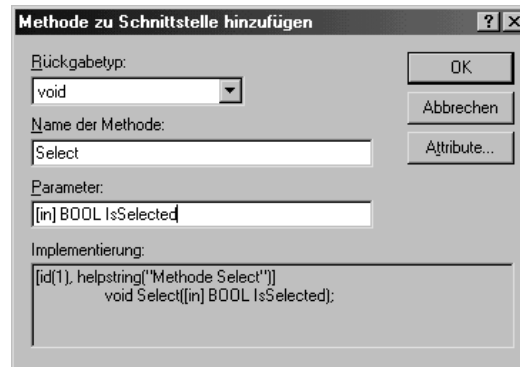


Abbildung 34.10:
Methode zu
Schnittstelle
hinzufügen

Nun gilt es die Schnittstellen zu den Verbindungspunkten zu implementieren, und zwar zu `IConnectionPoint` und zu `IConnectionPointContainer`

5. Klicken Sie in der Klassen-Ansicht mit der rechten Maustaste auf die `CACTL`-Klasse, und wählen Sie im Kontextmenü den Eintrag `VERBINDUNGSPUNKT IMPLEMENTIEREN` aus.

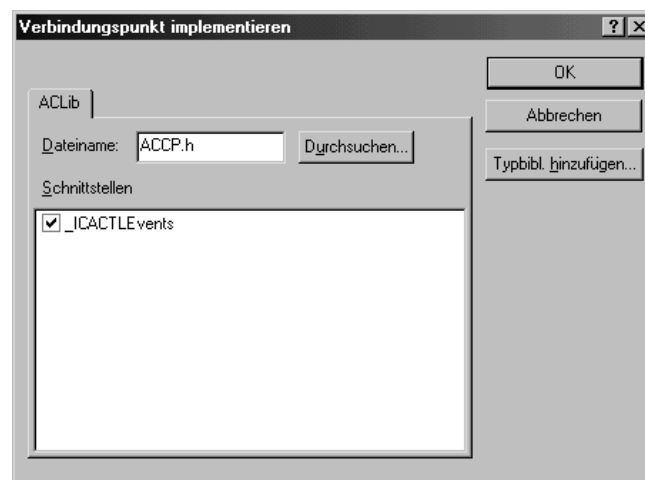


Abbildung 34.11:
Verbindungs-
punkt imple-
mentieren

6. Wählen Sie im erscheinenden Dialogfeld die `_ICACTLEvents`-Schnittstelle aus.
7. Klicken Sie auf OK.



Unter Umständen ist es erforderlich den von Visual C++ erzeugten Code zu korrigieren, indem Sie in der `BEGIN_CONNECTION_POINT_MAP` aus der Header-Datei der Steuerelement-Klasse den Eintrag `CONNECTION_POINT_ENTRY(IID_IACtlEvents)` in `CONNECTION_POINT_ENTRY(DIID_IACtlEvents)` abwandeln.

Visual C++ erstellt nun ein Proxy-Klassentemplate namens `CProxy_IACtlEvents`. In diesem Template ist die Elementfunktion `Fire_Select` definiert, die wir zur Auslösung unseres Ereignisses aufrufen.

Der Verbindungspunkt besteht nun. Wir können nun das Ereignis definieren, d.h., als Antwort auf einen Mausklick das Ereignis auslösen.

8. Klicken Sie in der Klassen-Ansicht mit der rechten Maustaste auf die `CACTL`-Klasse, und wählen Sie im Kontextmenü den Eintrag `BEHANDLUNGSRoutine FÜR Windows-Nachrichten Hinzufügen` aus.
9. Wählen Sie im Nachrichten/Ereignisse-Feld `WM_LBUTTONDOWN`, und lassen Sie eine neue Behandlungsroutine hinzufügen.
10. Springen Sie in den Quelltext der Behandlungsfunktion, und geben Sie den Code aus Listing 34.3 ein.

Listing 34.3:
Die `CACTL::OnLButtonDown`-
ButtonDown-
Elementfunktion

```
LRESULT CACTL::OnLButtonDown(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL&
bHandled)
{
    m_bSelected = !m_bSelected;
    FireViewChange();
    Fire_Select(m_bSelected);
    bHandled = TRUE;
    return S_OK;
}
```

Das Ereignis ist nun dem Steuerelement hinzugefügt worden.

Ereignisverarbeitung testen

Sie können das Steuerelement und die Ereignisverarbeitung nun bereits im Test-Container ausprobieren.

1. Rufen Sie den Befehl `ERSTELLEN/AUSFÜHREN VON` auf, und wählen Sie im erscheinenden Dialogfenster über die Pfeil-Schaltfläche den Untermenüeintrag `TEST-CONTAINER FÜR ACTIVEX-STEUERELEMENTE` aus.
2. Im Test-Container rufen Sie den Befehl `BEARBEITEN/NEUES STEUERELEMENT EINFÜGEN` auf und wählen die Klasse des Steuerelements aus.

3. Jetzt können Sie im Test-Container die Ereignisverarbeitung beim Klicken auf das ActiveX-Steuerelement verfolgen.

34.2.6 Hinzufügen einer Eigenschaftenseite

Unserem Steuerelement muß außerdem noch ein Eigenschaftendialog hinzugefügt werden, in dem der Anwender die Eigenschaftenseite des Steuerelements öffnen kann. Wir werden einen einfachen Dialog erstellen, in dem ein Kombinationsfeld enthalten ist, das drei Figuren zur Auswahl anbietet.

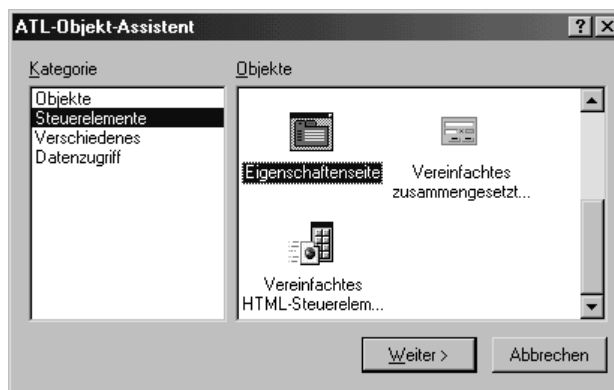


Abbildung 34.12:
Hinzufügen
einer Eigen-
schaftenseite

1. Wählen Sie aus dem Menü EINFÜGEN den Eintrag NEUES ATL-OBJEKT aus.
2. Selektieren Sie in dem daraufhin angezeigten Dialog den Eintrag STEUERELEMENTE.
3. Wählen Sie EIGENSCHAFTENSEITE aus (Abbildung 34.12), und betätigen Sie die Schaltfläche WEITER >.
4. Geben Sie in dem Dialog EIGENSCHAFTEN VON ATL-OBJEKT-ASSISTENT den Namen `ACTLProp` als Kurzbezeichnung der Eigenschaftenseite ein (Abbildung 34.13). Die verbleibenden Felder werden währenddessen automatisch aktualisiert.
5. Öffnen Sie nun das Register ZEICHENFOLGEN (Abbildung 34.14). Geben Sie in die Felder TITEL und DOK.-ZEICHENFOLGE die entsprechenden Texte ein. Löschen Sie den Inhalt des Eingabefelds unter HILFEDATEI. Unser Steuerelement verfügt über keine Hilfedatei.

Abbildung 34.13:
Namen der
Eigenschaften-
seite

The screenshot shows the 'Eigenschaften von ATL-Objekt-Assistent' dialog box with the 'Namen' tab selected. It contains two main sections: 'C++' and 'COM'. The 'C++' section has fields for 'Kurzbez.' (ACTLProp), 'Klasse' (CACTLProp), 'H-Datei' (ACTLProp.h), and '.CPP-Datei' (ACTLProp.cpp). The 'COM' section has fields for 'Co-Klasse' (ACTLProp), 'Schnittstelle' (IACTLProp), 'Typ' (ACTLProp Class), and 'Programm-ID' (AC.ACTLProp). At the bottom are 'OK' and 'Abbrechen' buttons.

Abbildung 34.14:
Eigenschaften-
seitenzeichen-
folgen

The screenshot shows the 'Eigenschaften von ATL-Objekt-Assistent' dialog box with the 'Attribute' tab selected. It contains fields for 'Titel' (&Form), 'Dok.-Zeichenfolge' (ACTL-Eigenschaften), and 'Hilfedatei'. At the bottom are 'OK' and 'Abbrechen' buttons.

Abbildung 34.15:
Die Dialogvor-
lage der Eigen-
schaftenseite

The screenshot shows the 'AC.rc - IDD_ACTLPROP (Dialog)' resource editor. It displays a dialog box template with a title bar and a main area containing a 'Form:' label and a text box. The editor includes a menu bar, a toolbar, and a scroll bar.

Wenn Sie die Schaltfläche OK betätigen, erstellt der ATL-Objektassistent die neue Eigenschaftenseite. Dazu werden eine neue Resource und der Dialog `IDD_ACTLPROP` generiert. Drei neue Dateien deklarieren die Klasse `CACTLProp` (in `ACTLPROP.H`), enthalten die Definition der Klasse (`ACTLPROP.CPP`) und stellen ein Registrierungsskript für das Eigenschaftenseitenobjekt zur Verfügung (`ACTLPROP.RGS`).

Die verbleibenden Schritte sind nicht schwierig.

6. Erstellen Sie zunächst eine Dialogvorlagenressource und fügen Sie dieser ein Kombinationsfeld und ein Textfeld hinzu, wie in Abbildung 34.15 dargestellt.
7. Setzen Sie den Bezeichner des Kombinationsfelds auf `IDC_CBSHAPE` und den Stil auf ein Dropdown-Listenfeld, dessen Einträge nicht sortiert dargestellt werden. Speichern Sie den Dialog.

Sie fragen sich nun möglicherweise, wieso wir nicht den Dialog-Editor von Visual Studio verwenden, um die gewünschten Auswahlmöglichkeiten in das Kombinationsfeld einzutragen. Dieses Feature ist eine spezifische MFC-Funktion. In Anwendungen, wie zum Beispiel unserem ATL-Steuerelement, die nicht die MFC verwenden, müssen wir die Einträge des Kombinationsfelds manuell bestimmen.

Wir müssen außerdem gewährleisten, daß die in diesem Dialog vorgenommene Auswahl korrekt von dem Steuerelement verarbeitet wird. Dazu modifizieren wir die Standardimplementierung von `CACTLProp::Apply`.

Wir benötigen zunächst zwei Nachrichtенbearbeiterfunktionen.

- ➡ Die erste dieser Funktionen reagiert auf `WM_INITDIALOG`-Nachrichten.
- ➡ Die zweite Funktion ist ein Anweisungsbearbeiter, der auf `CBN_SELCHANGE`-Nachrichten von dem Kombinationsfeld reagiert.

Diese Funktionen müssen in der `CACTLProp`-Klassendeklaration deklariert und zur Nachrichtenzuordnungstabelle hinzugefügt werden. Modifizieren Sie `ACTLPROP.H` dazu wie folgt:

```
BEGIN_MSG_MAP(CACTLProp)
    MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)
    COMMAND_HANDLER(IDC_CBSHAPE, CBN_SELCHANGE,
        OnShapeSelchange);
    CHAIN_MSG_MAP(IPropertyPageImpl<CACTLProp>)
END_MSG_MAP()

LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam,
    BOOL& bHandled);
LRESULT OnShapeSelchange(WORD wNotify, WORD wID, HWND hWnd,
    BOOL& bHandled);
```

Die in Listing 34.4 aufgeführte Implementierung der beiden Funktionen wird in der Datei *ACTLPROP.CPP* vorgenommen. *CACTLProp::OnInitDialog* fügt drei Zeichenfolgen in das Kombinationsfeld ein. Außerdem richtet die Funktion das Kombinationsfeld derart ein, daß dieses die aktuelle Figur des Steuerelements wiedergibt. Beachten Sie bitte den Vergleich `m_nObject == 1`;. Diese Überprüfung berücksichtigt die Situationen, in denen die Eigenschaftenseite für mehr als ein Steuerelement aufgerufen wird. Geschieht dies, wird das Kombinationsfeld nicht auf einen bestimmten Wert gesetzt, da verschiedene Steuerelemente unterschiedliche Figuren darstellen können.

Listing 34.4:
Nachrichtenbear-
beiterfunktionen
in *CACTLProp*

```
LRESULT CACTLProp::OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL&
bHandled)
{
    bHandled = TRUE;
    HWND hCB = GetDlgItem(IDC_CBSHAPE);
    ::SendMessage(hCB, CB_RESETCONTENT, 0, 0);
    ::SendMessage(hCB, CB_ADDSTRING, 0, (LPARAM)_T("Ellipse"));
    ::SendMessage(hCB, CB_ADDSTRING, 0, (LPARAM)_T("Rechteck"));
    ::SendMessage(hCB, CB_ADDSTRING, 0, (LPARAM)_T("Dreieck"));
    if (m_nObjects == 1)
    {
        CComQIPtr<IACTL, &IID_IACTL> pACTL(m_ppUnk[0]);
        short nShape;
        pACTL->get_Shape(&nShape);
        ::SendMessage(hCB, CB_SETCURSEL, nShape, 0);
    }
    return S_OK;
}

LRESULT CACTLProp::OnShapeSelchange(WORD wNotify, WORD wID, HWND hWnd, BOOL&
bHandled)
{
    bHandled = TRUE;
    SetDirty(TRUE);
    return S_OK;
}
```

Die *CACTLProp::OnShapeSelchange*-Elementfunktion ruft *IPropertyPage-Impl::SetDirty* auf, um anzuzeigen, daß eine neue Auswahl in der Eigenschaftenseite getroffen wurde. Die Schaltfläche ÜBERNEHMEN kann daraufhin beispielsweise in dem Eigenschaftendialog freigegeben werden, der die Eigenschaftenseite enthält.

Die Funktion *CACTLProp::Apply*, die nach einem Klick auf die Schaltfläche ÜBERNEHMEN aufgerufen wird, ist in Listing 34.5 aufgeführt. Diese Funktion aktualisiert das Steuerelement gemäß der Auswahl des Anwenders in der Eigenschaftenseite.

Listing 34.5:
Die *CACTLProp::*
Apply-Element-
funktion

```
STDMETHOD(Apply)(void)
{
    ATLTRACE(_T("CACTLProp::Apply\n"));
    for (UINT i = 0; i < m_nObjects; i++)
    {
        CComQIPtr<IACTL, &IID_IACTL> pACTL(m_ppUnk[i]);
        HWND hCB = GetDlgItem(IDC_CBSHAPE);
```

```

        int nShape = ::SendMessage(hCB, CB_GETCURSEL, 0, 0);
        if (nShape < 0) nShape = 0;
        if (nShape > 2) nShape = 2;
        pACTL->put_Shape(nShape);
    }
    m_bDirty = FALSE;
    return S_OK;
}

```

Binden Sie *AC.H* nun in die Datei *ACTLPROP.H* ein, da das Projekt andernfalls nicht korrekt kompiliert wird:

```

#ifndef __ACTLPROP_H_
#define __ACTLPROP_H_

#include "resource.h"          // Hauptsymbole

#include "AC.h"

```

Modifizieren Sie nun noch *ACTL.H*, um einen Verweis auf die neue Eigenschaftenseite einzubinden und den vom Anwendungsassistenten generierten Beispielverweis auf die Standard-Farbeigenschaftenseite zu entfernen:

```

BEGIN_PROPERTY_MAP(CACTL)
    // Example entries
    // PROP_ENTRY("Property Description", dispid, clsid)

    PROP_ENTRY("Shape", 1, CLSID_ACTLProp)
END_PROPERTY_MAP()

```

34.2.7 Hinzufügen einer Bitmap

Unsere letzte Aufgabe besteht darin, eine Bitmap zu definieren, die von der Container-Anwendung verwendet wird, um unser Steuerelement beispielsweise in einer Werkzeugleiste zu repräsentieren. Der ATL-COM-Anwendungsassistent hat bereits einen Registrierungseintrag für diese Bitmap in dem Registrierungsskript des Steuerelements generiert. Wenn Sie die Datei *ACTL.RGS* öffnen, finden Sie dort die folgende Programmzeile:

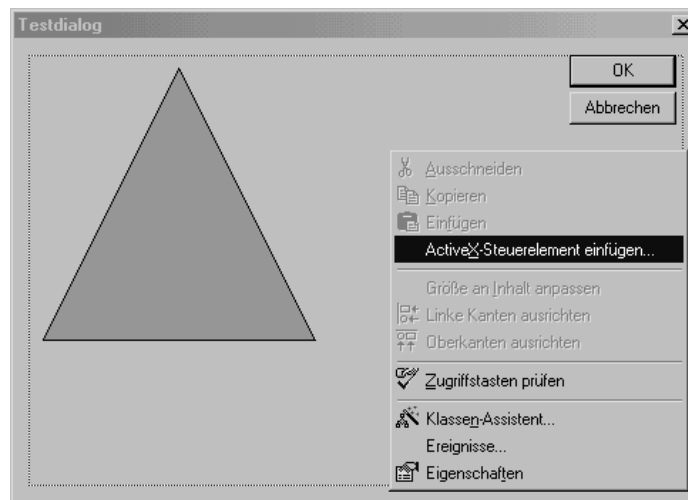
```
ForceRemove 'ToolboxBitmap32' = s '%MODULE%', 101'
```

Diese Zeile bestimmt die Bitmap mit dem Bezeichner 101 als die Bitmap, die das Steuerelement repräsentiert. Öffnen Sie die Bitmap, und bearbeiten Sie sie nach Ihren Wünschen.

Kompilation

Das neue Steuerelement kann nun kompiliert und verwendet werden (Abbildung 34.16).

Abbildung 34.16:
Das vollständige
ACTL-Steuer-
element



34.3 Zusammenfassung

Die ActiveX-Templatebibliothek repräsentiert eine neue Möglichkeit der Erstellung von ActiveX-Objekten. Der Vorteil dieser Bibliothek besteht in der geringen Größe des generierten Programmcodes, die beispielsweise eine Distribution über das Internet ermöglicht. Ein wesentlicher Nachteil ist der komplexere Vorgang, der zum Erstellen von Objekten erforderlich ist.

Nachfolgend sind die Schritte aufgeführt, die zur Erstellung eines ActiveX-Steuerelements mit der ATL ausgeführt werden müssen:

- ➔ Erstellen eines ATL-COM-Projektgerüsts mit Hilfe des ATL-COM-Anwendungsassistenten.
- ➔ Hinzufügen eines Steuerelementobjekts zu dem Projekt.
- ➔ Hinzufügen der Eigenschaften und Methoden.
- ➔ Zeichnen des Steuerelements.
- ➔ Hinzufügen von Ereignissen.
- ➔ Hinzufügen von Eigenschaftenseiten.
- ➔ Erstellen der Bitmap des Steuerelements.

Einige dieser Schritte sind äußerst komplex und erfordern ein umfassendes Verständnis des Komponentenobjektmodells und der ActiveX-Templatebibliothek.

ActiveX- Steuerelemente verwenden

Kapitel

35

Microsoft hat mit der Visual-Basic-Programmierungsumgebung eine neue Art der Software-Entwicklung eingeführt. Benutzerdefinierte Visual-Basic-Steuerelemente, die auch als VBX-Zusatzsteuerelemente bezeichnet werden, bilden die Grundlage für die Komponentenprogrammierung.

Obwohl VBX-Zusatzsteuerelemente keine revolutionäre Entwicklung sind (sie repräsentieren lediglich eine Variante der DLLs), erfüllen sie die Voraussetzungen für die Wiederverwendbarkeit in einer Weise, die bisher nicht möglich war. Sogar Programmierer, die den Begriff DLL noch nie gehört haben, bearbeiten VBX-Zusatzsteuerelemente mit Leichtigkeit und erstellen beinahe professionelle Visual-Basic-Anwendungen in wenigen Tagen.

VBX wurde ursprünglich speziell für Visual Basic entwickelt. Der unerwartete Erfolg dieser Technologie führte jedoch dazu, daß Microsoft VBX in das Visual-C++-Entwicklungssystem implementierte. Das Einbinden der VBX-Bibliothek in ein C-Programm ist nicht sehr schwierig, da die VBX selbst in der Programmiersprache C geschrieben wurde. Ein Problem ergibt sich dadurch, daß die verwendeten Datentypen die Basic-Sprache und nicht C berücksichtigen.

Leider war die Unterstützung der VBX-Zusatzsteuerelemente in C-Programmen nicht von Dauer. Kurz nachdem VBX eingeführt wurde, stellte Microsoft die Weiterentwicklung der 16-Bit-Version für Visual C++ ein. Die 32-Bit-Version für Windows NT und Windows 95 war vollständig inkompatibel mit den 16-Bit-VBX-Bibliotheken.

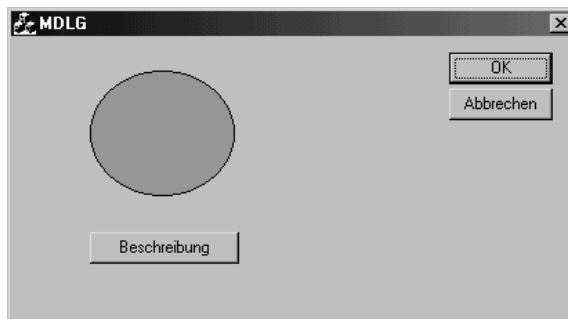
Microsoft entschloß sich daher, einen neuen benutzerdefinierten Steuerelement-Mechanismus zu entwickeln, der auch die bisherigen VBX-Einschränkungen aufheben sollte. Eine Technologie, die sich auf eine

einzigste Programmierungsumgebung bezieht, sollte es nie wieder geben. Die neuen benutzerdefinierten OLE-Steuerelemente verwendeten die OLE-Technik für die Kommunikation mit einer STEUERELEMENT-CONTAINER-ANWENDUNG und waren ebenso einfach zu bedienen wie ihre VBX-Korrelate. Diese benutzerdefinierten OLE-Steuerelemente haben sich im Laufe der Zeit zu den ActiveX-Komponenten entwickelt.

Dieses Kapitel erörtert die ActiveX-Steuerelemente aus der Sicht des STEUERELEMENT-ANWENDERS. Damit ist der Anwendungsprogrammierer gemeint, der ein ActiveX-Steuerelement als eine Art Black Box verwendet und diese Komponente in die Anwendung einbindet. Diese Vorgehensweise unterscheidet sich von der des STEUERELEMENT-ENTWICKLERS, der ein ActiveX-Steuerelement als eine OCX-Bibliothek erstellt, sowie von der Vorgehensweise des ENDANWENDERS, der die von dem Steuerelement-Anwender erstellte Anwendung nutzt.

Ein einfaches ActiveX-Steuerelement mit der Bezeichnung MCTL demonstriert diese neue Technologie. MCTL ist eine Schaltfläche, die, nachdem sie betätigt wurde, eine rechteckige, elliptische oder dreieckige Figur darstellt. Außerdem ändert sich die Farbe, wenn die Schaltfläche mit der Maus angeklickt wird, um den Status der Selektion anzuzeigen (Abbildung 35.1).

Abbildung 35.1:
Das MCTL-Steuerelement



Ein ActiveX-Steuerelement ist ein Objekt, auf das der Programmierer über Eigenschaften, Methoden und Ereignisse zugreift.

- ➔ **STEUERELEMENT-EIGENSCHAFTEN** sind Werte verschiedener Typen, wie zum Beispiel Integer oder Zeichenfolgen, die gesetzt oder ermittelt werden können.
- ➔ **STEUERELEMENT-METHODEN** sind Prozeduren, die Sie aufrufen, um das Steuerelement eine bestimmte Aufgabe ausführen zu lassen.
- ➔ **EREIGNISSE** sind Nachrichten, die das Steuerelement an das übergeordnete Fenster sendet. Ereignisse zeigen bestimmte Begebenheiten an, wie zum Beispiel einen Mausklick.

Die meisten Steuerelemente präsentieren eine sichtbare Oberfläche, obwohl dies nicht unbedingt erforderlich ist. Sie können Steuerelemente erstellen, die Eigenschaften anbieten, Methoden ausführen und Ereignisnachrichten versenden, ohne sich dem Endanwender zu zeigen.

35.1 Hinzufügen von ActiveX-Steuerelementen zu Ihrer Anwendung

Die folgenden Schritte sind notwendig, um einer Anwendung benutzerdefinierte Steuerelemente hinzuzufügen:

- ➔ Gewährleisten Sie, daß Ihre Anwendung benutzerdefinierte Steuerelemente unterstützt. Rufen Sie dazu `AfxEnableControlContainer` auf.
- ➔ Fügen Sie der Dialogvorlage ein benutzerdefiniertes Steuerelement hinzu.
- ➔ Setzen Sie die Eigenschaften des Steuerelements.
- ➔ Definieren Sie Elementvariablen, die das Steuerelement selbst oder dessen Eigenschaften repräsentieren.
- ➔ Definieren Sie einen Nachrichtенbearbeiter für die Nachrichten der Steuerelemente.

Ich habe zur Erläuterung der benutzerdefinierten Steuerelemente eine Anwendung mit der Bezeichnung MDLG erstellt. Diese verwendet einen Dialog mit dem MCTL-Steuerelement aus Kapitel 33. MDLG ist eine vom Anwendungsassistenten generierte, dialogfeldbasierende Anwendung mit der Standard-Unterstützung für ActiveX-Steuerelemente.

Wenngleich in diesem Kapitel ActiveX-Steuerelemente in Verbindung mit Dialogen beschrieben werden, ist der Gebrauch benutzerdefinierter Steuerelemente nicht auf Dialogfelder beschränkt. ActiveX-Steuerelemente können auch in HTML-Seiten (World Wide Web) verwendet werden.

35.1.1 Erstellen eines Steuerelement-Containers

Damit Sie ActiveX-Steuerelemente verwenden können, muß Ihre Anwendung ein Steuerelement-Container sein. Sie erreichen dies während der Initialisierung des Programms mit einem Aufruf von `AfxEnableControlContainer`. Die von dem Anwendungsassistenten erstellten

ActiveX-Anwendungen nehmen diesen Aufruf in der `InitInstance`-Elementfunktion des Anwendungsobjekts vor. Erstellen Sie ein Programm mit dem Anwendungsassistenten, für das Sie die ActiveX-Unterstützung nicht aktiviert haben, können Sie den Aufruf jederzeit nachtragen.

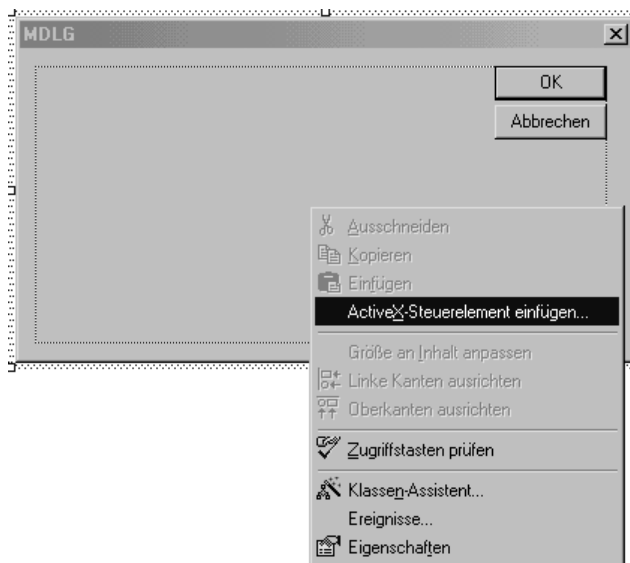
35.1.2 Hinzufügen eines ActiveX-Steuerelements zu einer Dialogvorlage

ActiveX-Steuerelemente können, wie andere Steuerelemente auch, mit dem Dialog-Editor einer Dialogvorlage hinzugefügt werden.

ActiveX-Steuerelemente werden nicht automatisch in der Steuerelementleiste angezeigt.

1. Sie müssen daher die rechte Maustaste betätigen, um ein Kontextmenü zu öffnen, aus dem Sie den Eintrag `ACTIVEX-STEUERELEMENT EINFÜGEN` auswählen (Abbildung 35.2).

Abbildung 35.2:
Einfügen eines
ActiveX-Steuerelements



Wenn Sie den Befehl `ACTIVEX-STEUERELEMENT EINFÜGEN` auswählen, wird der in Abbildung 35.3 dargestellte Dialog geöffnet, in dem alle auf Ihrem System installierten ActiveX-Steuerelemente aufgeführt sind.

2. Selektieren Sie das gewünschte Steuerelement, und betätigen Sie die Schaltfläche `OK`.

Das Steuerelement wird daraufhin in der linken oberen Ecke Ihres Dialogs angeordnet.

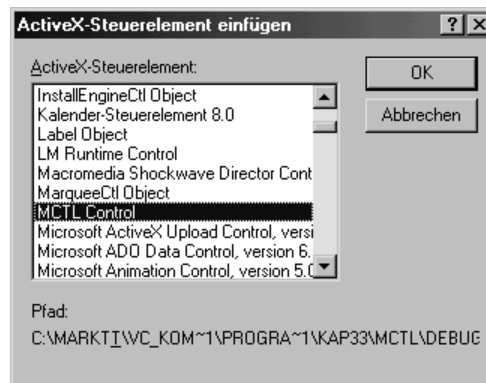


Abbildung 35.3:
Die installierten
ActiveX-Steuer-
elemente

Nachdem sich das Steuerelement in dem Dialog befindet, können Sie es wie ein gewöhnliches Steuerelement an eine neue Position verschieben und seine Größe ändern.

Ich verwende in der MDLG-Anwendung zusätzlich zu dem MCTL-Steuerelement eine Schaltfläche. Diese wird von dem Endanwender betätigt, um sich die *Shape*-Eigenschaft des Steuerelements anzeigen zu lassen. Der Bezeichner der Schaltfläche lautet `IDC_BUTTON1`. Das MCTL-Steuerelement ist mit `IDC_MCTLCTRL1` bezeichnet.

35.1.3 Steuerelement-Eigenschaften setzen

Sobald Sie ein Steuerelement in Ihre Dialogvorlage einfügen, wird der Programmcode des Steuerelements aktiviert. Das Visual Studio lädt den Programmcode der Bibliothek aus der OCX-Datei und aktiviert das Steuerelement im ENTWURFSMODUS. Das Verhalten eines Steuerelements variiert abhängig davon, ob es im Entwurfsmodus oder im ANWENDERMODUS aktiviert ist. Im Anwendermodus wird eine Anwendung, die das Steuerelement verwendet, ausgeführt.

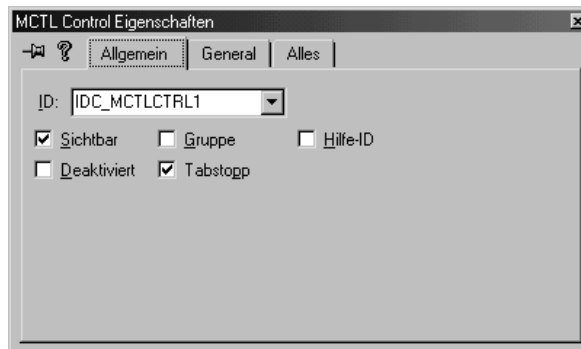
Die meisten ActiveX-Steuerelemente bieten eine Eigenschaftenseite an, in der die Eigenschaften des Steuerelements gesetzt werden können. Eine große Zahl dieser Steuerelement-Eigenschaften sind *beständig*: Einstellungen, die zur Entwurfszeit vorgenommen werden, definieren die Art der Darstellung und das Verhalten des Steuerelements während der Laufzeit.

Das MCTL-Steuerelement verfügt über zwei Eigenschaften.

- ➡ `IsSelected`, eine Boolesche Eigenschaft, zeigt an, ob sich das Steuerelement in einem selektierten Status befindet. `IsSelected` ist eine Eigenschaft mit einem ausschließlichen Lesezugriff (und kann somit nicht zur Entwurfszeit gesetzt werden).

- ➔ **Shape** bestimmt, ob das Steuerelement als Ellipse, Rechteck oder Dreieck dargestellt werden soll. Diese Eigenschaft kann zur Entwurfszeit gesetzt werden. (Versuchen Sie hingegen, **Shape** zur Laufzeit einen Wert zuzuweisen, führt dies zu einem Fehler.)

Abbildung 35.4:
Die Eigenschaften eines
ActiveX-Steuerelements



Sie bestimmen die Eigenschaften eines Steuerelements, indem Sie den Mauszeiger im Dialog-Editor auf das gewünschte Steuerelement bewegen, einmal die rechte Maustaste betätigen und aus dem daraufhin angezeigten Kontextmenü den Eintrag **EIGENSCHAFTEN** auswählen (Abbildung 35.4).

Die erste und letzte Eigenschaftenseite wird vom Developer Studio zur Verfügung gestellt. Die mittlere Seite wird von dem Steuerelement selbst angeboten. Das MCTL-Steuerelement verfügt beispielsweise über eine einzelne Eigenschaftenseite, in der Sie den Wert der **Shape**-Eigenschaft setzen (Abbildung 35.5).

Abbildung 35.5:
Die MCTL-Eigen-
schaftenseite



Die letzte Seite des Eigenschaftendialogs (Abbildung 35.6) bildet eine Schnittstelle zu allen Eigenschaften des Steuerelements. Dazu zählen ebenfalls die Eigenschaften mit Lesezugriff, deren Werte nicht verändert werden können.

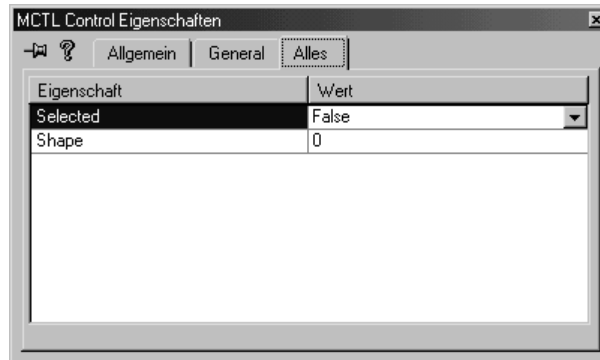


Abbildung 35.6:
Alle Eigenschaften des MCTL-
Steuerelements

35.1.4 Hinzufügen von Elementvariablen

Die Vorgehensweise zum Hinzufügen einer Elementvariable zu einem ActiveX-Steuerelement unterscheidet sich nicht von dem Verfahren, das für gewöhnliche Steuerelemente angewendet wird.

1. Selektieren Sie das Steuerelement innerhalb des Dialog-Editors.
2. Rufen Sie den Klassen-Assistenten auf.
3. Selektieren Sie dort das Register MEMBER-VARIABLEN, und führen Sie einen Doppelklick auf der Zeile aus, die den Bezeichner des Steuerelements enthält.

Nun geschieht etwas Ungewöhnliches. Der Klassen-Assistent reagiert mit einem Dialog, der in Abbildung 35.7 dargestellt ist.

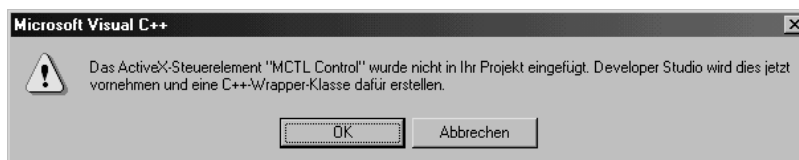
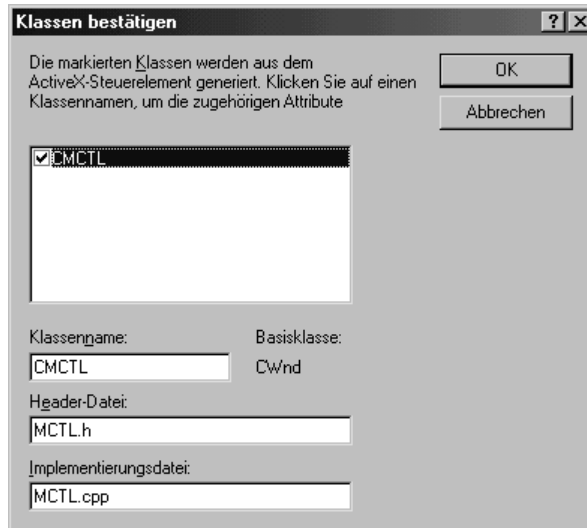


Abbildung 35.7:
Erstellen einer
Klasse für das
ActiveX-Steuer-
element

Damit Sie Variablen erstellen können, die sich auf das ActiveX-Steuerelement beziehen, müssen Sie zunächst eine von `CWnd` abgeleitete Klasse erzeugen, die das Steuerelement repräsentiert. Der Klassen-Assistent führt diese Aufgabe automatisch aus, wenn Sie dem Steuerelement erstmals eine Elementvariable hinzufügen.

4. Wenn Sie jetzt die Schaltfläche OK betätigen zeigt das Visual Studio einen weiteren Dialog an (Abbildung 35.8), in dem Sie die von dem Klassen-Assistenten generierten Klassen- und Dateinamen verändern können.

Abbildung 35.8:
Bestätigen der
Klassenbezeich-
nungen des
ActiveX-Steuer-
elements



5. Nachdem die neuen Klassen erzeugt wurden, definieren Sie die Elementvariable (`m_CMCTL`) für das ActiveX-Steuerelement.

Diese Vorgehensweise muß lediglich einmal in dieser Form ausgeführt werden. Alle weiteren Elementvariablen werden dem Steuerelement wie gewohnt hinzugefügt.

Betrachten Sie nun den vom Klassen-Assistenten generierten Programmcode. Als ich der MDLG-Anwendung eine Elementvariable für das MCTL-Steuerelement hinzufügte, erstellte der Klassen-Assistent daraufhin die `CMCTL`-Klasse, die Header-Datei `MCTL.H` sowie die Implementierungsdatei `MCTL.CPP`.

Die in Listing 35.1 aufgeführte Deklaration der Klasse `CMCTL` enthält alle Elementfunktionen, die zur Erstellung des Steuerelements, zum Einrichten der Eigenschaften und zum Aufrufen der Methoden erforderlich sind.

Listing 35.1:
CMCTL-Klassen-
deklaration

```
class CMCTL : public CWnd
{
protected:
    DECLARE_DYNCREATE(CMCTL)
public:
    CLSID const& GetClsid()
    {
        static CLSID const clsid
            = { 0x20e26da6, 0x584a, 0x11d2, { 0xb4, 0xf9, 0x9b, 0x7,
                0x8c, 0x21, 0x32, 0x15 } };
        return clsid;
    }
    virtual BOOL Create(LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle,
        const RECT& rect,
```

```

        CWnd* pParentWnd, UINT nID,
        CCreateContext* pContext = NULL)
{ return CreateControl(GetClsid(), lpszWindowName, dwStyle,
rect, pParentWnd, nID); }

BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
const RECT& rect, CWnd* pParentWnd, UINT nID,
CFile* pPersist = NULL, BOOL bStorage = FALSE,
BSTR bstrLicKey = NULL)
{ return CreateControl(GetClsid(), lpszWindowName, dwStyle,
rect, pParentWnd, nID, pPersist, bStorage, bstrLicKey); }

// Attribute
public:
    BOOL GetSelected();
    void SetSelected(BOOL);
    short GetShape();
    void SetShape(short);

// Operationen
public:
    void AboutBox();
};

```

Die Implementierungsdatei (Listing 35.2) hält ebenfalls keine Überraschungen für uns bereit. Denken Sie bitte daran, die Warnungen zu Beginn dieser Datei zu beachten. Dieser Programmabschnitt wurde generiert, um eine Schnittstelle zu anderen Steuerelementen zur Verfügung zu stellen. Ihre Aufgabe als Anwendungsprogrammierer besteht nicht darin, das Verhalten des Steuerelements zu modifizieren (dies ist außerdem nicht möglich, solange Sie keinen Zugriff auf den Quellcode des Steuerelements erhalten).

```

//////////////////////////////////// CMCTL

IMPLEMENT_DYNCREATE(CMCTL, CWnd)

//////////////////////////////////// Eigenschaften
CMCTL

BOOL CMCTL::GetSelected()
{
    BOOL result;
    GetProperty(0x1, VT_BOOL, (void*)&result);
    return result;
}

void CMCTL::SetSelected(BOOL propVal)
{
    SetProperty(0x1, VT_BOOL, propVal);
}

short CMCTL::GetShape()
{
    short result;
    GetProperty(0x2, VT_I2, (void*)&result);
    return result;
}

void CMCTL::SetShape(short propVal)
{

```

Listing 35.2:
CMCTL-Klassen-
implementierung

```

    SetProperty(0x2, VT_I2, propVal);
}

//////////////////////////////////// Operationen
CMCTL

void CMCTL::AboutBox()
{
    InvokeHelper(0xffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

```

Der MDLG-Beispielanwendung habe ich eine einzelne Elementvariable vom Typ CMCTL hinzugefügt, die mit m_CMCTL bezeichnet ist. Der Schaltfläche BESCHREIBUNG wurde eine Nachrichtenerarbeiterfunktion hinzugefügt, die diese Variable nutzt. Die Funktion ist in Listing 35.3 aufgeführt. Dort wird die Shape-Eigenschaft des Steuerelements mit einem Aufruf von CMCTL::GetShape ermittelt. Das Ergebnis wird in einem Nachrichtefeld angezeigt.

Listing 35.3:
Der Zugriff auf
die Eigenschaften
des Steuer-
elements

```

void CMDLGDlg::OnButton1()
{
    // TODO: Nachrichtenerarbeiter hier einfügen
    CString shape;
    switch (m_CMCTL.GetShape())
    {
        case 0: shape = _T("Ellipse"); break;
        case 1: shape = _T("Rechteck"); break;
        case 2: shape = _T("Dreieck"); break;
    }
    AfxMessageBox(_T("Die aktuell ausgewählte Form: ") +
        shape + _T("."));
}

```

35.1.5 Bearbeiten von Nachrichten

Das Bearbeiten der Nachrichten von einem ActiveX-Steuerelement ist einfacher als die Bearbeitung der Eigenschaften. Sie können mit Hilfe des Klassen-Assistenten Bearbeiter für jede Nachricht definieren, die das Steuerelement unterstützt. Abbildung 35.9 demonstriert, wie ein Bearbeiter für das einzige Ereignis definiert wird, das das MCTL-Steuerelement generieren kann: das Select-Ereignis.

Abhängig von dem Typ des Ereignisses, empfängt der Ereignisbearbeiter möglicherweise Parameter. Der Bearbeiter für das Select-Ereignis des MCTL-Steuerelements empfängt einen Booleschen Parameter, der angibt, ob das Steuerelement selektiert ist oder nicht. Die Bearbeiterfunktion in MDLG (Listing 35.4) zeigt in diesem Fall ein Meldungsfenster an.

Listing 35.4:
Bearbeiten eines
ActiveX-
Steuerelement-
ereignisses

```

void CMDLGDlg::OnSelectMctlctrl1(BOOL IsSelected)
{
    // TODO: Code für die Behandlungsroutine der
    // Steuerelement-Benachrichtigung hier einfügen
    if (IsSelected)
        AfxMessageBox(_T("Das Steuerelement ist ausgewählt.));
}

```

```

else
    AfxMessageBox(_T("Das Steuerelement ist nicht "
        "ausgewählt.));
}

```

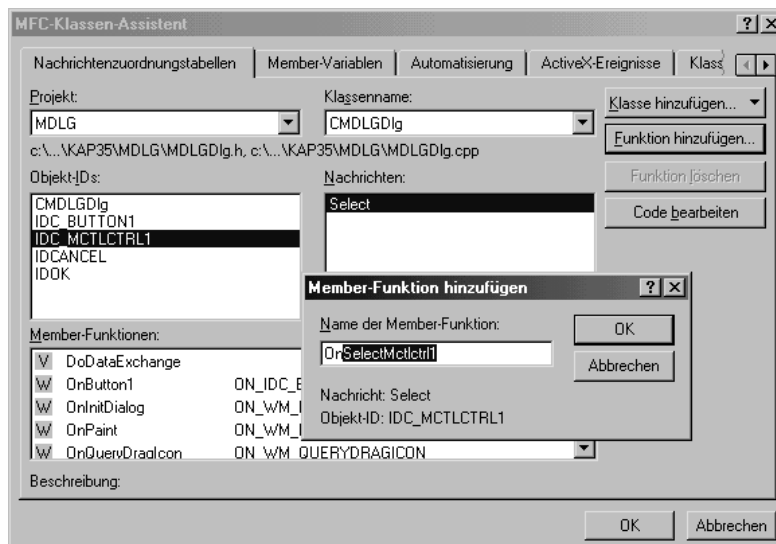


Abbildung 35.9:
Hinzufügen
eines Nachricht-
tabelleintrags für ein
ActiveX-Steuer-
elementereignis

Wie Sie sehen, ist die Vorgehensweise zum Einfügen eines ActiveX-Steuerelements nicht schwieriger als das Einfügen eines gewöhnlichen Steuerelements. Zudem müssen Sie nicht die Details der Implementierung kennen, wenn Sie ein ActiveX-Steuerelement effizient nutzen möchten. Sie benötigen lediglich eine Dokumentation über die Verwendungsmöglichkeiten und das Verhalten eines Steuerelements sowie dessen Eigenschaften, Methoden und Ereignisse.

Ich denke, daß die in der ActiveX-Technologie von Microsoft implementierte Komponentenprogrammierung sehr bald eine der Grundlagen der modernen C++-Programmierung sein wird.

35.2 Visual-C++-ActiveX-Steuerelemente

Microsoft bietet verschiedene ActiveX-Steuerelemente mit Visual C++ an. Einige dieser Steuerelemente gleichen in Ihrer Funktion und in Ihrem Aufbau VBX-Steuerelementen früherer Versionen von Visual Basic.

Das Microsoft-Animation-Control-Steuerelement. Dieses flexible Steuerelement implementiert animierte und Multistatus-Schaltflächen.

Das Microsoft-Communications-Steuerelement ist ein Beispiel für ein Steuerelement, das zur Laufzeit nicht sichtbar ist. Dieses Steuerelement stellt eine Schnittstelle zu einem Kommunikationsport zur Verfügung.

Das Microsoft-Masked-Edit-Steuerelement ist ein Eingabefeld, für das eine Eingabemaske vorgegeben werden kann.

Das Microsoft-Multimedia-Steuerelement bildet eine klar umrissene Multimedia-Schnittstelle.

Das Picture-Clip-Steuerelement wird ebenfalls zur Laufzeit nicht angezeigt. Es bietet einen effizienten Mechanismus zur Organisation einer großen Zahl kleinerer Symbole oder Bitmaps.

Diese und weitere Steuerelemente finden Sie alle im Dialogfeld ACTIVE-X-STEUERELEMENT EINFÜGEN (Abbildung 35.2).

35.3 Zusammenfassung

ActiveX-Steuerelemente repräsentieren eine Technologie, die ein 32-Bit-Nachfolger der benutzerdefinierten Visual-Basic-Steuerelemente (VBX) ist.

Damit eine Anwendung als ActiveX-Steuerelement-Container verwendet werden kann, muß während der Initialisierung der Anwendung die Funktion `AfxEnableControlContainer` aufgerufen werden.

ActiveX-Steuerelemente werden wie gewöhnliche Steuerelemente in einer Anwendung verwendet. Sie können einer Dialogvorlage mit Hilfe des Dialog-Editors hinzugefügt werden. Das Steuerelement wird über einige Eigenschaftenseiten eingerichtet, die in dem EIGENSCHAFTEN-Dialog des Dialog-Editors angezeigt werden.

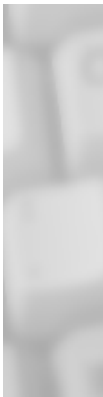
Definieren Sie mit dem Klassen-Assistenten Elementvariablen, mit deren Hilfe Sie in Ihrer Anwendung auf die Eigenschaften des Steuerelements zugreifen. Erstellen Sie das erste Mal eine Elementvariable für einen bestimmten Steuerelementtyp, generiert der Klassen-Assistent eine Mantelklasse für das Steuerelement und fügt diese dem Projekt hinzu.

Der Klassen-Assistent kann außerdem dazu verwendet werden, Bearbeiterfunktionen für Steuerelementereignisse zu definieren.

Microsoft bietet verschiedene benutzerdefinierte Steuerelemente für die Visual-C++-Entwicklungsumgebung an.

Datenbankprogrammierung

Teil **V**

- 
- 36. Datenbankprogrammierung mit ODBC
 - 37. DAO – Datenzugriffsobjekte
 - 38. OLE DB und ADO
 - 39. Datenbank- und Abfragendesign, SQL-Debugging
- 

Datenbank- programmierung mit ODBC

Kapitel 36

 ODBC (Open Database Connectivity) ist ein plattformunabhängiger Mechanismus für den Zugriff auf die Daten unterschiedlicher Datenquellen.

ODBC-Treiber sind für verschiedene Datenquellen erhältlich. Sie können ODBC verwenden, um Daten aus Textdateien, dBase-Tabellen, Excel-Blättern, SQL-Server-Datenbanken und vielen anderen Dateien einzulesen.

Die meisten ODBC-Treiber dürfen ohne Einschränkungen distribuiert werden. Sie können Ihre Software somit mit dem erforderlichen ODBC-Treiber und der Software zur Treiberinstallation und Verwaltung anbieten.

ODBC kann auf Datenquellen bezogene SQL-Anweisungen (Structured Query Language) ausführen. Sie finden in diesem Kapitel daher eine kurze Übersicht über SQL. Möchten Sie ODBC umfassend nutzen, sollten Sie weiterführende Literatur lesen, wie zum Beispiel C. J. DATE, A GUIDE TO THE SQL STANDARD.

Die MFC-Bibliothek unterstützt das Erstellen von ODBC-Anwendungen. Einige Klassen nehmen ODBC-Datenbanken, Tabellen und Datensätze auf. Der Anwendungsassistent und der Klassen-Assistent ermöglichen das Generieren von ODBC-Anwendungen.

36.1 ODBC im Einsatz

Der folgende Abschnitt bietet eine Zusammenfassung der wesentlichen Konzepte von ODBC. Diese sollten Ihnen bekannt sein, bevor Sie eine ODBC-Anwendung erstellen.

Mit Hilfe des ODBC-Setups, das Sie über die Systemsteuerung oder als selbständiges Programm aufrufen, können Sie Datenquellen registrieren.

Was ist eine Datenquelle? Doch was ist eine Datenquelle? Die Antwort auf diese Frage ist von dem verwendeten Treiber abhängig.

- Verwenden Sie beispielsweise einen SQL-Server-Treiber, kann die Datenquelle eine Datenbank oder ein Server sein.
- Die Microsoft-Access- und Excel-Treiber bearbeiten Datenbanken, die in Dateien gespeichert sind (MDB- oder XLS-Datei).
- Datenquellen, auf die über den Texttreiber von Microsoft zugegriffen wird, sind Verzeichnispfade, die Textdateien enthalten. Diese Dateien dienen aus der Sicht des Treibers als Tabellen in einer Datenbank.

Typen von ODBC-Datenquellen Die aktuelle Version von ODBC unterscheidet zwischen drei Quellentypen.

- Eine ANWENDERDATENQUELLE ist lediglich für den Anwender sichtbar, der die Quelle erstellt hat.
- Auf eine SYSTEMDATENQUELLE können alle Anwender zugreifen, die an einem Computer arbeiten.
- Die Spezifikationen einer DATEIDATENQUELLE sind in einer Datei gespeichert und können gemeinsam von Anwendern genutzt werden, die unterschiedliche Rechner verwenden.

Neue Datenquelle installieren



Aufbau und Reihenfolge der Dialogfelder der ODBC-Unterstützung weichen unter Windows 95 und Windows 98 leicht voneinander ab.

1. Um eine Datenquelle auf einem System zu installieren, rufen Sie das 32-Bit-ODBC-Setup aus der Systemsteuerung auf.
2. Öffnen Sie das Register, das dem gewünschten Datenquellentyp zugeordnet ist, und klicken Sie auf HINZUFÜGEN.

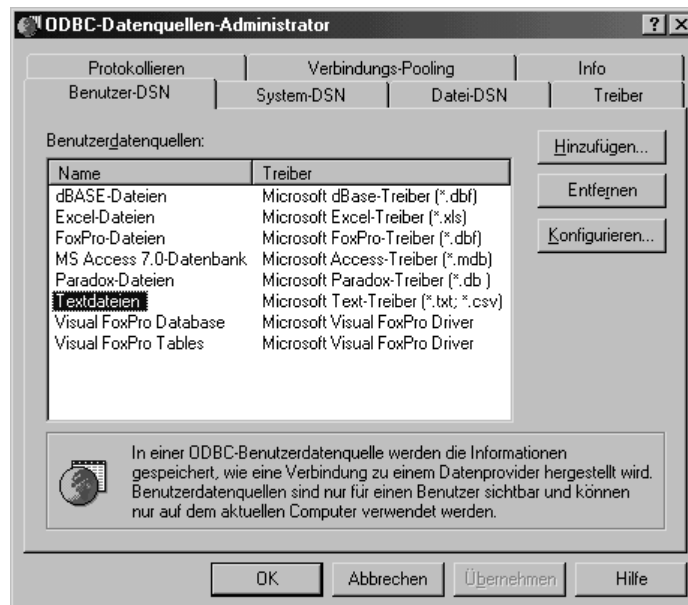


Abbildung 36.1:
Der ODBC-Administrator

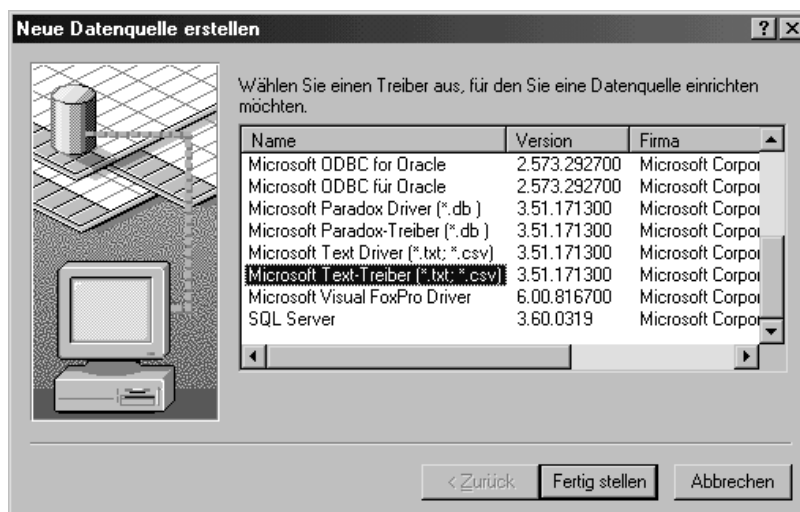


Abbildung 36.2:
Der ODBC-Dia-
log Neue Daten-
quelle erstellen

3. Wählen Sie den zu Ihrer Datenquelle passenden Treiber aus, und klicken Sie auf FERTIGSTELLEN.
4. Im Anschluß daran wird ein spezifischer Treiberdialog angezeigt (Abbildung 36.3), in dem Sie die entsprechende Datenbank angeben und die Eigenschaften des Treibers einstellen können.

Abbildung 36.3:
ODBC-Text-
Setup

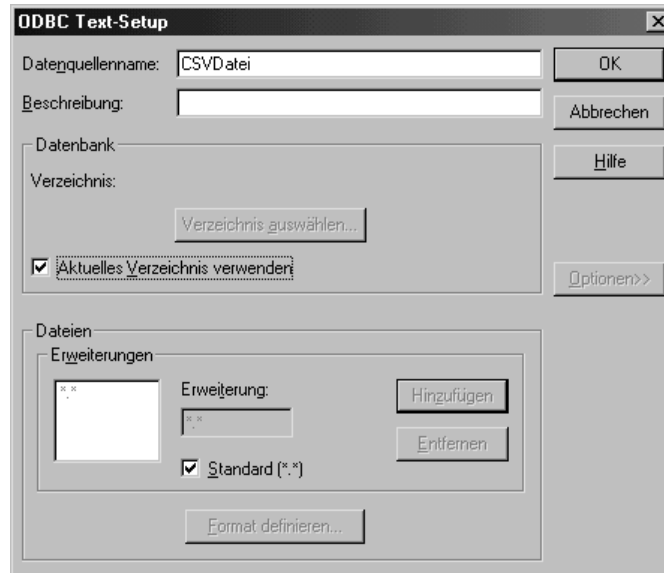
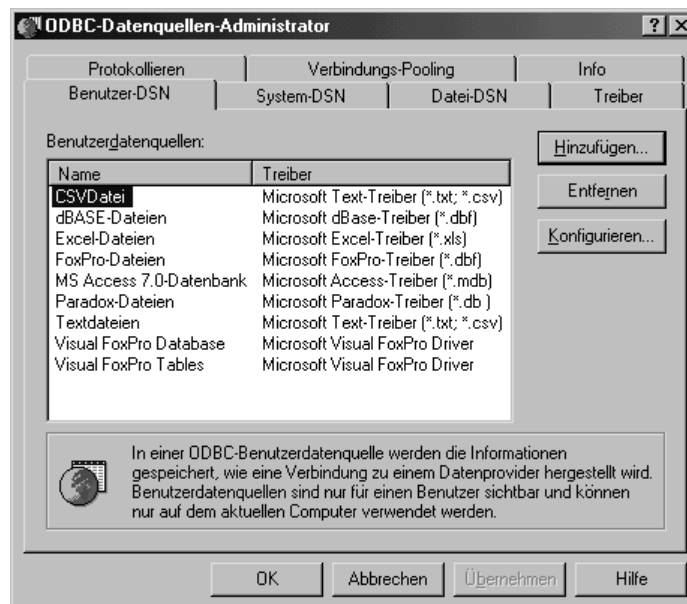


Abbildung 36.4:
ODBC-Setup



- Der Hauptdialog des ODBC-Setup-Programms (Abbildung 36.4) führt alle installierten Datenquellen auf. Sie können Datenquellen hinzufügen, entfernen oder bereits installierte Datenquellen modifizieren.

36.1.1 ODBC-API-Konzepte

Die Treiber

Anwendungen, die ODBC nutzen, verwenden ODBC-Treiber, um auf die gewünschten Daten zuzugreifen. Diese Treiber können ein- oder mehrstufig sein.

- ➔ Einstufige Treiber bearbeiten ODBC-Aufrufe und SQL-Anweisungen.
- ➔ Mehrstufige Treiber bearbeiten ODBC-Aufrufe und übergeben der Datenquelle die SQL-Anweisungen (dieses Verfahren wird gewöhnlich von Netzwerk-Servern genutzt).

ODBC-Ebenen

Der ODBC-Standard definiert drei Koordinationsebenen.

- ➔ Der API-Kern enthält die grundlegenden ODBC-Aufrufe, die für einen Zugriff auf eine Datenquelle und das Ausführen von SQL-Anweisungen erforderlich sind.
- ➔ Der API-Level-1 verfügt über weitere Aufrufe, die Informationen über Datenquellen und den Treiber ermitteln.
- ➔ Der API-Level-2 enthält Aufrufe, die beispielsweise Parameter bearbeiten und Arrays erzeugen.

Da einige Treiber Level-2-Aufrufe nicht unterstützen, wird jede Anweisung mit dem entsprechenden API-Level ausgezeichnet, um Anwendungsfehler auszuschließen.

ODBC und SQL

ODBC definiert eine grundlegende Syntax und zwei Varianten: eine minimale SQL-Syntax und eine erweiterte Syntax.

ODBC ist nicht mit der eingebetteten SQL identisch. Die eingebettete SQL verwendet SQL-Anweisungen in Quellprogrammen, die in einer anderen Programmiersprache geschrieben wurden. Solch ein Programm wird zunächst vorkompiliert, bevor es an den Compiler der Host-Programmiersprache übergeben wird.

Im Gegensatz dazu interpretiert ODBC die SQL-Anweisungen zur Laufzeit. Das Host-Programm muß nicht erneut kompiliert werden, um unterschiedliche SQL-Anweisungen ausführen zu können. Das Kompilieren mehrerer Versionen des Host-Programms für verschiedene Datenquellen ist ebenfalls nicht notwendig.

Eine ODBC-Anwendung muß bestimmte Aufgaben ausführen, um eine Verbindung mit einer Datenquelle herzustellen und SQL-Anweisungen auszuführen. Diese Schritte sind in Abbildung 36.5 dargestellt.

Abbildung 36.5:
Typische Reihenfolge der ODBC-Aufrufe

```

SQLAllocEnv();           // Reserviert die ODBC-Umgebung

SQLAllocConnect();       // Reserviert Speicher für die Verbindung

SQLConnect();            // Lädt Treiber, verbindet mit Datenquelle

SQLAllocStmt();          // Reserviert Speicher für SQL-Anweisung

// ODBC-Anweisungen ausführen
SQLPrepare();
SQLExecute();
SQLBindCol();
SQLFetch();
...

SQLFreeStmt();           // Gibt den Anweisungsspeicher frei

SQLDisconnect();         // Gibt Treiberspeicher frei, löst Verbindung

SQLFreeConnect();        // Gibt den Verbindungsspeicher frei

SQLFreeEnv();            // Gibt den Umgebungsspeicher frei, Programmende

```

- SQLAllocEnv** Der erste Aufruf in Abbildung 36.5 ist mit `SQLAllocEnv` bezeichnet und reserviert eine ODBC-Umgebung. Der Aufruf initialisiert die ODBC-Bibliothek und gibt eine Umgebungszugriffsnummer vom Typ `SQLENVH` zurück.
- SQLAllocConnect** Der zweite Aufruf, `SQLAllocConnect`, reserviert Speicher für die Verbindung. Der Handle, der von dieser Funktion zurückgegeben wird, ist vom Typ `SQLHDBC` und wird in den nachfolgenden ODBC-Funktionen verwendet, um auf eine bestimmte Verbindung zu verweisen. Eine Anwendung kann mehrere geöffnete Verbindungen verwalten.
- SQLConnect** Nun wird die Funktion `SQLConnect` aufgerufen. Die Funktion stellt die Verbindung her, indem sie den Treiber lädt und eine Verbindung zur Datenquelle aufbaut. Alternativ dazu können Sie beispielsweise `SQLDriverConnect` verwenden, um eine Verbindung zu einer Datenquelle herzustellen, die nicht mit dem ODBC-Setup installiert wurde.
- SQLAllocStmt** Der Speicher für eine SQL-Anweisung wird mit `SQLAllocStmt` reserviert. Mit dem Reservieren des Anweisungsspeichers in einem separaten Schritt, bietet ODBC einen Mechanismus an, mit dessen

Hilfe Befehle erstellt, verwendet und erneut verwendet werden können, bevor der für die Anweisungen reservierte Speicher freigegeben wird.

Nachdem diese vier Aufrufe ausgeführt wurden, ruft eine gewöhnliche ODBC-Anwendung einige Funktionen auf, um die auf die Datenbank bezogenen SQL-Anweisungen ausführen zu lassen.

Die Anwendung verwendet `SQLPrepare`, um einen SQL-Befehl für die Ausführung vorzubereiten (zu kompilieren). `SQLExecute` führt die Anweisung aus. Das Programm ruft verschiedene Funktionen auf, um Variablen an Befehle zu binden und das Ergebnis einer Anweisung zu ermitteln.

Wurden alle gewünschten Aufgaben ausgeführt, sollte die Anwendung die reservierten ODBC-Ressourcen wieder freigeben.

<code>SQLFreeStmt</code>	Der Anweisungs-Handle wird mit <code>SQLFreeStmt</code> freigegeben.
<code>SQLDisconnect</code>	Die Verbindung wird mit <code>SQLDisconnect</code> beendet.
<code>SQLFreeConnect</code>	<code>SQLFreeConnect</code> gibt die Reservierung des Verbindungsspeichers frei.
<code>SQLFreeEnv</code>	Schließlich wird die ODBC-Umgebung mit <code>SQLFreeEnv</code> dispensiert.

36.1.2 Ein einfaches ODBC-Beispiel

Um das soeben beschriebene in die Praxis umzusetzen, habe ich eine einfache ODBC-Anwendung entwickelt, die die Zeilen eines Excel-Tabellenblatts ausliest. Während des Zugriffs auf ein Excel-Arbeitsblatt mit einem Microsoft-Excel-ODBC-Treiber repräsentiert das Arbeitsblatt die Datenbanktabelle. Die Zeilen des Tabellenblatts entsprechen somit den Datensätzen einer Datenbanktabelle.

Das Arbeitsblatt ist in Abbildung 36.6 dargestellt. Diese einfache Tabelle enthält Nachnamen, Vornamen und Altersangaben.

Wir werden das Excel-Arbeitsblatt nicht als Datenquelle mit Hilfe des ODBC-Setup installieren. Statt dessen nutzen wir die `SQLDriverConnect`-Funktion. Diese Funktion ermöglicht Ihnen die Verbindung zu einer Datenquelle, die nicht zuvor installiert wurde.

Die in Listing 36.1 aufgeführte Anwendung kann über die Kommandozeile kompiliert werden. Geben Sie dazu

```
CL ADR.C ODBC32.LIB
```

ein. Die Anwendung sucht die Datei *adr.xls* im aktuellen Verzeichnis. (Natürlich muß ebenfalls der Excel-ODBC-Treiber installiert sein.)

Abbildung 36.6:
Ein einfaches
Excel-Arbeits-
blatt, auf das
über ODBC
zugegriffen wer-
den soll

	A	B	C	D	E
1	Nachname	Vorname	Alter		
2	Koch	Manuela	29		
3	Koch	Christine	26		
4	Thiemann	Franz-Josef	33		
5	Berglar	Christine	44		
6					
7					
8					
9					

Listing 36.1:
Eine einfache
ODBC-Konsolen-
anwendung

```
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#define CONNSTR "DBQ=ADR.XLS;DRIVER={Microsoft Excel-Treiber \ (*.xls)}"
#define CONNLEN (sizeof(CONNSTR)-1)
#define SQLTRY(x,y) \
{ \
    rc = y; \
    if (rc != SQL_SUCCESS) \
    { \
        char szState[6]; \
        char szMsg[255]; \
        SDWORD sdwNative; \
        SWORD swMsgLen; \
        SQLError(hEnv, hDBC, hStmt, szState, &sdwNative, \
                szMsg, sizeof(szMsg), &swMsgLen); \
        printf("Fehler %d waehrend %s\nSQLStatus = %s\n\
                SQL-Meldung = %s\n",rc,x,szState,szMsg);\
        goto Terminate; \
    } \
}

void main(void)
{
    SQLHENV hEnv = 0;
    SQLHDBC hDBC = 0;
    SQLHSTMT hStmt = 0;
    SQLCHAR szConnStr[255];
    SQLCHAR szStmt[255];
    SQLCHAR szFirstName[255];
    SQLCHAR szLastName[255];
    long nAge;
    SWORD cbConnStr;
    RETCODE rc;
    SDWORD sdwLNLen;
    SDWORD sdwFNLen;
```

```

SDWORD sdwALen;
int i;
char szResult[1000];

SQLTRY("SQLAllocEnv", SQLAllocEnv(&hEnv))
SQLTRY("SQLAllocConnect", SQLAllocConnect(hEnv, &hDBC))
SQLTRY("SQLDriverConnect", SQLDriverConnect(hDBC, NULL,
    CONNSTR, CONNLEN, szConnStr, sizeof(szConnStr),
    &cbConnStr, SQL_DRIVER_NOPROMPT))
SQLTRY("SQLAllocStmt", SQLAllocStmt(hDBC, &hStmt))

sprintf(szStmt, "SELECT * FROM [Sheet1$]");
SQLTRY("SQLPrepare", SQLPrepare(hStmt, szStmt,
    strlen(szStmt)))

SQLTRY("SQLBindCol", SQLBindCol(hStmt, 1, SQL_C_CHAR,
    (PTR)szLastName, sizeof(szLastName), &sdwLNLen))
SQLTRY("SQLBindCol", SQLBindCol(hStmt, 2, SQL_C_CHAR,
    (PTR)szFirstName, sizeof(szFirstName), &sdwFNLen))
SQLTRY("SQLBindCol", SQLBindCol(hStmt, 3, SQL_C_SLONG,
    (PTR)&nAge, sizeof(nAge), &sdwALen))

SQLTRY("SQLExecute", SQLExecute(hStmt))

for (i = 1; (rc = SQLFetch(hStmt)) == SQL_SUCCESS; i++)
{
    printf("Datensatz #%d\tNachname: %s\tVorname: %s\tAlter: \n",
        i, szLastName, szFirstName, nAge);
}
if (rc != SQL_NO_DATA_FOUND)
{
    SQLTRY("SQLFetch", rc)
}

printf("Operation war erfolgreich.\n");

Terminate:
if (hStmt) SQLFreeStmt(hStmt, SQL_CLOSE);
if (hDBC) SQLDisconnect(hDBC);
if (hDBC) SQLFreeConnect(hDBC);
if (hEnv) SQLFreeEnv(hEnv);
}

```

Da sehr viele Umstände während eines ODBC-Aufrufs zu einem Fehler führen können, verfügt unser Beispiel über eine Fehlerüberprüfung. ODBC-Aufrufe nutzen einen einheitlichen Mechanismus zur Fehlermeldung. Ich habe daher das einfache Makro `SQLTRY` generiert und für die Benachrichtigung über Fehler verwendet. In komplexen Anwendungen nutzen Sie natürlich die Ausnahmehandlung zu diesem Zweck.

*Fehler-
behandlung*

Nach den obligatorischen Aufrufen von `SQLAllocEnv` und `SQLAllocConnect` ruft das Programm die Funktion `SQLDriverConnect` auf. Diese Funktion öffnet eine Tabelle, die nicht mit dem ODBC-Setup installiert wurde. Ein Dialog zur Auswahl der Tabelle wird dem Anwender nicht angezeigt.

*Öffnen der
Daten-
banktabelle*

Beachten Sie bitte die Konstanten `CONNSTR` und `CONNLEN`. `CONNSTR` nimmt den Treibernamen auf. Wird dieser nicht korrekt angegeben, führt der Aufruf zu einem Fehler.

Nachdem eine Verbindung zu der Datenbank (Arbeitsblatt) aufgebaut wurde, wird eine einzelne SQL-Anweisung ausgeführt:

```
SELECT * FROM [Sheet1$]
```

Der Name `Sheet1$` (in eckigen Klammern, da das Zeichen `$` nicht von der SQL unterstützt wird) ist die Treiberbezeichnung für das erste Arbeitsblatt einer Excel-Mappe. Die SQL-Anweisung `SELECT` ermittelt einen Datensatz oder eine Datensatzgruppe. In dem Beispiel selektiert die Anweisung die Felder aller Datensätze.

*Zugriff auf die
Tabellenspalten*

Die nächsten drei Aufrufe binden C-Variablen an Tabellenspalten. Dazu wird die `SQLBindCol`-Funktion verwendet. Werden anschließend Datensätze ausgelesen, werden die Feldinhalte der Datensätze automatisch diesen Variablen zugewiesen.

Die Datensätze werden mit `SQLFetch` ausgelesen und mit `printf` angezeigt. `SQLFetch` wird wiederholt aufgerufen, bis der Rückgabewert dieser Funktion nicht mehr `SQL_SUCCESS` entspricht. Ein Rückgabewert vom Typ `SQL_NO_DATA_FOUND` zeigt an, daß der letzte Datensatz ermittelt wurde. Jeder andere Wert ist ein Fehler und wird entsprechend bearbeitet.

Das Programm endet mit den Aufrufen von `SQLFreeStmt`, `SQLDisconnect`, `SQLFreeConnect` und `SQLFreeEnv`. Diese Funktionen geben die Ressourcen frei und beenden die Verbindung zur Datenquelle.

Wenn Sie das Programm starten, erhalten Sie eine Ausgabe, die der folgenden ähnlich ist:

```
Datensatz #1 Nachname: Koch Vorname: Manuela Alter: 29
Datensatz #2 Nachname: Koch Vorname: Christine Alter: 26
Datensatz #3 Nachname: Thiemann Vorname: Franz-Josef Alter: 33
Datensatz #4 Nachname: Berglar Vorname: Christine Alter: 44
Operation war erfolgreich.
C:\>
```

36.1.3 Weitere ODBC-Aufrufe

Das Beispielprogramm in Listing 36.1 demonstriert einige der grundlegenden Features von ODBC. Natürlich bestehen weitere ODBC-Funktionen, die von Anwendungen genutzt werden können.

*Daten-
verbindung*

Zusätzlich zu `SQLConnect` und `SQLDriverConnect` stellt auch `SQLBrowseConnect` eine Verbindung zu einer Datenquelle her. Diese Funktion ermöglicht Anwendungen, Datenquellen in einer Schleife zu durchsuchen.

Verschiedene Verbindungsoptionen, die sich auf den Transaktionsvorgang, die Zeichensatzübersetzung, Timeouts, die Überwachung und andere Features beziehen, können mit Hilfe der Funktion `SQLSetConnectOption` gesetzt werden. Mit `SQLGetConnectOption` ermitteln Sie die aktuellen Einstellungen.

*Verbindungs-
optionen*

Informationen über Treiber, Datenquellen und andere Optionen werden mit verschiedenen Funktionen ermittelt. Dazu zählen `SQLDataSource`, `SQLDrivers`, `SQLGetFunctions`, `SQLGetInfo` und `SQLGetTypeInfo`.

*Treiberinforma-
tionen*

Anweisungsebenenoptionen können mit `SQLSetStmtOption` bestimmt werden.

*Anweisungs-
optionen*

Alternativ zu `SQLPrepare` und `SQLExecute` kann die Funktion `SQLExecDirect` aufgerufen werden, um SQL-Anweisungen in einem Schritt auszuführen. Der Vorteil dieser Funktion ist die Möglichkeit der wiederholten Ausführung einer vorbereiteten Anweisung. Außerdem können mit der Funktion Informationen über das Ergebnis ermittelt werden, bevor die Anweisung ausgeführt wird.

*SQL-
Anweisungen*

Die vom Treiber übersetzte Version einer SQL-Anweisung kann mit `SQLNativeSql` ermittelt werden.

Einige SQL-Anweisungen erfordern Parameter. Mit `SQLBindParameter` weisen Sie den Variablen Ihres Programms SQL-Abfragen zu. Nachfolgend ist eine solche SQL-Anweisung aufgeführt:

SQL-Parameter

```
INSERT INTO [Sheet1$] (LastName, FirstName, Age) VALUES (?, ?, ?)
```

Bevor diese Anweisung ausgeführt wird, können Sie `SQLBindParameter` dreimal aufrufen, um Programmvariablen die Abfragen in der SQL-Anweisung zuzuweisen. Die Funktion wird zusammen mit `SQLParamData` und `SQLPutData` verwendet, die wiederum ausgeführt werden, wenn `SQLExecute` den Wert `SQL_NEED_DATA` zurückgibt.

`SQLParamOptions`, eine Level-2-ODBC-Erweiterung, ermöglicht einer Anwendung das Einrichten mehrerer Werte. Eine weitere Level-2-Erweiterung mit der Bezeichnung `SQLExtendedFetch` gibt die Daten mehrerer Zeilen in einem Array zurück.

Informationen über die Parameter einer Anweisung können mit `SQLDescribeParam` und `SQLNumParams` ermittelt werden.

Informationen über das Ergebnis einer Anweisung werden mit `SQLNumResultCols`, `SQLColAttributes` und `SQLDescribeCol` analysiert. Die `SQLRowCount`-Funktion gibt die Anzahl der Zeilen zurück, die von einer der SQL-Operationen UPDATE, INSERT oder DELETE betroffen sind. Die Anzahl der Zeilen in einer Datensatzgruppe wird von dieser Funktion jedoch nicht in jedem Fall korrekt zurückgegeben. Nur wenige SQL-Treiber unterstützen diese Funktionalität.

SQL-Ergebnisse

- Spaltenzugriff* Sie müssen nicht `SQLBindCol` verwenden, um Spalten zu binden. Eine Anwendung kann ebenfalls `SQLGetData` aufrufen, um Daten aus nicht gebundenen Spalten auszulesen.
- SQL-Marken* ODBC unterstützt das Positionieren von SQL-Marken. Die erweiterte Level-2-Funktion `SQLSetPos` setzt die Marke in eine bestimmte Zeile und aktualisiert, löscht oder fügt der Zeile Daten hinzu.
- Transaktionen* Die `SQLTransact`-Funktion unterstützt den Transaktionsvorgang.
- Datenquelleninformationen* Informationen über Datenquellen können mit `SQLTables`, `SQLTablePrivileges`, `SQLColumns`, `SQLColumnPrivileges`, `SQLPrimaryKeys`, `SQLForeignKeys`, `SQLSpecialColumns`, `SQLStatistics`, `SQLProcedures` und `SQLProcedureColumns` ermittelt werden. Die Informationen werden als Ergebnisgruppe zurückgegeben, auf die mit `SQLBindCol` oder `SQLFetch` zugegriffen werden kann.
- Asynchrone Ausführung* ODBC ermöglicht die asynchrone Ausführung von Funktionen. Rufen Sie dazu `SQLSetStmtOption` oder `SQLSetConnectOption` mit `SQL_ASYNC_ENABLE` auf. Wird anschließend eine Funktion aufgerufen, die die asynchrone Ausführung unterstützt, gibt diese die Steuerung mit dem Rückgabewert `SQL_STILL_EXECUTING` zurück. Ein wiederholter Aufruf dieser Funktion (mit Parametern, die zulässig sein müssen aber abgesehen von dem ersten `hStmt`-Parameter ignoriert werden) ermittelt, ob die Funktion vollständig ausgeführt wurde.
- Fehler* Informationen über ODBC-Fehler können mit `SQLError` abgefragt werden.

36.2 Der SQL-Standard und ODBC

SQL (Structured Query Language) ist ein offizieller (ANSI) Standard für die Bearbeitung relationaler Datenbanken. Die meisten ODBC-Operationen (einschließlich der ODBC-Operationen einer MFC-Anwendung) senden SQL-Anweisungen an den Treiber. Sie sollten daher grundlegend mit dieser Sprache vertraut sein. In diesem Abschnitt ist eine kurze Übersicht über SQL enthalten. Unsere besondere Aufmerksamkeit gilt dabei der Anwendung dieser Anweisungen im Zusammenhang mit ODBC.

Den Kern der SQL bilden

- die DATENMANIPULATIONSANWEISUNGEN und
- die DATENDEFINITIONSANWEISUNGEN.

Datenmanipulationsanweisungen lesen Daten aus einem Datensatz aus, löschen Daten in einem Datensatz oder fügen diesem neue Daten hinzu. Datendefinitionsanweisungen definieren das Layout der Datenbank.

36.2.1 Datenmanipulationsanweisungen

Es gibt vier wesentliche Datenmanipulationsanweisungen:

- ➡ SELECT,
- ➡ INSERT,
- ➡ UPDATE und
- ➡ DELETE.

SELECT

SELECT-Operationen werden gewöhnlich in dem Format

"SELECT-FROM-WHERE"

angegeben. Eine SELECT-Anweisung kann beispielsweise wie folgt aufgebaut sein:

```
SELECT Vorname, Nachname FROM ANGESTELLTE WHERE ANGESTELLTE.Alter<30
```

Weitere Klauseln und Einschränkungen können verwendet werden, um die Abfrage einer SELECT-Anweisung detaillierter zu gestalten.

Eines der interessantesten Features einer relationalen Datenbank ist die Möglichkeit, VERKNÜPFUNGSOPERATIONEN auszuführen. Verknüpfungsoperationen kombinieren zwei oder mehrere Tabellen zu einer Ergebnisgruppe. Betrachten Sie dazu die folgende Anweisung:

Zugriff auf verknüpfte Tabellen

```
SELECT ANGESTELLTE.Vorname, ANGESTELLTE.Nachname, PLAN.Name  
FROM ANGESTELLTE, PLAN WHERE ANGESTELLTE.Alter < PLAN.MaxAlter
```

Diese Anweisung wird für die beiden Tabellen KUNDEN und PLAN ausgeführt. Die erste Tabelle nimmt die Daten der Angestellten eines Unternehmens auf. Die zweite Tabelle enthält Daten über die Sozialpakete (Mutterschaftsgeld, Kindergeld etc.), die das Unternehmen anbietet. Die SELECT-Anweisung ermittelt, welche Sozialpakete den Angestellten zustehen. Beachten Sie bitte, daß einige Angestellte mehrere Sozialpakete erhalten können. Die Namen dieser Angestellten werden mehrmals in der Ergebnisgruppe aufgeführt.

Möchten Sie mit der SELECT-Anweisung alle Felder einer Zeile selektieren, verwenden Sie ein Malzeichen als Kürzel. Für eine KUNDEN-Tabelle mit den drei Feldern Vorname, Nachname und Alter könnten also die beiden folgenden Anweisungen verwendet werden:

Kürzel

```
SELECT Vorname, Nachname, Alter FROM KUNDEN  
SELECT * FROM KUNDEN
```

Aggregatfunktionen SQL bietet einige AGGREGATFUNKTIONEN an. Zu diesen Funktionen zählen COUNT, MIN, MAX, AVG und SUM. Um beispielsweise in der KUNDEN-Tabelle die Anzahl der Angestellten zu zählen, deren Nachnamen sich voneinander unterscheiden, verwenden Sie die folgende Anweisung:

```
SELECT COUNT (DISTINCT KUNDEN.Nachname) FROM KUNDEN
```

Um die Summe des Alters der gesamten Belegschaft zu berechnen, verwenden Sie diese Anweisung:

```
SELECT SUM (KUNDEN.Alter) FROM KUNDEN
```

Einige SELECT-Anweisungsarten werden für mehrere Zeilen ausgeführt und geben eine Zeilengruppe als Ergebnis zurück. Der SQL-Standard definiert das Konzept einer Positionsmarke (Cursor), die dazu verwendet wird, Zeilen aus einer Ergebnisgruppe zu extrahieren. Die ODBC-Funktionen SQLBindCol und SQLFetch basieren auf demselben Prinzip.

INSERT, UPDATE, DELETE

Die INSERT-Anweisung fügt einer Tabelle weitere Zeilen hinzu.

Die UPDATE-Anweisung modifiziert bereits bestehende Zeilen.

Die DELETE-Anweisung entfernt Zeilen.

Die Syntax dieser Befehle entspricht weitgehend der Syntax der SELECT-Anweisung. Diese Anweisungen verfügen über Varianten, die Cursor nutzen und die von den Positionsmarken keinen Gebrauch machen. Betrachten Sie zum Beispiel die folgenden Anweisungen, die jedes Jahr am 31. Dezember ausgeführt werden, um die Angestelltdatenbank des Unternehmens zu aktualisieren (niemand in dem Unternehmen wird älter als 30 Jahre):

```
UPDATE KUNDEN  
SET KUNDEN.Alter = KUNDEN.Alter + 1  
WHERE KUNDEN.Alter < 30
```

Diese Aktualisierung wird für alle Zeilen ausgeführt, auf die die WHERE-Klausel zutrifft. Sie erfordert keine Positionsmarke, um ausgeführt zu werden. Andere Anwendungsmöglichkeiten dieser Anweisung verwenden Cursor. ODBC unterstützt diese Operationen über SQLBindParameter und ähnlichen Funktionen.

36.2.2 Ansichten

Eine Ansicht ist eine »virtuelle« Tabelle.

CREATE VIEW

Eine Ansicht repräsentiert eine Zeilengruppe, die nicht im physikalischen Speicher vorhanden ist, sondern dynamisch mit der `CREATE VIEW`-Anweisung erstellt wird.

Eine Ansicht kann mit einer `SELECT`-Anweisung generiert werden. Um beispielsweise eine Ansicht zu erzeugen, die alle Angestellten enthält, die jünger als 30 Jahre sind, geben Sie die folgende Zeile ein:

```
CREATE VIEW JUNGEKUNDEN (Nachname, Vorname, Alter)
AS SELECT KUNDEN.Nachname, KUNDEN.Vorname, KUNDEN.Alter
   FROM KUNDEN
  WHERE KUNDEN.Alter < 30
```

In den nachfolgenden Operationen, können Sie diese Ansicht wie eine gewöhnliche Tabelle verwenden. Die folgende `SELECT`-Anweisung demonstriert dies:

```
SELECT * FROM JUNGEKUNDEN
```

36.2.3 Datendefinitionsanweisungen

Datendefinitionsanweisungen werden dazu verwendet, Tabellen und Indizes in einer Datenbank zu erstellen und zu aktualisieren.

CREATE TABLE

Die `CREATE TABLE`-Anweisung erstellt eine Tabelle. Um die in den vorangegangenen Abschnitten verwendete `KUNDEN`-Tabelle zu generieren, geben Sie die folgende Zeile ein:

```
CREATE TABLE KUNDEN
( Nachname CHAR(30) NOT NULL,
  Vorname  CHAR(30),
  Alter    INTEGER
)
```

Die `CREATE TABLE`-Anweisung unterstützt Einschränkungsklauseln. Dazu zählen `UNIQUE` (ein Feldwert muß eindeutig sein) und `CHECK` (Festlegen einer Bedingung). Die `KUNDEN`-Tabellendefinition könnte auch wie folgt vorgenommen werden:

```
CREATE TABLE KUNDEN
( Nachname CHAR(30) NOT NULL,
  Vorname  CHAR(30) NOT NULL,
  Alter    INTEGER,
  UNIQUE (Nachname, Vorname),
  CHECK (Alter < 30)
)
```

Tabellen können außerdem mit Indizes erstellt werden. Um eine KUNDEN-Tabelle zu generieren, deren Nachname-Feld indiziert ist, verwenden Sie folgende Syntax:

```
CREATE TABLE KUNDEN
( Nachname CHAR(30) NOT NULL,
  FirstName CHAR(30),
  Alter      INTEGER,
  PRIMARY KEY (Nachname)
)
```

CREATE INDEX

Die `CREATE INDEX`-Anweisung erzeugt einen Index für eine bereits bestehende Tabelle.

ALTER TABLE

Die Anweisung `ALTER TABLE` kann dazu verwendet werden, die Struktur einer bestehenden Tabelle zu modifizieren.

DROP

Die `DROP`-Anweisung löscht eine Tabelle oder einen Index aus einer Datenbank.

GRANT, REVOKE

Die Befehle `GRANT` und `REVOKE` setzen oder löschen die Sicherheitsprivilegien bestimmter Tabellen.

36.3 ODBC in MFC-Anwendungen

Das Verwenden der ODBC wird durch die MFC-Bibliothek erleichtert. Einfache Anwendungen, die über ODBC auf Tabellen zugreifen, können mit wenigen Mausklicks erstellt werden. Dazu verwenden Sie den Anwendungsassistenten und den Klassen-Assistenten. Einige MFC-Klassen unterstützen den Zugriff auf Datenbanken und Datensätze.

Sie lernen die ODBC-Features der MFC-Bibliothek kennen, indem Sie ein einfaches Beispiel erstellen.

36.3.1 Einrichten einer Datenquelle

Bevor eine MFC-ODBC-Anwendung mit dem Anwendungsassistenten erstellt werden kann, muß eine Datenquelle erzeugt werden, mit der die Anwendung arbeitet. Diese Datenquelle wird mit Hilfe des bereits bekannten ODBC-Setups eingerichtet und bezeichnet.

Die in unserer Beispielanwendung verwendete Datenquelle ist eine Textdatei. Um auf diese Datei zugreifen zu können, benötigen wir den Microsoft-Text-ODBC-Treiber. (Wenn Sie diesen Treiber nicht während der Installation von Visual C++ ausgewählt haben, führen Sie das C++-Setup-Programm bitte erneut aus.)

Die Datendatei *ADR.TXT* enthält einige Datensätze mit Vornamen, Nachnamen und Altersangaben. Die erste Zeile der Datei bildet die Überschriftenzeile. Die Felder der Datei sind durch Semikola voneinander getrennt:

```
Nachname;Vorname;Alter  
Koch;Manuela;22  
Koch;Christine;26  
Thiemann;Franz-Josef;33  
Berglar;Christine;44
```

Nachdem diese Datei erzeugt wurde, geben wir dem 32-Bit-ODBC-Setup die Datenquelle an.

1. Rufen Sie das ODBC-Setup aus der Systemsteuerung auf, und betätigen Sie die Schaltfläche HINZUFÜGEN.

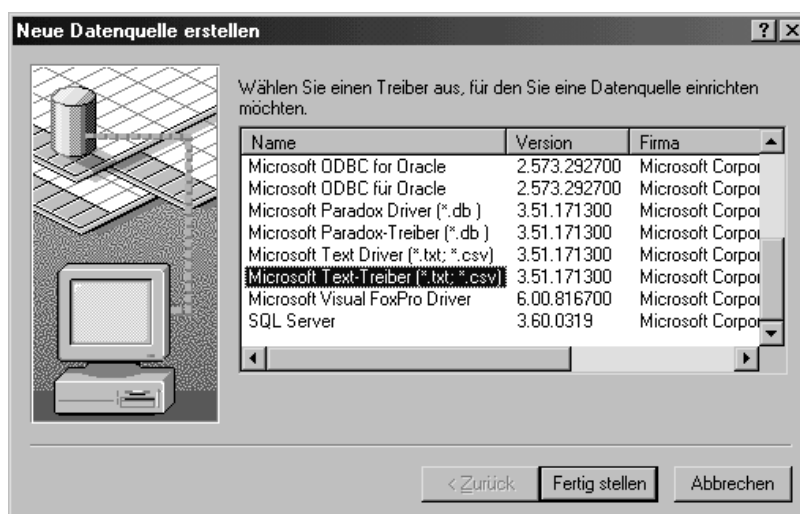
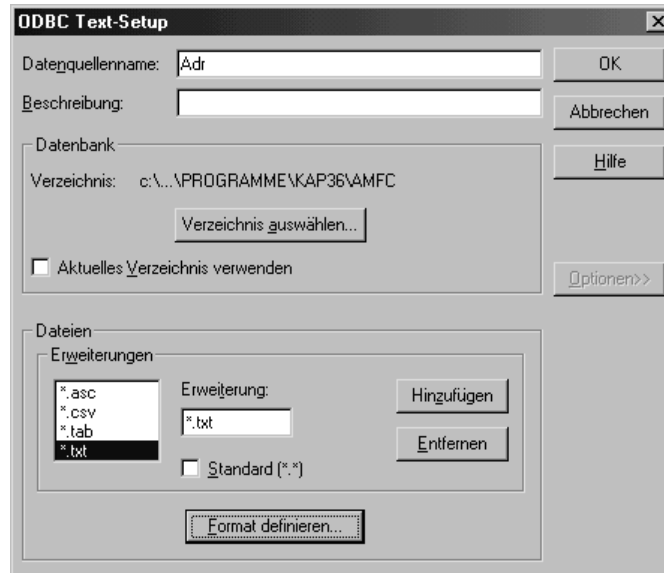


Abbildung 36.7:
Hinzufügen
einer Textdaten-
quelle

2. Selektieren Sie den in Abbildung 36.7 dargestellten Microsoft-Texttreiber.
3. Klicken Sie im Anschluß daran auf die Schaltfläche FERTIGSTELLEN, um den Dialog ODBC TEXT-SETUP aufzurufen (Abbildung 36.8).

Abbildung 36.8:
ODBC-Text-
Setup



Der Microsoft-Texttreiber verwaltet Verzeichnisse als Datenbanken und die einzelnen Textdateien als Tabellen in der Datenbank. Sie können entweder das aktuelle Verzeichnis oder ein bestimmtes Verzeichnis als Datenquelle einrichten.

4. Deaktivieren Sie die Option AKTUELLES VERZEICHNIS VERWENDEN, und klicken Sie auf VERZEICHNIS AUSWÄHLEN, um die Datei *Adr.txt* auszuwählen.
5. Klappen Sie das Dialogfeld auf (Schalter Optionen >>), legen Sie die Dateierweiterung fest, und klicken Sie auf die Schaltfläche FORMAT DEFINIEREN (Abbildung 36.8).
6. Im Dialog TEXTFORMAT DEFINIEREN bestimmen Sie das Format einzelner Tabellen (Textdateien). Aktivieren Sie für *ADR.TXT* die Option SPALTENNAMEN IN ERSTER ZEILE, um die Schaltfläche ANALYSIEREN freizugeben. Klicken Sie auf diese Schaltfläche, um die Namen der Felder und deren Typ ermitteln zu lassen.
7. Schließen Sie die Dialoge.

In dem AMFC-Verzeichnis, in dem die Datei *ADR.TXT* vorhanden ist, wurde eine weitere Datei von dem ODBC-Setup generiert. Diese Datei trägt die Bezeichnung *SCHEMA.INI* (Listing 36.2) und enthält Informationen über die Eigenschaften der ODBC-Datenquelle, die wir zuvor spezifiziert haben.

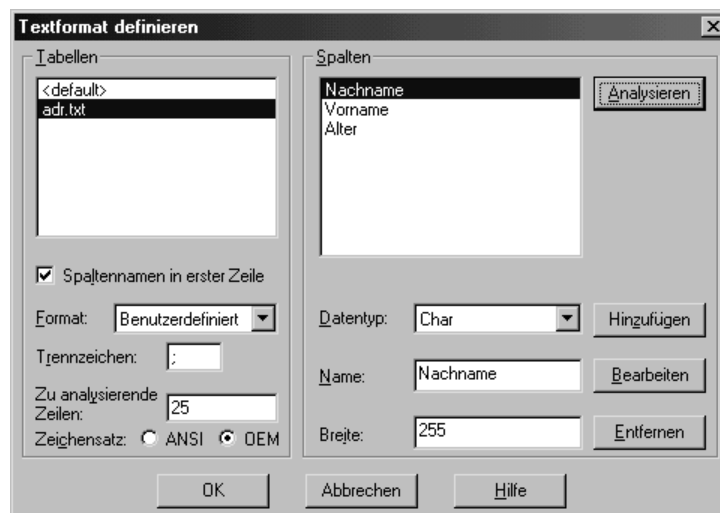


Abbildung 36.9:
Definieren des
Formats einer
Texttabelle

```
[adr.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=25
CharacterSet=OEM
Col1=NACHNAME Char Width 255
Col2=VORNAME Char Width 255
Col3=ALTER Integer
```

Listing 36.2:
Die von dem
ODBC-Setup ge-
nerierte Datei
SCHEMA.INI

Nachdem Sie die Datenquelle eingerichtet und bezeichnet haben, wenden Sie sich dem Anwendungsassistenten zu, um ein Gerüst für unsere Anwendung zu erstellen.

36.3.2 Erstellen eines ODBC-Anwendungsgerüsts mit dem Anwendungsassistenten

1. Rufen Sie den Anwendungsassistenten auf, und erstellen Sie ein Projekt mit der Bezeichnung AMFC. Das Projekt sollte eine Anwendung erstellen, die ein einzelnes Dokument (SDI) verwendet (Anwendungsassistent, Schritt 1).
2. Die Datenbankoptionen werden im zweiten Schritt des Assistenten (Abbildung 36.10) bestimmt. Aktivieren Sie dort die Option DATENBANKANSICHT OHNE DATEIUNTERSTÜTZUNG.
3. Betätigen Sie im Anschluß daran die Schaltfläche DATENQUELLE, und geben Sie eine Datenquelle für die Anwendung an. Wählen Sie die zuvor erstellte Datenquelle ADR in dem Dialog DATENBANK-OPTIONEN aus (Abbildung 36.11).

Abbildung 36.10:
Bestimmen der
Datenbankunterstützung im
Anwendungsassistenten

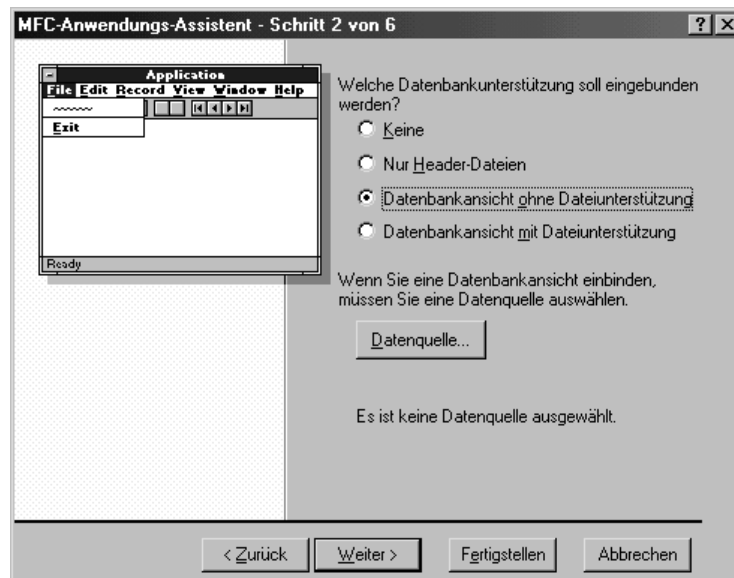
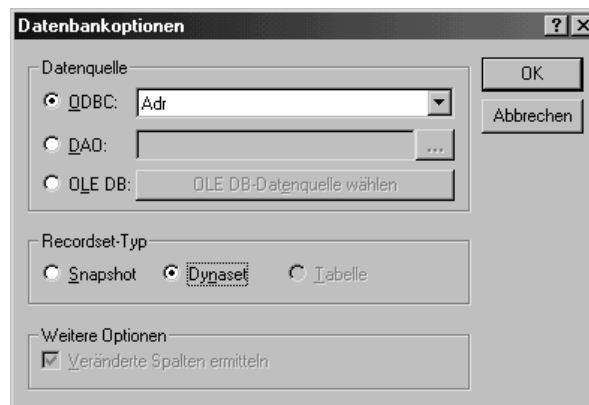


Abbildung 36.11:
Bestimmen
einer Daten-
quelle



4. Sie können ebenfalls den Recordset-Typ bestimmen. Ein Datensatz vom Typ SNAPSHOT repräsentiert eine statische Ansicht der zugrundeliegenden Daten. Diese Datensätze können somit Änderungen von Daten wiedergeben, die nach dem Erstellen der Datensätze vorgenommen wurden (dazu zählen nicht nur hinzugefügte oder entfernte Datensätze). Der Typ TABELLE ermöglicht die direkte Manipulation von Tabellen in einer relationalen Datenbank. Während Snapshots die sicherste Auswahl bilden, bieten DYNASET-Datensätze einen Performance-Vorteil. Unsere Anwendung wird mit einem Dynaset-Datensatz erstellt. Wählen Sie diesen Typ daher in dem Dialog DATENBANKOPTIONEN aus.

5. Betätigen Sie die Schaltfläche OK, um sich einen weiteren Dialog anzeigen zu lassen (Abbildung 36.12). In diesem Dialog selektieren Sie die Datenbanktabelle. Wählen Sie *ADR.TXT* aus, und klicken Sie auf OK.

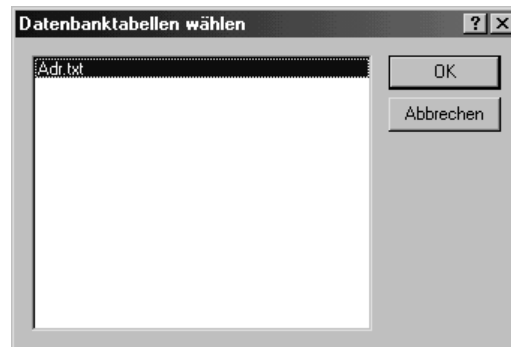


Abbildung 36.12:
Auswählen der
Datenbank-
tabelle

6. Sie befinden sich nun wieder in dem Dialog des Anwendungsassistenten. Weitere Optionen, die von den Standardeinstellungen abweichen, müssen nicht selektiert werden. Betätigen Sie daher die Schaltfläche FERTIGSTELLEN. Die AMFC-Anwendung wird daraufhin von dem Anwendungsassistenten erstellt.

Betrachten Sie bitte die vom Anwendungsassistenten generierten Klassen (Abbildung 36.13). Sie werden feststellen, daß einige Klassen und Elementvariablen erzeugt wurden, die in Anwendungen ohne Datenbankunterstützung nicht vorhanden sind.

Die neue Klasse *CAMFCSet* ist von *CRecordset* abgeleitet. Die Klassendeklaration (Listing 36.3) zeigt, daß der Anwendungsassistent nicht nur die Klasse erstellt, sondern dieser ebenfalls Elementvariablen hinzugefügt hat, die die Felder der von uns angegebenen Datenbanktabelle repräsentieren.

```
class CAMFCSet : public CRecordset
{
public:
    CAMFCSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(CAMFCSet)

    // Feld-/Parameterdaten
    ///{AFX_FIELD(CAMFCSet, CRecordset)
    CString m_NACHNAME;
    CString m_VORNAME;
    long m_ALTER;
    ///}AFX_FIELD

    // Überladungen
    // Vom Klassenassistenten generierte Überladungen
    // virtueller Funktionen
    ///{AFX_VIRTUAL(CAMFCSet)
```

Listing 36.3:
Deklaration von
CAMFCSet

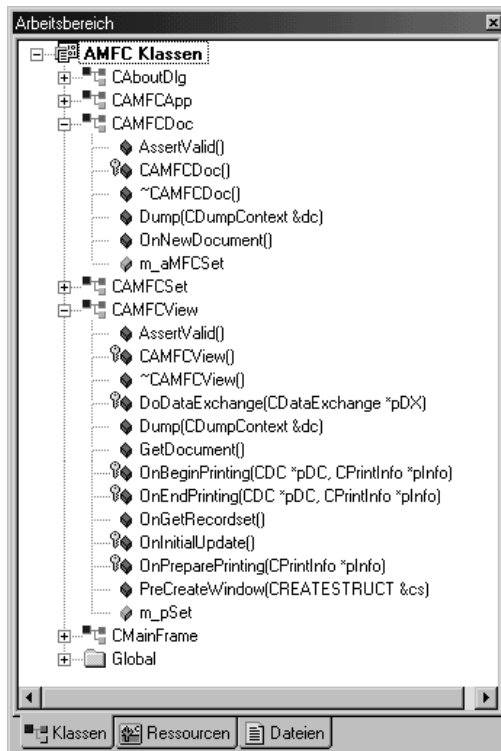
```

public:
    // Standard-Verbindungszeichenfolge
    virtual CString GetDefaultConnect();
    // Standard-SQL für Recordset
    virtual CString GetDefaultSQL();
    // RFX-Unterstützung
    virtual void DoFieldExchange(CFieldExchange* pFX);    ///AFX_VIRTUAL

// Implementierung
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
};

```

Abbildung 36.13:
ODBC-Anwen-
dungsklassen



Diese Elementvariablen sind ebenfalls in der Klassenimplementierungsdatei, (Listing 36.4), in der Konstruktorfunktion und in der Funktion DoFieldExchange aufgeführt. Die zuletzt genannte Funktion wird von dem Anwendungsrahmen aufgerufen, um Daten zwischen den Elementvariablen eines Datensatzes und den entsprechenden Spalten in der Datenbanktabelle auszutauschen.

```

IMPLEMENT_DYNAMIC(CAMFCSet, CRecordset)

CAMFCSet::CAMFCSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    ///{AFX_FIELD_INIT(CAMFCSet)
    m_NACHNAME = _T("");
    m_VORNAME = _T("");
    m_ALTER = 0;
    m_nFields = 3;
    ///}AFX_FIELD_INIT
    m_nDefaultType = dynaset;
}

CString CAMFCSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=Adr");
}

CString CAMFCSet::GetDefaultSQL()
{
    return _T("[Adr].[txt]");
}

void CAMFCSet::DoFieldExchange(CFieldExchange* pFX)
{
    ///{AFX_FIELD_MAP(CAMFCSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[NACHNAME]"), m_NACHNAME);
    RFX_Text(pFX, _T("[VORNAME]"), m_VORNAME);
    RFX_Long(pFX, _T("[ALTER]"), m_ALTER);
    ///}AFX_FIELD_MAP
}

```

*Listing 36.4:
Implementie-
rung von CAMFC-
Set*

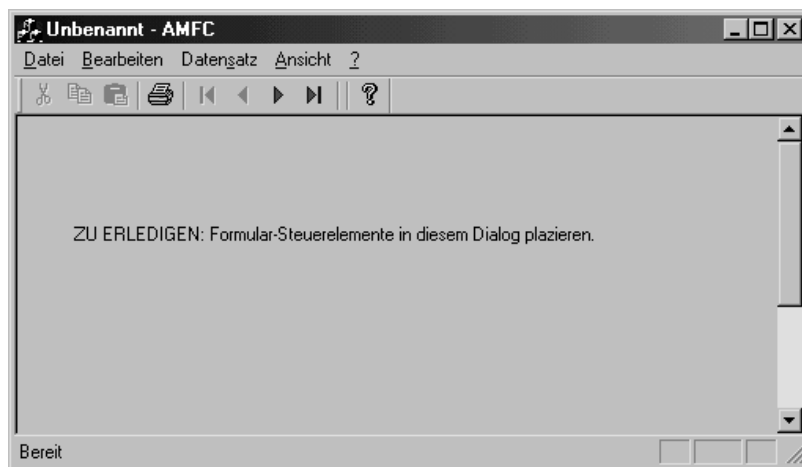
Visual C++ enthält einen schwerwiegenden Fehler. Beachten Sie bitte, daß die `GetDefault-Elementfunktion` die Zeichenfolge `"[Adr].[txt]"` zurückgibt. Dies führt zu einem SQL-Syntaxfehler, wenn Sie versuchen, die Anwendung ausführen zu lassen. Die korrekte Zeichenfolge muß `"[Adr.txt]"` lauten. Ändern Sie die Implementierung dieser Funktion, so daß diese die erforderliche Zeichenfolge zurückgibt.



In der Dokumentklasse und der Ansichtsklasse sind lediglich geringfügige Änderungen enthalten. Die Dokumentklasse verfügt über eine neue Elementvariable mit der Bezeichnung `m_aMFCSet`, die vom Typ `CAMFCSet` ist. Diese Variable repräsentiert die Tabelle, mit der das Anwendungsdokument verknüpft ist. In der Ansichtsklasse ist ebenfalls eine neue Elementvariable enthalten, deren Name `m_pSet` lautet. Diese verweist gewöhnlich auf das `m_aMFCSet`-Element der Dokumentklasse. In der Ansichtsklasse befindet sich außerdem die neue Funktion `OnGetRecordset`. Die Standardimplementierung dieser Funktion gibt den Wert von `m_pSet` zurück.

Das Anwendungsgerüst kann bereits kompiliert und ausgeführt werden. In dem gegenwärtigen Zustand ist dies jedoch nicht empfehlenswert. Wie in Abbildung 36.14 dargestellt, zeigt die Anwendung nicht mehr als einen leeren Dialog an. Wenngleich die Befehle zur Auswahl eines Datensatzes bereits ausgeführt werden können, sehen Sie lediglich das Sperren und die Freigabe von Menüeinträgen und Schaltflächen, wenn das Ende oder der Anfang der Tabelle erreicht wird. Sie sollten zunächst den Anwendungsdialog modifizieren, bevor Sie das Programm verwenden.

Abbildung 36.14:
Das ausgeführte Anwendungsgerüst



36.3.3 Bearbeiten der ODBC-Anwendung

Die Bearbeitung der ODBC-Anwendung zur Darstellung der Datensätze erfordert keine zusätzliche Programmzeile. Sie fügen dem Dialog der Anwendung lediglich einige Steuerelemente hinzu und verwenden den Klassen-Assistenten, um die entsprechenden Elementvariablen zu definieren.

Datenbankformular aufsetzen

Öffnen Sie zunächst den Dialog `IDD_AMFC_FORM` zur Bearbeitung. Entfernen Sie das statische `TODO`-Steuerelement, und fügen Sie dem Dialog drei statische Textfelder und drei Eingabefeldern hinzu, wie in Abbildung 36.15 dargestellt. Bezeichnen Sie die Textfelder mit `IDC_NACHNAME`, `IDC_VORNAME` und `IDC_ALTER`.

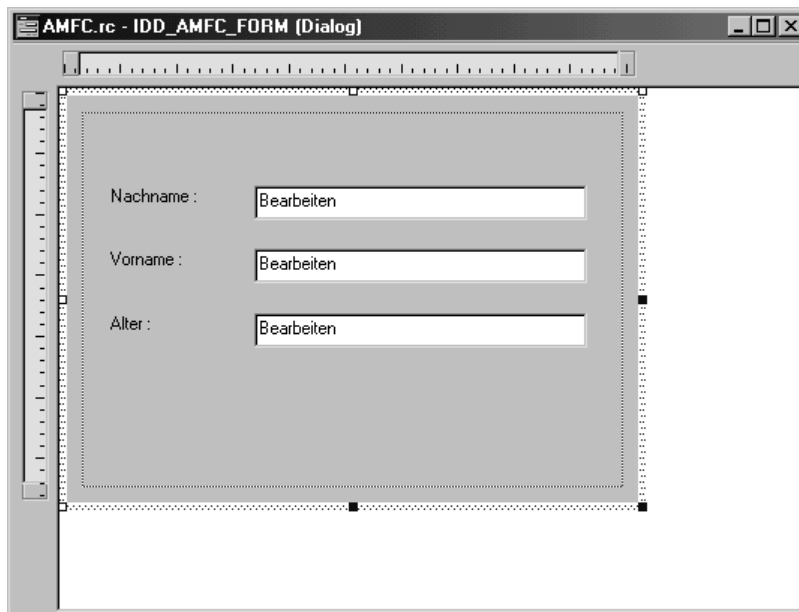


Abbildung 36.15:
Hinzufügen von
Steuerelemen-
ten zum AMFC-
Dialog

Elementvariable einrichten

1. Um der Klasse mit Hilfe des Klassen-Assistenten eine Elementvariable hinzuzufügen, drücken und halten Sie die Taste ([Strg]) und führen einen Doppelklick auf einem der Eingabefelder aus. Daraufhin wird der Klassen-Assistenten-Dialog MEMBER-VARIABLE HINZUFÜGEN geöffnet. Der Klassen-Assistent gibt darin den Namen der neuen Elementvariablen gemäß der Bezeichnung des entsprechenden Datenfelds vor (Auswahl über Listefeld, siehe Abbildung 36.16).

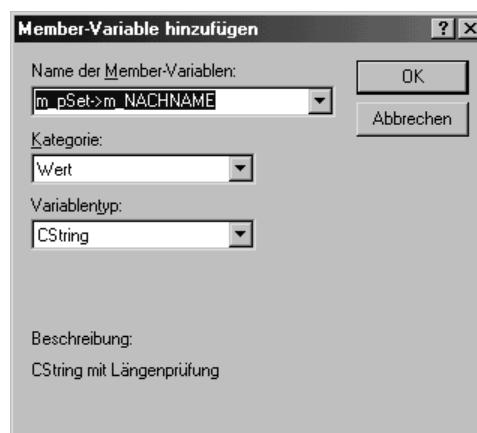
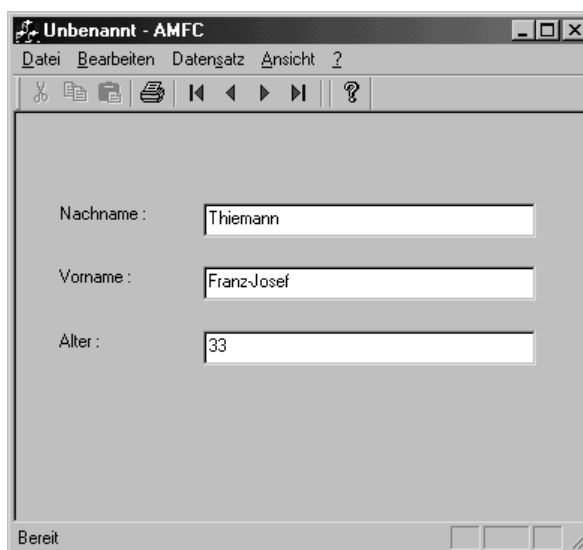


Abbildung 36.16:
Zuweisen einer
Datensatz-Ele-
mentvariablen
zu einem Dia-
logfeld-Steuer-
element

2. Wiederholen Sie diesen Schritt für die beiden verbleibenden Felder.
3. Kompilieren Sie die Anwendung anschließend erneut.

AMFC ist nun eine funktionsfähige Anwendung (Abbildung 36.17), die zur Darstellung der Datensätze in *ADR.TXT* verwendet werden kann.

Abbildung 36.17:
Die AMFC-
Anwendung



Dieser einfache Datensatz-Browser demonstriert natürlich nicht die gesamte ODBC-Funktionalität der MFC. Die verbleibenden Abschnitte des Kapitels beschreiben daher, welche Features außerdem von den ODBC-Klassen der MFC-Bibliothek unterstützt werden.

36.3.4 ODBC-Klassen in der MFC

Die von der MFC-Bibliothek angebotenen ODBC-Klassen sind in Abbildung 36.18 aufgeführt. Die beiden wichtigsten Klassen sind `CDatabase` und `CRecordset`.

CDatabase

Die `CDatabase`-Klasse repräsentiert eine Verbindung zu einer Datenquelle. Die Elementvariable dieser Klasse, die mit `m_hdbc` bezeichnet ist, nimmt einen ODBC-Verbindungs-Handle auf. Die Elementfunktionen `Open` und `Close` stellen die Verbindung zu der Datenquelle her oder beenden diese.

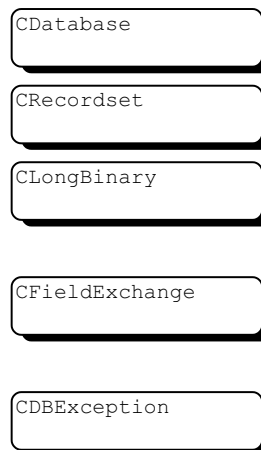


Abbildung 36.18:
ODBC-Klassen
in der MFC

Andere Elementfunktionen setzen oder ermitteln Verbindungseinstellungen. Dazu zählen `GetConnect` (gibt die ODBC-Verbindungszeichenfolge zurück), `IsOpen`, `GetDatabaseName`, `CanUpdate`, `CanTransact`, `InWaitForDataSource`, `SetLoginTimeout`, `SetQueryTimeout` und `SetSynchronousMode`. Die `CDatabase`-Klasse verwendet den asynchronen Modus, um auf die Datenquelle zuzugreifen. Die Ausführung einer asynchronen Operation kann mit der `Cancel`-Elementfunktion unterbrochen werden.

Die Transaktionsbearbeitung wird von den Elementfunktionen `BeginTrans`, `CommitTrans` und `Rollback` unterstützt.

Die `CDatabase`-Klasse bietet außerdem zwei überladbare Funktionen an. `OnSetOptions` setzt die Optionen für die Standardverbindung. `OnWaitForDataSource` wird von dem Anwendungsrahmen aufgerufen, um Prozessorzeit während des Ausführens einer zeitintensiven Operation abzugeben.

Die `ExecuteSQL`-Elementfunktion führt eine SQL-Anweisung direkt aus. Diese Funktion kann nicht in Verbindung mit SQL-Anweisungen verwendet werden, die Datensätze zurückgeben.

CRecordset

Die `CRecordset`-Klasse verfügt über die Funktionalität einer ODBC-SQL-Anweisung und dient der Bearbeitung der Spaltengruppe, die von dieser Anweisung zurückgegeben wird. Die Elementvariablen der Klasse bezeichnen den ODBC-Anweisungs-Handle, die Nummern der Felder und Parameter in dem Datensatz, das `CDatabase`-Objekt, über das ein Datensatz mit einer Datenquelle verbunden wird, und zwei Zeichenfolgen, die den SQL-Klauseln `WHERE` und `ORDER BY` entsprechen.

Datensatztypen Die beiden wichtigsten Datensatztypen sind `Dynasets` und `Snapshots`. Der Typ eines Datensatzes wird während eines Aufrufs von `CRecordset::Open` bestimmt.

- ➔ `Snapshots` repräsentieren eine statische Ansicht der Daten. Die Daten werden so dargestellt, wie sie zum Zeitpunkt der Erstellung des Snapshot vorlagen. Diese Art der Darstellung ist für das Generieren von Berichten sehr nützlich.
- ➔ `Dynasets` stellen eine dynamische Ansicht der Daten dar, die jede Änderung berücksichtigen, die von anderen Anwendern oder über andere Datensätze Ihrer Anwendung vorgenommen wurden.

Wird ein Datensatz mit der `Open`-Elementfunktion geöffnet, greift die Funktion auf die Tabelle zu und führt die Abfrage aus, die den Datensatz repräsentiert. Der Datensatz und der entsprechende Anweisungs-Handle können mit der `Close`-Elementfunktion geschlossen werden.

Die Attribute des Datensatzes werden mit den Elementfunktionen `CanAppend`, `CanRestart`, `CanScroll`, `CanTransact`, `CanUpdate`, `GetRecordCount`, `GetStatus`, `GetTableName`, `GetSQL`, `IsOpen`, `IsBOF`, `IsEOF` und `IsDeleted` ermittelt.

Einzelne Datensätze werden mit den Funktionen `Move`, `MoveFirst`, `MoveLast`, `MoveNext` und `MovePrev` selektiert.

Mit `AddNew`, `Delete`, `Edit` und `Update` fügen Sie der Datensatzgruppe neue Datensätze hinzu, löschen oder bearbeiten Datensätze und aktualisieren diese.

Weitere Datensatzfunktionen führen verschiedene Verwaltungsarbeiten aus.

Verwenden Sie `CRecordset` niemals direkt. Sie sollten eine eigene Klasse von `CRecordset` ableiten. Fügen Sie dieser Klasse die Elementvariablen hinzu, die den Feldern (Spalten) der Tabelle entsprechen, in der die Datensätze enthalten sind. Überschreiben Sie anschließend die `DoFieldExchange`-Elementfunktion. Die Funktion führt den Datenaustausch zwischen Elementvariablen und Datenbankfeldern mit Hilfe von `RFX_`-Funktionen aus. Diese Funktionen, deren Syntax und Konzepte denen der Dialog-Datenaustauschfunktionen (`DDX_`) ähnlich sind, finden Sie in Tabelle 36.1 aufgeführt.

Funktionsname	Feldtyp	ODBC-SQL-Typ
RFX_Binary	CByteArray	SQL_BINARY, SQL_LONGVARBINARY, SQL_VARBINARY
RFX_Bool	BOOL	SQL_BIT
RFX_Byte	BYTE	SQL_TINYINT
RFX_Date	CTime	SQL_DATE, SQL_TIME, SQL_TIMESTAMP
RFX_Double	double	SQL_DOUBLE
RFX_Int	int	SQL_SMALLINT
RFX_Long	LONG	SQL_INTEGER
RFX_LongBinary	CLongBinary	SQL_LONGVARCHAR
RFX_Single	float	SQL_REAL
RFX_Text	CString	SQL_CHAR, SQL_DECIMAL, SQL_LONGVARCHAR, SQL_NUMERIC, SQL_VARCHAR

Tabelle 36.1:
RFX_-Funktionen

CFieldExchange

Der Austausch von Feldinhalten geschieht mit Hilfe der `CFieldExchange`-Klasse. Ein Objekt dieser Klasse enthält Informationen über das Feld, dessen Inhalt mit `DoFieldExchange` einer Elementvariablen zugewiesen wird.

CRecordView

Die `CRecordView`-Klasse ist eine Ansichtsklasse, die von `CFormView` abgeleitet ist und die Datensätze einer Datenbank in Formularen darstellt. Objekte vom Typ `CRecordView` nutzen den Dialog-Datenaustausch (DDX) und den Datensatzfeldaustausch (RFX), um Daten zwischen dem Formular und der Datenquelle zu übertragen. Die von `CRecordView` abgeleiteten Objekte werden zusammen mit von `CRecordset` abgeleiteten Objekten verwendet.

ODBC-Operationen nutzen `CDBException` für die Fehlermeldung über den MFC-Ausnahmemechanismus.

36.4 Zusammenfassung

ODBC ist ein leistungsfähiger, auf SQL basierender und plattformunabhängiger Mechanismus für den Zugriff auf die Daten einer Datenquelle.

Den ODBC-Kern bilden ODBC-Treiber. Diese distribuierbaren DLLs implementieren den Zugriff auf unterschiedliche Datenquellen. Einstufige Treiber stellen die Verbindung zu der Datenquelle her und bearbeiten SQL-Anweisungen. Mehrstufige Treiber stellen die Verbindung her und übergeben SQL-Anweisungen an die Datenquelle. ODBC-Datenquellen können lokale Dateien (zum Beispiel Textdateien, dBase-Dateien oder Excel-Dateien) und entfernte Daten-Server sein (beispielsweise SQL-Server oder Oracle-Server).

Datenquellen werden gewöhnlich mit dem ODBC-Setup (in der Systemsteuerung) installiert. Mit einem Aufruf von `SQLDriverConnect` können Sie jedoch eine Verbindung zu einer Datenquelle aufbauen, die nicht auf diese Weise installiert wurde.

Während einer ODBC-Sitzung werden Funktionen aufgerufen, die die Verbindung herstellen, SQL-Anweisungen übergeben und das Ergebnis bearbeiten. Die ODBC-API definiert diese Funktionen. Die API bestimmt außerdem drei Koordinationsebenen (den Kern, Level 1 und Level 2). Die meisten Treiber unterstützen jedoch ausschließlich Level-1-ODBC-Funktionen.

ODBC unterstützt verschiedene Varianten des Standards der SQL-Syntax. Datenmanipulationsanweisungen, wie zum Beispiel `SELECT`, `INSERT`, `UPDATE` und `DELETE`, sowie Datendefinitionsanweisungen, wie `CREATE TABLE`, `DROP TABLE`, `CREATE INDEX`, `DROP INDEX` und `ALTER TABLE`, können von Ihnen verwendet werden. ODBC unterstützt außerdem die `CREATE-VIEW-SQL`-Anweisung.

Die MFC-Bibliothek enthält zwei ODBC-Klassen. Die `CDatabase`-Klasse repräsentiert eine ODBC-Verbindung. Die `CRecordset`-Klasse repräsentiert eine ODBC-SQL-Anweisung und die Datensätze, die von dieser Anweisung zurückgegeben werden. Anwendungen leiten eine Klasse gewöhnlich von `CRecordset` ab und fügen dieser Elementvariablen hinzu, die den Tabellenspalten entsprechen. Die `CRecordset`-Klasse verfügt über Elementfunktionen, mit deren Hilfe Datensätze selektiert und bearbeitet werden können.

DAO – Datenzugriffsobjekte

Kapitel 37

Datenzugriffsobjekte (DAOs – Data Access Objects) gehören zu Microsofts aktuellsten Innovationen in der Datenbankzugriffstechnologie. Diese Technologie wird für den Datenbankzugriff unter Visual Basic, Microsoft Access und Visual Basic für Anwendungen verwendet. Seit Visual C++ 4 stehen die Datenzugriffsobjekte ebenfalls dem C++-Programmierer über einige MFC-Klassen zur Verfügung.

DAO wird in Form von Komponenten angeboten, die von dem Anwender distribuiert werden dürfen.

37.1 DAO-Übersicht

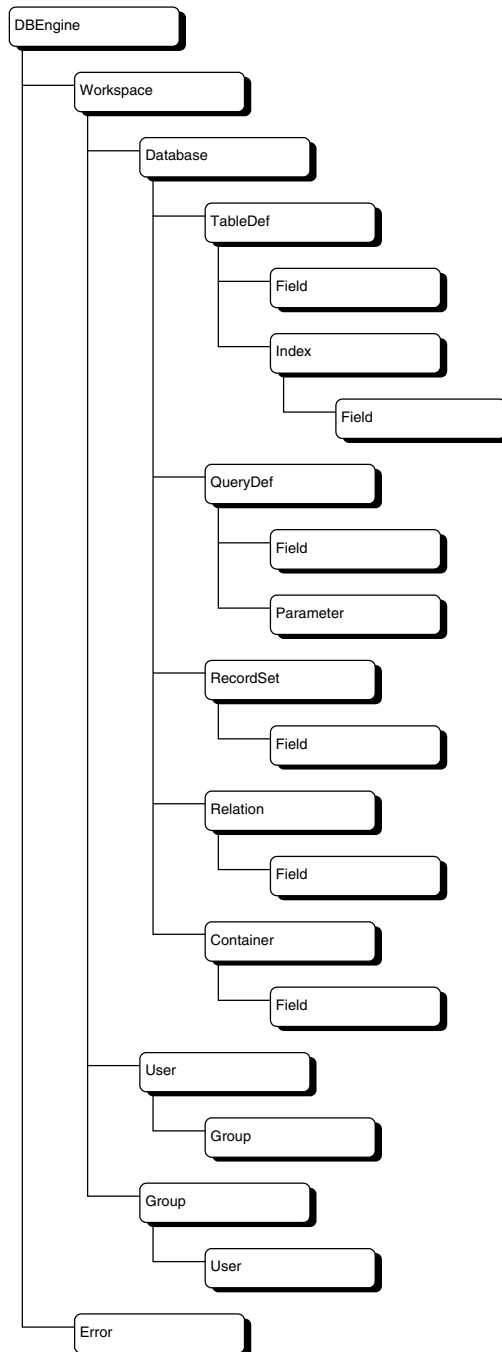
Datenzugriffsobjekte ermöglichen Ihnen den Zugriff auf Datenbanken und deren Bearbeitung über das Microsoft-Jet-Datenbankmodul. Mit Hilfe dieses Moduls können Sie auf Daten in Microsoft-Access-Datenbanken zugreifen (MDB-Dateien). Diese Technologie ermöglicht Ihnen ebenfalls den Zugriff auf lokale und entfernte Datenbanken über ODBC-Treiber.

Die DAO-Technologie basiert auf OLE. Abbildung 37.1 stellt die Hierarchie der Datenzugriffsobjekte dar. Die DAO-Klassen der MFC haben diese Hierarchie vereinfacht übernommen.

Sehr viele DAO-Funktionen nutzen SQL-Anweisungen (Structured Query Language).

- ➡ Verwenden Sie die SQL-Anweisung SELECT, um Daten aus einer Datenbank einzulesen.
- ➡ Die SQL-Anweisungen UPDATE, INSERT und DELETE modifizieren den Inhalt einer Datenbank.

Abbildung 37.1:
DAO-Objekt-
hierarchie



Eine einfache Möglichkeit, SQL-Anweisungen zu erstellen, die mit DAO-Objekten verwendet werden können, besteht darin, die Abfrage mit Microsoft Access zu generieren und in der Datenbank zu speichern. Sie können anschließend mit einem `QueryDef`-Objekt auf die Abfrage zugreifen.

Visual C++ unterstützt das Erstellen von DAO-Anwendungen mit Hilfe des Anwendungsassistenten. Mit dem Assistenten erzeugen Sie Anwendungen, die auf DAO-Klassen basieren. In diesem Kapitel werden Sie eine einfache DAO-Anwendung generieren und untersuchen.

37.2 Erstellen einer DAO-Anwendung

Das Erstellen einer DAO-Anwendung ist sehr einfach. Sie legen zunächst eine Datenquelle an. Für die in diesem Kapitel vorgestellte Anwendung verwenden wir eine einfache Access-Datenbank mit zwei Tabellen. Anschließend erzeugen Sie mit dem Anwendungsassistenten das Anwendungsgerüst. Schließlich bearbeiten Sie die Anwendung, so daß diese unseren Anforderungen gerecht wird.

Die DAO-Anwendung wird ein einfacher Browser sein, mit dessen Hilfe die Navigation innerhalb einer relationalen Verknüpfung möglich ist. Die Verknüpfung besteht aus den Datensätzen der beiden Tabellen.

37.2.1 Die Datenbank

Die für das Beispielprogramm verwendete Datenbank enthält zwei Tabellen.

- ➔ Eine Tabelle nimmt den Vornamen, Nachnamen und das Alter der Angestellten eines Unternehmens auf.
- ➔ Die zweite Tabelle enthält die mit Buchstaben bezeichneten Gehaltsstufen sowie das Alter, bis zu dem ein Angestellter mit der entsprechenden Gehaltsstufe vergütet wird.

Die Anwendung, die Sie bitte ADAO nennen, führt für jeden Angestellten die Gehaltsstufe auf, die dieser erreichen kann.

Erstellen Sie die Datenbank mit Microsoft Access.



Beachten Sie bitte, daß die folgende Beschreibung auf Access für Windows 95 basiert. Wenn Sie eine spätere Access-Version verwenden, können die Datenbank und der vom Anwendungsassistenten generierte Programmcode von der Datenbank und dem Programm abweichen, die in diesem Kapitel erläutert werden. Die Benutzeroberfläche von Access für Windows 95 unterscheidet sich ebenfalls von der Oberfläche späterer Versionen. Berücksichtigen Sie diesen Umstand bitte bei den Abbildungen und Beschreibungen.

1. Starten Sie Access, und öffnen Sie eine leere Datenbank mit der Bezeichnung *ADAO.MDB* in einem Verzeichnis Ihrer Wahl.
2. Öffnen Sie im daraufhin angezeigten Dialog das Register *TABELLEN*, und betätigen Sie die Schaltfläche *NEU*.
3. Selektieren Sie in dem Dialog *NEUE TABELLE* den Eintrag *ENTWURFSANSICHT*.

Abbildung 37.2:
Erstellen der
Kunden-Tabelle

Feldname	Felddatentyp	Beschreibung
Nachname	Text	
Vorname	Text	
Alter	Zahl	

Feldeigenschaften

Allgemein | Nachschlagen

Feldgröße: 50
 Format:
 Eingabeformat:
 Beschriftung:
 Standardwert:
 Gültigkeitsregel:
 Gültigkeitsmeldung:
 Eingabe erforderlich: Nein
 Leere Zeichenfolge: Nein
 Indiziert: Nein

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

4. Abbildung 37.2 zeigt die neu erstellte Tabelle *Kunden*. Wie Sie sehen, wurden der Tabelle drei Felder hinzugefügt (*Nachname*, *Vorname* und *Alter*). Die ersten beiden dieser Felder können bis zu fünfzig Zeichen aufnehmen. Das dritte Feld ist numerisch. Weisen Sie der Kombination der Felder *Nachname* und *Vorname* den Primärschlüssel zu. Speichern Sie die Tabelle unter dem Namen *Kunden*.



Abbildung 37.3:
Erstellen der
Gehalt-Tabelle

5. Erstellen Sie eine weitere Tabelle (Abbildung 37.3). Diese Tabelle enthält Informationen über die Gehaltsstufen. Sie besteht aus zwei Feldern. Weisen Sie dem ersten Feld mit der Bezeichnung `Name` den Primärschlüssel zu. Speichern Sie die Tabelle bitte unter dem Namen `Gehalt`.

Nachdem Sie die beiden Tabellen erstellt haben, zeigt das Datenbankfenster diese an, wie in Abbildung 37.4 dargestellt.

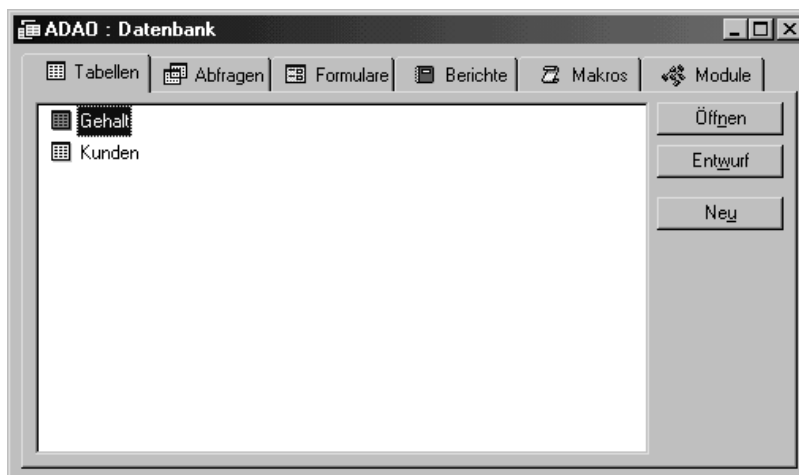


Abbildung 37.4:
Die Tabellen in
Adao.mdb

Fügen Sie den Tabellen nun die erforderlichen Daten hinzu.

1. Führen Sie dazu einen Doppelklick auf dem Tabellennamen im Datenbankfenster aus, und geben Sie dann Ihre Daten in die Tabelle ein.

Abbildung 37.5 zeigt die vier Datensätze, die Sie in `Kunden` anlegen. Abbildung 37.6 führt die drei Datensätze der Tabelle `Gehalt` auf.

Abbildung 37.5:
Datensätze in
der Kunden-
Tabelle

	Nachname	Vorname	Alter
	Koch	Manuela	22
	Koch	Christine	46
	Thiemann	Franz-Joseph	64
	Berglar	Christine	18
*			0

Abbildung 37.6:
Datensätze in
der Gehalt-
Tabelle

	Name	MaxAlter
	Plan 'A'	25
	Plan 'B'	40
	Plan 'C'	65
*		0

Sie können Microsoft Access nun beenden. Die MDB-Datei kann anschließend von unserer C++-DAO-Anwendung verwendet werden.

37.2.2 Erstellen des Anwendungsgerüsts

1. Rufen Sie den Anwendungsassistenten auf, um die ADAO-Anwendung zu erstellen.
2. Selektieren Sie bitte im ersten Schritt des Assistenten die Option EINZELNES DOKUMENT (SDI) und im zweiten Schritt DATENBANKANSICHT OHNE DATEIUNTERSTÜTZUNG (Abbildung 37.7).
3. Betätigen Sie die Schaltfläche DATENQUELLE, um eine Datenquelle auszuwählen. Selektieren Sie im anschließend angezeigten Dialog DATENBANKOPTIONEN die Option DAO (Abbildung 37.8).
4. Klicken Sie nun auf die mit dem Auslassungszeichen (...) beschriftete Schaltfläche. Im daraufhin dargestellten Dialog können Sie die Datenbankdatei auswählen.

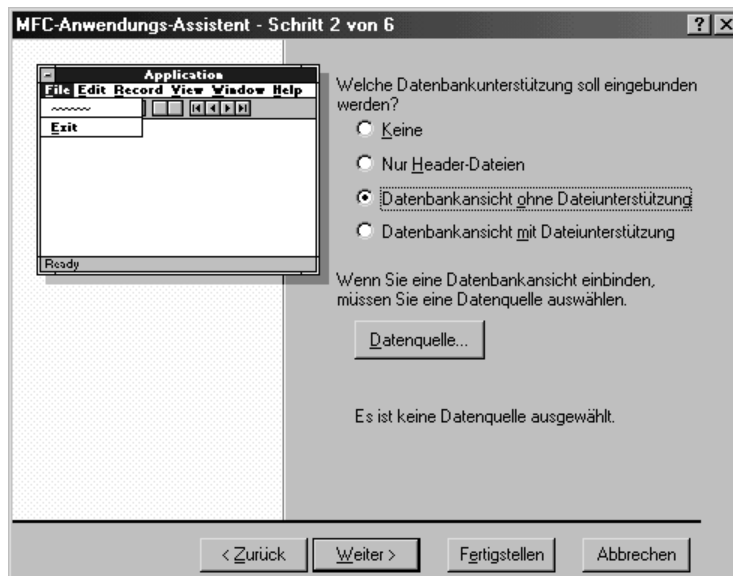


Abbildung 37.7:
Hinzufügen der
Datenbankun-
terstützung zum
Anwendungsge-
rüst

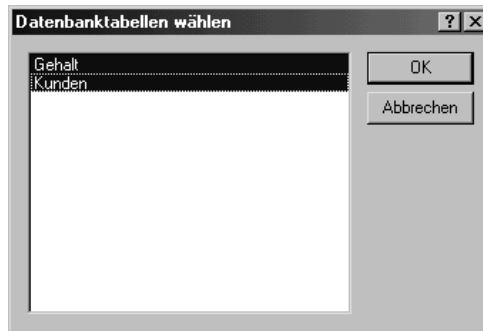


Abbildung 37.8:
Hinzufügen
einer DAO-
Datenquelle

5. Wählen Sie in diesem Dialog die von Ihnen mit Access generierte Datei *ADAO.MDB* aus.
6. Verlassen Sie danach den Dialog DATENBANKOPTIONEN mit einem Klick auf OK. Ein weiterer Dialog wird angezeigt, in dem Sie die Tabellen auswählen können.
7. Selektieren Sie dort sowohl die Tabelle *Kunden* als auch *Gehalt*, und betätigen Sie anschließend die Schaltfläche OK (Abbildung 37.9).

Sie sehen nun wieder den Dialog des Anwendungsassistenten vor sich. Unter der Schaltfläche DATENQUELLE wird die ausgewählte Datenquelle angezeigt.

Abbildung 37.9:
Die Auswahl der
Tabellen



Alle weiteren Optionen des Assistenten bleiben unverändert.

8. Betätigen Sie daher die Schaltfläche FERTIGSTELLEN, und das Anwendungsgerüst wird erstellt.

37.2.3 Das DAO-Anwendungsgerüst

Die Klassen des vom Anwendungsassistenten generierten Anwendungsgerüsts sind in Abbildung 37.10 dargestellt.

Im Vergleich zu einer Anwendung, die keine Datenbankfunktionalität unterstützt, verfügt ADAO über eine zusätzliche Klasse `CDAOSet` sowie einige neue Elementvariablen und Funktionen in der Dokument- und Ansichtsklasse. Ein weiterer Unterschied besteht darin, daß die Ansichtsklasse von `CDaoRecordView` abgeleitet ist.



Wenn Sie mit der MFC-ODBC-Programmierung vertraut sind, werden Sie feststellen, daß die Struktur der Anwendung kaum von einer mit dem Anwendungsassistenten generierten ODBC-Anwendung differiert.

Die neue Klasse `CDAOSet` ist von `CDaoRecordset` abgeleitet und repräsentiert die Datensatzgruppe, die wir in der Verknüpfung der Tabellen `Kunden` und `Gehalt` selektieren. Der Anwendungsassistent hat in die Klassendeklaration (Listing 37.1) Elementvariablen eingefügt, die den Spalten (Feldern) der beiden Tabellen entsprechen.

Listing 37.1:
CDAOSet-Klassendeklaration

```
class CDAOSet : public CDaoRecordset
{
public:
    CDAOSet(CDaoDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(CDAOSet)

// Feld-/Parameterdaten
//{{AFX_FIELD(CDAOSet, CDaoRecordset)
    CString m_Name;
    long m_MaxAlter;
```

```

CString  m_Nachname;
CString  m_Vorname;
long     m_Alter;
//))AFX_FIELD

// Überladungen
// Vom Klassenassistenten generierte Überladungen virtueller
// Funktionen
//))AFX_VIRTUAL(CADAOSet)
public:
virtual CString GetDefaultDBName(); // ÜBERARBEITEN: Hier
                                   // einen Befehl holen
virtual CString GetDefaultSQL();   // Standard-SQL für
                                   // Recordset
virtual void DoFieldExchange(CDaoFieldExchange* pFX); // RFX-
                                                       // Unterstützung
//))AFX_VIRTUAL

// Implementierung
#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
};

```

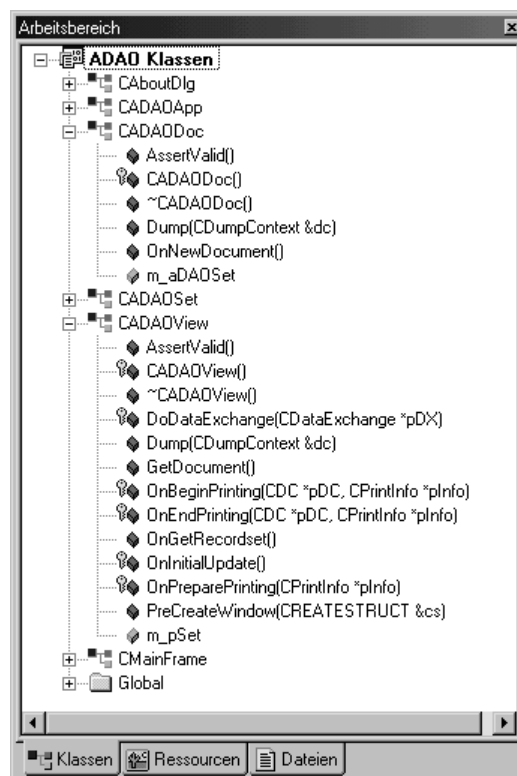


Abbildung 37.10:
Die Klassen des
Anwendungsge-
rüsts

Die Implementierung von `CADAOSet` (Listing 37.2) zeigt, wie die Elementvariablen in dem Konstruktor der Klasse initialisiert werden. Auch die vom Anwendungsassistenten erzeugte `DoFieldExchange`-Elementfunktion greift auf diese Variablen zu. Die Funktion tauscht Daten zwischen Elementvariablen der Klasse und Feldern der Datenbank aus.

Listing 37.2:
CADAOSet-
Klassenimple-
mentierung

```
IMPLEMENT_DYNAMIC(CADAOSet, CDAORecordset)

CADAOSet::CADAOSet(CDAODatabase* pdb)
: CDAORecordset(pdb)
{
    //{AFX_FIELD_INIT(CADAOSet)
    m_Name = _T("");
    m_MaxAlter = 0;
    m_Nachname = _T("");
    m_Vorname = _T("");
    m_Alter = 0;
    m_nFields = 5;
    //}AFX_FIELD_INIT
    m_nDefaultType = dbOpenDynaset;
}

CString CADAOSet::GetDefaultDBName()
{
    return _T("C:\\Markt&T\\VC_Kompodium\\Programme\\Kap37\\ADA0\\Adao.mdb");
}

CString CADAOSet::GetDefaultSQL()
{
    return _T("[Gehalt],[Kunden]");
}

void CADAOSet::DoFieldExchange(CDAOFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(CADAOSet)
    pFX->SetFieldType(CDAOFieldExchange::outputColumn);
    DFX_Text(pFX, _T("[Name]"), m_Name);
    DFX_Long(pFX, _T("[MaxAlter]"), m_MaxAlter);
    DFX_Text(pFX, _T("[Nachname]"), m_Nachname);
    DFX_Text(pFX, _T("[Vorname]"), m_Vorname);
    DFX_Long(pFX, _T("[Alter]"), m_Alter);
    //}AFX_FIELD_MAP
}
```

`DoFieldExchange` verwendet `DFX_-`Funktionen. Diese gleichen den `RFX_-`Funktionen für den ODBC-Datensatzfeldaustausch. Die `DFX_-`Funktionen sind in Tabelle 37.1 aufgeführt.

Tabelle 37.1:
DFX_-Funktionen

Funktionsname	Feldtyp	ODBC-SQL-Typ
DFX_Binary	CByteArray	DAO_BYTES
DFX_Bool	BOOL	DAO_BOOL
DFX_Byte	BYTE	DAO_BYTES
DFX_Currency	COleCurrency	DAO_CURRENCY

Funktionsname	Feldtyp	ODBC-SQL-Typ
DFX_DateTime	ColeDateTime	DAO_DATE
DFX_Double	double	DAO_R8
DFX_Long	long	DAO_I4
DFX_LongBinary	CLongBinary	DAO_BYTES
DFX_Short	short	DAO_I2
DFX_Single	float	DAO_R4
DFX_Text	CString	DAO_CHAR, DAO_WCHAR

Anwendungen sollten nicht die *DFX_LongBinary-Funktion*, sondern *statt dessen DFX_Binary verwenden*. *DFX_LongBinary dient der Kompatibilität mit ODBC.*



Die neuen Elementvariablen und Funktionen in der Ansichts- und Dokumentklasse unserer Anwendung, haben einfache Aufgaben. Die Dokumentklasse *CADAODoc* enthält die neue Elementvariable *m_aDAOSet* vom Typ *CADAOSet*. Diese Variable repräsentiert die Datensatzgruppe, mit der das Dokument verknüpft ist.

CADAODoc

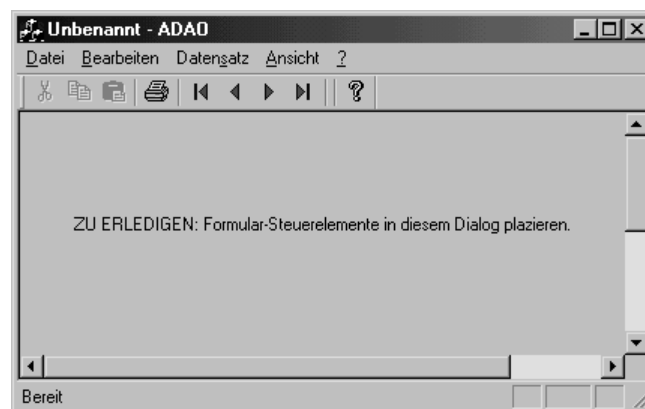


Abbildung 37.11:
Ausführen des
ADAO-Anwen-
dungsgerüsts

Die Ansichtsklasse verfügt über einen Zeiger vom Typ *CADAOSet* (*m_pSet*). In der Standardimplementierung wird dieser Zeiger auf das Element *m_aDAOSet* des Dokumentobjekts gesetzt. Die Ansichtsklasse enthält außerdem die neue Elementfunktion *OnGetRecordset*, die in der Standardimplementierung *m_pSet* zurückgibt.

CADAView

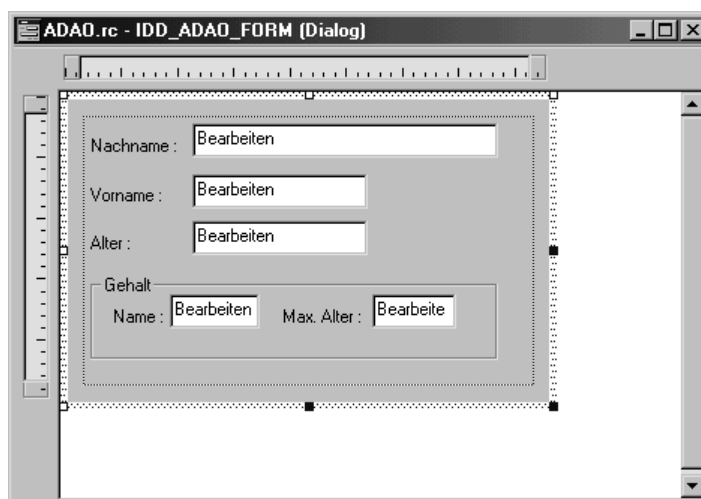
Sie können die ADAO-Anwendung bereits kompilieren. Abbildung 37.11 zeigt jedoch, daß die Anwendung in diesem Zustand keine besondere Funktion ausführt. Sie müssen zunächst den Dialog bearbeiten. Außerdem müssen der Anwendung die erforderlichen Operationen hinzugefügt werden, die die von uns selektierten Datensätze anzeigen.

37.2.4 Bearbeiten der Anwendung

Sie werden zunächst den Hauptdialog der ADAO-Anwendung bearbeiten.

1. Öffnen Sie den mit `IDD_ADAO_FORM` bezeichneten Dialog zur Bearbeitung.
2. Entfernen Sie das statische TODO-Steuerelement, und fügen Sie dem Dialog die in Abbildung 37.12 dargestellten Bezeichnungs- und Eingabefelder hinzu.

Abbildung 37.12:
Bearbeiten des
ADAO-Dialogs



3. Nennen Sie die fünf Eingabefelder `IDC_NACHNAME`, `IDC_VORNAME`, `IDC_ALTER`, `IDC_NAME` und `IDC_MAXALTER`.

Bevor Sie den Dialog schließen, sollten Sie den Dialogfeldern die entsprechenden Elementvariablen mit Hilfe des Klassen-Assistenten zuweisen.

4. Halten Sie die Taste `Strg` gedrückt, und führen Sie anschließend einen Doppelklick über dem Eingabefeld `IDC_NACHNAME` aus.

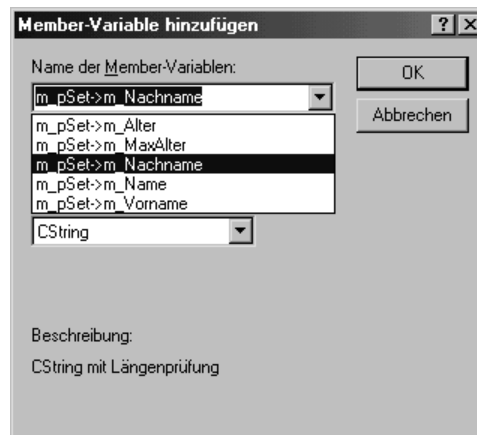


Abbildung 37.13:
Hinzufügen
einer Datensatz-
Elementvariablen

5. Daraufhin wird der Klassen-Assistenten-Dialog MEMBER-VARIABLE HINZUFÜGEN aufgerufen. Der Name der Elementvariable kann dort bereits aus dem zugehörigen Listensfeld ausgewählt werden (Abbildung 37.13).

Verfahren Sie analog für die restlichen Eingabefelder im Dialog.

Sie können den Dialog schließen. Bestimmen Sie nun die Auswahlkriterien, die unsere Anwendung lediglich die Datensätze anzeigen läßt, die die zulässigen Gehaltsstufen jedes Angestellten repräsentieren.

*Datensätze
filtern*

6. Öffnen Sie die `CADAOSet::GetDefaultSQL`-Funktion, um die Auswahlkriterien zu ändern.

Die Standardimplementierung dieser Funktion gibt die Bezeichnungen der Tabellen zurück, die die Datensätze enthalten. Wir fügen der Funktion zusätzliche Kriterien hinzu, um ausschließlich solche Datensätze in der Tabelle zu selektieren, die in den Feldern der Anwendung angezeigt werden sollen.

Die gewünschte Auswahl kann in der strukturierten Abfragesprache SQL formuliert werden. Verwenden Sie dazu die folgende **SELECT-Anweisung**:

```
SELECT Kunden.Nachname, Kunden.Vorname, Kunden.Alter, Gehalt.Name,
       Gehalt.MaxAlter
FROM Kunden, Gehalt
WHERE Kunden.Alter < Gehalt.MaxAlter
ORDER BY Kunden.Nachname, Kunden.Vorname, Gehalt.Name
```

Um unsere Auswahl zu spezifizieren, könnten wir `GetDefaultSQL` modifizieren, so daß die Funktion eine Zeichenfolge zurückgibt, die der zuvor aufgeführten SQL-SELECT-Anweisung entspricht.

Eine andere Möglichkeit besteht darin, die Elementvariablen der CDaoRecordset-Klasse zu nutzen.

CDaoRecordset verfügt über zwei Elementvariablen. Die Variable m_strFilter entspricht der SQL-Klausel WHERE, und m_strSort besitzt die Funktionalität der ORDER BY-Klausel. Die neue Version von CDAOSet::GetDefaultSQL (Listing 37.3) verwendet diese Elementvariablen, um die gewünschten Datensätze zu selektieren.

*Listing 37.3:
Die aktualisierte
Version von
CDAOSet::Get-
DefaultSQL*

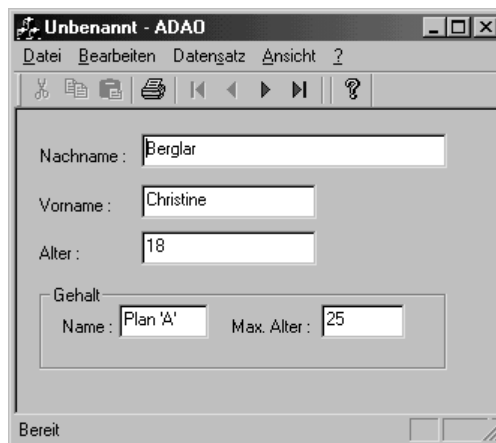
```
CString CDAOSet::GetDefaultSQL()
{
    m_strFilter = _T("[Kunden].[Alter] < [Gehalt].[MaxAlter]");
    m_strSort =
        _T("[Kunden].[Nachname],[Kunden].[Vorname],[Gehalt].[Name]");
    return _T("[Kunden],[Gehalt]");
}
```

Das ADAO-Projekt ist nun vollständig.

7. Kompilieren Sie die Anwendung, und starten Sie das Programm (Abbildung 37.14).

Die Anwendung zeigt die Gehaltsstufen an, mit denen die Angestellten vergütet werden können. Die Datensätze sind nach dem Namen des Angestellten und der Bezeichnung der Gehaltsstufen sortiert.

*Abbildung 37.14:
Die ausgeführte
ADAO-
Anwendung*



37.3 DAO-Klassen

Wenngleich die ADAO-Anwendung demonstriert, wie ein einfaches DAO-Programm erstellt werden kann, implementiert sie nicht alle DAO-Feature. Die verbleibenden Abschnitte dieses Kapitels geben daher eine kurze Übersicht über die MFC-DAO-Klassen und deren Möglichkeiten.

Die von der MFC zur Verfügung gestellten DAO-Klassen sind in Abbildung 37.15 aufgeführt. Zusätzlich zur `CDaoRecordset`-Klasse, die wir während der Erstellung unseres Projekts kennengelernt haben, bestehen fünf weitere Hauptklassen und zwei Helferklassen. Verglichen mit der Vielzahl unterschiedlicher DAO-Objekte (die zu Beginn dieses Kapitels in Abbildung 37.1 dargestellt sind), sind diese Klassen vorteilhafter, da ihre Anzahl geringer ist.

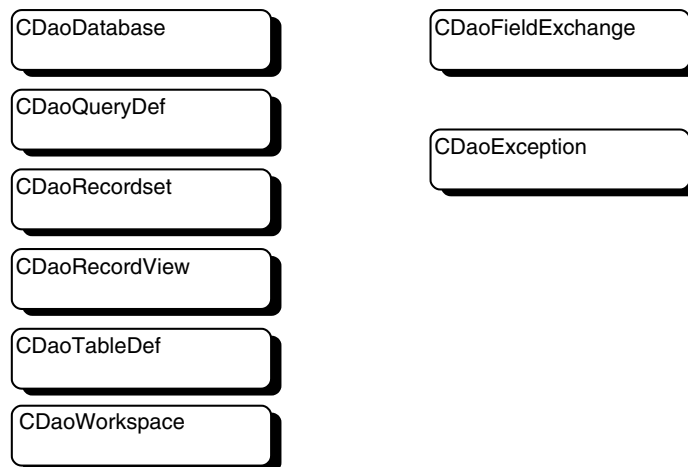


Abbildung 37.15:
DAO-Klassen

Funktionsweise der Datenzugriffsobjekte

Bevor wir die Feature dieser Klassen erläutern, erfahren Sie, wie Datenzugriffsobjekte arbeiten. Betrachten Sie dazu nochmals die Abbildung 37.1.

Alle DAO-Objekte werden vom `DBEngine`-Objekt abgeleitet, alle Datenbankobjekte wiederum von DAO-Arbeitsbereichsobjekten. Sie greifen nur dann auf diese Objekte zu, wenn Sie Sicherheitsdatenbanken bearbeiten möchten. In allen anderen Situationen genügt ein Standardarbeitsbereichsobjekt für alle Transaktionen.

Die Datenbank- und Datensatzobjekte repräsentieren Datenbanken und selektierte Datensatzgruppen (Tabellen, Datensätze oder Dynasets) in diesen Datenbanken.

Abfragedefinitionsobjekte (`QueryDef`) führen spezifische SQL-Abfragen in einer Datenbank aus. Abfragedefinitionen werden gewöhnlich in Verbindung mit Datensätzen verwendet, um über eine bestimmte Abfrage auf die Daten einer Datenbank zuzugreifen.

Tabellendefinitionsobjekte (`TableDef`) repräsentieren die Struktur der Tabellen einer Datenbank. Mit Hilfe dieser Objekte können neue Tabellen erstellt und die Struktur sowie die Eigenschaften bereits bestehender Tabellen modifiziert werden.

Weitere DAO-Objekte, wie zum Beispiel `Field`, `Parameter`, `Index`, `User`, `Group` und `Error`, werden nicht durch MFC-Klassen repräsentiert. Statt dessen geschieht der Zugriff auf diese Objekte über andere DAO-MFC-Klassen.

37.3.1 Die `CDaoRecordset`-Klasse

`CDaoRecordset`-Objekte repräsentieren Datensatzgruppen. Eine Datensatzgruppe enthält die Datensätze einer Tabelle, eines Dynasets oder eines Snapshot. Eine Tabellendatensatzgruppe kann aktualisiert werden und enthält die Datensätze einer oder mehrerer Tabelle(n), die den Anforderungen einer Abfrage entsprechen. Dynaset-Datensatzgruppen können ebenfalls aktualisiert werden. Ein Snapshot kann ebenfalls die Felder einer oder mehrerer Tabellen enthalten. Diese können jedoch nicht aktualisiert werden. Das Snapshot ist eine statische Kopie der Datensätze und wird dazu verwendet, nach Datensätzen zu suchen oder Berichte zu generieren.

Datensatzgruppen erstellen Eine Datensatzgruppe wird mit `CDaoRecordset::Open` erstellt. Drei Varianten dieser Funktion ermöglichen Ihnen das Erzeugen einer Datensatzgruppe anhand einer SQL-Zeichenfolge sowie das Erstellen eines `CDaoTableDef`-Objekts oder eines `CDaoQueryDef`-Objekts.

Elementfunktionen Die `CDaoRecordset`-Klasse bietet eine Vielzahl verschiedener Elementfunktionen an. Die wohl wichtigsten Funktionen dienen der Navigation innerhalb einer Datenbank und der Aktualisierung von Datensatzgruppen. Zu den Navigationsfunktionen zählen `Find`, `FindFirst`, `FindLast`, `FindNext`, `FindPrev`, `Move`, `MoveFirst`, `MoveLast`, `MoveNext` und `MovePrev`. Zu den Aktualisierungsfunktionen zählen `AddNew`, `CancelUpdate`, `Delete`, `Edit` und `Update`.

Weitere Navigationsfunktionen sind `GetAbsolutePosition`, `GetBookmark`, `GetPercentPosition`, `SetAbsolutePosition`, `SetBookmark` und `SetPercentPosition`.

Die `CDaoRecordset`-Klasse bietet verschiedene Attributfunktionen an, mit deren Hilfe die Attribute von Datensatzgruppen gesetzt und ermittelt werden können. Die `CanUpdate`-Funktion bestimmt beispielsweise, ob eine Datensatzgruppe aktualisiert werden kann. Die `SetCurrentIndex`-Funktion setzt den aktuellen Index in einer Tabellendatensatzgruppe.

Gewöhnlich leiten Sie Ihre eigene Klasse von `CDaoRecordset` ab, fügen dieser Elementvariablen hinzu, die die Felder repräsentieren, und überschreiben die `DoFieldExchange`-Funktion, um den Datenaustausch zwischen der Datenbank und den Elementvariablen zu ermöglichen. Darüber hinaus bestehen jedoch weitere Elementfunktionen, die eine Alternative zu dieser Vorgehensweise anbieten. Dazu zählen `GetFieldValue` und `SetFieldValue`, die den direkten Zugriff auf den Wert eines Feldes gestatten. Dieses Verfahren wird im Gegensatz zum statischen Binden mit `DoFieldExchange` als dynamisches Binden bezeichnet.

Eigene Klassen ableiten

Andere Datensatzgruppenoperationen steuern den lokalen Cache-Speicher der Datensätze und bearbeiten Datensatzgruppenindizes.

37.3.2 Die `CDaoRecordView`-Klasse

Die `CDaoRecordView`-Klasse ist eine Ansichtsklasse zur Darstellung von Datenbankinhalten in passenden Steuerelementen. `CDaoRecordView` ist von `CFormView` abgeleitet und stellt eine dialogbasierte Ansicht dar, die mit einem `CDaoRecordSet`-Objekt verbunden ist und dessen Datenbankfelder mit Steuerelementen verknüpft.

`CDaoRecordView` verfügt über drei Attributelementfunktionen: `OnGetRecordset`, `IsOnLastRecord` und `IsOnFirstRecord`.

37.3.3 Die `CDaoDatabase`-Klasse

Die `CDaoDatabase`-Klasse repräsentiert eine Verbindung zu einer Datenbank. Diese Verbindung wird mit einem Aufruf von `CDaoDatabase::Open` erstellt und mit `CDaoDatabase::Close` geschlossen. Eine neue Datenbank erzeugen Sie mit `CDaoDatabase::Create`.

`CDaoDatabase` verfügt über einige Attributelementfunktionen. Die `GetName`-Elementfunktion ermittelt beispielsweise den Namen der Datenbank. `IsOpen` bestimmt, ob die Verbindung geöffnet ist.

Weitere Elementfunktionen bearbeiten die Auflistung der Tabellendefinitions- und Abfragedefinitionsobjekte, die für eine Datenbank definiert wurden. Die `DeleteTableDef`-Elementfunktion löscht nicht ausschließlich ein `DAO-TableDef`-Objekt, sondern ebenfalls die zugrundeliegende Tabelle und die darin enthaltenen Daten.

37.3.4 Die `CDaoWorkspace`-Klasse

Die `CDaoWorkspace`-Klasse repräsentiert eine Datenbanksitzung. Sie müssen gewöhnlich keine Objekte von dieser Klasse erstellen. Dies ist nur dann erforderlich, wenn Sie bestimmte Funktionen dieser Klasse nutzen oder auf durch ein Paßwort geschützte Datenbanken zugreifen möchten.

Ein DAO-Arbeitsbereich wird mit `CDaoWorkspace::Create` erstellt. Die Parameter der Funktion bestimmen die Bezeichnung des Arbeitsbereichs, den Anwendernamen und das Paßwort. Ein bereits bestehendes Arbeitsbereichsobjekt kann mit `CDaoWorkspace::Open` geöffnet werden. Wenn Sie dieser Funktion einen `NULL`-Parameter übergeben, öffnen Sie den Standardarbeitsbereich.

Mehrere Elementfunktionen der Klasse bearbeiten Datenbanken und das Datenbankmodul. Sie können eine Datenbank beispielsweise komprimieren und reparieren, indem Sie die Funktionen `CompactDatabase` und `RepairDatabase` aufrufen. Andere Funktionen manipulieren den Anwendernamen, das Paßwort und weitere Datenbankattribute.

37.3.5 Die `CDaoQueryDef`-Klasse

Die `CDaoQueryDef`-Klasse repräsentiert Abfragedefinitionen. Sie erstellen mit der `CDaoQueryDef::Create`-Elementfunktion eine neue Abfragedefinition. Um auf eine Abfragedefinition zuzugreifen, die in einer Datenbank gespeichert ist, verwenden Sie `CDaoQueryDef::Open`. Sie können der Datenbank eine neue Abfrage hinzufügen, indem Sie die `CDaoQueryDef::Append`-Elementfunktion aufrufen.

`CDaoQueryDef`-Objekte können zusammen mit `CRecordSet`-Objekten verwendet werden, um Daten aus einer Datenbank auszulesen. `CDaoQueryDef`-Objekte werden ebenfalls direkt verwendet. Um eine Abfrage auszuführen, die Daten in einer Datenbank modifiziert, rufen Sie die `CDaoQueryDef::Execute`-Elementfunktion auf.

Weitere `CDaoQueryDef`-Elementfunktionen setzen oder ermitteln die Attribute der Abfragedefinitionen und bearbeiten Abfragefelder sowie Parameter.

37.3.6 Die CDaoTableDef-Klasse

Die `CDaoTableDef`-Klasse repräsentiert Tabellendefinitionen. Eine Tabellendefinition beschreibt die Struktur und die Attribute einer Datenbanktabelle.

Sie öffnen mit `CDaoTableDef::Open` eine bestehende Tabellendefinition. Eine neue Tabellendefinition wird mit `CDaoTableDef::Create` erstellt. Um der Datenbank eine Tabelle hinzuzufügen, die einer neuen Definition entspricht, rufen Sie die `AppendElement`-Funktion auf.

Die Felder einer Tabelle können mit `CreateField` und `DeleteField` erzeugt und gelöscht werden. Indizes werden für eine Tabelle mit `CreateIndex` und `DeleteIndex` erstellt und entfernt. Weitere Elementfunktionen setzen oder ermitteln verschiedene Tabellenattribute. `GetFieldCount` gibt beispielsweise die Anzahl der Felder in der Tabelle zurück. `SetValidationRule` weist einem Feld eine Gültigkeitsregel zu.

37.3.7 Weitere DAO-Klassen

Zusätzlich zu den fünf wesentlichen DAO-Klassen, verwenden DAO-Operationen zwei weitere Klassen: `CDaoFieldExchange` und `CDaoException`.

- ➡ `CDaoFieldExchange` wird in Aufrufen von `CDaoRecordset::DoFieldExchange` verwendet. Ein Objekt vom Typ `CDaoFieldExchange` definiert das Feld, das von der Datensatzfeldaustauschfunktion betroffen ist. Außerdem stellt das Objekt Parameter zur Verfügung, über die Attribute gesetzt werden können, die den Feldaustausch beschreiben.
- ➡ Alle DAO-Klassen nutzen Ausnahmeobjekte vom Typ `CDaoException`, um Fehler zu melden.

37.4 Zusammenfassung

Datenzugriffsobjekte repräsentieren eine Technologie, die auf OLE basiert und in Visual Basic, Visual Basic für Anwendungen und Microsoft Access verwendet wird, um über das Microsoft-Jet-Datenbankmodul auf Datenbanken zuzugreifen. Die MFC und der Visual-C++-Anwendungsassistent unterstützen das Entwickeln von DAO-Anwendungen in Visual C++. DAO-Bibliotheken werden in Form von distribuierbaren Komponenten angeboten, die Sie mit Ihrer Visual-C++-Anwendung vertreiben können.

Das Erstellen einer DAO-Anwendung ist mit der Hilfe des Anwendungsassistenten sehr einfach. Bestimmen Sie zunächst die Datenquelle, modifizieren Sie anschließend den Hauptdialog der Anwendung und fügen Sie dem Dialog Datensatz-Elementvariablen hinzu.

Die DAO-Objekthierarchie ist eine komplexe Struktur verschiedener Objekte. Die MFC vereinfacht diese Gliederung, indem sie fünf DAO-Klassen und zwei Helferklassen zur Verfügung stellt. Die Klassen `CDaoQueryDef` und `CDaoRecordset` repräsentieren Abfragen in einer Datenbank und das Ergebnis dieser Abfragen. Die Datenbanken selbst werden durch `CDaoDatabase` repräsentiert. Die Klasse `CDaoWorkspace` wird weniger häufig verwendet. Möchten Sie nicht auf Sicherheitsdatenbanken zugreifen, können Sie den Standardarbeitsbereich nutzen. Das Erstellen eines Objekts vom Typ `CDaoWorkspace` ist in diesem Fall nicht notwendig. `CDaoTableDef` repräsentiert eine Tabellenstruktur. Mit dieser Klasse fügen Sie Ihrer Datenbank weitere Tabellen hinzu und bearbeiten bereits bestehende Tabellen.

OLE DB und ADO

Kapitel 38

Ein unsterbliches Zitat, das I. I. Rabi, einem der Größen aus der goldenen Ära der Physik um 1930 zugeordnet wird, kommentiert die Entdeckung des Myons: »Wer hat das bestellt?« Er gab damit sicherlich der Frustration vieler seiner Kollegen Ausdruck, als sie von einem neuen Elementarteilchen hörten, das die bestehenden Theorien zerrüttete.

Viele Windows-Programmierer fühlen wohl ähnlich, wenn aus Redmond fast täglich andere Technologien kommen. Denn so wie das Myon seinen Platz in neuen physikalischen Theorien erhielt, die zu einem tieferen Verständnis der Kräfte der Natur führten, würde uns ein besseres Verständnis von Microsofts Entwicklungsstrategie helfen, die Rolle vieler ihrer neuen Entwicklungswerkzeuge und Komponenten zu verstehen.

OLE DB und ADO sind da keine Ausnahme. Der erfahrene Programmierer fragt sich, welchen Bedarf es für diese beiden neuen Spezifikationen gibt, wenn wir bereits zwei konkurrierende Datenzugriffstechnologien für den Einsatz durch den C++-Programmierer haben: ODBC und DAO.

Weder OLE DB noch ADO sind ein Ersatz für existierende DBMS-Technologien. Statt dessen wagen sie sich an eine Stelle, wo noch kein DBMS-Programmierer je gewesen ist:

➡ ADO steht für ActiveX-Datenobjekte, und stellt eine Technologie für den Datenzugriff mit Objekten geringen Umfangs dar (teilweise auf OLE DB beruhend). Mit geringem Umfang meine ich, daß die resultierenden Codeobjekte (ausführbare Programme oder Bibliotheken) eine Größe haben, die sie für Internet-Inhalte interessant machen.

- ➔ OLE DB ist ein Versuch, eine COM-basierte Schnittstellenspezifikation zu definieren, die traditionelle Datenquellen (DBMS) und andere Datenquellen, von einfachen Dateisystemen bis hin zu elektronischer Post, vereinheitlicht. Mittlerweile kann OLE DB auch zusammen mit der ATL verwendet werden und wird von verschiedenen Anwendungsassistenten unterstützt. OLE DB ist gegenüber ADO die leistungsfähigere und professionellere Technik.

Dieses Kapitel bietet eine kurze Übersicht über beide Ansätze. Mehr Lesestoff zu diesen sich schnell ändernden Technologien, einschließlich brandaktueller Informationen, gibt es auf Microsofts Web Site.

38.1 OLE DB

OLE DB ist ein Satz von Spezifikationen zum Komponentenobjektmodell (COM). Diese Schnittstellen werden von Anbietern gestellt und von Verbrauchern verwendet. Beachten Sie, daß diese Unterscheidung rein konzeptionell ist; eine einzelne Anwendung oder Bibliothek kann in beiden Rollen auftreten. In der Tat: Das ist es, was OLE-DB-Dienstanbieter tun; statt Daten zu »besitzen«, kapseln diese Komponenten Daten, die ein anderer Dienst zur Verfügung stellt.

38.1.1 SDK und Templates

Ich unterstreiche noch einmal, daß OLE DB eine Spezifikation ist, nicht eine Bibliothek oder ein Programm. Das Produkt, das Sie auf der Visual-C++-CD-ROM erhalten, ist das OLE DB Software Development Kit (SDK), das Header-Dateien, Bibliotheken und Beispiele für OLE-DB-Anbieter und -Konsumenten enthält.

Ab Visual C++ 6.0 gibt es nun auch eine anwenderfreundlichere Unterstützung von OLE DB, die auf einer Reihe von Klassentemplates basiert und Unterstützung durch den Anwendungsassistenten bereitstellt. Wir werden uns diese Unterstützung im zweiten Beispiel zu OLE DB ansehen.

38.1.2 Grundkonzepte

Die OLE-DB-Spezifikation definiert eine Reihe von Komponentenobjekten. Diese werden kurz in den folgenden Abschnitten besprochen.

38.1.3 Datenquellobjekte

Die Sitzung eines OLE-DB-Client beginnt mit der Erzeugung eines Datenquellobjekts mit Hilfe der Funktion `CoCreateInstance`. Diese aktiviert den entsprechenden OLE-DB-Anbieter und bereitet ihn für die Sitzung vor. Datenquellobjekte stellen die Schnittstelle `IDBProperties` bereit, durch die Verbindungs- und Authentifizierungsinformation mitgeteilt werden kann (wie der Name einer Datenquelle oder das Paßwort eines User). Sie stellen auch die Schnittstelle `IDBInitialize` bereit, durch die die eigentliche Verbindung zur Datenquelle hergestellt werden kann.

38.1.4 Sitzungen (Sessions)

Sitzungen werden mit der Schnittstelle `IDBCreateSession` erzeugt, die ebenfalls von Datenquellobjekten bereitgestellt wird.

Ein Session-Objekt erzeugt Zeilen-Set-, Befehls- und Transaktionsobjekte.

38.1.5 Befehle

Befehle werden mit der Schnittstelle `IDBCreateCommand` einer Session erzeugt. Der Text des Befehls ist anbieterspezifisch und wird mit der Schnittstelle `ICommandText`, die vom Befehlsobjekt bereitgestellt wird, festgelegt. Der Text ist oft ANSI-SQL oder eine Obermenge derselben; jedenfalls ist er in der OLE-DB-Spezifikation nicht vorgeschrieben.

38.1.6 Zeilen-Sets

Zeilen-Sets werden erzeugt, wenn Befehle mit `ICommand::Execute` ausgeführt werden. Beachten Sie, daß Befehlsobjekte nicht immer ein Objekt zurückgeben, wenn diese Methode aufgerufen wird. Zum Beispiel wird ein Zeilen-Set zurückgegeben, wenn der Befehl eine SQL `SELECT`-Anweisung ist, doch nicht, wenn der Befehl eine Aktualisier- oder Löschabfrage darstellt.

Zeilen-Sets kann man auch mit der Schnittstelle `IOpenRowset`, die von Session-Objekten bereitgestellt wird, erzeugen. Diese Methode ist vorzuziehen, wenn der Anbieter keine Befehle unterstützt.

38.1.7 Transaktionen

Transaktionen bieten ein Mittel zur Koordinierung der Prozesse auf einem Multiuser-System. Falls ein Anbieter Transaktionen unterstützt, stellt das Session-Objekt die Schnittstelle `ITransactionLocal` bereit. Durch ihre Methoden `StartTransaction` und `Commit` kann die Verarbeitung der Transaktionen implementiert werden.

38.1.8 Enumeratoren

Enumerator-Objekte können zum Aufzählen anderer Objekte eingesetzt werden.

Enumeratoren werden durch Aufrufen von `CoCreateInstance` mit der Klassen-ID des Enumerators erzeugt.

Ein Root-Enumerator, der mit dem OLE DB SDK geliefert wird, zählt Datenquellen und andere Enumeratoren auf. Der Einsatz dieses Enumerators ist die vorzuziehende Alternative zum Durchsuchen der Registrierung, um die Anzahl verfügbarer Datenquellen zu bestimmen.

Weitere Enumeratoren sind anbieterspezifisch; zum Beispiel können Enumeratoren zum Durchsuchen eines Dateisystems eingesetzt werden.

38.1.9 Fehler

OLE-DB-Fehlerobjekte erweitern die Fähigkeiten der Automation-Fehlerobjekte um die Rückgabe mehrfacher Fehler-Records und anbieterspezifischer Fehlermeldungen.

Der Zugriff auf ein Fehlerobjekt erfolgt über `GetErrorInfo` von der Automation-DLL und mit der Abfrage einer `IErrorInfo`-Schnittstelle von diesem Objekt. Anbieter können auch mit `CoCreateInstance` Fehlerobjekte erzeugen.

38.2 Ein OLE-DB-SDK-Arbeitsbeispiel

Nun fügen wir alles zusammen zu einer einfachen Win32-Konsolenanwendung.

Dieses Beispiel verwendet keine Befehle, Transaktionen, Enumeratoren oder Fehlerobjekte. Es ist eine OLE-DB-Konsumentenanwendung, die über einen Microsoft-OLE-DB-Anbieter für ODBC auf eine Microsoft-Access-Datenbank zugreift und den Inhalt einer Tabelle aus dieser Datenbank an die Konsole in einem Format mit Tabulatoren als Trennzeichen ausgibt. Um den Code einfach zu halten, wird vorausgesetzt, daß alle Felder in der Tabelle Textfelder sind.

Das vollständige Programm ist in Listing 38.1 gezeigt. Da es nur aus einer `main`-Funktion besteht, ist es ein ziemlich einfaches Beispiel für die grundlegenden OLE-DB-Aufrufe und Funktionsweise. Dieses Programm kann mit folgender Befehlszeile kompiliert werden:

```
CL OD.CPP OLEDB.LIB OLE32.LIB OLEAUT32.LIB
```

Beachten Sie, daß Sie das OLE DB SDK installiert haben müssen, um dieses Beispiel erfolgreich zu kompilieren.



*Listing 38.1:
Ein Beispiel zu
OLE DB*

```
#define DBINITCONSTANTS

#include <iostream.h>
#include <assert.h>
#include <stddef.h>
#include <ole2ver.h>
#include <oledb.h>
#include <msdasql.h>

// Das Programm verwendet den deutschen Access-Treiber. Sollten
// Sie nur den englischen Treiber zur Verfügung haben, so
// schreiben Sie:
// const WCHAR szDSNtemplate[] =
//     L"DRIVER={Microsoft Access Driver (*.mdb)};DBQ=";

const WCHAR szDSNtemplate[] = L"DRIVER={Microsoft Access-Treiber (*.mdb)};DBQ=";

void main(int argc, char *argv[])
{
    IMalloc* pIMalloc;
    IDBInitialize *pIDBInit;
    IDBProperties *pIDBProperties;
    IDBCreateSession *pIDBCreateSession;
    IOpenRowset *pIOpenRowset;
    IRowset *pIRowset;
    IColumnsInfo *pIColumnsInfo;
    IAccessor *pIAccessor;

    HACCESSOR hAccessor;
    HROW *rghRows = NULL;

    DBPROP dbProp;
    DBPROPSSET dbPropSet;
    DBID dbID;
    DBCOLUMNINFO *pColumnInfo;
    WCHAR *pColumnNames;
    ULONG cColumns;
    ULONG cBindings;
    ULONG cRowsObtained;
    DBBINDING *rgBindings;

    BYTE *pData;
    struct COLUMNDATA
    {
        DWORD dwLength;
        DWORD dwStatus;
        BYTE bData[1];
    } *pColumn;

    DWORD dwOffset;
    int i, j, l;
    char *p;
    WCHAR *pwszDSN;
    WCHAR *pwszTable;

    if (argc != 3)
```

```

    {
        cerr << "Aufruf: " << argv[0]
            << " Datenbankname Tabellenname" << endl;
        exit(1);
    }
    l = strlen(argv[1]) + 1;
    pwszDSN = new WCHAR[wcslen(szDSNtemplate) + 1];
    wcscpy(pwszDSN, szDSNtemplate);
    MultiByteToWideChar(CP_ACP, 0, argv[1], 1,
        pwszDSN + wcslen(pwszDSN), 1);
    l = strlen(argv[2]) + 1;
    pwszTable = new WCHAR[l];
    MultiByteToWideChar(CP_ACP, 0, argv[2], 1, pwszTable, 1);

    CoInitialize(NULL);
    CoCreateInstance(CLSID_MSDASQL, 0, CLSCTX_INPROC_SERVER,
        IID_IDBInitialize, (void**) &pIDBInit);
    CoGetMalloc(MEMCTX_TASK, &pIMalloc);

    VariantInit(&dbProp.vValue);
    dbProp.dwOptions = DBPROPOPTIONS_REQUIRED;
    dbProp.dwPropertyID = DBPROP_INIT_PROVIDERSTRING;
    dbProp.colid = DB_NULLID;
    dbProp.vValue.vt = VT_BSTR;
    dbProp.vValue.bstrVal = SysAllocString(pwszDSN);
    delete[] pwszDSN;

    dbPropSet.rgProperties = &dbProp;
    dbPropSet.cProperties = 1;
    dbPropSet.guidPropertySet = DBPROPSET_DBINIT;

    pIDBInit->QueryInterface(IID_IDBProperties,
        (void **) &pIDBProperties);
    pIDBProperties->SetProperties(1, &dbPropSet);
    if (pIDBInit->Initialize() == E_FAIL)
    {
        cerr << "Ungueltiger Datenbankname" << endl;
        goto NODB;
    }
    pIDBInit->QueryInterface(IID_IDBCreateSession,
        (void **) &pIDBCreateSession);
    pIDBCreateSession->CreateSession(NULL, IID_IOpenRowset,
        (LPUNKNOWN *) &pIOpenRowset);

    dbID.eKind = DBKIND_NAME;
    dbID.uName.pwszName = pwszTable;
    pIOpenRowset->OpenRowset(NULL, &dbID, NULL, IID_IRowset, 0,
        NULL, (LPUNKNOWN*) &pIRowset);
    if (pIRowset == NULL)
    {
        cerr << "Ungueltiger Tabellenname" << endl;
        goto NOTABLE;
    }

    pIRowset->QueryInterface(IID_IColumnsInfo,
        (void**) &pIColumnsInfo);
    pIColumnsInfo->GetColumnInfo(&cColumns, &pColumnInfo,
        &pColumnNames);
    pIColumnsInfo->Release();

    for (i = 0, cBindings = 0; i < cColumns; i++)
    {
        if (pColumnInfo[i].wType != DBTYPE_VECTOR) cBindings++;
    }
    rgBindings = new DBBINDING[cBindings];

```

```

dwOffset = 0;
for (i = 0, j = 0; i < cColumns; i++)
{
    if (pColumnInfo[i].wType == DBTYPE_VECTOR) continue;
    assert(j < cBindings);
    if (j > 0) cout << '\t';
    l = wcslen(pColumnInfo[i].pwszName) + 1;
    p = new char[l];
    WideCharToMultiByte(CP_ACP, 0, pColumnInfo[i].pwszName, l,
                        p, l, NULL, NULL);

    cout << p;
    delete[] p;
    rgBindings[j].dwPart = DBPART_VALUE | DBPART_LENGTH
                          | DBPART_STATUS;
    rgBindings[j].eParamIO = DBPARAMIO_NOTPARAM;
    rgBindings[j].iOrdinal = pColumnInfo[i].iOrdinal;
    rgBindings[j].wType... = DBTYPE_STR;
    rgBindings[j].pTypeInfo = NULL;
    rgBindings[j].obValue = dwOffset + offsetof(COLUMNDATA,
                                                bData);
    rgBindings[j].obLength = dwOffset + offsetof(COLUMNDATA,
                                                dwLength);
    rgBindings[j].obStatus = dwOffset + offsetof(COLUMNDATA,
                                                dwStatus);
    rgBindings[j].cbMaxLen = pColumnInfo[i].wType
        == DBTYPE_STR ? pColumnInfo[i].ulColumnSize +
            sizeof(char) : 100;
    rgBindings[j].pObject... = NULL;
    dwOffset += rgBindings[j].cbMaxLen + offsetof(COLUMNDATA,
                                                bData);

    dwOffset = (dwOffset + 7) & ~7;
    j++;
}
cout << endl;
pData = new BYTE[dwOffset];

pIRowset->QueryInterface(IID_IAccessor, (void**)&pIAccessor);
pIAccessor->CreateAccessor(DBACCESSOR_ROWDATA, cBindings,
                          rgBindings, 0, &hAccessor, NULL);

while (SUCCEEDED(pIRowset->GetNextRows(NULL, 0, 100,
    &cRowsObtained, &rghRows)) && cRowsObtained)
{
    for (i = 0; i < cRowsObtained; i++)
    {
        pIRowset->GetData(rghRows[i], hAccessor, pData);
        for (j = 0; j < cBindings; j++)
        {
            pColumn = (COLUMNDATA *)((BYTE *)pData +
                rgBindings[j].obLength);
            if (j > 0) cout << '\t';
            cout << (char *)pColumn->bData;
        }
        cout << endl;
    }
    pIRowset->ReleaseRows(cRowsObtained, rghRows, NULL, NULL,
        NULL);
}

pIAccessor->ReleaseAccessor(hAccessor, NULL);
pIAccessor->Release();
pIRowset->Release();
pIMalloc->Free(pColumnInfo);
pIMalloc->Free(pColumnNames);

```



```

        pIMalloc->Free(rghRows);
        delete[] rgBindings;
        delete[] pData;
    NOTABLE:
        pIOpenRowset->Release();
        pIDBCreateSession->Release();
    NODB:
        pIDBProperties->Release();
        pIDBInit->Release();
        pIMalloc->Release();
        CoUninitialize();
        delete[] pwszTable;
    }

```

Aufruf Das Programm beginnt mit der Untersuchung der Befehlszeile; es sollte mit zwei Parametern aufgerufen werden:

- Der erste enthält den Dateinamen der Microsoft-Access-Datenbank, auf die zugegriffen werden soll.
- Der zweite ist der Name der auszugebenden Tabelle.

Wenn die Tabelle beispielsweise *Autoren* und die Datenbank *BUECHER.MDB* heißt und sich in *C:\BUECHER* befindet, so wird das Programm (*OD.EXE* genannt) folgendermaßen aufgerufen:

```
OD C:\BUECHER\BUECHER.MDB AUTOREN
```

Der Datenbankstring wird dann zu einem SQL-Verbindungsstring kombiniert und nach Unicode umgewandelt, wie 32-Bit-OLE es erfordert. Der Einsatz eines SQL-Verbindungs-String hilft uns zu vermeiden, daß wir eine ODBC-Datenquelle über den ODBC-Manager installieren müssen.

Als nächstes werden die COM-Bibliotheken initialisiert und ein Datenquellobjekt erzeugt. Seine Schnittstelle *IDBInitialize* wird dabei offengelegt.

Datenquelle initialisieren Um die Datenquelle zu initialisieren, müssen wir ihr den OLE-Verbindungsstring als Eigenschaft zuordnen. Wir verwenden die anbieterspezifische Eigenschaft *DBPROP_INIT_PROVIDERSTRING*, obwohl *DBPROP_INIT_DATASOURCE* der Standard ist. Der Grund: Letztere erlaubt uns nicht, einen Verbindungs-String zu verwenden, sondern nur den Namen einer Datenquelle, die über den ODBC-Manager installiert ist.

Daten abfragen Nachdem eine Sitzung erfolgreich erzeugt ist, erzeugen wir ein Zeilen-Set mit der Schnittstelle *IOpenRowset*. Komplexere Anwendungen können statt dessen ein Befehlsobjekt verwenden, das zum Beispiel das Übermitteln eines SQL-Abfrage-String an den Anbieter ermöglicht.

Wenn wir das Zeilen-Set haben, werden Spalten gezählt und der Spaltenkopf ausgegeben. Zuletzt holt das Programm die Daten, iteriert durch die Einträge und gibt ihre Inhalte aus.

Wenn das Programm seine Aufgabe beendet hat, gibt es alle COM-Objekte sowie den Speicher, der lokal oder durch COM-Methoden zugeteilt wurde, wieder frei.

```
C:\MARKT&T\VC_KOMPENDIUM\PROGRAMME\KAP38\OD>OD BUECHER
AUTOREN
Name      Nationalitaet
Asimov, Isaac    USA
Rand, Ayn        USA
Harrison, Harry  USA
Bradbury, Ray    USA
Verne, Jules     Frankreich
Gabor, Aron      Ungarn
Calvino, Italo   Italien
Hofstadter, Douglas R. USA
```

Beispielsitzung

38.3 Ein OLE-DB-MFC-Anwendungsbeispiel

In diesem Abschnitt werden Sie den MFC-Anwendungsassistenten zum Aufbau einer OLE-DB-Konsumenten-Anwendung nutzen. Die Anwendung greift wiederum auf die Datenbank BUECHER zu und erlaubt dem Anwender die Datensätze der Tabelle BUECHER einzusehen.

38.3.1 Erstellen des Anwendungsgerüsts

1. Rufen Sie den Anwendungsassistenten auf, um die OLE-DB-Anwendung zu erstellen.

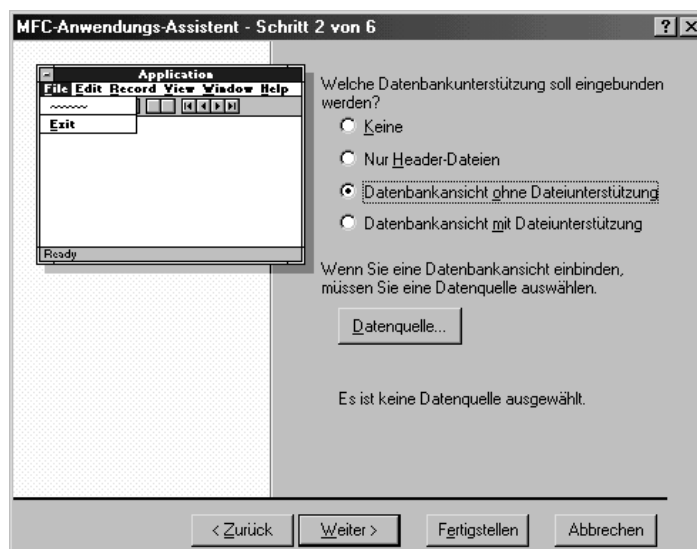
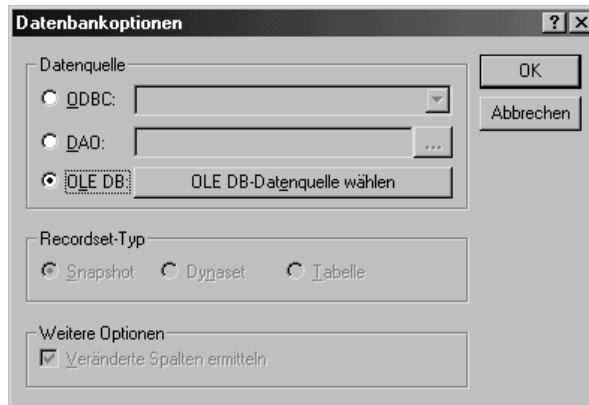


*Abbildung 38.1:
Hinzufügen der
Datenbankunterstützung zum
Anwendungsgerüst*

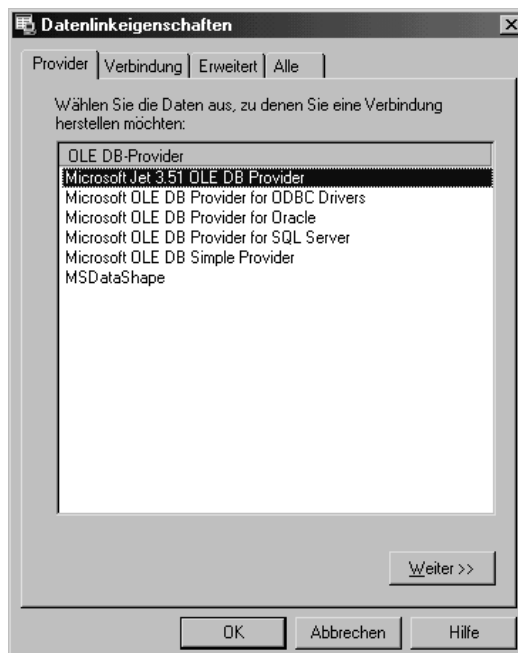
2. Selektieren Sie bitte im ersten Schritt die Option EINZELNES DOKUMENT (SDI) und im zweiten Schritt DATENBANKANSICHT OHNE DATEIUNTERSTÜTZUNG (Abbildung 38.1).

Abbildung 38.2:
Hinzufügen
einer OLE-DB-
Datenquelle



3. Betätigen Sie die Schaltfläche DATENQUELLE, um eine Datenquelle auszuwählen. Selektieren Sie im anschließend angezeigten Dialog DATENBANKOPTIONEN die Option OLE DB. und klicken Sie dann auf den Schalter OLE-DB-DATENQUELLE WÄHLEN (Abbildung 38.2).

Abbildung 38.3:
Auswahl des
OLE-DB-Provi-
ders



4. Wählen Sie als OLE-DB-Dienstanbieter die Microsoft Jet Engine aus (Abbildung 38.3). Klicken Sie dann auf WEITER, um die Datenbank auszuwählen.

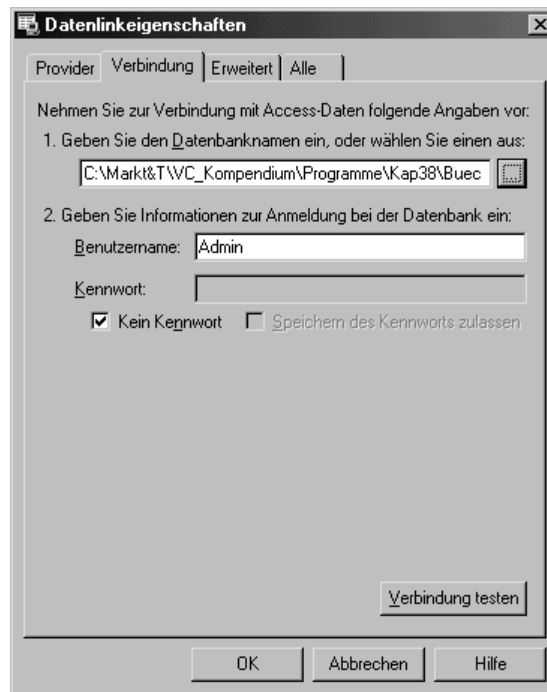


Abbildung 38.4:
Auswahl der
Datenbank

5. Wählen Sie in diesem Dialog den Pfad zur Access-Datenbank BUECHER.MDB aus. Klicken Sie danach auf den Schalter VERBINDUNG TESTEN und warten Sie, bis Ihnen eine Erfolgsmeldung angezeigt wird.

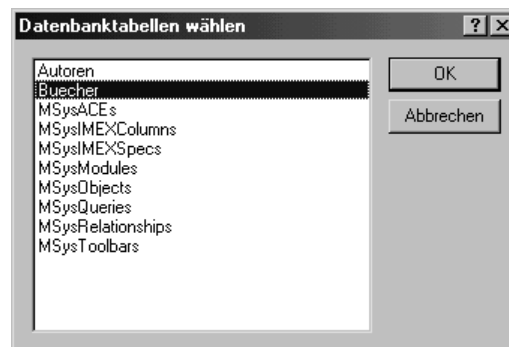


Abbildung 38.5:
Auswahl der
Tabelle

6. Verlassen Sie danach den Dialog mit einem Klick auf OK. Ein weiterer Dialog wird angezeigt, in dem Sie die Tabellen auswählen können. Selektieren Sie dort die Tabelle `BUECHER`, und betätigen Sie anschließend die Schaltfläche OK (Abbildung 38.5).

Sie sehen nun wieder den Dialog des Anwendungsassistenten vor sich. Unter der Schaltfläche `DATENQUELLE` wird die ausgewählte Datenquelle angezeigt.

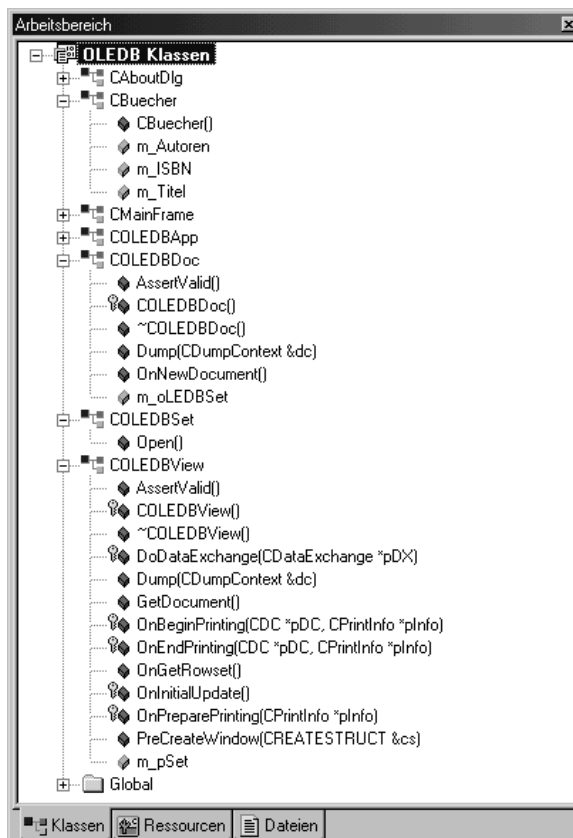
Alle weiteren Optionen des Assistenten bleiben unverändert.

7. Betätigen Sie daher die Schaltfläche `FERTIGSTELLEN`, um das Anwendungsgerüst erstellen zu lassen.

38.3.2 Das OLE-DB-Anwendungsgerüst

Die Klassen des vom Anwendungsassistenten generierten Anwendungsgerüsts sind in Abbildung 38.6 dargestellt.

Abbildung 38.6:
Die Klassen des
Anwendungs-
gerüsts



Im Vergleich zu einer Anwendung, die keine Datenbankfunktionalität unterstützt, verfügt OLE DB über eine zusätzliche Klasse `COLEDBSet` sowie einige neue Elementvariablen und Funktionen in der Dokument- und der Ansichtsklasse.

Im Vergleich zu einer DAO-Anwendung ist die Ansichtsklasse von der neuen MFC-Klasse `COleDBRecordView` abgeleitet. Zudem wurde eine Klasse `CBuecher` zur Repräsentation der BUECHER-Tabelle angelegt.

Diese Klasse wird beispielsweise für die Ableitung der Klasse `COLEDBSet` benötigt. Diese Klasse wird nämlich nicht von einer einfachen Basis-klasse abgeleitet, sondern von einer Instanzierung des `CCommand`-Templates für die Klasse `CBuecher`:

```
class COLEDBSet : public CCommand<CAccessor<CBuecher> >
```

Die Klasse für die Datenbanktabelle

Listing 38.2 zeigt die Deklaration der Klasse `CBuecher`.

```
class CBuecher
{
public:
    CBuecher()
    {
        memset( (void*)this, 0, sizeof(*this) );
    };

    char m_Autoren[51];
    char m_ISBN[26];
    char m_Titel[101];

    BEGIN_COLUMN_MAP(CBuecher)
        COLUMN_ENTRY_TYPE(3, DBTYPE_STR, m_Autoren)
        COLUMN_ENTRY_TYPE(1, DBTYPE_STR, m_ISBN)
        COLUMN_ENTRY_TYPE(2, DBTYPE_STR, m_Titel)
    END_COLUMN_MAP()
};
```

*Listing 38.2:
Die Klasse
CBuecher*

Hier interessieren uns vor allem die Elementvariablen `m_Autoren`, `m_ISBN` und `m_Titel`, die für uns die Verbindung zu den Feldern der Buecher-Tabelle herstellen.

Die Klasse `COLEDBSet`

Die neue Klasse `COLEDBSet` definiert als einziges eigenes Element die Elementfunktion `Open()`, in der die Verbindung zur Datenbank hergestellt wird. Hier finden Sie auch die Spezifikation der Datenbank (Name und Pfad) sowie den SQL-String zur Abfrage der Daten aus der Datenbank, die Sie bei Bedarf bearbeiten können.

Listing 38.3:
COLEDBSet-Klas-
sendeclaration

```
class COLEDBSet : public CCommand<CAccessor<CBuecher> >
{
public:
    HRESULT Open()
    {
        CDataSource db;
        CSession session;
        HRESULT hr;

        CDBPropSet dbinit(DBPROPSET_DBINIT);
        dbinit.AddProperty(DBPROP_AUTH_CACHE_AUTHINFO, true);
        dbinit.AddProperty(DBPROP_AUTH_ENCRYPT_PASSWORD, false);
        dbinit.AddProperty(DBPROP_AUTH_MASK_PASSWORD, false);
        dbinit.AddProperty(DBPROP_AUTH_PASSWORD, "");
        dbinit.AddProperty(DBPROP_AUTH_PERSIST_ENCRYPTED, false);
        dbinit.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO,
                           false);
        dbinit.AddProperty(DBPROP_AUTH_USERID, "Admin");
        // Der nachfolgende String haengt von Ihrer
        // Verzeichnisstruktur ab.
        dbinit.AddProperty(DBPROP_INIT_DATASOURCE,
                           "C:\\Programme\\Kap38\\Buecher.mdb");
        dbinit.AddProperty(DBPROP_INIT_MODE, (long)16);
        dbinit.AddProperty(DBPROP_INIT_PROMPT, (short)4);
        dbinit.AddProperty(DBPROP_INIT_PROVIDERSTRING,
                           ";COUNTRY=0;CP=1252;LANGID=0x0409");
        dbinit.AddProperty(DBPROP_INIT_LCID, (long)1033);

        hr = db.OpenWithServiceComponents(
                           "Microsoft.Jet.OLEDB.3.51",&dbinit);

        if (FAILED(hr))
            return hr;

        hr = session.Open(db);
        if (FAILED(hr))
            return hr;

        CDBPropSet propset(DBPROPSET_ROWSET);
        propset.AddProperty(DBPROP_CANFETCHBACKWARDS, true);
        propset.AddProperty(DBPROP_IRowsetScroll, true);
        propset.AddProperty(DBPROP_IRowsetChange, true);
        propset.AddProperty(DBPROP_UPDATABILITY,
                           DBPROPVAL_UP_CHANGE |
                           DBPROPVAL_UP_INSERT |
                           DBPROPVAL_UP_DELETE );

        hr = CCommand<CAccessor<CBuecher> >::Open(session,
                           "SELECT * FROM Buecher", &propset);

        if (FAILED(hr))
            return hr;

        return MoveNext();
    }
};
```

38.3.3 Bearbeiten der Anwendung

Sie werden zunächst den Hauptdialog der OLE-DB-Anwendung bearbeiten.

1. Öffnen Sie den mit `IDD_OLEDB_FORM` bezeichneten Dialog zur Bearbeitung.
2. Entfernen Sie das statische TODO-Steuerelement, und fügen Sie dem Dialog die in Abbildung 38.7 dargestellten Bezeichnungs- und Textfelder hinzu.

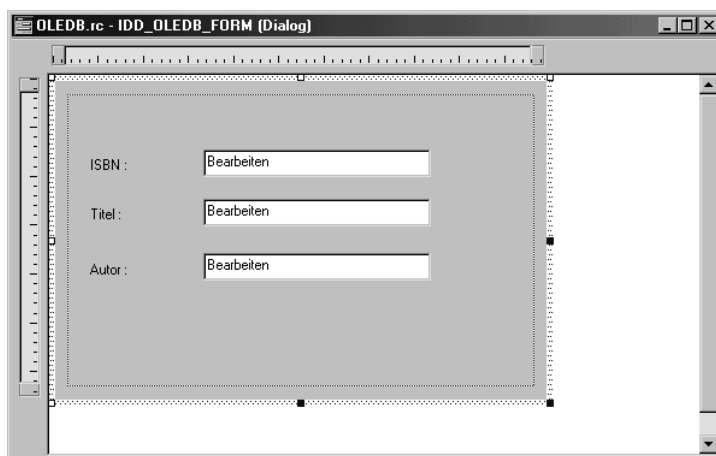


Abbildung 38.7:
Bearbeiten des
OLE-DB-Dialogs

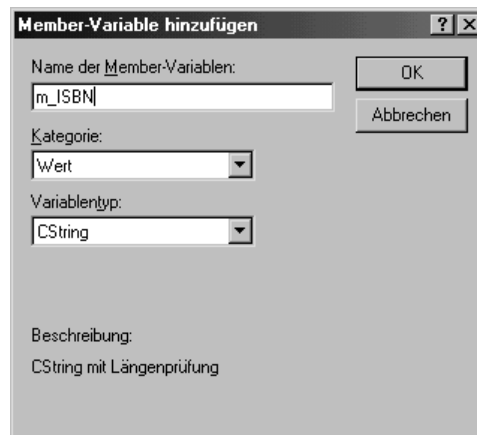
3. Nennen Sie die drei Eingabefelder `IDC_ISBN`, `IDC_TITEL`, und `IDC_AUTOR`.

Bevor Sie den Dialog schließen, sollten Sie den Dialogfeldern die entsprechenden Elementvariablen mit Hilfe des Klassen-Assistenten zuweisen.

4. Halten Sie die Taste `[Strg]` gedrückt, und führen Sie anschließend einen Doppelklick über dem Eingabefeld `IDC_ISBN` aus.
5. Daraufhin wird der Klassen-Assistenten-Dialog `MEMBER-VARIABLE HINZUFÜGEN` aufgerufen. Geben Sie einen Namen für die Elementvariable ein (Abbildung 38.8).
6. Verfahren Sie analog für die restlichen Eingabefelder im Dialog.

Um den Datentransfer komplett zu machen, müssen wir noch die `COLEDBView`-Elementfunktion `DoDataExchange` bearbeiten. Zwischen dem Aufruf der Basisklassenversion und den vom Klassen-Assistenten generierten DDX-Funktionen fügen wir den Code zur Komplettierung der Datentransferkette ein (siehe Listing 38.4).

Abbildung 38.8:
Hinzufügen
einer Datensatz-
Elementvariablen



Listing 38.4:
COLEDBView::
DoDataExchange

```
void COLEDBView::DoDataExchange(CDataExchange* pDX)
{
    COleDBRecordView::DoDataExchange(pDX);

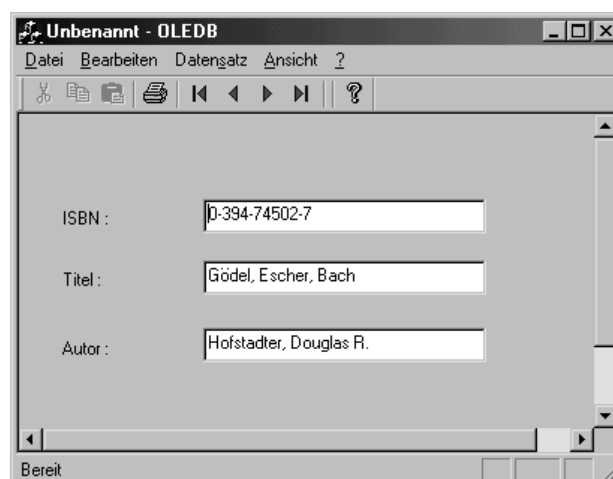
    m_ISBN = m_pSet->m_ISBN;
    m_Titel = m_pSet->m_Titel;
    m_Autor = m_pSet->m_Autoren;

    ///{AFX_DATA_MAP(COLEDBView)
    DDX_Text(pDX, IDC_ISBN, m_ISBN);
    DDX_Text(pDX, IDC_TITEL, m_Titel);
    DDX_Text(pDX, IDC_AUTOR, m_Autor);
    ///}AFX_DATA_MAP
}
```

Das ADAO-Projekt ist nun vollständig.

7. Kompilieren Sie die Anwendung, und starten Sie das Programm (Abbildung 38.9).

Abbildung 38.9:
Die OLE-DB-
Anwendung



38.4 ActiveX-Datenobjekte

ActiveX-Datenobjekte kennzeichnen eine Technologie, die speziell für Client-Anwendungen, die auf Daten in einer Datenbank zugreifen und diese manipulieren, entwickelt wurde. ADO ist schnell und ressourcensparend und zur Entwicklung von Webinhalten bestens geeignet.

ADO ist eine Spezifikation und kein Produkt. Dennoch gibt es im OLE DB SDK eine spezifische ADO-Implementation. ADODB ist eine ADO-Implementation, die OLE-DB-Anbieter verwenden, einschließlich der Microsoft-OLE-DB-Anbieter für ODBC.

38.5 Übersicht der ADO-Objekte

ADO spezifiziert eine Reihe von sieben Objekttypen und vier Collection-Typen. Diese werden in den folgenden Abschnitten kurz beschrieben.)

38.5.1 Das Verbindungsobjekt

Ein Verbindungsobjekt stellt eine Verbindung zu einer Datenquelle dar. Verbindungsobjekte werden unabhängig von vorher erzeugten Objekten erzeugt. In ADO-Programmen ist gewöhnlich der erste Schritt das Erzeugen eines Verbindungsobjekts.

Verbindungsobjekte bieten Methoden für die Verwaltung von Verbindungen und Verarbeitung der Transaktionen.

38.5.2 Die Eigenschafts-Collection und Eigenschaftsobjekte

Eigenschaftsobjekte stellen dynamische Merkmale dar, die nicht durch Methoden und Eigenschaften der ADO-Objekte implementiert sind. Die Bedeutung der Eigenschaftsobjekte wird vom Anbieter definiert.

Eigenschaftsobjekte sind in Eigenschaftskollektionen organisiert. Verbindungsobjekte, Befehlsobjekte, Recordset-Objekte und Feldobjekte haben eine Eigenschaftskollektion.

38.5.3 Die Fehler-Collection und Fehlerobjekte

Wenn während einer ADO-Operation ein Fehler auftritt, werden ein oder mehrere Fehlerobjekte in die Fehlerkollektion des Verbindungsobjekts plaziert.

Fehlerobjekte haben einige Eigenschaften wie Beschreibung, Anzahl, Quelle oder SQL-Zustand, die eine genaue Beschreibung des aufgetretenen Fehlers liefern.

38.5.4 Das Befehlsobjekt

Befehlsobjekte werden zum Erzeugen und Ausführen eines Befehls (etwa einer SQL-Anweisung) auf einem Anbieter verwendet. Befehlsobjekte können von sich selbst erzeugt werden, wobei dann Verbindungsobjekte nach Bedarf miterzeugt werden. Falls Sie mehrere Befehlsobjekte verwenden, ist es besser, wenn zuerst ein Verbindungsobjekt erzeugt wird und die Eigenschaft `ActiveConnection` jedes neuen Befehls als Referenz darauf gesetzt wird.

Befehlsobjekte können eingesetzt werden, um Records von der Datenquelle in Form von `Recordset`-Objekten zu erhalten.

38.5.5 Die Parameter-Collection und Parameterobjekte

Befehlsobjekte können gespeicherte Prozeduren oder parametrisierte Abfragen referenzieren, wobei es nötig sein kann, Parameter an das Befehlsobjekt vor der Ausführung zu übergeben. Dies wird durch den Einsatz der Parameterkollektion des Befehlsobjekts erreicht.

Parameterobjekte unterstützen die Eigenschaften `Name` und `Value`, über die individuelle Parameter definiert werden können.

38.5.6 Das Recordset-Objekt

Ein `Recordset`-Objekt stellt eine Kollektion von Records dar. Eine solche Kollektion kann beispielsweise das Ergebnis-Set einer `SELECT`-Abfrage sein.

`Recordset`-Objekte können unabhängig erzeugt werden, wobei implizit ein Verbindungsobjekt erzeugt wird. Alternativ dazu können `Recordset`-Objekte mit einem vorher definierten Verbindungsobjekt geöffnet werden.

`Recordset`-Objekte unterstützen Methoden zum Durchqueren von Records und Abrufen von Daten. Die Methode `GetRows` vor allem kann zum effizienten Abrufen von Informationen eingesetzt werden, da sie den Inhalt mehrerer Zeilen auf einmal zurückgibt.

38.5.7 Die Feld-Collection und Feldobjekte

Eine Feld-Collection ist jedem `Recordset`-Objekt zugeordnet. Über die Feld-Collection können Spalten einer Tabelle oder Ergebnisssets aufgezählt werden und Informationen, wie der Name der Spalte, gewonnen werden.

Spalten werden durch Feldobjekte in einer Feld-Collection dargestellt. Feldobjekte haben Eigenschaften wie `Name`, `Typ` oder `Genauigkeit` und beschreiben die Spalte, der sie zugeordnet sind.

38.6 Ein Arbeitsbeispiel

ADO ist bestens zum Einsatz in Skriptsprachen wie Visual Basic geeignet. Dennoch ist es relativ einfach, einfache ADO-Anwendungen mit C++ zu schreiben: Das Beispielprogramm in diesem Abschnitt greift in weniger als 90 Zeilen Code auf eine OLE-DB-Datenquelle zu, führt eine SQL-SELECT-Abfrage aus und gibt die Ergebnisse in Tabulatorformat aus.

Es referenziert eine Microsoft-Access-Datenbank namens `BOOKS.MDB`. Diese Datenbank enthält zwei Tabellen:

- ➡ die Tabelle `AUTHORS` listet Namen und Nationalität der Autoren auf;
- ➡ die Tabelle `BOOKS` listet die ISBN-Nummern, Titel und Autoren der Bücher auf.

Die Abfrage, die im Beispielprogramm aus Listing 38.5 durchgeführt wird, listet alle Bücher amerikanischer Autoren auf.

Dieses Programm ist eine Win32-Konsolenanwendung, die leicht von der Befehlszeile mit dem Befehl

```
CL AD.CPP OLE32.LIB OLEAUT32.LIB ADOID.LIB
```

kompiliert werden kann.

Um das Programm erfolgreich zu kompilieren und auszuführen, müssen Sie auf Ihrem System das OLE DB SDK installiert haben.

```
#include <iostream.h>
#include <objbase.h>
#include <adoid.h>
#include <adoint.h>
```

```
// Das Programm verwendet den deutschen Access-Treiber. Sollten
// Sie nur den englischen Treiber zur Verfügung haben, so
```



*Listing 38.5:
Ein Beispiel zu
ADO*

```

// schreiben Sie:
// const WCHAR wszConnection[] =
//      L"DRIVER={Microsoft Access Driver (*.mdb)};...";

const WCHAR wszConnection[] = L"DRIVER={Microsoft Access-Treiber
(*.mdb)};DBQ=BUECHER.MDB";

const WCHAR wszSQL[] =
    L"SELECT BUECHER.*, Autoren.Nationalitaet "
    L"FROM { oj Buecher LEFT OUTER JOIN Autoren ON "
    L"Buecher.Autoren = Autoren.Name } "
    L"WHERE (Autoren.Nationalitaet = 'USA')";

void main(void)
{
    ADORecordset *pRecordset;
    ADOFields *pFields;
    ADOField *pField;
    VARIANT varSQL;
    VARIANT varConnection;
    VARIANT varStart;
    VARIANT varField;
    VARIANT varRows;
    VARIANT varItem;
    BSTR strColumn;
    LONG lColumns, lRows;
    LONG lIndex[2];
    LONG l;
    char *p;

    VariantInit(&varConnection);
    VariantInit(&varSQL);
    VariantInit(&varStart);
    VariantInit(&varField);
    VariantInit(&varItem);
    varConnection.vt = VT_BSTR;
    varConnection.bstrVal = SysAllocString(wszConnection);
    varSQL.vt = VT_BSTR;
    varSQL.bstrVal = SysAllocString(wszSQL);
    varStart.vt = varField.vt = VT_ERROR;
    varStart.scode = varField.scode = DISP_E_PARAMNOTFOUND;

    CoInitialize(NULL);
    CoCreateInstance(CLSID_CADORecordset, NULL, CLSCTX_INPROC_SERVER, IID_IADORecordset,
    (void **)&pRecordset);
    pRecordset->Open(varSQL, varConnection, adOpenForwardOnly, adLockReadOnly,
    adCmdText);

    pRecordset->GetRows(adGetRowsRest, varStart, varField, &varRows);
    SafeArrayGetUBound(varRows.parray, 1, &lColumns);
    SafeArrayGetUBound(varRows.parray, 2, &lRows);

    pRecordset->get_Fields(&pFields);
    for (lIndex[0] = 0; lIndex[0] <= lColumns; lIndex[0]++)
    {
        varItem.vt = VT_INT;
        varItem.intVal = lIndex[0];
        pFields->get_Item(varItem, &pField);
        pField->get_Name(&strColumn);
        if (lIndex[0] > 0) cout << '\t';
        l = wcslen(strColumn) + 1;
        p = new char[l];
        WideCharToMultiByte(CP_ACP, 0, strColumn, l, p, l, NULL, NULL);
        cout << p;
    }
}

```

```
delete[] p;
SysFreeString(strColumn);
pField->Release();
}
cout << endl;
pFields->Release();

for (lIndex[1] = 0; lIndex[1] <= lRows; lIndex[1]++)
{
    for (lIndex[0] = 0; lIndex[0] <= lColumns; lIndex[0]++)
    {
        SafeArrayGetElement(varRows.parray, &lIndex[0],
                            &varField);
        if (lIndex[0] > 0) cout << '\t';
        l = wcslen(varField.bstrVal) + 1;
        p = new char[l];
        WideCharToMultiByte(CP_ACP, 0, varField.bstrVal, l, p, l,
                            NULL, NULL);

        cout << p;
        delete[] p;
    }
    cout << endl;
}

pRecordset->MoveFirst();
pRecordset->Release();
SysFreeString(varConnection.bstrVal);
SysFreeString(varSQL.bstrVal);
}
```

Der Ablauf dieses Programms ist sehr leicht nachvollziehbar und braucht kaum Erklärung. Nach einigen Variableinitialisierungen beginnt die eigentliche Arbeit mit dem Erzeugen eines ADO-Recordset-Objekts. Dieses Objekt wird zum Herstellen einer Verbindung mit einem ODBC-Verbindungsstring verwendet. Der Einsatz eines Verbindungsstring statt des Namens der Quelldatei hilft uns, die Installation der Datenquelle über den ODBC-Manager zu vermeiden.

Die Abfrage wird ausgeführt und die Ergebnisse durch den Aufruf der Methode `GetRows` des Recordset-Objekts abgefragt. Der Rest der Anwendung ist kaum mehr als Schöndrucken. Zuerst wird die Feldinformation für jede Spalte abgefragt, so daß eine Kopfzeile ausgegeben werden kann. Als nächstes wird eine Iteration über alle Zeilen ausgeführt und die Daten an die Standardausgabe geschickt. Beachten Sie, daß vorausgesetzt wird, daß alle Felder dieser Abfrage einen Textwert zurückgeben. Falls Sie dieses Programm als Ausgangspunkt für Ihre eigenen Anwendungen verwenden, müßten Sie Code hinzufügen, der den Feldtyp untersucht, statt blindlings einen – ggf. falschen – Typ anzunehmen.

38.7 Zusammenfassung

OLE DB ist eine Schnittstellenspezifikation, die mit Hilfe des Komponentenobjektmodells (COM) den Zugang zu verschiedenen Datenquellen ermöglicht. Diese Datenquellen können DBMS sein, aber auch andere wie Dateisysteme oder elektronische Post umfassen.

OLE-DB-Anwendungen können Anbieter oder Konsumenten sein. Anbieter stellen Schnittstellen bereit; Konsumenten verwenden Schnittstellen zum Ausführen von Aufgaben an Datenbanken.

Die OLE-DB-Spezifikation definiert eine Reihe von Objekten: Datenquellenobjekte, Session-Objekte, Befehlsobjekte, Rowset-Objekte, Enumeratorobjekte und Fehlerobjekte. Konsumenten erzeugen ein Datenquellenobjekt über den Aufruf von `CoCreateInstance` und verwenden Schnittstellen dieses Objekts zum Erzeugen einer Session. Sessions ihrerseits werden zum Erzeugen von Befehlen und Rowsets für den Datenzugriff verwendet.

Das OLE DB SDK ist ein separates Produkt auf Ihrer Visual-C++-CD-ROM. Es enthält auch den Microsoft-OLE-DB-Anbieter für ODBC, eine Anbieterimplementation, die Zugriff auf ODBC-Datenquellen bietet.

ActiveX-Datenobjekte bieten eine schnelle, ressourcensparende Technologie für den Datenzugriff. Als solche sind sie geeignet für Anwendungen für das Web, können aber auch von gewöhnlichen C++-Anwendungen eingesetzt werden. ADO definiert eine Reihe von sieben Objekten und vier Collections. Verbindungen zu Datenquellen werden von Verbindungsobjekten dargestellt. Die Ergebnisse von Abfragen der Datenquelle werden von Recordset-Objekten dargestellt. Weitere Objekttypen umfassen Feldobjekte (in Feld-Collections), Befehlsobjekte und entsprechende Parameterobjekte (in Parameter-Collections), Fehler-Collections, die Fehlerobjekte enthalten, und Eigenschafts-Collections, die Eigenschaftsobjekte enthalten.

Datenbank- und Abfragedesign, SQL- Debugging

Kapitel 39

Visual C++ Enterprise Edition integriert eine Reihe datenbankenbezogener Werkzeuge in Visual Studio. Es ist nun möglich, Datenbanken zu erzeugen, Tabellen zu füllen, Abfragen zu erzeugen und zu testen und gespeicherte Prozeduren auf einem lokalen oder entfernten Server zu debuggen – all das, ohne die Umgebung von Visual Studio zu verlassen.

Viele der Datendesignfähigkeiten von Visual Studio sind für die meisten Datenquellen verfügbar. Andere, wie die Fähigkeit, Tabellen zu erstellen, sind spezifisch für den Microsoft SQL Server.

Zwei Features sind für den C++-Programmierer von besonderem Interesse.

- ➡ Das eine ist der Abfrage-Designer, der das Erstellen und Testen von SQL-Abfragen innerhalb von Visual Studio ermöglicht;
- ➡ Das andere ist der SQL-Debugger, der das Debuggen von gespeicherten SQL-Prozeduren, die auf einem Microsoft SQL Server laufen, ermöglicht.

39.1 Visual-Datenbankwerkzeuge

Um die Datenbankfähigkeiten von Visual Studio vorzuführen, habe ich eine einfache Datenbank in Microsoft Access erzeugt. Diese Datenbank von Büchern und Autoren enthält zwei Tabellen: Die Tabelle BUECHER listet Bücher mit ihren Titeln, Autoren und ISBN-Nummern; die Tabelle AUTOREN listet Autoren und ihre Nationalität. Die beiden Tabellen sind durch eine Viele-zu-Eins-Beziehung verknüpft, wie die Abbildung des Microsoft-Access-Bildschirms in Abbildung 39.1 zeigt.

Abbildung 39.1:
Beziehungen in
der Datenbank
Buecher

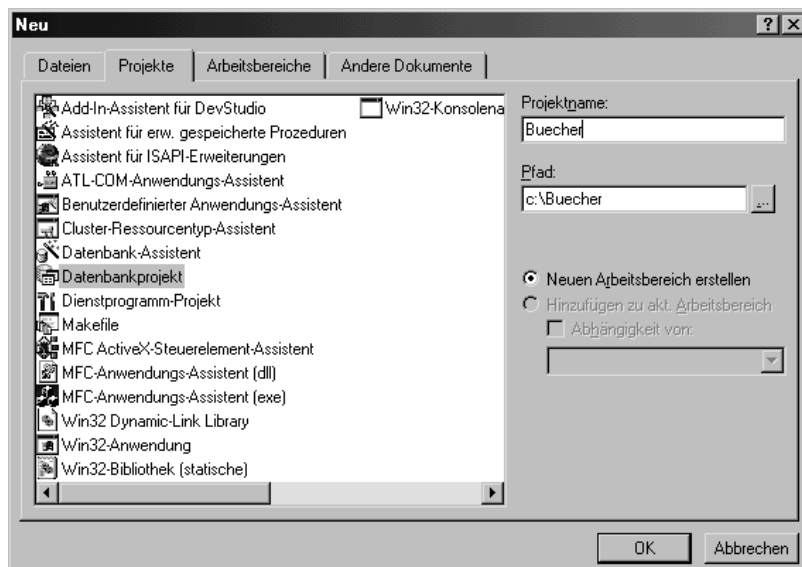


39.1.1 Erzeugen eines Datenbankprojekts

Um aus Visual Studio auf die Datenbank BUECHER zugreifen zu können, müssen Sie ein Datenbankprojekt erzeugen.

1. Legen Sie ein Verzeichnis *Buecher* an, und kopieren Sie die Datenbank *Buecher.mdb* in dieses Verzeichnis.

Abbildung 39.2:
Erzeugen eines
Datenbankprojekts



2. Wählen Sie NEU im DATEI-Menü von Visual Studio und auf der Registerseite PROJEKTE den Eintrag DATENBANKPROJEKT (Abbildung 39.2). Achten Sie darauf, daß das Projekt in dem Verzeichnis erstellt wird, in dem die Datenbank steht.
3. Wenn Sie OK klicken, fragt Visual Studio nach der Datenquelle.



Abbildung 39.3:
Wählen einer
Datenquelle

4. Sie können für unsere Datenbank eine ODBC-Datenquelldatei erzeugen, indem Sie den Schalter NEU im Dialog zum Wählen der Datenquelle betätigen (Abbildung 39.3).

Wenn Sie umgekehrt bereits eine ODBC-Treiberverbindung zu Ihrer Datenbank eingerichtet haben, können Sie diese auf der Registerseite Computerdatenquelle auswählen.

Was als nächstes erscheint, ist der erste Dialog des Assistenten für das Erzeugen neuer Datenquellen (nun, er ist zwar nicht als Assistent gekennzeichnet, aber er funktioniert wie einer, so wird mir vielleicht verziehen, wenn ich ihn so nenne).

5. Hier können Sie den Treiber für die neue Datenquelle angeben – in unserem Fall, der Microsoft-Access-Treiber (Abbildung 39.4). Der Schalter ERWEITERT erlaubt die Angabe zusätzlicher treiberspezifischer Informationen, doch im Moment brauchen Sie sich nicht darum zu kümmern. (Dennoch, wenn Sie zum Beispiel auf eine paßwortgeschützte Datenbank zugreifen wollen, könnten Sie diese Möglichkeit brauchen.)
6. Klicken Sie auf WEITER, um zum zweiten Dialog des Assistenten zu gelangen.

Abbildung 39.4:
Auswählen
eines Treibers

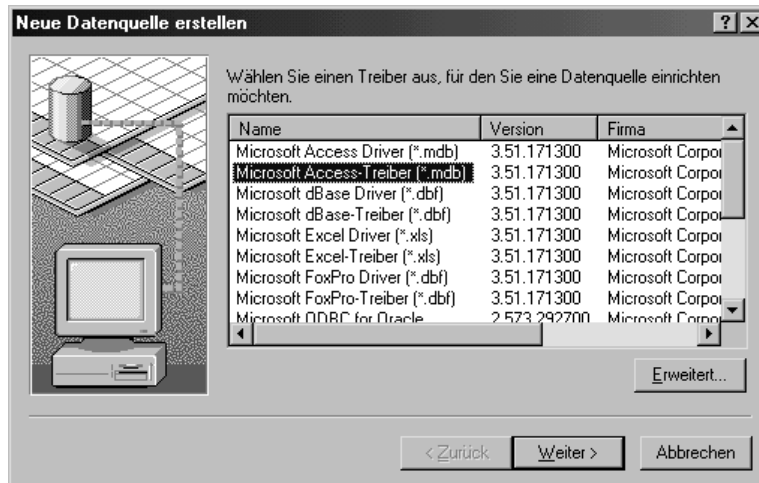
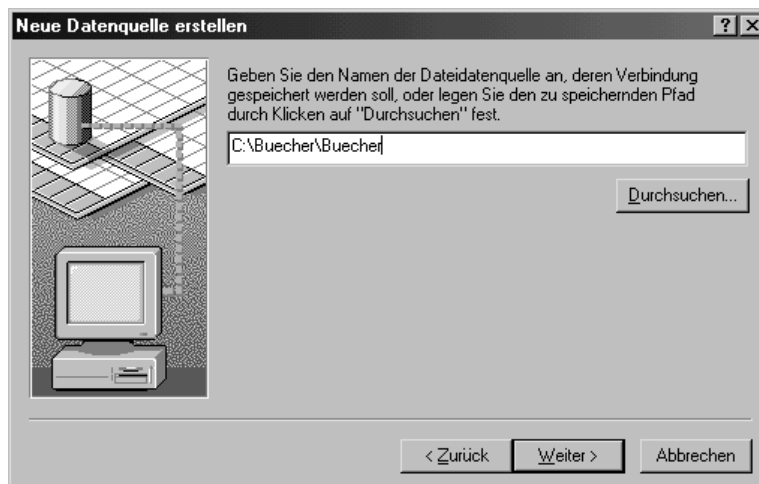


Abbildung 39.5:
Angabe eines
Dateinamens für
die Datenquelle



- Hier können Sie einen Dateinamen für die Datenquelle angeben. Dies wird eine Datei mit der Erweiterung .DSN sein, die zum Speichern von Datenquelleninformationen verwendet wird (daher der Name »Dateidatenquelle«, um sie von den Datenquellen zu unterscheiden, die über Einträge in der Registratur angegeben werden). Sie können einen Dateinamen eingeben (Abbildung 39.5) oder einen aus dem Standarddialog SPEICHERN UNTER wählen, indem Sie den Schalter DURCHSUCHEN klicken.

Der letzte Dialog des Assistenten (NEUE DATENQUELLE ERZEUGEN) gibt eine Übersicht Ihrer gewählten Einstellungen.

8. Wenn Sie jetzt den Schalter FERTIGERSTELLEN klicken, wird ein treiberspezifischer Dialog aufgerufen. Im Falle des von Ihnen gewählten Microsoft-Access-Treibers läßt Sie dieser Dialog (Abbildung 39.6) die Access-Datenbankdatei angeben, die Sie mit Ihrer Datenquelle verbinden wollen.

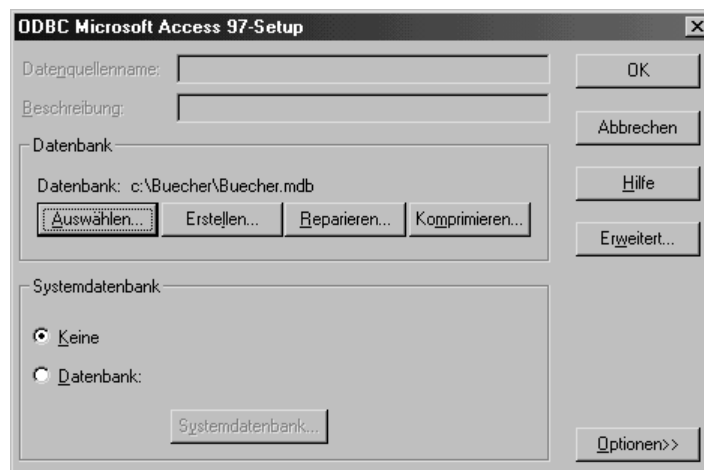


Abbildung 39.6:
Auswählen einer
Access-Daten-
bank

9. Klicken Sie auf den Schalter AUSWÄHLEN, um die Datenbankdatei zu wählen.
10. Wenn Sie diesen Dialog schließen, kehrt die Kontrolle zu dem Datenquellen-Dialog zurück, wo Sie die neu erzeugte Dateidatenquelle (.DSN-Datei) wählen und durch Klicken von OK weitermachen.

39.2 Arbeiten mit einer Datenbank

Werfen Sie mal einen Blick auf das Arbeitsbereichfenster für das neu erstellte Datenbankprojekt (Abbildung 39.7).

In der Datenansicht (Abbildung 39.8) sehen Sie auch die Tabellen der Datenbank. Hätte Ihre Access-Datenbank Abfragen definiert, würden diese hier auch erscheinen.

Was können Sie mit einer Datenbank tun, auf die so zugegriffen wird? Einfach gesagt, zwei Dinge:

- ➡ Sie können den Inhalt von Tabellen ansehen und ändern, und
- ➡ Sie können SQL-Abfragen über die Datenbank entwerfen.

Abbildung 39.7:
Datenbankpro-
jekt

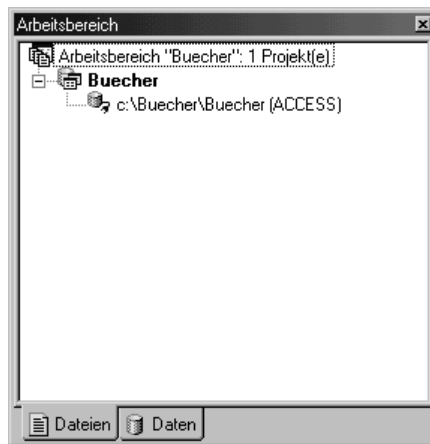
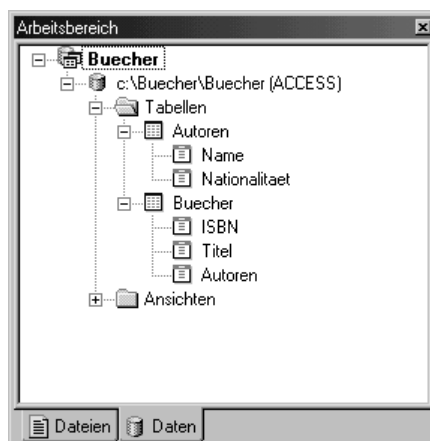


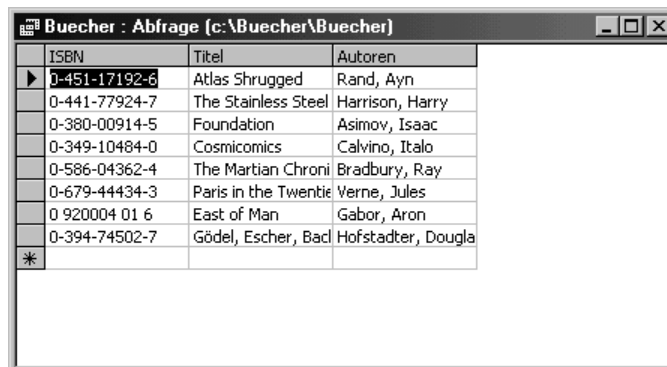
Abbildung 39.8:
Ein Datenbank-
projekt in der
Datenansicht



Dinge, die Sie nicht tun können, ist das Erstellen neuer Tabellen oder Ändern des Aufbaus der bereits bestehenden; so etwas ist nur möglich, wenn sich die Datenbank auf einem Microsoft SQL Server befindet.

39.2.1 Hinzufügen oder Ändern von Daten

Doppelklicken Sie auf die Tabelle BUECHER in der Datenansicht, um einen neuen Fenstertyp zu öffnen, das Abfragefenster (Abbildung 39.9). Dieses Fenster kann zur Eingabe, zum Ändern oder Löschen von Datensätzen aus jeder Tabelle der Datenbank dienen.



ISBN	Titel	Autoren
0-451-17192-6	Atlas Shrugged	Rand, Ayn
0-441-77924-7	The Stainless Steel	Harrison, Harry
0-380-00914-5	Foundation	Asimov, Isaac
0-349-10484-0	Cosmicomics	Calvino, Italo
0-586-04362-4	The Martian Chroni	Bradbury, Ray
0-679-44434-3	Paris in the Twentie	Verne, Jules
0 920004 01 6	East of Man	Gabor, Aron
0-394-74502-7	Gödel, Escher, Bac	Hofstadter, Dougla
*		

Abbildung 39.9:
Einige Lieblings-
bücher des
Autors

39.2.2 Abfragen erstellen

Das Abfragenfenster ist viel mehr als bloß ein Werkzeug zum Eingeben oder Ändern von Daten. Es bietet auch die Möglichkeit, komplexe Abfragen zu erstellen und zu testen.

Angenommen, Sie möchten zum Beispiel eine Liste aller amerikanischen Autoren in der Datenbank sehen.

1. Klicken Sie in der Dateien-Ansicht mit der rechten Taste auf den Namen der Datenbank.
2. Rufen Sie in dem erscheinenden Kontextmenü den Befehl NEUE ABFRAGE auf.

Es erscheint ein neues, leeres Abfragefenster, zusammen mit der Abfragen-Werkzeugleiste (Abbildung 39.10).



Abbildung 39.10:
Die Abfragen-
Werkzeugleiste

Wenn die Abfragen-Werkzeugleiste nicht erscheint, können Sie sie durch Klick mit der rechten Maustaste auf den Hintergrund einer beliebigen Symbolleiste aus der Liste der Werkzeugleisten auswählen.



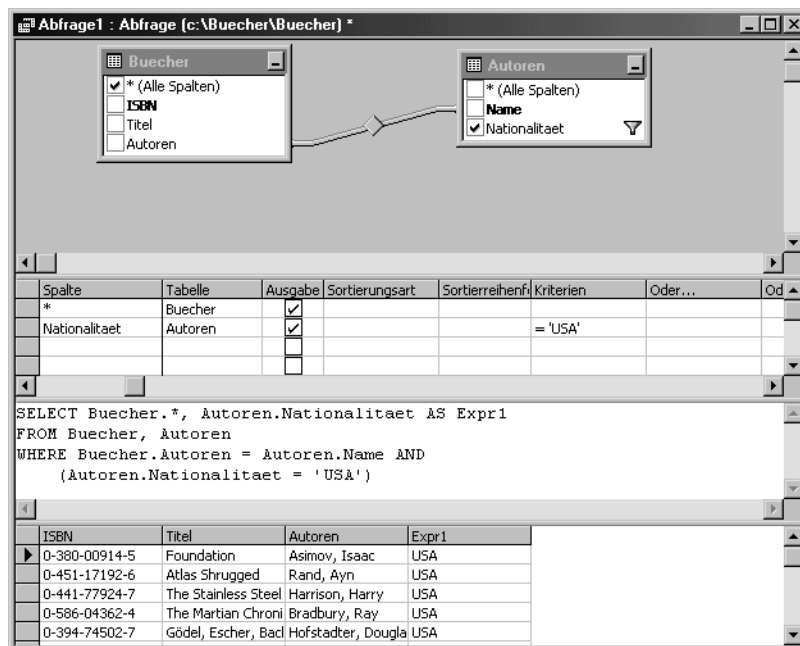
Diese Werkzeugleiste kann zum Ein- und Ausschalten verschiedener Bereiche des Abfragefensters verwendet werden, wie des Diagrammbereichs, des Tabellenbereichs, des SQL-Bereichs und des Ergebnisbereichs.

Tabelle aufnehmen Das Aufnehmen einer Tabelle in eine Abfrage ist einfach.

1. Schalten Sie die Diagrammansicht im Abfragefenster ein, und ziehen Sie einfach den Namen der Tabelle aus der Datenansicht in die Diagrammansicht.
2. Das Wählen einer SQL-Verbindung ist nicht schwieriger; Sie können Felder, die an einer Verbindung teilnehmen, von einer Tabelle zur anderen ziehen. Zum Beispiel können Sie das Autorenfeld von der Büchertabelle zum Namensfeld in der Autorentabelle ziehen, und Visual Studio wird automatisch eine SQL-Verbindung herstellen. Die Verbindung ist von Haus aus intern; um den Typ der Verbindung zu ändern, klicken Sie auf das Symbol, das die beiden Tabellen verbindet.

Abbildung 39.11 zeigt die vier Flächen des Abfragefensters mit einer fertigen Abfrage:

Abbildung 39.11:
Eine SELECT-
Abfrage



- ➡ Im Diagrammbereich sieht man die verknüpften Tabellen.
- ➡ Im Tabellenbereich sieht man die ausgewählten Spalten und die zugehörigen Filterkriterien.
- ➡ Im SQL-Bereich steht die SELECT-Anweisung, die der Abfrage entspricht.

- ➔ Im letzten Bereich wird schließlich das Ergebnis der Abfrage angezeigt, nachdem zuvor in der Abfragen-Werkzeugleiste auf das Ausrufezeichen (!) zum Austesten der Abfrage geklickt wurde.

SELECT-Abfragen sind nicht die einzigen SQL-Anweisungen, die mit Visual Studio erstellt werden können. Sie können auch INSERT, UPDATE und DELETE-Anweisungen erstellen.



39.2.3 Werkzeuge von Visual Database und C++-Anwendungen

Sie werden sich wohl die ganze Zeit gefragt haben: Was hat all das mit Visual C++-Programmierung zu tun?

Offensichtlich wird bei der Arbeit mit den Datenbankfähigkeiten von Visual Studio vorausgesetzt, daß Sie an einem datenbankbezogenen Projekt arbeiten. Wenn das der Fall ist, so kann das Aufstellen eines Datenbankprojekts in vielerlei Hinsicht von Vorteil sein.

- ➔ Erstens werden Sie in der Lage sein, die Daten direkt zu manipulieren, ohne auf externe Werkzeuge angewiesen zu sein, wie Microsoft Access.
- ➔ Zweitens werden Sie Abfragen testen können, bevor Sie sie in Ihre C++-Programme einverleiben. Wenn sich eine Abfrage als korrekt erweist, können Sie die SQL-Anweisung leicht in Ihre ODBC- oder DAO-Recordset-Klasse kopieren.
- ➔ Schließlich können Sie auch Tabellen und Ansichten erzeugen und ändern, wenn Ihre Datenbank auf einem Microsoft SQL Server liegt. Die Fähigkeiten der SQL Server werden in den folgenden Abschnitten beschrieben.

39.3 SQL Server anwenden

Wie bereits erwähnt, bietet Visual Studio eine Reihe Feature, die für den Microsoft SQL Server spezifisch sind – einschließlich der Fähigkeit, Tabellen und Ansichten zu erzeugen und zu ändern, gespeicherte SQL-Prozeduren zu erstellen und zu testen sowie neue Datenbanken mit Hilfe des Datenbank-Assistenten zu erstellen.

Der nächste Abschnitt gibt eine Übersicht über diese Fähigkeiten und zeigt, wie der Assistent einzusetzen ist, um die Datenbank *Buecher.mdb* auf dem SQL Server zu reproduzieren.



Es besteht die Möglichkeit, daß Ihre Einstellungen den meinen ähnlich sind und daß Sie SQL Server auf demselben System wie Visual Studio haben. Wenn das der Fall ist, müssen Sie sicherstellen, daß Microsoft SQL Server (MSSQLServer) unter demselben Konto-bezeichner läuft wie Visual Studio, ansonsten werden bestimmte Teile, wie das Debuggen gespeicherter Prozeduren, nicht richtig funktionieren.

39.3.1 Einrichten des SQL-Servers

Nachdem Sie den SQL-Server installiert haben (der SQL-Server ist Teil der Enterprise Edition von Visual C++), sind noch einige Konfigurationsarbeiten erforderlich, bevor der SQL-Server eingesetzt werden kann.

ODBC-Verbindung zum SQL-Server

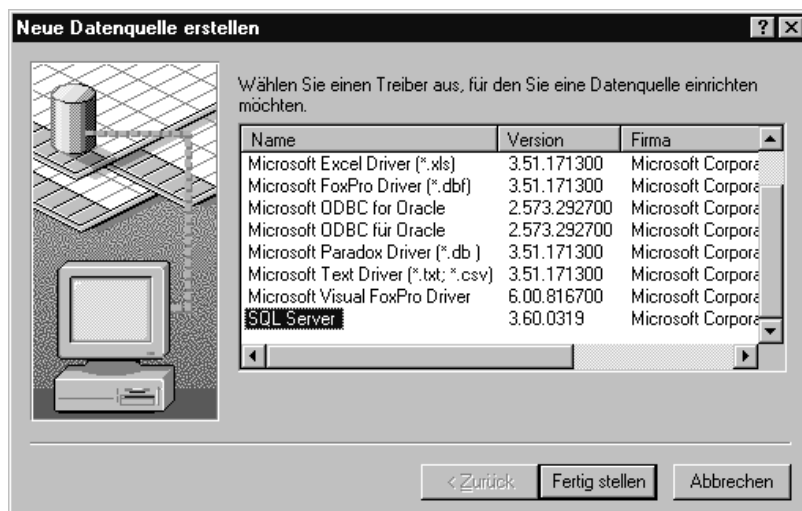
Als erstes richten Sie eine ODBC-Treiberverbindung ein.



Beachten Sie, daß Aufbau und Anordnung der Dialoge je nach Version des Betriebssystems leicht abweichen können.

1. Doppelklicken Sie in der Systemsteuerung auf das Symbol des ODBC-Setups.
2. Drücken Sie auf der Registerseite Benutzer-DSN auf die Schaltfläche HINZUFÜGEN.

Abbildung 39.12:
SQL-Server-Treiber auswählen



3. Wählen Sie im Dialog NEUE DATENQUELLE ERSTELLEN (Abbildung 39.12) den SQL Server als Treiber aus. Klicken Sie dann auf FERTIGSTELLEN.

Auf den nächsten Seiten richten wir die SQL-Server-Verbindung ein.

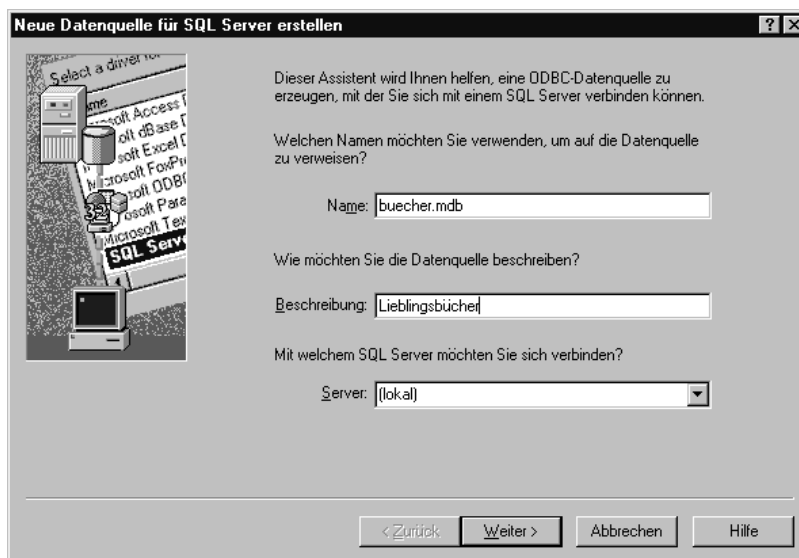


Abbildung 39.13:
Datenbank
angeben

4. Im ersten Dialogfeld NEUE DATENQUELLE FÜR SQL SERVER ERSTELLEN geben Sie einen Namen und eine Beschreibung für die Datenquelle ein. Wichtig ist die Angabe des SQL-Servers. Befindet sich der Server auf Ihrem Rechner, können Sie den Eintrag *(lokal)* auswählen.
5. Entscheiden Sie sich im nächsten Dialog (Abbildung 39.14) für die SQL SERVER-AUTHENTIFIZIERUNG, und geben Sie im unteren Teil des Dialogfelds LOGIN-ID und KENNWORT ein.
Die beiden letzten Eingaben sind nicht willkürlich, sondern sind für das Login auf den SQL Server erforderlich. Die LOGIN-ID *sa* und der Verzicht auf ein KENNWORT sind die Standardeinstellungen, die beim Setup des SQL Servers eingerichtet werden (sofern diese nicht von Ihnen geändert wurden).
6. Bestätigen Sie Vorgaben der nächsten Dialogfelder bis Sie zum abschließenden Dialog ODBC Microsoft SQL Server – Setup kommen.

Abbildung 39.14:
SQL Server-
Authentifizierung

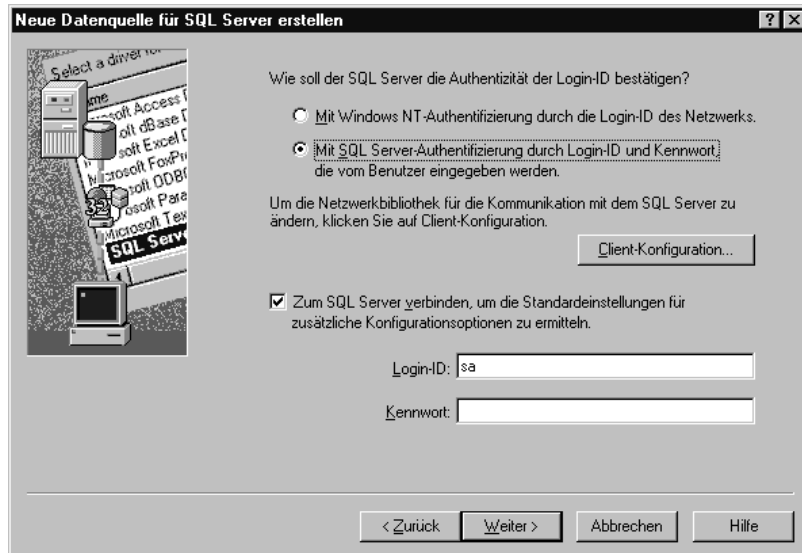


Abbildung 39.15:
Abschließendes
Dialogfeld



7. Drücken Sie auf den Schalter DATENQUELLE TESTEN. Danach sollten Sie eine ähnliche Meldung wie in Abbildung 39.16 angezeigt bekommen.

Die ODBC-Datenquelle ist jetzt eingerichtet.



Abbildung 39.16:
Test der Daten-
quelle

Registrieren des SQL-Servers

Jetzt muß der SQL-Server (lokal) noch registriert werden.

1. Rufen Sie den SQL Enterprise Manager (*sqlw.exe*) auf.
2. Wählen Sie im Menü SERVER den Befehl SERVER REGISTRIEREN aus.



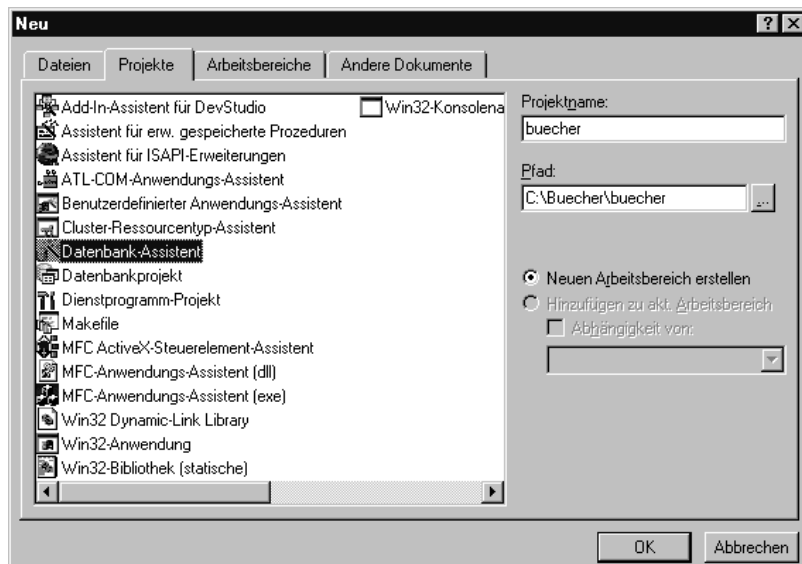
Abbildung 39.17:
SQL-Server
registrieren

3. Geben Sie den Namen des zu registrierenden SERVERS, die LOGIN-ID und das KENNWORT ein.
4. Drücken Sie auf die Schaltfläche REGISTRIEREN.

39.3.2 Erzeugen einer Datenbank mit dem Datenbank-Assistenten

1. Der Datenbank-Assistent kann als einer der möglichen Projekttypen gewählt werden, wenn Sie den NEU-Befehl aus dem DATEI-Menü von Visual C++ verwenden (Abbildung 39.18).

Abbildung 39.18:
Datenbank-Assistent auswählen



Der erste Dialog des Datenbank-Assistenten (Abbildung 39.19) erlaubt die Auswahl eines Servers.

2. Geben Sie den Namen des Rechners ein, auf dem der Server läuft; Sie müssen auch einen Microsoft-SQL-Server-Kontonamen und ein Paßwort angeben. (Beachten Sie, daß das der Microsoft-SQL-Server-Konto-Name ist (wie sa für den Systemverwalter) und nicht eine Windows-NT-Konto-ID.)
3. Der nächste Dialog (Abbildung 39.20) erlaubt das Wählen des Microsoft-SQL-Server-Datenbankgeräts, das zum Speichern Ihrer Datenbank und der Logs verwendet wird. Sie können bei Bedarf mit diesem Dialog auch neue Datenbankgeräte erzeugen.

Datenbank-Assistent -- Schritt 1 von 4

SQL Server für Ihre Datenbank angeben

Mit diesem Assistenten können Sie eine neue Microsoft SQL Server-Datenbank auf einem vorhandenen SQL Server erstellen.
Welchen SQL Server möchten Sie verwenden?

Geben Sie die Anmelde-ID und das Kennwort eines Kontos mit CREATE DATABASE-Privilegien auf diesem Server an.

Anmelde-ID:

Kennwort:

< Zurück Weiter > Fertigstellen

Abbildung 39.19:
Server auswählen

Datenbank-Assistent -- Schritt 2 von 4

Datenbank-Geräte auswählen

Mit dem Datenbank-Assistenten können Sie Ihre Datenbank auf einem vorhandenen SQL Server-Gerät einrichten oder neue SQL Server-Geräte erstellen.
Wählen Sie aus jeder Liste ein vorhandenes Gerät aus, oder wählen Sie "<Neues Gerät>" aus, um ein neues Gerät zu erstellen.

Welche Datenbank-Geräte möchten Sie verwenden?

Freier Speicherplatz: 2MB

Welches Protokoll-Gerät möchten Sie verwenden?

Freier Speicherplatz: 2MB

< Zurück Weiter > Fertigstellen

Abbildung 39.20:
Auswahl des
Datenbankge-
räts

Abbildung 39.21:
Datenbank
benennen

Datenbank-Assistent -- Schritt 3 von 4

Datenbank benennen

Name der neuen SQL Server-Datenbank?
Buecher

Reservierter Speicher für diese Datenbank?
1 MB

Für Protokoll reservierter Speicher?
1 MB

< Zurück Weiter > Fertigstellen

4. Der dritte Dialog (Abbildung 39.21) ist die Stelle, wo Sie den Namen Ihrer neuen Datenbank angeben können. Sie können auch die Größe des Plattenspeichers angeben, der für Datenbank und Logs reserviert werden soll.
5. Der vierte Schritt dient bloß der Bestätigung. Wenn Sie FERTIGSTELLEN klicken, wird die neue Datenbank auf Microsoft SQL Server erzeugt.

39.3.3 Hinzufügen oder Ändern von Tabellen auf Microsoft SQL Server

1. Um Ihrer Datenbank eine neue Tabelle hinzuzufügen, klicken Sie in der Datenansicht mit der rechten Maustaste auf den Ordner TABELLEN (Abbildung 39.22).

Visual Studio wird zuerst nach dem Namen der neuen Tabelle fragen und dann ihre Spalten in einem eigenen Fenster anzeigen (Abbildung 39.23).

Die Tabelle wird erst erzeugt, wenn Sie die Felder angegeben und die Änderungen gespeichert haben.

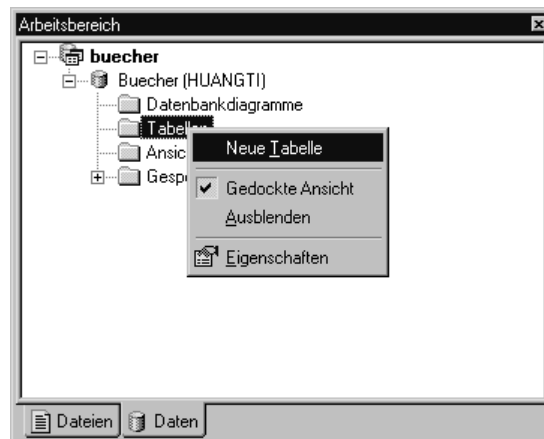


Abbildung 39.22:
Hinzufügen
einer neuen
Tabelle zu einer
Microsoft-SQL-
Server-Daten-
bank

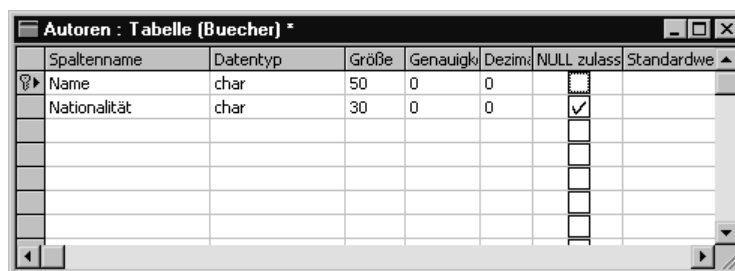


Abbildung 39.23:
Definieren der
Felder in einer
neuen Tabelle

2. Sie können (und sollten) eine der Spalten als Primärschlüssel angeben, bevor Sie die Tabelle speichern. Das tun Sie durch Klicken des entsprechenden Schalters in der Werkzeugleiste, die automatisch angezeigt wird, wenn die neue Tabelle im Entwurfsmodus geöffnet wird. Sie können aber auch den entsprechenden Befehl aus dem BEARBEITEN-Menü des Visual Studio verwenden. Beide werden erst aktiviert, wenn mindestens eine Spalte Ihrer Tabelle (im Entwurfsraster als Zeilen dargestellt) selektiert ist.
3. Definieren Sie analog die Tabelle BUECHER (Abbildung 39.24).

Primärschlüssel

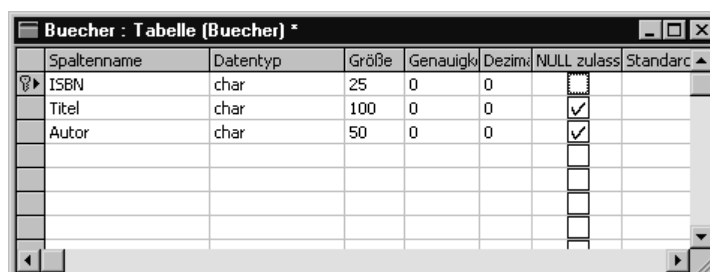


Abbildung 39.24:
Definieren der
Tabelle Buecher

Nachdem diese und die Autorentabelle definiert sind, können Sie zum nächsten Schritt schreiten: Definieren der Beziehungen.

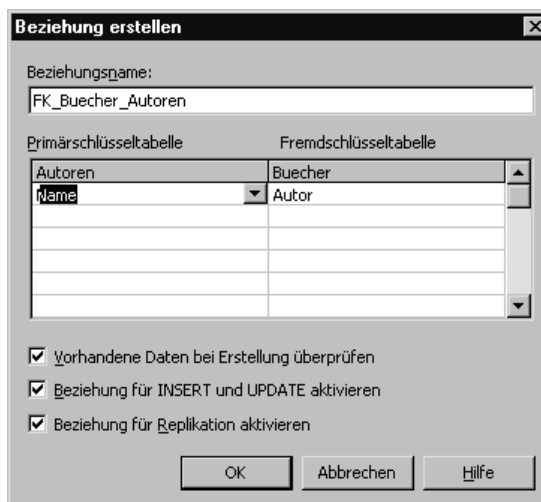
Beziehungen einrichten

Abbildung 39.25:
Diagramm
erstellen



4. Um eine neue Beziehung zu erzeugen, klicken Sie mit der rechten Maustaste in der Datenansicht das Symbol DATENBANKDIAGRAMME an und wählen im erscheinenden Kontextmenü NEUES DIAGRAMM aus.
5. Es erscheint ein leeres Diagramm, in das Sie die beiden Tabellen aus der Datenansicht ziehen.

Abbildung 39.26:
Definieren der
Beziehung



Die Beziehung, die wir zwischen Buecher und Autoren aufstellen wollen, ist eine Eine-zu-Vielen-Beziehung zwischen dem Namensfeld der Autorentabelle und dem Feld AUTOR in der Büchertabelle.

6. Um diese Beziehung zu definieren, ziehen Sie das Feld NAME über das Feld AUTOR. Daraufhin erscheint der Dialog **BEZIEHUNG ERSTELLEN** (Abbildung 39.26), wo die Eigenschaften der neuen Beziehung festgelegt oder geändert werden können.
7. Für die Beziehung, die wir aufstellen wollen, muß nichts geändert werden; klicken Sie also OK.

Wenn der Dialog verschwindet, sieht man im Diagramm eine grafische Darstellung der neuen Beziehung (Abbildung 39.27).

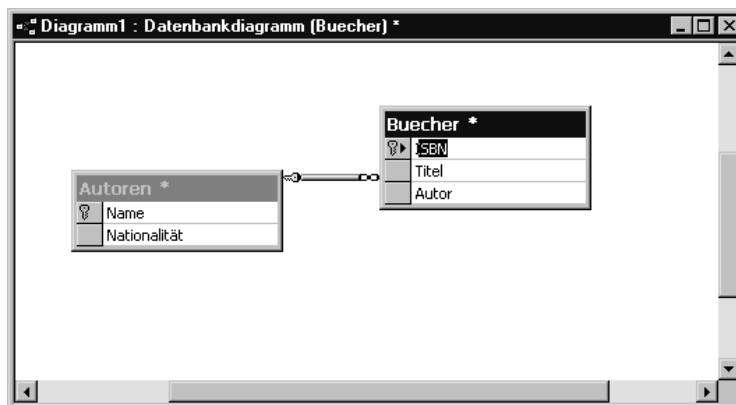


Abbildung 39.27:
Darstellung der
Beziehung

Daten eingeben

Nun können Sie die Tabellen öffnen und Daten eingeben. Beachten Sie, daß aufgrund der bereits definierten Beziehung referentielle Integrität erzwungen wird; wenn Sie versuchen, ein Buch einzugeben, dessen Autor noch nicht in der Autorentabelle steht, wird ein Fehler gemeldet.

39.3.4 Erstellen und Testen gespeicherter Prozeduren

Eine weitere Fähigkeit von Visual Studio, spezifisch für Microsoft SQL Server, ist das Erstellen, Ändern und Testen gespeicherter Prozeduren.

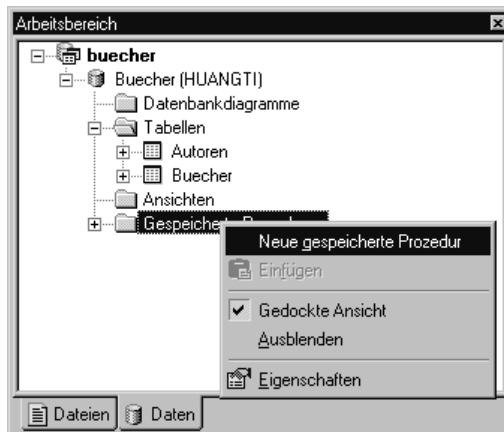
Gespeicherte Prozeduren sind SQL-Programme, die zum Ausführen auf dem Server entwickelt wurden. Sie können manuell aufgerufen oder aus anderen SQL-Anweisungen heraus referenziert werden, einschließlich in C-Code eingebetteten SQL-Anweisungen. Gespeicherte Prozeduren werden auch effizienter als andere SQL-Anweisungen ausgeführt, da sie vom Server direkt ausgeführt werden. Bei andernorts

*Gespeicherte
Prozeduren*

gespeicherten Prozeduren werden dem Server die Anweisungen einzeln übergeben. Es ist daher sinnvoll, für oft eingesetzte SQL-Anweisungen gespeicherte Prozeduren zu erstellen.

Das Erstellen einer gespeicherten Prozedur ist in Visual Studio einfach.

Abbildung 39.28:
Gespeicherte
Prozedur
anlegen



1. Um Ihrer Datenbank eine gespeicherte Prozedur hinzuzufügen, klicken Sie in der Datenansicht mit der rechten Maustaste auf den Ordner GESPEICHETERE PROZEDUREN (Abbildung 39.28).
2. Geben Sie in dem erscheinenden Editierfenster den SQL-Code für die Prozedur ein.

Abbildung 39.29:
Aufsetzen einer
gespeicherten
Prozedur

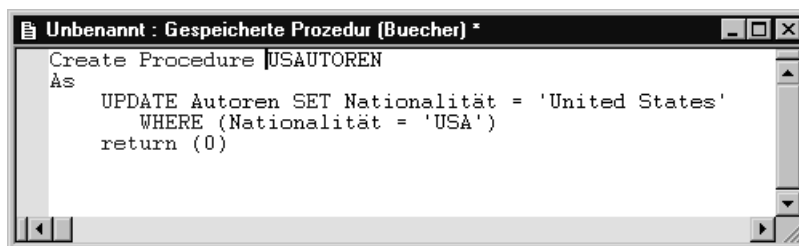


Abbildung 39.29 zeigt eine einfache gespeicherte Prozedur zum Ändern der Nationalität amerikanischer Autoren mit einer UPDATE-Abfrage.

Sie können die bekannten Debug-Fähigkeiten von Visual Studio nutzen, um die Prozedur zu debuggen. Zum Beispiel dient die Taste **F10** zum Ausführen des SQL-Codes im Einzelschritt.

39.4 Zusammenfassung

Visual C++ bietet eine Vielzahl von Werkzeugen, um Datenbanken und die Daten darin zu editieren. Einige dieser Werkzeuge sind für die meisten ODBC-Datenquellen verfügbar; andere sind spezifisch für den Microsoft SQL Server.

Ein Datenbankprojekt kann zum Zugriff auf Informationen, die in einer Datenbank gespeichert sind, erzeugt werden. Daten können editiert, hinzugefügt oder gelöscht werden. Ebenso können Abfragen erstellt werden, die über den Inhalt einer Tabelle berichten oder ihn verändern.

Mit Microsoft SQL Server sind einige zusätzliche Features verfügbar. Eine neue Datenbank kann mit dem Datenbank-Assistenten erzeugt werden. Tabellen können hinzugefügt oder Spalten vorhandener Tabellen geändert werden. Beziehungen können mit Hilfe der Datenbankdiagramme hergestellt werden.

Schließlich können auch die Debug-Fähigkeiten von Visual Studio im Zusammenhang mit Microsoft SQL Server eingesetzt werden. Gespeicherte SQL-Prozeduren können debuggt werden, indem Sie die Ihnen bereits von der Arbeit mit C++ vertraute Debug-Schnittstelle einsetzen.

Internet- und Netzwerkpro- grammierung

Teil **VI**

- 
- 40. Anwenden der WinInet-API
 - 41. MFC-Internet-Klassen
 - 42. Nachrichtenfähige Anwendungen mit MAPI
 - 43. TCP/IP-Programmierung mit WinSock
 - 44. Telefonie-Anwendungen mit TAPI
 - 45. Netzwerkprogrammierung mit Pipes und Aufruf von Remote Procedures

Anwenden der WinInet-API

Kapitel

40

Die neuen MFC-Klassen, die erweiterte Unterstützung für HTTP-, FTP- und Gopher-Operationen im Internet bieten, beruhen alle auf einer Erweiterung der Win32-API. Diese Erweiterung, entweder als Win32-Internet-Funktionen oder als WinInet-Bibliothek bezeichnet, enthält eine Reihe von Funktionen, die die Aufgabe, Kontakt zu einem WWW-, Gopher- oder FTP-Server aufzunehmen und Daten zu übertragen, stark vereinfachen.

Die WinInet-Bibliothek wurde ursprünglich von Microsoft als Zusatzkomponente zu Windows 95 und Windows NT vertrieben. Die Bibliothek ist als Datei *WININET.DLL* implementiert und darf von Entwicklern weitergegeben werden. Diese Funktionsbibliothek wird eventuell in zukünftige Windows-Versionen aufgenommen.

WinInet.dll

Wann brauchen Sie WinInet? Die Antwort ist einfach: Jedesmal, wenn Sie zu einem WWW-, Gopher- oder FTP-Server Verbindung aufnehmen und ein Dateien sollten Sie die WinInet-Bibliothek verwenden. Die Bibliothek ermöglicht das Entwickeln von Anwendungen ohne detaillierte Kenntnis der zugrundeliegenden Protokolle und ohne die TCP/IP-Verbindungen direkt über die WinSock-Bibliothek zu manipulieren. Sie behandelt auch die Eigenheiten verschiedener Server; zum Beispiel muß Ihre Anwendung nicht detailliert über die ASCII-Verzeichnis-Ausgabeformate verschiedener FTP-Server Bescheid wissen.

Sie können WinInet direkt oder indirekt über die MFC-Internet-Klassen verwenden.

- ➔ WinInet-Funktionen können von reinem C-Code aufgerufen werden; sie können in anderen Programmiersprachen, wie Visual Basic, die eine Schnittstelle zum Aufruf von Funktionen aus externen DLLs bietet, eingesetzt werden.
- ➔ Die MFC-Internet-Klassen sind dagegen C++-spezifisch.

Obwohl der Sinn der WinInet-Bibliothek die Befreiung des Programmiers von den Details der Protokollimplementationen ist, kann ihre Arbeitsweise leichter verstanden werden, wenn Sie mindestens ein oberflächliches Verständnis von WWW, Gopher und FTP haben. Deshalb beginnt dieses Kapitel mit einer kurzen Übersicht über diese Protokolle, gefolgt von der Antwort auf die Frage, wie WinInet eine stabile, leicht verwendbare Umgebung für sie bietet.

40.1 Internet-Protokolle

Wie erwähnt, bietet die WinInet-Bibliothek eine API für drei der am weitesten verbreiteten Internet-Protokolle:

- ➔ FTP,
- ➔ Gopher und
- ➔ HTTP.

Von diesen drei Protokollen ist FTP das älteste. Es bietet Möglichkeiten zum bidirektionalem Dateitransfer.

Gopher kann als unmittelbarer Vorgänger des WWW betrachtet werden, der ein effizientes menügesteuertes Navigationssystem zum Suchen und Abrufen von Texten und anderer Dokumente bietet.

HTTP ist das Protokoll, das hinter dem WWW steht.

Sowohl HTTP als auch Gopher sind unidirektional; sie können zum Abrufen der Dokumente vom Server eingesetzt werden, doch nicht zum Hochladen.

40.1.1 Das Dateitransferprotokoll FTP

Das älteste der drei Protokolle, FTP, wird schon viele Jahre im Internet von Servern eingesetzt, die Dateien zur Verfügung stellen. Gewöhnlich wird mit einem FTP-Server über eine Client-Anwendung wie das Windows-95/98-Programm *FTP.EXE* kommuniziert. Hier ist ein Beispiel einer typischen FTP-Sitzung:

*Beispiel für eine
FTP-Sitzung*

```
C:\>ftp vt11
Connected to vt11.
220 vt11 FTP server (Version wu-2.4(1) Sat Feb 18 13:40:36 CST 1995) ready.
User (vt11:(none)): ftp
331 Guest login ok, send your complete e-mail address as password.
Password:
230-Welcome, archive user! This is an experimental FTP server. If have any
230-unusual problems, please report them via e-mail to root@vt11
230-If you do have problems, please try using a dash (-) as the first character
230-of your password -- this will turn off the continuation messages that may
230-be confusing your ftp client.
230-
230 Guest login ok, access restrictions apply.
ftp> cd pub
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 2
drwxr-xr-x  2 root   wheel   1024 Apr  9 1996 .
drwxr-xr-x  8 root   wheel   1024 Apr  9 1996 ..
-rwxrwxr-x  1 root   wheel      0 Jul 10 1993 dummy_test_file
226 Transfer complete.
198 bytes received in 0.00 seconds (198000.00 Kbytes/sec)
ftp> get dummy_test_file
200 PORT command successful.
150 Opening ASCII mode data connection for dummy_test_file (0 bytes).
226 Transfer complete.
ftp> bye
221 Goodbye.
C:\>
```

Selbstverständlich ist das, was Sie hier sehen, nicht das Protokoll selbst, sondern bloß die Anwenderschnittstelle des FTP-Client. Die meisten Befehle sind einfach und leicht zu verstehen; andere, wie der Befehl `ls`, können für jemanden, der mit Unix nicht vertraut ist, etwas undurchsichtig erscheinen (`ls` ist das Unix-Äquivalent des MS-DOS-Befehls `DIR`, der die Inhalte von Verzeichnissen auflistet).

Doch wie funktioniert all das? Wie kommunizieren FTP-Clients und Server tatsächlich? Statt eine sinnlose und mühsame Diskussion über die Interna des Protokolls zu beginnen, machen wir ein praktisches Experiment, indem wir mit unserem Telnet-Client Kontakt zu einem FTP-Server aufnehmen. Die meisten Telnet-Programme, einschließlich der Windows-95/98- und Windows-NT-Versionen von *TELNET.EXE*, akzeptieren einen optionalen Parameter, der den TCP-Port bezeichnet, zu dem man Verbindung aufnimmt. Wohlbekannte Portnummern haben auch einen symbolischen Namen, der in einer Datei namens *services* (im Hauptverzeichnis von Windows, oder unter Unix in */etc*) gespeichert ist. Die Portnummer für FTP, Port 21, hat den symbolischen Identifizierer (nicht überrascht sein!) `ftp`, somit lautet unsere Telnet-Befehlszeile:

```
C:\>TELNET VTT1 FTP
```

Sehen wir uns an, was geschieht, wenn wir versuchen, eine FTP-Session manuell aufzubauen:

```
Connected to vt11.
Escape character is '^J'.
220 vt11 FTP server (Version wu-2.4(1) Sat Feb 18 13:40:36 CST 1995) ready.
USER anonymous
331 Guest login ok, send your complete e-mail address as password.
PASS user@host.net
230-Welcome, archive user! This is an experimental FTP server. If have any
230-unusual problems, please report them via e-mail to root@vt11
230-If you do have problems, please try using a dash (-) as the first character
230-of your password -- this will turn off the continuation messages that may
230-be confusing your ftp client.
230-
230 Guest login ok, access restrictions apply.
CWD pub
250 CWD command successful.
LIST
425 Can't build data connection: Connection refused.
```

Hoppla, weiter geht's nicht. Um tatsächlich eine Datei zu übertragen (Verzeichnislisten werden von FTP als Textdateien behandelt), müßten wir eine zweite Verbindung aufbauen, da FTP zwei Verbindungen verwendet: eine für die Steuerung, die andere für Datenübertragung.

Trotzdem sollte diese kleine Telnet-Session lehrreich gewesen sein. Wie Sie sehen können, sind die meisten Befehle und Antworten für den Bediener lesbar; Daten werden getrennt übertragen. Wenn Verzeichnislisten übertragen werden, haben FTP-Clients zwei Möglichkeiten: die Liste als solche anzuzeigen oder ihren Inhalt zu analysieren und zu interpretieren. Im zweiten Fall muß der Client wissen, wie verschiedene Server Verzeichnisinhalte darstellen.

Das FTP-Protokoll kann für zahlreiche Aufgaben abgesehen vom Anfordern von Dateien eingesetzt werden. Da es bidirektional ist, können; über FTP auch Dateien zum Server gesendet werden. Das Protokoll bietet auch einige Befehle zum Manipulieren des Dateisystems, also zum Löschen von Dateien und zum Erzeugen, Entfernen und Wechseln von Verzeichnissen.

40.1.2 Das Gopher-Protokoll

Gopher ist ein Internet-Protokoll, das auch heute noch sehr verbreitet ist, und das Internet als Reihe von Menüs und Dokumenten präsentiert. Eine typische Gopher-Sitzung beginnt mit der Verbindung zu einem beliebigen Gopher-Root-Server, der dem User eine Reihe von Menüpunkten präsentiert; das Wählen dieser Menüpunkte verbindet den User mit anderen Gopher-Servern, um entweder Dokumente zu laden oder in weiteren Menüs zu navigieren.

In der Vergangenheit war Gopher-Client-Software typischerweise textbasiert. Heutzutage können die meisten Web-Clients wie Netscape oder Internet Explorer mit Gopher-Servern Verbindung aufnehmen. Sie machen somit andere Programme überflüssig.

Gopher wurde ursprünglich an der Universität Minnesota entwickelt, wo der heute noch am häufigsten besuchte Server betrieben wird. Auf den meisten Rechnern werden Sie nach dem Starten eines Gopher-Client folgendes Menü sehen:

```
Internet Gopher Information Client v2.1.-1
Home Gopher server: gopher2.tc.umn.edu

--> 1. Information About Gopher/
    2. Computer Information/
    3. Discussion Groups/
    4. Fun & Games/
    5. Internet file server (ftp) sites/
    6. Libraries/
    7. News/
    8. Other Gopher and Information Servers/
    9. Phone Books/
    10. Search Gopher Titles at the University of Minnesota <?>
    11. Search lots of places at the University of Minnesota<?>
    12. University of Minnesota Campus Information/

Press ? for Help, q to Quit                                     Page: 1/1
```

Der User kann den Cursor mit den Pfeiltasten nach oben und nach unten bewegen, ein Punkt kann mit der -Taste ausgewählt werden.

Wie liefert das Protokoll eigentlich diese Informationen? Versuchen wir, uns manuell mit dem Server der Universität von Minnesota zu verbinden (das heißt über Telnet) und dasselbe Menü anzufordern. Dazu müssen wir lediglich, ein Wagenrücklaufzeichen + an den Server senden, nachdem die Verbindung hergestellt wurde:

```
$ telnet gopher2.tc.umn.edu gopher
Trying 134.84.132.29...
Connected to gopher.tc.umn.edu.
Escape character is '^J'.
1Information About Gopher<tab>1/Information About
Gopher<tab>gopher.tc.umn.edu<tab>70<tab>+
1Computer Information<tab>1/computer<tab>spinaltap.micro.umn.edu<tab>70
1Discussion Groups<tab>1/Mailing Lists<tab>gopher.tc.umn.edu<tab>70<tab>+
1Fun & Games<tab>1/fun<tab>spinaltap.micro.umn.edu<tab>70
1Internet file server (ftp) sites<tab>1/FTP
Searches<tab>gopher.tc.umn.edu<tab>70<tab>+
1Libraries<tab>1/Libraries<tab>gopher.tc.umn.edu<tab>70<tab>+
1News<tab>1/News<tab>gopher.tc.umn.edu<tab>70<tab>+
1Other Gopher and Information Servers<tab>1/Other Gopher and Information
Servers<tab>gopher.tc.umn.edu<tab>70<tab>+
1Phone Books<tab>1/Phone Books<tab>gopher.tc.umn.edu<tab>70<tab>+
7Search Gopher Titles at the University of
Minnesota<tab><tab>mudhoney.micro.umn.edu<tab>4325
7Search lots of places at the University of Minnesota<tab><tab>mindex:/lotsoplaces
spinaltap.micro.umn.edu<tab>70
1University of Minnesota Campus Information<tab>1/
```

```
uofm<tab>gopher.tc.umn.edu<tab>70<tab>+
.
Connection closed by foreign host.
$
```

Wie Sie sehen, kommt der Inhalt des Menüs, das Gopher am Bildschirm ausgibt, in einem stark strukturierten Format. Jede Zeile beginnt mit einem einzelnen Zeichen, das den Informationstyp des Menüpunktes bezeichnet. Typ 1, der im Menü am häufigsten vorkommt, ist ein Gopher-Verzeichnis (das heißt ein weiteres Menü), während Typ 7 einen Suchdienst darstellt. Natürlich gibt es zahlreiche weitere Typen.

Das Typenzeichen wird von einigen Strings gefolgt, die durch Tabulatoren getrennt sind. Der erste ist der Selektor-String, der lesbare Titel des Menüpunktes. Der zweite, dritte und vierte String bezeichnen den internen Identifizierer des Menüpunktes, den Host und die Portnummer, von wo dieser angefordert werden kann.

Die Ausgabe von Gopher ist einfacher, wenn ein Dokument angefordert wird. Der Client fordert das Dokument an, indem er seinen Identifizierer sendet, gefolgt von einem Wagenrücklaufzeichen. Der Server antwortet mit der Ausgabe des Inhalts des Dokuments an den Client.

Sicherlich wird eine vereinfachte Übersicht wie diese dem Protokoll nicht gerecht, doch ich hoffe, es hat gereicht, um ein grundlegendes Verständnis dafür zu entwickeln, wie Gopher Informationen liefert.

40.1.3 Das Hypertext-Transfer-Protokoll HTTP

Das bei weitem populärste Protokoll im Internet ist heutzutage HTTP. Dieses Protokoll wird vom World Wide Web verwendet, um Multimedia-Informationen als Hypertext Markup Language (HTML)-Dokumente, kurz Webseiten, zu versenden.

Dank der Popularität des WWW glaube ich nicht, daß Webseiten aus der Sicht des Users eingeführt werden müssen. Ich bezweifle ernsthaft, daß es auch nur einen einzigen C++-, geschweige denn Visual-C++-Entwickler auf diesem Planeten gibt, der noch nie einen Web-Browser in Aktion gesehen hat.

Springen wir somit weiter und sehen wir uns an, wie das Protokoll funktioniert. Angenommen, Sie versuchen, unter dem URL `http://localhost/blank.html` ein Dokument aus dem Web anzufordern. Eine einfache Telnet-Session zeigt, wie dies manuell gemacht werden kann:

```
$telnet localhost http
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /index.html
```

```
<HTML>
<HEAD>
<TITLE>Microsoft Internet Explorer</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
Connection closed by foreign host.
$
```

Bisher keine großen Geheimnisse. Der URL wird offensichtlich vom Browser in zwei Teile geteilt.

- ➡ Der erste Teil, `http://localhost`, identifiziert sowohl den Host, der das angeforderte Material liefert, als auch das verwendete Protokoll.
- ➡ Der zweite Teil, `/index.html`, wird an den Server in Form eines GET-Befehls gesendet, worauf der Server mit dem Inhalt des Dokuments antwortet.

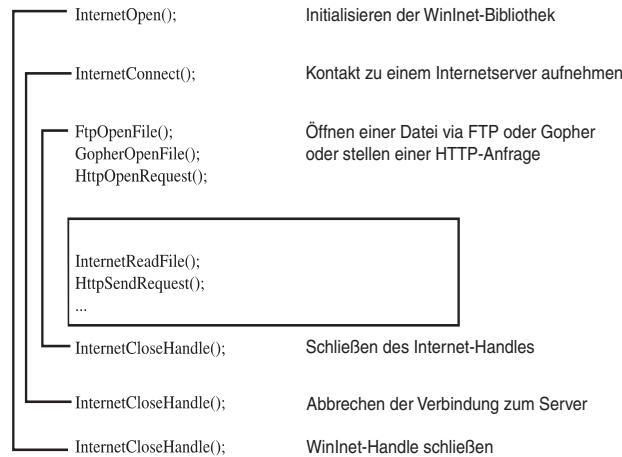
Wiederum ist das tatsächliche Protokoll natürlich viel komplexer, als diese einfache Session andeutet. Zum Beispiel kann ein HTTP-Client seinen Typ identifizieren, Zugriffsberechtigungsinformationen für den Zugang zu paßwortgeschützten Webseiten senden und so weiter. Trotzdem sollte dieses einfache Beispiel einige der Geheimnisse lüften, indem es zeigt, was im Hintergrund passiert, wenn eine einfache Anforderung von einem Web-Server bedient wird.

40.2 Die WinInet-Bibliothek

Die vorherigen Beispiele veranschaulichen, wie einfache Dateien von Servern über die Protokolle FTP, Gopher und HTTP angefordert werden. Das Abwickeln des Zwiegesprächs mit einem Server aus einem C- oder C++-Programm wäre ein mühsames Unterfangen. Die Anwendung müßte einen oder mehrere Sockets öffnen und verwalten. Sie müßte detaillierte Kenntnisse über die verwendeten Protokolle einbringen, einschließlich server- und versionsspezifischer Eigenheiten. Da diese Art von Verbindungen häufig in der Internet-Programmierung eingesetzt werden, ist das Erstellen einer wiederverwendbaren Bibliothek ein selbstverständlicher Schritt, und genau das, was die WinInet-Bibliothek bietet.

Das WinInet-Programmiermodell sollte Windows-Programmierern mit Erfahrung in ODBC oder TAPI vertraut sein. Wie die Schichten einer Zwiebel besteht auch eine WinInet-Transaktion aus Erzeugen einer Sitzung, einer Verbindung mit dieser Sitzung und einer Dateiübertragung innerhalb dieser Verbindung, wie Abbildung 40.1 zeigt.

Abbildung 40.1:
Typische Folge
von WinInet-
Aufrufen



Alle WinInet-Sessions beginnen mit dem Erzeugen eines Handles auf eine WinInet-Session durch Aufruf der Funktion `InternetOpen`. Die meisten Anwendungen werden diese Funktion nur einmal aufrufen, da der Handle wiederverwendbar ist. Trotzdem können Anwendungen mehrere Handles verwenden, beispielsweise wenn mehrere Proxy-Server im Einsatz sind.

Mit einem Sitzungs-Handle ausgerüstet, stellen Anwendungen Verbindungen zum Internet über einen Aufruf von `InternetConnect` her. Diese Funktion wird zum Herstellen von Verbindungen zu allen drei Servertypen verwendet. `InternetConnect` gibt ebenfalls ein Handle zurück, das zu nachfolgenden Aufrufen von Funktionen wie `FtpOpenFile`, `GopherOpenFile` oder `HttpOpenRequest` eingesetzt werden kann.

Die folgenden Abschnitte enthalten einfache Beispiele, die Dateien von FTP, Gopher und HTTP-Servern anfordern.

40.2.1 Anfordern von Dateien von einem FTP-Server

Listing 40.1 enthält ein Beispielprogramm, das eine bestimmte Datei von einem FTP-Server im Internet anfordert.

Listing 40.1:
Einfache FTP-
Anwendung

```

#include <windows.h>
#include <wininet.h>
#include <stdio.h>

void main(void)
{
    HINTERNET hIS, hIC, hIF;
    DWORD dwBytes;
    char c;

    hIS = InternetOpen("FTPGET", INTERNET_OPEN_TYPE_DIRECT, NULL,
        NULL, 0);

```

```
hIC = InternetConnect(hIS, "ftp.microsoft.com",
                    INTERNET_DEFAULT_FTP_PORT, NULL, NULL,
                    INTERNET_SERVICE_FTP, 0, 0);
hIF = FtpOpenFile(hIC, "disclaimer.txt", GENERIC_READ,
                FTP_TRANSFER_TYPE_ASCII, 0);
while (InternetReadFile(hIF, &c, 1, &dwBytes))
{
    if (dwBytes != 1) break;
    putchar(c);
}

InternetCloseHandle(hIF);
InternetCloseHandle(hIC);
InternetCloseHandle(hIS);
}
```

Um dieses Beispiel zu kompilieren, geben Sie folgende Kommandozeile ein:

```
CL FTPGET.C WININET.LIB
```

Das fertige Programm, *ftpget.exe*, sollte erfolgreich den Microsoft-FTP-Server anwählen und den Inhalt der Datei *disclaimer.txt*, die sich auf jenem Server befindet, ausgeben.

Das Programm enthält keine Überraschungen.

Verbindung herstellen

Wie vorhin besprochen, sind drei Aufrufe nötig, um die Verbindung herzustellen und die erwünschte Datei zu öffnen:

- Ein Aufruf von `InternetOpen` initialisiert die WinInet-Bibliothek.
- Ein Aufruf von `InternetConnect` stellt eine anonyme FTP-Verbindung zum Server her.
- Ein Aufruf von `FtpOpenFile` greift tatsächlich auf die Datei zu.

Datei lesen

Nachdem die Datei mit Erfolg geöffnet wurde, werden wiederholte Aufrufe von `InternetReadFile` verwendet, um deren Inhalte abzurufen. In diesem Beispiel findet der Abruf zeichenweise statt. In echten Anwendungen würde man einen viel größeren Puffer für eine effizientere Übertragung einsetzen.

Mit einigen Änderungen kann diese Anwendung jeden Server ansprechen und eine Datei Ihrer Wahl abrufen. Einige weitere Änderungen könnten der Anwendung neue Feature hinzufügen, z. B. die Fähigkeit, zwischen ASCII- oder binärer Dateiübertragung zu wählen oder Dateien auf den Server zu laden.

Die WinInet-Bibliothek enthält auch Funktionen, die das Browsen in FTP-Verzeichnissen erleichtern. Die Funktion `FtpFindFirstFile` bei-

spielsweise kann verwendet werden, um Verzeichnisinformationen vom FTP-Server anzufordern und zu parsen. Der Hauptvorteil dieses Ansatzes ist, daß Ihre Anwendung nicht wissen muß, in welchem Format die Verzeichnisinformationen von verschiedenen FTP-Servern präsentiert werden.

40.2.2 Anforderung von Dateien von einem Gopher-Server

Dateien von einem Gopher-Server anzufordern ist nicht schwieriger als von einem FTP-Server, wie Listing 40.2 zeigt.

*Listing 40.2:
Einfache Gopher-
Anwendung*

```
#include <windows.h>
#include <wininet.h>
#include <stdio.h>

void main(void)
{
    HINTERNET hIS, hIC, hIF;
    DWORD dwBytes;
    char c, *p;

    hIS = InternetOpen("GOPHRGET", INTERNET_OPEN_TYPE_DIRECT,
        NULL, NULL, 0);
    hIC = InternetConnect(hIS, "localhost", 0, NULL, NULL,
        INTERNET_SERVICE_GOPHER, 0, 0);
    hIF = GopherOpenFile(hIC, "0\\t\\tgopher2.tc.umn.edu\\t70",
        NULL, 0, 0);

    while (InternetReadFile(hIF, &c, 1, &dwBytes))
    {
        if (dwBytes != 1) break;
        putchar(c);
    }

    InternetCloseHandle(hIF);
    InternetCloseHandle(hIC);
    InternetCloseHandle(hIS);
}
```

Diese Anwendung kann mit einem einfachen Befehl:

```
CL GOPHRGET.C WININET.LIB
```

von der Befehlszeile aus kompiliert werden.

Die ersten beiden WinInet-Aufrufe in diesem Programm sind dieselben wie die Aufrufe im FTP-Beispiel.



Beachten Sie trotzdem, daß die an InternetConnect übergebenen Parameter nicht den Host angeben, der später angesprochen wird. Das ist so, weil der Internet-Host und die Portnummer im Aufruf von GopherOpenFile angegeben und der Hostname und die Portnummer, die an InternetConnect übergeben werden, scheinbar ignoriert

werden. Ich habe in der WinInet-Dokumentation keinen konkreten Hinweis zu diesem Effekt gefunden, somit könnte dieses Verhalten in künftigen Versionen der WinInet-Bibliothek geändert werden.

Der Aufruf von `GopherOpenFile` unterscheidet sich von seinem FTP-Gegenstück darin, daß der zweite Parameter keinen Dateinamen, sondern einen Gopher-Locatorstring angibt. Ein Gopher-Locatorstring ist im Format den Strings aus den Gopher-Menüs ähnlich. Er enthält durch Tabulatoren getrennte Felder, die Typ, Titel und Identifizierer des Dokuments angeben, sowie den Host und die Portnummer, wo das Dokument angefordert werden kann. In unserem Beispiel wird der String

```
"0\t\tgopher2.tc.umn.edu\t70"
```

verwendet. Dieser wird folgendermaßen übersetzt:

0	Typ ist '0'; Titel ist leer
<leerer String>	Identifizierer des Root-Menüs
gopher2.tc.umn.edu	Hostname
70	Standard Gopher TCP-Portnummer

Obwohl in diesem Beispiel ein manuell erstellter Locator verwendet wurde, wäre dies typischerweise im richtigen Leben nicht der Fall. Viel wahrscheinlicher würden Sie einen Locatorstring verwenden, der von der Funktion `GopherFindFirstFile` zurückgegeben wurde. Diese Funktion kann ein Gopher-Menü abrufen und parsen und den Inhalt in der Form von Locatorstrings, die in einer Struktur vom Typ `GOPHER_FIND_DATA` eingebettet sind, präsentieren.

40.2.3 Anfordern von Dateien von einem HTTP-Server

Das Beispielprogramm aus Listing 40.3 führt die nun vertraute Aufgabe des Anforderns einer Datei durch, diesmal von einem HTTP-Server.

```
#include <windows.h>
#include <wininet.h>
#include <stdio.h>

void main(void)
{
    HINTERNET hIS, hIC, hIR;
    DWORD dwBytes;
    char c;
    hIS = InternetOpen("HTTPGET", INTERNET_OPEN_TYPE_DIRECT,
        NULL, NULL, 0);
    hIC = InternetConnect(hIS, "www.microsoft.com",
        INTERNET_DEFAULT_HTTP_PORT, NULL, NULL,
        INTERNET_SERVICE_HTTP, 0, 0);
```

*Listing 40.3:
Einfache HTTP-
Anwendung*

```

hIR = HttpOpenRequest(hIC, NULL, "/default.asp", NULL, NULL,
                     NULL, 0, 0);

if (HttpSendRequest(hIR, NULL, 0, NULL, 0))
{
    while (InternetReadFile(hIR, &c, 1, &dwBytes))
    {
        if (dwBytes != 1) break;

        putchar(c);
    }
}

InternetCloseHandle(hIR);
InternetCloseHandle(hIC);
InternetCloseHandle(hIS);
}

```

Um dieses Programm von der Kommandozeile aus zu kompilieren, geben Sie folgende Zeile ein:

```
CL HTTPGET.C WININET.LIB
```

Wie auch seine FTP- und Gopher-Gegenstücke, beginnt diese Anwendung mit Aufrufen von `InternetOpen` und `InternetConnect`; diesmal wird eine HTTP-Verbindung zu Microsofts Web-Hauptserver angegeben.

Das Öffnen einer HTTP-Anforderung mit `HttpOpenRequest` teilt die Anforderung nicht automatisch dem Server mit. Dafür ist ein Aufruf von `HttpSendRequest` nötig. Nachdem die Anforderung erfolgreich gesendet wurde, kann die Antwort des Servers mit `InternetReadFile` gelesen werden, wie bei FTP und Gopher.

40.2.4 Weitere WinInet-Feature und -Funktionen

Asynchroner Modus Die Beispiele in diesem Kapitel verwenden alle WinInet-Funktionen im synchronen (Blocking) Modus. Mit anderen Worten, die Funktionen kehren nicht zurück, bis die angegebene Anforderung erfüllt ist oder ein Fehler auftritt. Die WinInet-Bibliothek unterstützt auch den asynchronen Modus, in dem Funktionen sofort zurückkehren und die Anforderung im Hintergrund abgearbeitet wird.

Asynchroner Modus kann mit dem Flag `INTERNET_FLAG_ASYNC` im fünften Parameter von `InternetOpen` angegeben werden. Nachdem dieses Flag angegeben wurde, arbeiten alle folgenden Aufrufe, die das Session-Handle verwenden, das von `InternetOpen` zurückgegeben wurde, im asynchronen Modus.

Der asynchrone Modus erfordert, daß eine Zustandsrückruffunktion mit `InternetStatusCallback` registriert wird. Wenn eine WinInet-Funktion im asynchronen Modus aufgerufen wird, kann sie den Fehlercode

`ERROR_IO_PENDING` zurückgeben. wenn die Operation beendet ist, ruft die WinInet-Bibliothek die Zustandsrückruffunktion mit dem Wert `INTERNET_STATUS_REQUEST_COMPLETED` auf.

Weitere Features der WinInet-Bibliothek umfassen die Fähigkeit, den Cache zu verwalten. Der Cache wird zum Speichern lokaler Kopien von aus dem Internet übertragenen Daten verwendet, was den wiederholten Zugriff auf dieselben Informationen beschleunigt, indem mehrfaches Laden vermieden wird. Cache-Funktionen in WinInet ermöglichen es Anwendungen, lokal gespeicherte Daten mit bestimmten URLs explizit zu identifizieren und die gespeicherten Daten später durch URLs zu referenzieren.

*Cache-
Verwaltung*

40.3 Zusammenfassung

Die WinInet-Bibliothek ist eine Zugabe von Microsoft zur Win32-API, um die Internet-Unterstützung zu verbessern. Die Bibliothek bietet Unterstützung für die Kommunikation mit drei Servertypen: FTP, HTTP und Gopher. Diese drei Protokolle sind die meistverbreiteten für den Zugang zu Dateiinhalten.

Die typische Aufruffolge besteht aus einem Aufruf von `InternetOpen`, gefolgt von `InternetConnect`, und von Funktionen, die für das verwendete Protokoll spezifisch sind. Viele WinInet-Funktionen geben Internet-Handles vom Typ `HINTERNET` zurück, die mit Aufrufen von `InternetCloseHandle` geschlossen werden können.

Außer Dateiübertragung unterstützt die WinInet-Bibliothek viele andere Operationen, die jeweils für die drei Servertypen spezifisch sind. Im einzelnen unterstützt sie Dateien- und Verzeichnismanipulation sowie Laden von Dateien auf FTP-Server, Parsen von FTP-Verzeichnislisten, Interpretieren von Gopher-Menüs und verschiedene HTTP-Operationen. Die Bibliothek unterstützt auch Cache-Operationen.

WinInet-Funktionen können im synchronen (blockierenden) oder asynchronen Modus arbeiten. Der asynchrone Modus erfordert die Installation einer Rückruffunktion, die von der Bibliothek zum Mitteilen von Zustandsinformationen über laufende Anforderungen verwendet wird.

MFC-Internet-Klassen

Kapitel

41

41.1 Internet- Unterstützungsklassen

Microsofts Gesinnungswandel seit den Tagen, als das Internet dem Unternehmen zu fremd und exotisch erschien – das Engagement des Unternehmens, Internet-Technologien in allen Produkten zu unterstützen – wird an den Änderungen und Zusätzen der MFC-Bibliothek deutlich. Eine gewisse Zeit bot die Bibliothek eine Unterstützung für Low-Level-WinSock-Operationen. Mit MFC 4.2 führte Microsoft einige Internet-Unterstützungsklassen ein, welche auf das Internet bezogene Programmieraufgaben erleichtern sollten. Zu diesen Klassen zählen unter anderem `CInternetSession`, `CInternetConnection` sowie die von `CStdioFile` und `CInternetFile` abgeleiteten Klassen.

41.2 Die MFC-Internet- Klassenarchitektur

Die MFC-Internet-Unterstützungsklassen nutzen die neue Microsoft-Win32-Internet-Bibliothek (WinInet). Diese Bibliothek ist auf der WinSocket-TCP/IP-Implementierung aufgebaut und bietet High-Level-Unterstützung für allgemeine Internet-Programmieraufgaben. Die WinInet-Bibliothek unterstützt außerdem FTP-, Gopher- und HTTP-Verbindungen.

Diese drei Verbindungstypen werden von den entsprechenden MFC-Mantelklassen

- `CFtpConnection`,
- `CGopherConnection` und
- `CHttpConnection`

angeboten. Zusätzliche Klassen implementieren das Konzept einer Internet-Datei, über die Daten von und zu einem Server transferiert werden können. Dies geschieht mit Hilfe einiger Standardmethoden.

41.3 Aufbau von Internet-Verbindungen

*Aufbau einer
Internet-Verbin-
dung*

Der Aufbau einer Internet-Verbindung erfordert gewöhnlich drei MFC-Objekte.

- Ein Objekt vom Typ `CInternetSession` initialisiert die WinInet-Bibliothek.
- Ein Verbindungsobjekt repräsentiert die Verbindung zwischen dem Client-Computer und dem Internet-Server.
- Schließlich wird über ein Dateiobjekt der eigentliche Datentransfer vollzogen.

41.3.1 Internet-Sitzungen – `CInternetSession`

Eine Internet-Sitzung wird eingerichtet, indem ein `CInternetSession`-Objekt erstellt wird. Das Erstellen dieses Objekts initialisiert die WinInet-Bibliothek und konfiguriert interne MFC-Datenstrukturen.

Dem `CInternetSession`-Konstruktor werden mehrere Parameter übergeben. Der dritte und der fünfte Parameter sind besonders interessant.

dwAccessType Der dritte Parameter `dwAccessType` steuert die Verwendung des Proxy-Servers. Eine Verbindung kann direkt sein oder über einen in der Registrierung angegebenen vorkonfigurierten Proxy-Server respektive über einen Proxy-Server aufgebaut werden, der in dem Aufruf der Konstruktorfunktion angegeben wird.

dwFlags Der fünfte Parameter `dwFlags` steuert die Cache-Nutzung und die asynchrone Ausführung. Gewöhnlich wird der Cache bestmöglich genutzt. Eine Verbindung wird nur dann zu einem Server aufgebaut, wenn diese Vorgehensweise erforderlich ist. Dieser Umstand ist von zwei Fakto-

ren abhängig. Verwenden Sie das Flag `INTERNET_FLAG_DONT_CACHE`, werden die Daten aus dem Internet nicht im Cache abgelegt. Das Flag `INTERNET_FLAG_OFFLINE` weist die Bibliothek an, keine Verbindung zu dem Server herzustellen, sondern zu versuchen, die angeforderten Daten aus dem Cache einzulesen.

Internet-Aufrufe werden gewöhnlich synchron (in Blöcken) ausgeführt. Wird eine Anfrage über eine Bibliotheksfunktion gestellt, ist die Funktion so lange nicht beendet, bis diese Anfrage bearbeitet wurde oder ein Fehler auftritt. Dieses Verhalten kann über das Flag `INTERNET_FLAG_ASYNC` modifiziert werden. Verwenden Sie dieses Flag, gibt eine Funktion auch dann umgehend die Steuerung zurück, wenn die Operation noch nicht beendet ist. Wurde eine Anfrage bearbeitet, empfängt die Anwendung Nachrichten über eine Rückruffunktion.

Dazu müssen Sie eine Klasse von `CInternetSession` ableiten und die Funktion `CInternetSession::OnStatusCallback` überschreiben. Um Rückrufe zu ermöglichen, lassen Sie die Funktion `CInternetSession::EnableStatusCallback` mit dem Parameter `TRUE` ausführen.

Komplexe Internet-Anwendungen führen mehrere Sitzungen mit unterschiedlichen Einstellungen gleichzeitig aus, um dem Anwender einen flexiblen Servicebereich anzubieten. Dazu erstellt eine Anwendung lediglich mehrere `CInternetSession`-Objekte.

41.3.2 Internet-Verbindungen – `CxxxConnection`

Sie müssen nun ein Verbindungsobjekt erstellen, um Daten über das Internet transferieren zu können. Die MFC bietet drei Verbindungsobjekte an:

- ➡ `CFtpConnection`,
- ➡ `CGopherConnection` und
- ➡ `CHttpConnection`.

Diese Objekte werden von der Klasse `CInternetConnection` abgeleitet, die über problemorientierte Konzepte einer Internet-Verbindung verfügt. Wie die Namen der Objekte bereits erkennen lassen, können diese für den Aufbau einer Verbindung zu FTP-, Gopher- und HTTP-Servern verwendet werden.

Die Objekte werden mit einem Aufruf der entsprechenden Elementfunktion in `CInternetSession` erzeugt und mit dem C++-Operator `delete` gelöscht.

FTP-Verbindungen

CFTPConnection FTP-Verbindungen werden mit Objekten vom Typ `CFTPConnection` hergestellt. Diese Objekte werden mit einem Aufruf von `CInternetSession::GetFtpConnection` erzeugt.

Eine manuell aufgebaute FTP-Sitzung erfordert die Angabe

- eines Benutzernamens (gewöhnlich »anonymous« oder »ftp«) und
- eines Paßworts (Ihre E-Mail-Adresse, wenn Sie sich anonym anmelden).

Diese beiden Parameter müssen der `CInternetSession::GetFtpConnection`-Funktion übergeben werden, um eine Verbindung zu einem FTP-Server herzustellen.

FTP-Verbindungsobjekte führen einige Aufgaben aus, die Sie gewöhnlich während einer manuellen Sitzung vornehmen.

Verzeichnisse bearbeiten Verzeichnisse können mit Hilfe der Funktionen `SetCurrentDirectory`, `GetCurrentDirectory`, `GetCurrentDirectoryAsURL`, `RemoveDirectory` und `CreateDirectory` bearbeitet werden.

Das Auflisten des Inhalts eines Verzeichnisses ist ein wichtiges FTP-Feature. FTP-Verzeichnisse werden in einer für uns lesbaren Form von dem Server zurückgegeben. Leider ist dieses Format nicht für alle Betriebssystemplattformen und Server-Implementierungen gleich. Anwendungen müßten daher komplexe Algorithmen zur Analyse der FTP-Verzeichnisse verwenden, wären diese nicht bereits in der Klasse `CFTPFileFind` implementiert. Möchten Sie sich den Inhalt eines Verzeichnisses anzeigen lassen, stellen Sie zunächst eine FTP-Verbindung her. Wechseln Sie anschließend zu dem gewünschten Verzeichnis, erstellen Sie ein `CFTPFileFind`-Objekt, und verwenden Sie dessen Elementfunktionen `FindFile` und `FindNextFile`.

Dateien bearbeiten Die Anwendung kann Dateien öffnen (`OpenFile`), umbenennen (`Rename`), löschen (`Remove`), vom Server herunterladen (`GetFile`) oder auf dem Server speichern (`PutFile`). Mit Hilfe der Elementfunktion `OpenFile` können Dateien als `CInternetFile`-Objekt geöffnet werden.

Verbindung schließen Sie schließen ein nicht mehr benötigtes `CFTPConnection`-Objekt mit einem Aufruf der Elementfunktion `Close`. Obwohl der Destruktor ebenfalls die Verbindung schließt, unterdrückt der Aufruf dieser Funktion die Anzeige eines auswertenden Nachrichtenfelds.

Gopher-Verbindungen

`CGopherConnection`-Objekte verwalten die Verbindungen zu Internet-Gopher-Servern. Sie bauen eine Gopher-Verbindung durch den Aufruf der Funktion `CInternetSession::GetGopherConnection` auf. *CGopher-Connection*

Beachten Sie bitte, daß die Syntax der `CInternetSession::GetGopherConnection`-Funktion die Angabe eines Server-Namens verlangt.

Ein Gopher-Server kann zwei unterschiedliche Datentypen an die Client-Anwendung übermitteln: Gopher-Menüs und Gopher-Dateien.

Gopher-Menüs bestehen aus mehreren Gopher-Locator-Elementen, die auf weitere Gopher-Menüs und Gopher-Dateien verweisen.

Gopher-Dateien können nicht von dem Server heruntergeladen oder bearbeitet werden.

`CGopherConnection`-Objekte verwenden eine Helferklasse für die Gopher-Locator-Elemente. Diese mit `CGopherLocator` bezeichnete Klasse verfügt über drei überladene Versionen ihres Konstruktors, so daß Locator-Objekte erstellt werden können, die auf unterschiedlichen Informationen basieren. *CGopherLocator*

Gopher-Menüs werden mit Hilfe der `CGopherFileFind`-Klasse analysiert. Die `FindFile`- und `FindNextFile`-Elementfunktionen dieser Klasse setzen ein `CGopherLocator`-Objekt auf den selektierten Gopher-Menüeintrag. Sie müssen Gopher-Locator-Objekte somit nicht manuell erstellen. *Menüs analysieren*

HTTP-Verbindungen

HTTP-Verbindungsobjekte vom Typ `CHttpConnection` repräsentieren Verbindungen zu einem World-Wide-Web-Server via HTTP. Abgesehen vom Konstruktor, enthält diese Klasse eine Elementfunktion mit der Bezeichnung `CHttpConnection::OpenRequest`. Die Funktion gibt einen Zeiger auf ein Objekt vom Typ `CHttpFile` zurück, der für den Transfer einer HTTP-Datei benötigt wird.

Zusätzlich zum Herunterladen von WWW-Inhalten unterstützt `CHttpConnection` das Web-Authoring. `CHttpConnection`-Objekte werden außerdem zum Einrichten von sicheren Verbindungen verwendet.

Objekte vom Typ `CHttpConnection` werden gewöhnlich mit einem Aufruf von `CInternetSession::GetHttpConnection` erstellt.

41.3.3 Internet-Dateien – CInternetFile

Der Internet-Dateitransfer wird in der MFC durch die Klasse `CInternetFile` und den davon abgeleiteten Klassen unterstützt. `CInternetFile` selbst wird von `CStdioFile` abgeleitet. Auf diese Weise können sehr viele MFC-Standard-Dateibearbeitungen für Internet-Dateiobjekte ausgeführt werden.

FTP-Dateien werden durch Objekte vom Typ `CInternetFile` repräsentiert. Die HTTP- und Gopher-Klassen `CHttpFile` und `CGopherFile` sind von `CInternetFile` abgeleitet.

41.3.4 Weitere Unterstützungsklassen

Andere Unterstützungsklassen erleichtern die effiziente Internet-Programmierung mit der MFC. Einige dieser Klassen wurden bereits in den vorherigen Absätzen erwähnt. Betrachten Sie die folgenden Abschnitte daher bitte als eine Zusammenfassung.

<i>CFindFile</i> <i>CGopherFindFile</i>	Die Suche nach Dateien auf FTP- oder Gopher-Servern geschieht mit Hilfe der von <code>CFindFile</code> abgeleiteten Klassen <code>CFindFile</code> und <code>CGopherFindFile</code> . Diese Klassen entbinden den Anwender von der manuellen Analyse der Inhalte von FTP- oder Gopher-Verzeichnissen. Dies gilt besonders für FTP-Verzeichnisse, deren Formate differieren können.
<i>CGopherLocator</i>	Gopher-Locator-Objekte verweisen auf Menüeinträge und Dateien auf Gopher-Servern. Ein Locator wird durch ein Objekt vom Typ <code>CGopherLocator</code> repräsentiert.
<i>CInternetException</i>	Internet-Klassen verwenden häufig MFC-Ausnahmen für die Anzeige von Fehlern. Spezifische Internet-Fehler werden durch die Klasse <code>CInternetException</code> repräsentiert. Da Verbindungsprobleme im Internet nicht selten sind, müssen die Anwendungen diese Fehler abfangen und analysieren, um unerwartete Verbindungsabbrüche zu vermeiden und dem Anwender die korrekten, angeforderten Daten zur Verfügung zu stellen.

41.4 MFC-Internet-Klassen in Anwendungen verwenden

In diesem Abschnitt werden Ihnen drei einfache Beispiele vorgestellt, die den Gebrauch der Internet-Klassen in FTP-, Gopher- und HTTP-Sitzungen demonstrieren.

41.4.1 Kommunikation mit einem FTP-Server

Das erste Beispiel (Listing 41.1) ist ein einfaches MFC-Programm, das eine Datei vom Microsoft-FTP-Server herunterlädt. Kompilieren Sie diese Konsolenanwendung mit der folgenden Kommandozeile:

```
CL /MT /GX FTPGET.CPP
```

```
#include <afxinet.h>
#include <iostream.h>

void main(void)
{
    CInternetSession is(_T("FTPGET"));
    CFtpConnection *pFC = NULL;
    CInternetFile *pIF = NULL;
    try
    {
        pFC = is.GetFtpConnection(_T("ftp.microsoft.com"));
        pIF = pFC->OpenFile(_T("disclaimer.txt"), GENERIC_READ,
                           FTP_TRANSFER_TYPE_ASCII);

        char c;
        while (pIF->Read(&c, 1) == 1) cout << c;
        pIF->Close();
        pFC->Close();
    }
    catch (CInternetException *pIE)
    {
        cout << "Internet error " << pIE->m_dwError << "."
              << endl;
    }
    delete pIF;
    delete pFC;
}
```

*Listing 41.1:
Herunterladen
einer Datei von
einem FTP-Server*

Die Ausführung beginnt mit der Erstellung eines `CInternetSession`-Objekts auf dem Stack. Der einzige Parameter, der dem Konstruktor übergeben wird, ist eine Zeichenfolge, die den Anwendungsnamen aufnimmt.

Der größte Abschnitt der `main`-Funktion ist durch einen `try...catch`-Block geschützt. Auf diese Weise wird gewährleistet, daß Internet-Fehler korrekt abgefangen und dem Anwender gemeldet werden.

Nachdem die Internet-Verbindung hergestellt wurde, muß eine Verbindung zu dem gewünschten FTP-Server aufgebaut werden. Dies ge-

schieht mit einem Aufruf von `CInternetSession::GetFtpConnection`. Wird diese Funktion erfolgreich ausgeführt, gibt sie einen Zeiger auf ein `CFtpConnection`-Objekt zurück. Anschließend wird die angegebene Datei mit Hilfe der Funktion `CFtpConnection::OpenFile` zum Lesen geöffnet. Die Funktion gibt einen Zeiger auf ein Objekt vom Typ `CInternetFile` zurück.

Die Datei wird mit `CStdioFile`-Funktionen eingelesen. Der Inhalt der Datei nimmt den Weg über die Standardausgabe. Nachdem das Einlesen beendet ist, werden sowohl das `CInternetFile`-Objekt als auch das `CInternetConnection`-Objekt über die entsprechenden `Close`-Elementfunktionen geschlossen und im Anschluß daran gelöscht.

41.4.2 Kommunikation mit einem Gopher-Server

Das zweite Beispiel (Listing 41.2) lädt eine Datei vom Gopher-Server der Universität von Minnesota. Die Struktur des Programms entspricht weitgehend der des FTP-Beispiels. Kompilieren Sie die Anwendung durch:

```
CL /MT /GX GOPHRGET.CPP
```

*Listing 41.2:
Laden einer Datei
von einem
Gopher-Server*

```
#include <afxinet.h>
#include <iostream.h>

void main(void)
{
    CInternetSession is("GOPHRGET");
    CGopherConnection *pGC = NULL;
    CGopherFile *pGF = NULL;
    try
    {
        pGC = is.GetGopherConnection(_T("localhost"));
        pGF = pGC->OpenFile(
            pGC->CreateLocator("0\t\tgopher2.tc.umn.edu\t70"));
        char c;
        while (pGF->Read(&c, 1) == 1) cout << c;
        pGF->Close();
        pGC->Close();
    }
    catch (CInternetException *pIE)
    {
        cout << "Internet error " << pIE->m_dwError << "."
            << endl;
    }
    delete pGF;
    delete pGC;
}
```

Zunächst wird ein `CInternetSession`-Objekt, dann ein `CGopherConnection`-Objekt und schließlich ein `CGopherFile`-Objekt erstellt.

Das `CGopherFile`-Objekt wird mit `CGopherConnection::OpenFile` erzeugt. Der dieser Funktion übergebene Parameter ist ein Gopher-Locator, der mit einem Aufruf von `CGopherConnection::CreateLocator` generiert

wird. Dazu wird der Funktion eine Locator-Zeichenfolge übergeben. In Ihrer Anwendung können Sie die `CGopherFindFile`-Klasse verwenden, um die Inhalte der Gopher-Menüs aufzuführen und einen Locator für den selektierten Menüeintrag zu erstellen.

41.4.3 Kommunikation mit einem HTTP-Server

Unser drittes Beispiel (Listing 41.3) stellt eine Verbindung zu einem World-Wide-Web-Server von Microsoft her und lädt von diesem die Hauptseite `DEFAULT.ASP`. Kompilieren Sie das Programm durch die Kommandozeile:

```
CL /MT /GX HTTPGET.CPP
```

```
#include <afxinet.h>
#include <iostream.h>

void main(void)
{
    CInternetSession is("HTTPGET");
    CHttpConnection *pHC = NULL;
    CHttpFile *pHF = NULL;
    try
    {
        pHC = is.GetHttpConnection(_T("www.microsoft.com"));
        pHF = pHC->OpenRequest(_T(""), _T("/default.asp"), NULL,
                               0, NULL, NULL, 0);

        pHF->SendRequest();
        char c;
        while (pHF->Read(&c, 1) == 1) cout << c;
        pHF->Close();
        pHC->Close();
    }
    catch (CInternetException *pIE)
    {
        cout << "Internet error " << pIE->m_dwError << ". "
              << endl;
    }
    delete pHF;
    delete pHC;
}
```

*Listing 41.3:
Laden einer Datei
von einem
HTTP-Server*

Auch hier beginnt die Programmausführung mit der Erstellung eines `CInternetSession`-Objekts. Anschließend wird eine HTTP-Verbindung hergestellt. Das Laden der Datei geschieht in zwei Schritten. Zunächst wird ein Objekt vom Typ `CHttpFile` mit einem Aufruf von `CHttpConnection::OpenRequest` generiert. Die Anfrage wird mit einem Aufruf von `CHttpFile::SendRequest` versendet.

Die MFC-Internet-Klassen unterstützen nicht direkt das Auswerten von HTML-Dokumenten. Keine Funktion bietet die Möglichkeit, die in einem Dokument angezeigten URLs zu bearbeiten. Möchten Sie die Auswahl einer Verknüpfung zu einer Internetseite in Ihrer Anwendung bearbeiten, müssen Sie die HTML-Dokumente selbst auswerten und die darin enthaltenen URLs extrahieren.

41.5 Zusammenfassung

Die WinInet-Bibliothek von Microsoft stellt Funktionen zur Verfügung, die FTP-, Gopher- und HTTP-Verbindungen implementieren. Die MFC kapselt die WinInet-Bibliothek in Form von MFC-Internet-Unterstützungsklassen. Auf diese Weise wird die MFC um die Low-Level-Unterstützung ergänzt, die bisher lediglich für die WinSock-Bibliothek zur Verfügung stand.

Der Datentransfer zwischen der Client-Anwendung und dem Internet-Host geschieht in drei Schritten:

- ➔ Zunächst wird eine Internet-Sitzung erstellt,
- ➔ dann eine Internetverbindung aufgebaut und
- ➔ schließlich die angegebene Datei für den Transfer geöffnet.

Internet-Sitzungen werden durch die `CInternetSession`-Klasse repräsentiert. Eine Anwendung kann mehrere dieser Objekte erstellen, die unterschiedliche Eigenschaften besitzen.

Die Verbindungen zu FTP-, Gopher- und HTTP-Servern werden durch Objekte vom Typ

- ➔ `CFtpConnection`,
- ➔ `CGopherConnection` und
- ➔ `CHttpConnection` repräsentiert.

Diese Objekte werden niemals direkt, sondern von Helferfunktionen in `CInternetSession` erstellt.

Ein Verbindungsobjekt wird geschlossen, wenn es gelöscht wird. Ein expliziter Aufruf der `Close`-Elementfunktion unterdrückt jedoch die Anzeige eines auswertenden Nachrichtenfelds.

Der Datentransfer zwischen dem Client und dem Internet-Server wird mit Hilfe der `CInternetFile`-Klasse und den davon abgeleiteten Klassen `CGopherFile` und `CHttpFile` vollzogen. Diese Klassen ermöglichen die Verwendung der Standard-Dateifunktionalität der MFC-Bibliothek.

Weitere Unterstützungsklassen werten FTP- und Gopher-Verzeichnisse aus, verwalten Gopher-Locator-Objekte und bearbeiten Internet-Fehler.

Nachrichtenfähige Anwendungen mit MAPI

Kapitel

42



Das Entwickeln nachrichtenfähiger Anwendungen ist nicht länger die Aufgabe einiger weniger Programmierer, die an unternehmensweiten Projekten zur Systemintegration arbeiten. Microsoft Exchange und MAPI-Dienste sind nun auf jedem Windows-95/98-Rechner verfügbar, und mit dem gegenwärtigen phänomenalen Wachstum des Internet verlangen immer mehr Anwender nach Nachrichtenfähigkeiten in Endanwendungen. Die für Unternehmensnetze gestern entwickelten Lösungen werden heute auf Heimcomputern eingesetzt. Falls diese Behauptung übertrieben scheinen sollte, bedenken Sie, daß ich die Seiten dieses Manuskripts einsende, indem ich sie mit der Maus auf das Microsoft-Exchange-Icon ziehe und loslasse.

Microsoft hat diese Tatsache erkannt, als entschieden wurde, den Anforderungen im Windows-95-Logo-Programm einen gewissen Umfang an MAPI-Unterstützung hinzuzufügen. Damit Anwendungen unter diesem Programm qualifiziert werden, müssen sie nun als Minimum einen Sendebefehl unterstützen, der sie in die Lage versetzt, ein Dokument zu einem MAPI-Behälter zu senden. (Offensichtlich ist eine MAPI-Unterstützung nicht erforderlich, wenn Ihre Anwendung ein ganz anderes Konzept hat, wie z. B. im Falle eines Spielprogramms.)

Das Akronym MAPI steht für Messaging Applications Programming Interface. Doch was genau ist MAPI? Wie ist MAPI implementiert? Und, noch wichtiger, wie können Anwendungen MAPI-Dienste am effizientesten nutzen? Dies sind die Fragen, auf die ich in diesem Kapitel einzugehen versuche.

Seien Sie sich bewußt, daß das Ziel dieses Kapitels nicht eine Einführung in MAPI auf dem für die Entwicklung von MAPI-Anbietern nötigen Niveau ist. Ein solches Ziel würde ein ganzes Buch erfordern. Statt

dessen versuche ich, eine Übersicht zu MAPI als Anwendungsschnittstelle zu bieten, mit dem Schwerpunkt Einsatz von MAPI in MFC-Anwendungen.



Die MAPI-Architektur, die ich in diesem Kapitel beschreibe, entspricht MAPI in Windows 95 oder in Windows NT Version 3.51 oder später. Frühere Versionen von Microsoft Mail, obwohl sie eingeschränkte Unterstützung für Simple MAPI und CMC bieten, unterstützen weder in vollem Umfang Erweitertes MAPI, noch sind sie mit der in diesem Kapitel beschriebenen Architektur kompatibel.

42.1 Die Architektur von MAPI

Wofür steht die MAPI? Was genau ist MAPI? Ist es der Microsoft Exchange Client, der später in Windows Messaging Client umbenannt wurde (um die Annahme, daß es sich bloß um einen Client zu Microsofts Exchange-Server-Mailsystem handelt, zu entkräften) und mit Windows 95 oder Windows NT 4.0 geliefert wird? Ist es der vieldiskutierte Exchange Server? Ist es der MAPI-Spooler, der still im Hintergrund läuft, wann immer Exchange gestartet wird?

Die Antwort: keines von alledem. MAPI ist keine Anwendung, keine DLL und kein Systemdienst; es besteht vielmehr aus einer Reihe von Spezifikationen. Obwohl Microsoft der Hauptanbieter von MAPI-Komponenten ist, ist es nicht nötig, auch nur eine einzige Microsoft-Komponente zu besitzen, um MAPI zu fahren (es ist auch nicht nötig, eine Windows- oder Win32-Plattform zu verwenden). Sie können ein MAPI-kompatibles System haben, mit Nachrichtenspeicher, Adreßbuch und Transportanbietern von Drittanbietern auf einem anderen Betriebssystem als Windows. MAPI ist ein Teil von dem, was Microsoft WOSA nennt – Windows Open Services Architecture – und aus einem gemeinsamen Satz von APIs zum verteilten Rechnen besteht.

MAPI kann eigentlich als zwei unabhängige Sätze von APIs gesehen werden, die Client-Anwendungen auf der einen Seite mit Dienstanbietern auf der anderen verbinden (Abbildung 42.1).

Wozu mehrere APIs? Die Gründe sind zum Teil geschichtlich. Zuerst hatte Microsoft die Spezifikationen für Simple MAPI herausgegeben, die ungefähr ein Dutzend grundlegende Aufrufe enthielten, um Anwendungen den Einsatz von Nachrichten zu ermöglichen. Die CMC-API (Common Messaging Calls) wurde als plattformunabhängiger Ersatz für Simple MAPI entwickelt.

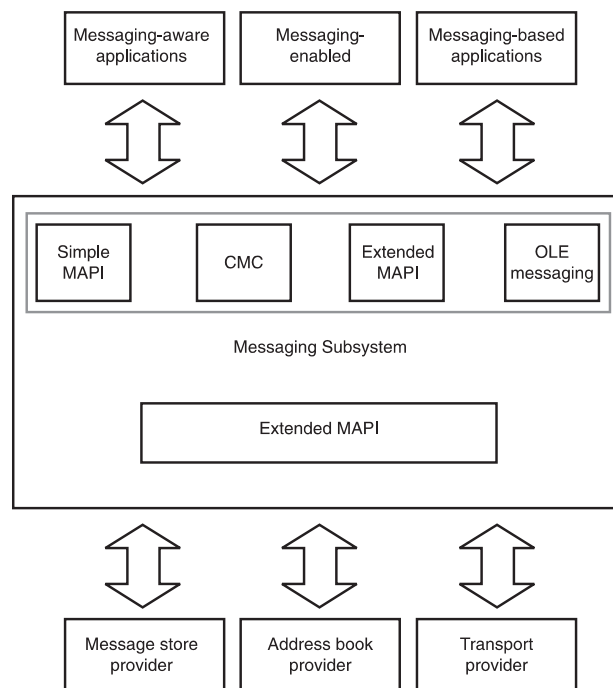


Abbildung 42.1:
Die MAPI-Architektur

Diese enthält auch um die zehn grundlegende Aufrufe, die den Zugang zu Nachrichtendiensten bieten. Demgegenüber ist Erweitertes MAPI eine umfangreiche, komplexe und sich weiterentwickelnde Spezifikation, die nicht nur für Client-Anwendungen eine Programmierschnittstelle bietet, sondern auch für Dienstanbieter. OLE-Nachrichten und eine API, speziell für Visual Basic und Visual C++-Anwendungen entwickelt, verwenden die OLE-Automation-Architektur.

Der nächste Abschnitt wirft einen detaillierteren Blick auf die MAPI-Architektur.

42.1.1 Typen von MAPI-Unterstützung

Anwendungen können drei getrennte Ebenen der MAPI-Unterstützung bieten.

Die einfachsten MAPI-Anwendungen sind NACHRICHTENBEWUSSTE Anwendungen. Diese Anwendungen sind nicht auf das Vorhandensein von MAPI angewiesen, um ihre Funktionen auszuführen. Sie bieten bloß eine vereinfachte MAPI-Funktionalität. Das Windows-95-WordPad, das in seinem Dateimenü einen Sendebefehl hat, ist ein gutes Beispiel für eine nachrichtenbewußte Anwendung.

Nachrichtenbewußte Anwendungen

Nachrichtenaktivierte Anwendungen Demgegenüber erfordern NACHRICHTENAKTIVIERTE Anwendungen das Vorhandensein von MAPI, um in vollem Umfang zu funktionieren. Obwohl sie auch funktionieren, wenn MAPI nicht vorhanden ist, können sie unter diesen Umständen nicht die gesamte Palette ihrer Dienste anbieten. Ein perfektes Beispiel für eine nachrichtenaktivierte Anwendung ist Microsoft Outlook. Während dieses Programm auch als eigenständiger persönlicher Informationsverwalter eingesetzt werden kann, haben viele seiner Fähigkeiten keinen Sinn, wenn es auf einem System läuft, das MAPI nicht installiert hat.

Nachrichtenbasierte Anwendungen An der Spitze der Hierarchie stehen NACHRICHTENBASIERTE Anwendungen. Das Ausführen dieser Anwendungen erfordert das Vorhandensein der gesamten Palette von MAPI-Diensten (Nachrichtenspeicher, Adreßbuch, Übertragung). Diese Anwendungen werden oft auch als Workgroup-Anwendungen bezeichnet. Ein gutes Beispiel für eine Workgroup-Anwendung ist der Microsoft Exchange Client; ich werde ihnen die Erklärung ersparen, warum diese Anwendung nicht auf einem eigenständigen System ohne MAPI-Unterstützung laufen kann.

Der nächste Abschnitt enthält eine kurze Tour durch die MAPI-Dienstanbieter. Dann können wir untersuchen, wie die verschiedenen APIs in Anwendungen, die Unterstützung auf diesen verschiedenen Ebenen bieten, verwendet werden.

42.1.2 Dienstanbieter

MAPI-Dienstanbieter sind die Komponenten, die insgesamt die MAPI-Dienste auf einem System implementieren. Es gibt drei Arten von Dienstanbietern:

- ➔ Adreßbücher,
- ➔ Nachrichtenspeicher und
- ➔ Übertragungswege.

Adreßbücher enthalten eine oder mehrere Listen von Behältern. Ein Adreßbuchanbieter implementiert eine Reihe von Schnittstellen, durch die Nachrichtenanwendungen oder weitere Anbieter Zugang zu Adressen oder Listen von Adressen erhalten.

Nachrichtenspeicher sind hierarchische Ablagen für MAPI-Nachrichten. Eine Nachricht besteht aus einer Vielzahl von Eigenschaften wie der Absender der Nachricht, der Behälter, das Datum, Betreff, Rumpf der Nachricht und viele andere. Anbieter von Nachrichtenspeichern implementieren irgendeine Form der dauerhaften Speicherung von Nachrichten (zum Beispiel eine lokale Datei auf Platte) und bieten eine Reihe von Schnittstellen, durch die

Anwendungen oder andere Anbieter Nachrichten im Speicher zählen, bestimmte Eigenschaften oder Nachrichten abrufen können.

Die Anbieter von Übertragungswegen implementieren die Verbindung zwischen dem lokalen und dem entfernten System. Übertragungsanbieter übernehmen ausgehende Nachrichten, stellen Verbindungen zu entfernten Systemen her und übertragen die Nachrichten in einem Format, das vom entfernten System verstanden wird. Übertragungsanbieter übernehmen auch ankommende Nachrichten vom entfernten System, wandeln diese nach Bedarf in MAPI-Objekte um und plazieren sie in den lokalen Nachrichtenspeicher. Ein gutes Beispiel eines Übertragungsanbieters ist der Microsoft-Internet-Übertragungsanbieter, der mit Hilfe von Windows Sockets und SMTP (Simple Mail Transfer Protocol) für ausgehende bzw. POP (Post Office Protocol) für ankommende Nachrichten Verbindungen mit entfernten Systemen aufbaut. Ausgehende Nachrichten werden in eine lesbare ASCII-Form umgewandelt, ankommende aus dieser Form in MAPI-Nachrichtenobjekte.

Alle Anbieter sind in Form von DLLs vertreten. Der Satz von DLLs, der für eine MAPI-Session geladen werden muß, wird durch MAPI-Profil bestimmt.

42.1.3 MAPI-Profil

Den »Kleber«, der die MAPI-Komponenten zusammenhält, stellt der MAPI-Spooler dar. Er ist ein eigenständiger Prozeß, der das Senden und Empfangen von Nachrichten ermöglicht. Typischerweise ist es der MAPI-Spooler, der abgeschickte Nachrichten an den Übertragungsanbieter weiterreicht und von ihm ankommende Nachrichten übernimmt. Der MAPI-Spooler implementiert eine Architektur zum Speichern und Weiterreichen, das heißt, abgeschickte Nachrichten werden an den entsprechenden Übertragungsanbieter »gespoolt«, sobald dieser verfügbar wird. Diese Fähigkeit ist auf Systemen mit großem Nachrichtenvolumen und zahlreichen entfernten Verbindungen lebenswichtig.

Der MAPI-Spooler

Eigentlich ist es ebenfalls der MAPI-Spooler, der für das Laden und Initialisieren der Dienstanbieter verantwortlich ist. Doch woher weiß der Spooler, welche Anbieter er laden soll?

Die Antwort ist das MAPI-Profil. MAPI-Profile sind Einträge in der Registrierdatenbank, die die aktuelle MAPI-Konfiguration bestimmen.

Das MAPI-Profil

Diese Einträge in der Registrierdatenbank können unter folgendem Schlüssel gefunden werden:

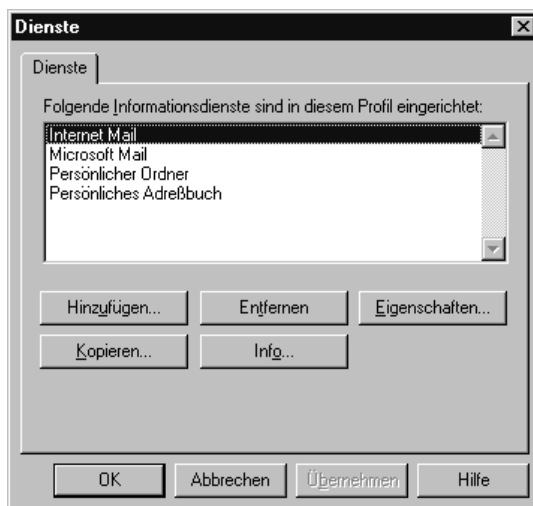
```
HKEY_CURRENT_USER\
  Software\
    Microsoft\
      Windows Messaging Subsystem\
        Profiles
```



Beachten Sie, daß die Einträge in der Registratur für die MAPI-Profil nicht manuell editiert werden sollten.

Für einen Anwender können mehrere Profile definiert sein. Jedes Profil beschreibt eine Reihe von Dienstanbietern und ihre Funktionsparameter. Zum Beispiel kann ein typisches Profil Microsoft Personal Address Book (Adreßbuchanbieter), Microsoft Fax (Übertragungsanbieter) und das Microsoft Network Online Service (Adreßbuch und Übertragung) enthalten. MAPI-Profile können mit dem Microsoft Exchange Client über den Befehl DIENSTE aus dem Menü TOOLS bearbeitet werden.

Abbildung 42.2:
Editieren eines
MAPI-Profiles



42.2 MAPI-APIs

Die verschiedenen APIs unterstützen verschiedene Arten von MAPI-Anwendungen.

Simple MAPI und CMC Nachrichterbewußte und nachrichtenaktivierte Anwendungen können sowohl Simple MAPI als auch CMC verwenden. Der Hauptvorteil beim Einsatz von CMC ist die völlige Plattformunabhängigkeit. Es wird empfohlen, daß neue Anwendungen CMC statt des älteren, plattformabhängigen Simple MAPI verwenden.

Nachrichtenbasierte Anwendungen (um von Dienst Anbietern ganz zu schweigen) erfordern Zugang zum vollständigen Satz von MAPI und sollten somit Erweitertes MAPI verwenden.

Erweitertes MAPI

Die OLE-Nachrichten werden in Visual Basic und Visual C++-Anwendungen eingesetzt. Ihre API ist reichhaltiger als CMC oder Simple MAPI, steht aber hinter Erweitertem MAPI zurück.

OLE-Nachrichten

42.2.1 Simple MAPI

Simple MAPI bietet eine Reihe von Funktionen, die es Anwendungen ermöglichen, eine MAPI-Sitzung zu starten, Nachrichtenaustauschfunktionen auszuführen und die Sitzung zu schließen.

Der einfachste Weg, Simple MAPI einzusetzen, führt über die Funktion `MAPI SendDocuments`. Diese Funktion kann zum Erzeugen einer Standard-MAPI-Nachricht mit einer oder mehreren angehängten Dateien eingesetzt werden. `MAPI SendDocuments` zeigt immer einen Dialog an, in dem der Anwender Behälter, Sendeoptionen und den Nachrichtentext angeben kann.

*MAPI Send-
Documents*

Dies wird vom Programm in Listing 42.1 veranschaulicht. Dieses Programm sendet eine Nachricht mit Hilfe von `MAPI SendDocuments`, mit der Datei `c:\autoexec.bat` darin eingebettet. Sie können dieses Programm von der Kommandozeile aus kompilieren:

```
CL SENDMSG.C USER32.LIB
```

```
#include <windows.h>
#include <map.h>

LPMAPISENDDOCUMENTS lpfnMAPI SendDocuments;

void SendMsg(HWND hwnd)
{
    (*lpfnMAPI SendDocuments)((ULONG)hwnd, ";",
                              "C:\\AUTOEXEC.BAT",
                              "AUTOEXEC.BAT", 0);
    MessageBox(hwnd, "Message sent", "", MB_OK);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_LBUTTONDOWN:
            SendMsg(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
}
```

*Listing 42.1:
Der Einsatz von
MAPI Send-
Documents*

```

    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;
    HANDLE hMAPILib;

    hMAPILib = LoadLibrary("MAPI32.DLL");
    lpfnMAPISendDocuments = (LPMAPISENDDOCUMENTS)GetProcAddress(
        hMAPILib, "MAPISendDocuments");

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HELLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("HELLO", "HELLO",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                       NULL, NULL, hInstance, NULL);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    FreeLibrary(hMAPILib);
    return msg.wParam;
}

```

MAPISendMail Ein viel flexiblerer Weg zum Versand von Nachrichten führt über MAPI-SendMail. Durch eine Reihe von Strukturen können Sie den Behälter und den Inhalt der Nachricht bestimmen. MAPISendMail macht es möglich, Mails zu versenden, ohne irgendeine Benutzerschnittstelle auszugeben. Somit kann MAPISendMail zum Beispiel auch von Kommandozeilenanwendungen eingesetzt werden.

MAPISendMail wird typischerweise innerhalb einer MAPI-Sitzung, die mit einem Aufruf von MAPILogon begonnen und mit MAPILogoff beendet wird, verwendet.

Dieser Einsatz von MAPISendMail wird im Programm aus Listing 42.2 gezeigt, das eine einfache Textnachricht nach `president@whitehouse.gov` sendet, ohne eine Anwenderschnittstelle auszugeben.



Aus Rücksicht auf die Nerven der Mitarbeiter von Herrn Clinton ändern Sie bitte die Adresse, bevor Sie mit dem Programm experimentieren!

Dieses Programm kann ohne irgendwelche Flags oder Bibliotheken von der Kommandozeile aus kompiliert werden:

CL CMDMSG.C

```
#include <windows.h>
#include <stdio.h>
#include <mapi.h>

LPMAPILOGON lpfnMAPILogon;
LPMAPISENDMAIL lpfnMAPISendMail;
LPMAPILOGOFF lpfnMAPILogoff;

MapiRecipDesc recipient =
{
    0, MAPI_TO,
    "Bill Clinton", "SMTP:president@whitehouse.gov",
    0, NULL
};

MapiMessage message =
{
    0, "Greetings",
    "Hello, Mr. President!\n",
    NULL, NULL, NULL, 0, NULL, 1, &recipient, 0, NULL
};

void main(void)
{
    LHANDLE lhSession;
    HANDLE hMAPILib;

    hMAPILib = LoadLibrary("MAPI32.DLL");
    lpfnMAPILogon =
        (LPMAPILOGON)GetProcAddress(hMAPILib, "MAPILogon");
    lpfnMAPISendMail =
        (LPMAPISENDMAIL)GetProcAddress(hMAPILib, "MAPISendMail");
    lpfnMAPILogoff =
        (LPMAPILOGOFF)GetProcAddress(hMAPILib, "MAPILogoff");

    (*lpfnMAPILogon)(0, NULL, NULL, 0, 0, &lhSession);
    (*lpfnMAPISendMail)(lhSession, 0, &message, 0, 0);
    (*lpfnMAPILogoff)(lhSession, 0, 0, 0);
    printf("Message to the White House sent.\n");

    FreeLibrary(hMAPILib);
}
```

*Listing 42.2:
Einsatz von Simple MAPI in einer Konsolenanwendung*

Simple MAPI kann auch verwendet werden, um ankommende Nachrichten zu verarbeiten. Aufrufe wie `MAPIFindNext` und `MAPIReadMail` können zum Beispiel zum Untersuchen neuer Nachrichten aus der Inbox des Users verwendet werden.

*MAPIFindNext
MAPIReadMail*

Simple MAPI bietet auch eine Anwenderschnittstelle zur Eingabe von Adressen. Statt `MAPISendDocuments` zu verwenden, setzen Sie den Aufruf von `MAPIAddress` ein, um den User in die Lage zu versetzen, Adressen zu wählen und einzugeben.

MAPIAddress

42.2.2 CMC – Common Messaging Calls

Common Messaging Calls sind eine Implementation der API der X.400 API Association. Sie bieten eine Reihe von Nachrichtenfunktionen der höheren Ebene, die ähnlich zu Simple MAPI eingesetzt werden.

Die Tabelle 42.1 vergleicht die Funktionen von CMC mit Simple MAPI.

Tabelle 42.1:
CMC und
Simple MAPI

CMC	Simple MAPI	Beschreibung
cmc_logon	MAPILogon	Anmelden beim Dienst
cmc_logoff	MAPILogoff	Abmelden vom Dienst
cmc_free	MAPIFreeBuffer	Freigabe allokierten Speichers
cmc_send	MAPISendMail	Sende eine Nachricht
cmc_send_documents	MAPISendDocuments	Sende Dateien in einer Nachricht
cmc_list	MAPIFindNext	Finde Nachrichten
cmc_read	MAPIReadMail	Lese Nachrichten
cmc_act_on	MAPISaveMail	Speichere oder lösche Nachrichten
	MAPIDeleteMail	
cmc_look_up	MAPIAddress MAPIDetails MAPIResolveName	
cmc_query_configuration	N/A	CMC-Konfigurationsdaten

Das Programm in Listing 42.3 veranschaulicht den Einsatz der CMC-Aufrufe aus einer Konsolenanwendung heraus. Sie können dieses Programm von der Kommandozeile aus kompilieren:

```
CL CMCMSG.C
```

Listing 42.3:
CMC in einer
Konsolenanwendung

```
#include <windows.h>
#include <xcmc.h>

typedef CMC_return_code (FAR PASCAL *LPFNCMCLOGON)(CMC_string,
    CMC_string, CMC_string, CMC_enum, CMC_ui_id, CMC_uint16,
    CMC_flags, CMC_session_id FAR *, CMC_extension FAR *);

typedef CMC_return_code (FAR PASCAL *LPFNCMCSEND)(CMC_session_id,
    CMC_message FAR *, CMC_flags, CMC_ui_id, CMC_extension FAR *);
```

```

typedef CMC_return_code (FAR PASCAL *LPFNCMCLOGOFF)(CMC_session_id,
    CMC_ui_id, CMC_flags, CMC_extension FAR *);

LPFNCMCLOGON lpfnCMCLogon;
LPFNCMCSEND lpfnCMCSend;
LPFNCMCLOGOFF lpfnCMCLogoff;

CMC_recipient recipient =
{
    "Bill Clinton", CMC_TYPE_INDIVIDUAL,
    "SMTP:president@whitehouse.gov", CMC_ROLE_TO,
    CMC_RECIP_LAST_ELEMENT, NULL
};

CMC_message message =
{
    NULL, "CMC: IPM", "Greetings", {0,0,0,0,0,0,0,0,0},
    "Hello, Mr. President!\n", &recipient, NULL,
    CMC_MSG_LAST_ELEMENT, NULL
};

void main(void)
{
    char msg[1000];
    CMC_session_id session;
    CMC_return_code retcode;
    HANDLE hMAPILib;

    hMAPILib = LoadLibrary("MAPI32.DLL");
    lpfnCMCLogon =
        (LPFNCMCLOGON)GetProcAddress(hMAPILib, "cmc_logon");
    lpfnCMCSend =
        (LPFNCMCSEND)GetProcAddress(hMAPILib, "cmc_send");
    lpfnCMCLogoff =
        (LPFNCMCLOGOFF)GetProcAddress(hMAPILib, "cmc_logoff");

    (*lpfnCMCLogon)(NULL, NULL, NULL, (CMC_enum)0, 0, 100,
        CMC_ERROR_UI_ALLOWED | CMC_LOGON_UI_ALLOWED, &session,
        NULL);
    (*lpfnCMCSend)(session, &message, 0, 0, NULL);
    (*lpfnCMCLogoff)(session, 0,
        CMC_ERROR_UI_ALLOWED | CMC_LOGOFF_UI_ALLOWED, NULL);

    printf("Message to the White House sent.\n");

    FreeLibrary(hMAPILib);
}

```

42.2.3 Erweitertes MAPI

Das Erweiterte MAPI ist eine umfangreiche, komplexe objektorientierte Programmierschnittstelle zu allen Aspekten von MAPI. Sein Programmiermodell beruht auf dem Component Object Model (COM).

Das erweiterte MAPI definiert eine Reihe von Objekttypen, einschließlich Nachrichten, Nachrichtenspeicher, Ordner, Mail-User, Adreßbücher, Anbieter, Sessions, und viele weitere Objekte.

MAPI-Objekte werden durch eine Reihe von Eigenschaften bestimmt, die ihrerseits als Objekte implementiert sind.

Der Zugriff auf MAPI-Objekte geschieht über Schnittstellen, die von der COM-Schnittstelle `IUnknown` abgeleitet sind. Zum Beispiel implementieren Eigenschaftsobjekte die Schnittstelle `IMAPIProp`, während Nachrichtenobjekte die `IMessage`-Schnittstelle implementieren.

Erweiterte MAPI-Programmierung bedeutet Manipulieren von MAPI-Objekten über ihre Eigenschaften und Methoden und über Benachrichtigungen über Ereignisse. MAPI-Eigenschaften können über Eigenschaftsobjekte manipuliert werden sowie über die Schnittstelle `IMAPIProp` oder über Eigenschaftstabellen. Die Methoden sind die aus `IUnknown` abgeleiteten Schnittstellen, die von den MAPI-Objekten zur Verfügung gestellt werden. Benachrichtigungen über Ereignisse bilden den Mechanismus zur Kommunikation zwischen MAPI-Objekten.

Erweitertes MAPI kann zur Implementierung nachrichtenbasierter Anwendungen verwendet werden, die eine genauere Kontrolle über die Nachrichtendienste erfordern, als sie Simple MAPI oder CMC bieten. Erweitertes MAPI kann auch für Verwaltungsaufgaben eingesetzt werden, wie das Wählen und Konfigurieren der Dienste und Erzeugen von Anwenderprofilen.

Ein weiteres Einsatzgebiet des erweiterten MAPI ist das Programmieren mit MAPI-Formen. MAPI-Formen sind spezielle Nachrichtentypen, die über eine Anwenderschnittstelle und einen Form-Server implementiert sind und die Wechselbeziehung des Anwenders mit der Formnachricht verwalten.

Erweitertes MAPI kann auch zum Erweitern des Microsoft-Clients eingesetzt werden. Kundenspezifische Versionen des Microsoft-Exchange-Clients können anwendungsspezifische Ansichten von Adreßbüchern, Nachrichtenspeichern und -objekten bieten sowie der Anwendung neue Such-, Adressierungs- und Workgroup-Features hinzufügen.

Schließlich ist das erweiterte MAPI auch die Programmierschnittstelle zum Erzeugen von MAPI-Dienstanbietern. Diese umfassen Adreßbücher, Nachrichtenspeicher und Übertragungswege.

42.2.4 OLE-Nachrichten

OLE-Nachrichten machen MAPI-Objekte über die Automatisierungsschnittstelle zugänglich. Das funktioniert am besten mit Automatisierungs-Clients wie Visual-Basic-Anwendungen oder Programmen, die in Visual Basic for Applications (VBA) geschrieben sind (das sind Microsoft Excel-, Access- oder Project-Programme).

Abbildung 42.3 veranschaulicht die Hierarchie der Automatisierungsobjekte, die über OLE-Nachrichten angesprochen werden.

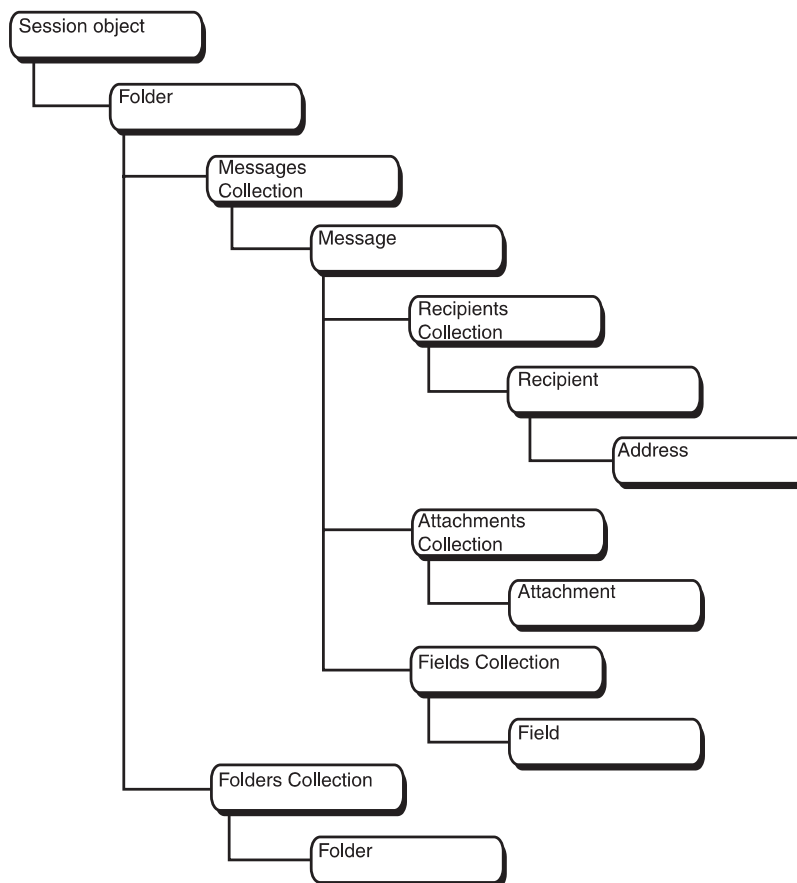


Abbildung 42.3:
OLE-Nachrichtenobjekte

Das Vorführen der OLE-Nachrichten in Visual C++ ist schwierig, da es das Erstellen einer Automatisierungs-Client-Anwendung erfordert. Um eine einfache OLE-Nachrichtensitzung zu zeigen, bin ich auf Visual Basic ausgewichen. Das Beispiel in Listing 42.4 zeigt, wie eine Visual-Basic-Anwendung eine einfache Nachricht senden kann, ohne eine Anwenderschnittstelle auszugeben.

```

Sub Command1_Click ()
    Dim objSession As Object
    Dim objMessage As Object
    Dim objRecip As Object

    Set objSession = CreateObject("MAPI.Session")
    objSession.Logon "", "", False, False

    Set objMessage = objSession.Outbox.Messages.Add
  
```

Listing 42.4:
OLE-Nachrichtenbeispiel in Visual Basic

```
objMessage.Subject = "Greetings"
objMessage.Text = "Hello again, Mr. President!"

Set objRecip = objMessage.Recipients.Add
objRecip.Name = "Bill Clinton"
objRecip.Address = "SMTP:president@whitehouse.gov"
objRecip.Type = 1
objRecip.Resolve (False)

objMessage.Send True, False
MsgBox "Message to the White House sent"
objSession.Logoff
Exit Sub
```

End Sub

Um dieses einfache Beispiel zu testen, weisen Sie den Code einer Schaltfläche in einem Visual-Basic-Form zu, führen Sie das Programm aus, und klicken Sie auf den Schalter.



Vergessen Sie ja nicht, die Adresse zu ändern, bevor Sie diesen Code testen!



Beachten Sie bitte, daß dieser Test nur funktionieren kann, wenn Sie den OLE-Nachrichtendienst auf Ihrem Rechner installiert haben. Die Komponenten des OLE-Nachrichtendienstes sind zur Zeit nur als Teil des MAPI-SDK verfügbar, das mit dem Microsoft -evel-oper-Network-Level-2-Abonnement geliefert wird.

42.3 MAPI-Unterstützung in MFC

Die Microsoft-Foundation-Classes-Bibliothek bietet MAPI-Unterstützung durch eine Reihe von Elementfunktionen in `CDocument`. Diese Unterstützung und wie sie in MFC-Rahmenanwendungen eingebaut ist, ist das Thema des letzten Teils dieses Kapitels.

42.3.1 MAPI-Unterstützung in `CDocument`

OnFileSendMail Die Klasse `CDocument` unterstützt MAPI durch die Elementfunktion `OnFileSendMail`.

`CDocument::OnFileSendMail` serialisiert das Dokument in eine temporäre Datei und ruft dann Funktionen von Simple MAPI auf, um eine Nachricht vorzubereiten und zu senden – mit der Datei als Anhang.

Eine Variante von `CDocument::OnFileSendMail` ist `COleDocument::OnFileSendMail`; diese Variante behandelt Compound-Dateien.

Die Elementfunktion `CDocument::OnUpdateFileSendMail` bestimmt, ob MAPI-Unterstützung auf dem System verfügbar ist und aktualisiert entsprechend Befehlsobjekte über ein `CCmdUI`-Objekt.

42.3.2 MAPI und Anwendungsassistent

Der Anwendungsassistent kann Anwendungsgerüste mit minimaler MAPI-Unterstützung generieren. Falls gefordert, kann der Anwendungsassistent das DATEI-Menü der zu erstellenden Anwendung mit einem NACHRICHT SENDEN-Befehl ausstatten. Dieser Befehl wird die Elementfunktion `OnFileSendMail` der Dokumentklasse der Anwendung aufrufen. Der Anwendungsassistent wird auch einen Handler zur Befehlsaktualisierung installieren, der die Elementfunktion `OnUpdateSendMail` referenziert.

Mit anderen Worten, falls Sie Ihre Anwendung nur nachrichtenbewußt machen wollen, müssen Sie nichts weiter tun, als die MAPI-Unterstützung zu aktivieren, wenn Sie ein Anwendungsgerüst mit dem Anwendungsassistenten erstellen. Dies ist alles, was Sie tun müssen, um die Anforderungen für das Windows-95-Logo zu erfüllen.

42.4 Zusammenfassung

MAPI, die Programmierschnittstelle für nachrichtenfähige Anwendungen, stellt eine umfangreiche Reihe von Spezifikationen dar, die nachrichtenfähige Anwendungen mit Dienst Anbietern verbinden, um so eine nachrichtenfähige Architektur zu bilden.

Nachrichtenfähige Anwendungen umfassen nachrichtenbewußte, nachrichtenaktivierte und nachrichtenbasierte Programme.

- ➡ Nachrichtenbewußte Anwendungen, wie das Windows-95/98-WordPad, hängen nicht vom Vorhandensein von MAPI ab, obwohl sie ein gewisses Niveau von MAPI-Funktionalität bieten.
- ➡ Nachrichtenaktivierte Anwendungen, wie Microsoft Outlook, erfordern das Vorhandensein von MAPI, um viele ihrer Funktionen durchführen zu können.
- ➡ Nachrichtenbasierte Anwendungen, wie der Microsoft Exchange Client, erfordern MAPI, um überhaupt zu funktionieren.

Auf der Dienstanbieterseite kennt MAPI Nachrichtenspeicher, Adreßbücher und Anbieter von Übertragungswegen. Diese Anbieter führen unter der Kontrolle des MAPI-Spoolers die Dienste durch, die zusammen das nachrichtenfähige System auf Ihrem Rechner bilden.

Die MAPI-Konfiguration wird in Form von MAPI-Profilen gespeichert. MAPI-Profile sind Einträge in der Registrierdatenbank, die aktive MAPI-Dienste und ihre Konfiguration identifizieren. Ein Anwender kann mehrere MAPI-Profile haben, wobei ein aktives Profil am Anfang einer MAPI-Sitzung gewählt ist.

Beim Entwickeln von MAPI-Anwendungen können Sie eine von mehreren APIs wählen.

- ➔ Simple MAPI bietet eine Reihe von Funktionen, die beim Adressieren einfacher Nachrichten und beim Anfordern ankommender Nachrichten von Bedeutung sind.
- ➔ Ähnliche Funktionen werden von CMC angeboten, jedoch ist CMC völlig plattformunabhängig. CMC ist die Implementation der gemeinsamen Nachrichtenaufrufe der X.400-Vereinigung.
- ➔ Visual-Basic- und Visual-C++-Anwendungen können auch OLE-Nachrichten für eine objektorientierte Schnittstelle zu den MAPI-Funktionen einsetzen.

TCP/IP- Programmierung mit WinSock

Kapitel

43

Vor der Informationsexplosion, die vor einigen Jahren begann, hatten nicht allzu viele Programmierer von TCP/IP gehört, dem Protokoll, das im Internet zur Kommunikation zwischen den Subnetzen verwendet wird.

Früher war es ein unbedeutendes Protokoll, doch heutzutage kann kein Betriebssystem von sich behaupten, komplett zu sein, wenn es dieses Protokoll nicht aufweist; Windows ist da keine Ausnahme.

Die am häufigsten eingesetzte Programmierschnittstelle für TCP/IP ist die Berkeley-Sockets-Bibliothek. Berkeley Sockets werden im gesamten Internet zum Implementieren von TCP/IP-Anwendungen verwendet, und so wählte auch Microsoft diese API für TCP/IP unter Windows, als die Zeit reif war.

Der Einsatz der Berkeley Sockets ist nicht auf die TCP/IP-Programmierung beschränkt. Berkeley Sockets stellen einen Mechanismus zur Kommunikation zwischen einzelnen Tasks dar, der TCP/IP oder andere Netzwerkprotokolle für die eigentliche Datenübertragung verwenden kann.

Zuerst folgt hier eine Übersicht der Grundlagen der TCP/IP-Netzwerke.

43.1 TCP/IP-Netzwerke und OSI

Ich möchte Ihnen eine weitere Geschichte des Entstehens von TCP/IP ersparen. Die Geschichte, wie es sich zum Protokoll von ARPANET und später von NFSNET entwickelte, wurde schon unzählige Male erzählt.

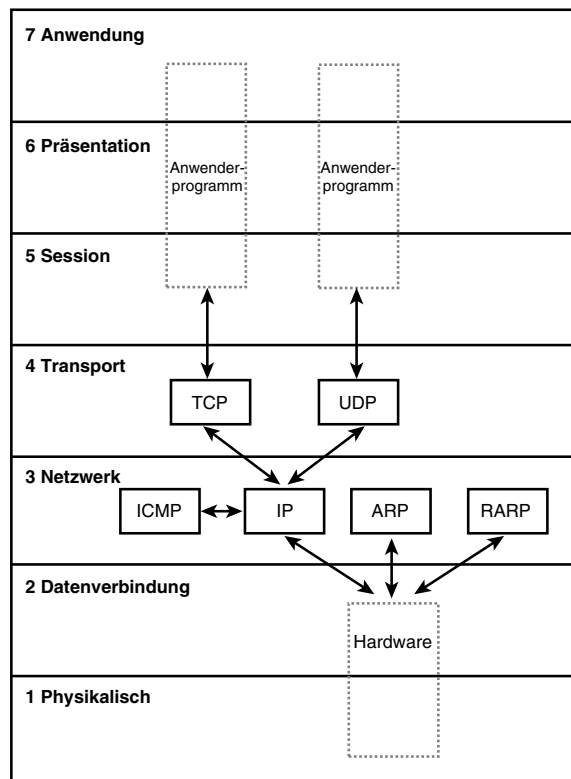
43.1.1 Die Internet-Protokollsuite

TCP/IP ist ein etwas ungeeigneter Bezeichner; die Internet-Protokollsuite, auf die es sich üblicherweise bezieht, besteht aus mehr Komponenten als nur TCP und IP. Abbildung 43.1 bietet dazu eine kurze Übersicht.



Im Ausdruck »Internet-Protokoll« ist der Begriff Internet allgemein gemeint. Es kann viele Internets geben (zum Beispiel das Netzwerk einer großen Firma, was heutzutage oft als Intranet bezeichnet wird). Dahingegen ist »das« Internet einzigartig und bei weitem das größte Netzwerk.

Abbildung 43.1:
Die Internet-
Protokollsuite
und die OSI-
Schichten



Was steckt hinter diesen seltsamen Abkürzungen?

TCP

TCP ist das Transmission Control Protocol, das Übertragungssteuerungsprotokoll. TCP bietet einen zuverlässigen Bytefluß zwischen zwei Prozessen. Zuverlässigkeit in diesem Kontext bedeutet, daß Anwendungen die Pakete

nicht nach solchen, die verlorengegangen oder zerstört wurden, überwachen müssen. TCP ist auch ein VERBINDUNGSORIENTIERTES Protokoll. Anwendungen stellen eine logische Verbindung her, bevor irgendein Datenaustausch stattfinden kann, und beenden die Verbindung, wenn der Datenaustausch erfolgt ist. Manchmal wird der Begriff VIRTUELLE SCHALTUNG für diese logische Verbindung verwendet.

- UDP UDP steht für USER DATAGRAM PROTOCOL. UDP ist ein VERBINDUNGSLOSES Protokoll. Datenpakete werden individuell adressiert und versendet. UDP ist kein zuverlässiges Protokoll; Anwendungen müssen darauf eingestellt sein, Datenverluste zu behandeln.
- ICMP Das INTERNET CONTROL MESSAGE PROTOCOL wird oft zum Versenden von Fehler- und Kontrollinformationen in TCP/IP-Netzen eingesetzt. ICMP-Pakete werden selten von Anwenderprozessen verwendet. Eine bemerkenswerte Ausnahme ist das Dienstprogramm `ping`, das zum Testen der Verfügbarkeit eines fremden Hosts verwendet wird. Dazu sendet der Rechner ein ICMP-»Echo«-Paket und überwacht seine Rückkehr.
- IP IP steht für INTERNET PROTOCOL (was sonst?). IP bietet den Paketdienst, auf dem TCP, UDP und ICMP beruhen.
- ARP ARP ist das ADDRESS RESOLUTION PROTOCOL. ARP wird zum Umsetzen einer Netzwerkadresse in eine Hardware-Adresse verwendet.
- RARP RARP ist das REVERSE ADDRESS RESOLUTION PROTOCOL. RARP wird zum Umsetzen einer Hardware-Adresse in eine Netzwerkadresse verwendet.

43.1.2 IP-Datagramme

Das Herzstück der TCP/IP-Datenübertragung ist das IP-Datagramm. Ein IP-Datagramm ist ein Paket, das Quell- und Zieladresse enthält, ebenso Diensttypinformation, User-Daten und Fehlerkorrekturinformation.

Ein IP-Datagramm besteht aus einem Header und einem Datenblock. Die Daten können beliebig sein, je nach Diensttyp und den Anforderungen des Users. Der Header andererseits enthält einen Satz von klar definierten Feldern, die im nächsten Abschnitt beschrieben werden.

43.1.3 IP-Header

Die Header eines IP-Datagramms, die IP-Header, bestehen normalerweise aus 20 Byte. Abbildung 43.2 zeigt die Felder dieses Headers.

Abbildung 43.2:
Der IP-Header

Version (4 Bit)	Headerlänge (4 Bit)	Diensttyp (8 Bit)
Paketlänge (16 Bit)		
Paketbezeichner (16 Bit)		
Fragmentierungs-Daten (16 Bit)		
Lebensdauer (8 Bit)	Protokoll (8 Bit)	
Header-Prüfsumme (16 Bit)		
Quelladresse (32 Bit)		
Zieladresse (32 Bit)		

Unter den Feldern aus Abbildung 43.2 sind einige, die für uns interessant sind, insbesondere

- ➔ das Protokoll,
- ➔ die Quelladresse und
- ➔ die Zieladresse.

Das Protokoll-Feld bestimmt, wie der Rest des IP-Pakets interpretiert wird. Einige Dutzende Werte wurden für dieses Feld definiert. Zum Beispiel bedeutet der Wert 6 ein TCP/IP-Paket. Ein Wert von 17 bedeutet ein UDP-Paket.

Die Adressen in einem Paket stellen Hostadressen dar. Hostadressen bestimmen eindeutig Hosts in einem Internet.

43.1.4 IP-Hostadressen und Routing

Eine IP-Hostadresse ist eine 32-Bit-Zahl, die eindeutig einen Internet-Host identifiziert. Um genauer zu sein: eine IP-Adresse identifiziert eindeutig eine bestimmte Schnittstelle eines Internet-Hosts; Gateway-Maschinen (die Schnittstellen zu mehr als einem Netzwerk haben), haben gewöhnlich mehrere Hostadressen.

Vereinbarungsgemäß werden die 4-Byte-Internet-Hostadressen gewöhnlich als Satz von vier Dezimalzahlen geschrieben, zum Beispiel 127.0.0.1.

Die Internet-Hostadresse wird gewöhnlich zweigeteilt in

Aufbau von Host-Adressen

- die NETZWERKADRESSE und
- die eigentliche HOSTADRESSE.

Die jeweilige Länge dieser zwei Adreßteile schwankt nach dem höherwertigen Byte der Adresse.

Klassen von Netzwerkadressen

Netzwerkadressen der Klasse A sind jene, deren höherwertiges Byte zwischen 0 und 127 liegt. Eine Adresse der Klasse A besteht aus einer 8-Bit-Netzwerkadresse und einer 24-Bit-Hostadresse. Da die Adressen, die mit 0 und 127 beginnen, reserviert sind, können höchstens 126 Unternetze der Klasse A in einem Internet sein. Ein Unternetz der Klasse A kann bis zu 16.777.214 Hosts enthalten (Adressen der Form nnn.0.0.0 und nnn.255.255.255. sind reserviert). *Klasse A*

Netzwerkadressen der Klasse B sind jene, deren höherwertiges Byte zwischen 128 und 191 liegt. Adressen der Klasse B bestehen aus einer 16-Bit-Netzwerkadresse und einer 16-Bit-Hostadresse. Es kann höchstens 65.534 Unternetze der Klasse B geben. Ein Unternetz der Klasse B kann bis zu 65.534 Hosts haben. (Die Adressen der Form nnn.mmm.0.0 und nnn.mmm.255.255 sind wiederum reserviert.) *Klasse B*

Eine Netzwerkadresse der Klasse C ist eine, deren höherwertiges Byte zwischen 192 und 223 liegt. Dies läßt insgesamt 2.097.152 Unternetze zu. Ein Unternetz der Klasse C kann bis zu 254 Hosts enthalten (Adressen der Form nnn.mmm.kkk.0 und nnn.mmm.kkk.255 sind reserviert.) *Klasse C*

Netzwerkadressen der Klasse D (höherwertiges Byte zwischen 224 und 255) sind für IP-MULTICASTING reserviert. Diese eingeschränkte Form von Broadcasting ist für die meisten WinSock-Programmierer nicht von Bedeutung. *Klasse D*

Routing

Ist die Zieladresse eines Pakets einmal bekannt, so können Hosts im Internet entsprechend ROUTEN. Für die meisten Hosts bedeutet Routen entweder den Versand des Pakets an ein bekanntes Ziel oder das Weiterleiten auf einer Default-Route. Falls Sie zum Beispiel ein Unternetz der Klasse C haben, das an einen Internet-Dienstanbieter über eine SLIP- oder PPP-Verbindung angeschlossen ist, ist diese Verbindung Ihre Default-Route. Alle Pakete, die nicht an einen Host aus Ihrem Unternetz adressiert sind, werden automatisch an Ihren Anbieter weitergeleitet.

Größere Hosts verwalten dynamisch aktualisierte Routing-Tabellen. Durch diese Tabellen kann die richtige Route der Pakete ermittelt werden, und im Fall eines Netzwerkfehlers können alternative Routen gefunden werden.

Sofern der WinSock-Programmierer betroffen ist, ist Routing eine im Hintergrund ablaufende Aktivität. Sie brauchen sich nicht darum zu kümmern, durch welche Zauberei das Internet ein Paket an einen WWW-Host in Neuseeland oder an eine FTP-Site in Alaska abliefern.

43.1.5 Hostnamen

Es wurde früh in der Entwicklung des Internet erkannt, daß numerische Hostadressen nicht immer geeignet sind. Zum einen sind sie schwer zu merken, zum anderen gäbe es ein heilloses Durcheinander, wenn aus irgendeinem Grund die Adresse eines Hosts geändert wird. Deshalb wurde ein Namenssystem eingeführt, das numerische IP-Adressen auf leicht zu merkende Hostnamen abbildet, und umgekehrt.

In den frühen Tagen der wenigen Hosts im Internet verwaltete jeder Host eine Datei (die Datei `hosts`) mit einer Liste aller Internet-Hosts und ihrer Adressen. Als das Internet sein phänomenales Wachstum begann, erwies sich diese Lösung als ungeeignet. Erstens mußte die Namensvergabe für Internet-Hosts standardisiert werden. Zweitens mußte ein Mechanismus gefunden werden, der die Notwendigkeit der Verwaltung einer Kopie der Datei `hosts` auf jedem am Internet angeschlossenen Rechner überflüssig machte.

Das Domain Name System

Die Antwort auf diese Probleme ist das Domain Name System (DNS), dessen Entwicklung in den frühen 80ern begann. DNS ist ein hierarchisches Namensvergabesystem. Das Format der DNS-Hostnamen ist jedem, der das Internet benützt, vertraut. Ein typischer Internet-Hostname hat die Form `host.subdomain.domain`.

Die Domain-Hierarchie An der Spitze der Hierarchie befindet sich die Root-Domain, bezeichnet mit einem einzelnen Punkt. Die nächsten sind die Toplevel-Domains. Die Namen der Toplevel-Domains wurden entweder nach Organisationen (meist in den Vereinigten Staaten) oder nach dem Land vergeben. Die am häufigsten verwendeten Toplevel-Domains sind folgende:

GOV:	Regierungsstellen
EDU:	Bildungsinstitute
COM:	kommerzielle Unternehmen

MIL: militärische Organisationen

ORG: andere Organisationen

Toplevel-Domains der Länder beruhen meist auf dem Standard ISO 3166 für Abkürzungen der Ländernamen mit zwei Buchstaben, zum Beispiel US (USA), CA (Canada), DE (Deutschland).

Wie wird ein Domänenname in eine numerische IP-Adresse übersetzt und umgekehrt? DNS definiert einen verteilten Namensdienst-Mechanismus, demnach Hosts als Namensserver für verschiedene Domains auftreten können. Namensserver kommunizieren ständig miteinander und tauschen Informationen über bestimmte Hosts aus. Ohne allzu sehr ins Detail zu gehen, reicht es, zu sagen, daß wenn Ihre Anwendung die IP-Adresse für einen Hostnamen anfordert, diese Anforderung an Ihren Default-Server geschickt wird, der wiederum einen anderen Namensserver im Internet nach der Information abfragen kann.

*Vom Domänen-
namen zur IP-
Adresse*

43.1.6 TCP- und UDP-Pakete, Portnummern und Sockets

Das Kennen der Adresse des Zielhosts für ein IP-Paket ist für die meisten Anwendungen nicht ausreichend. Schließlich kann ein Host mehrere offene Verbindungen verwalten und in einige TCP/IP-Gespräche gleichzeitig verwickelt sein. Wenn ein IP-Paket empfangen wird, wie kann der Host ermitteln, zu welchem Gespräch dieses Paket gehört?

Portnummern

TCP- und UDP-Pakete enthalten außer dem IP-Header noch zusätzliche Headerinformationen. Die ersten 4 Byte der TCP- und UDP-Header enthalten Portnummern, 2 Byte für die Quelle und 2 Byte für das Ziel.

Socket

Die Portnummer identifiziert zusammen mit der IP-Nummer ein Socket, das einmalig im gesamten Internet ist. Ein Socket-Paar identifiziert eindeutig eine Verbindung (zumindest im Fall von TCP, einem verbindungsorientierten Protokoll).

TCP- und UDP-Ports sind nicht äquivalent. Zum Beispiel beziehen sich TCP-Port 25 und UDP-Port 25 auf zwei verschiedene Dienste.



43.1.7 Internet-Dienste

Es gibt viele verschiedene Dienste auf den weitverbreiteten IP-Netzwerken, darunter FTP, Telnet, Gopher, das WWW, Archie, der DNS-Namensdienst, whois, finger und viele andere.

Die für diese Dienste verwendeten Protokolle werden in den Requests For Comment, den RFCs, definiert (einige der relevantesten RFCs sind am Ende dieses Kapitels aufgelistet). Die Dienste selber sind gewöhnlich unter wohlbekannten Portnummern erreichbar. Um z.B. mit einem Telnet-Server auf einem beliebigen Host im Internet Verbindung aufzunehmen, werden Sie versuchen, den TCP-Port 23 auf der Maschine anzusprechen. Wohlbekannte Ports, die ein Host erkennen soll, werden typischerweise in der Datei `services` des Systems identifiziert. Diese Datei wird sowohl unter Unix als auch in den Windows-Implementationen von TCP/IP verwendet.

Auf den meisten Systemen sind die TCP- und UDP-Portnummern 0 bis 1023 für privilegierte Prozesse reserviert.

43.2 Die WinSock-API

Die Berkeley-Socket-Schnittstelle kann für die Kommunikation sowohl mit verbindungsorientierten Protokollen (TCP) als auch verbindungslosen Protokollen (UDP) verwendet werden. Das Programmiermodell ist das Client-Server-Modell; Server warten auf ankommende Anforderungen, während Clients Sitzungen beginnen.

Es gibt einige Unterschiede in den Implementationen von WinSock und der Unix-Version der Berkeley Sockets. Der bedeutendste ist vielleicht die Tatsache, daß Socket-Deskriptoren und Datei-Deskriptoren nicht untereinander ausgetauscht werden können. Das hat einen bemerkenswerten Effekt bei der Portierung von Anwendungen, die diese Äquivalenz voraussetzen.

Ein weiterer Unterschied ist die Notwendigkeit, die WinSock-Bibliothek zu initialisieren. Anwendungen, die WinSock-Funktionen einsetzen wollen, müssen zuerst die Funktion `WSAStartup` aufrufen. Wenn Ihre Arbeit mit der WinSock-Bibliothek beendet ist, sollten Sie `WSACleanup` zum ordnungsgemäßen Beenden aufrufen.

Die WinSock-API führt auch einige WinSock-spezifische Funktionen für asynchrone E/A über Sockets ein. Diese helfen Ihnen beim Entwickeln ansprechender GUI-WinSock-Anwendungen.

43.2.1 Initialisieren von WinSock

Die WinSock-Bibliothek wird durch einen Aufruf von `WSAStartup` initialisiert. Die Anwendung, die diese Funktion aufruft, übergibt die Adresse einer `WSDATA`-Struktur, die Initialisierungsinformationen enthält. *WSAStartup*

Während des Aufrufs von `WSAStartup` handeln Anwendung und WinSock-Bibliothek eine Versionsnummer aus. Die Initialisierung scheitert, wenn zwischen der von der Anwendung unterstützten Versionsnummer und der von der WinSock-Bibliothek unterstützten Versionsnummer keine Übereinstimmung herrscht.

Wenn ein Fehler auftritt, gibt `WSAStartup` einen von Null verschiedenen Wert zurück. Anwendungen können über die Funktion `WSAGetLastError` ausführliche Fehlerinformationen abrufen. *WSAGetLastError*

43.2.2 Erzeugen und Verwenden der Sockets

Ein Socket wird durch Aufrufen der Funktion `socket` erzeugt. Die Parameter dieser Funktion bestimmen den Socket-Typ, den Typ der Netzwerkadresse und das verwendete Protokoll. Zum Beispiel erzeugt der Aufruf *Sockets erzeugen*

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

ein TCP-Socket.

Um ein Socket mit einer Hostadresse und einer Portnummer zu verbinden, rufen Anwendungen typischerweise die Funktion `bind` auf. Außer dem Identifizierer des Sockets, oder Socket-Deskriptor (der vom `socket`-Aufruf zurückgegeben wird), erwartet `bind` einen Parameter, der ein Zeiger auf eine Struktur mit der Socket-Adresse ist. Diese Struktur ist folgendermaßen definiert: *Sockets einsetzen*

```
struct sockaddr
{
    u_short sa_family;
    char sa_data[14];
};
```

Das Element `sa_family` dieser Struktur bestimmt den Typ der Adresse. Für Internet-Adressen wird der Wert auf `AF_INET` gesetzt. Das `sa_data` enthält die eigentliche Adresse.

Anwendungen können leicht auf die verschiedenen Komponenten der Adresse zugreifen, indem sie sich auf eine Struktur `sockaddr_in` beziehen (statt `sockaddr` zu verwenden). Hier die Definition dieser Struktur:

```
struct sockaddr_in
{
    short sin_family;
    u_short sin_port;
```



```

    struct in_addr sin_addr;
    char sin_zero[8];
};

```

Unter diesen Elementen ist `sin_port` die 16-Bit-Portnummer und `sin_addr` die 32-Bit-Hostadresse.

43.2.3 Namensdienste

Hostadresse ermitteln Um dem Feld mit der Hostadresse in der Struktur `sockaddr_in` sinnvolle Werte zuzuweisen, müssen Sie zuerst eine 32-Bit-Hostadresse erhalten. Um eine Adresse zu erhalten, wenn der symbolische Name eines Hosts bekannt ist, verwenden Sie die Funktion `gethostbyname`.

gethostbyname Beim Aufruf von `gethostbyname` übergeben Anwendungen den symbolischen Namen des Hosts und erhalten einen Zeiger auf eine `hostent`-Struktur zurück. Die Struktur `hostent` ist folgendermaßen definiert:

```

struct hostent
{
    char FAR * h_name;
    char FAR * FAR * h_aliases;
    short h_addrtype;
    short h_length;
    char FAR * FAR * h_addr_list;
};

```

Diese Struktur ist nötig, da ein Hostname mit mehreren Hostadressen in Verbindung gebracht werden kann (umgekehrt gilt dies ebenso). In den meisten Fällen nehmen Anwendungen einfach die erste (oftmals die einzige) Adresse aus `h_addr_list`; um dies zu erleichtern, ist das Symbol `h_addr` als `h_addr_list[0]` definiert.

inet_addr Falls eine Anwendung statt dessen eine numerische Adresse verwenden will, kann sie die Funktion `inet_addr` einsetzen, um einen String mit einer numerischen Adresse in einen 32-Bit-Adreßwert umzuwandeln. Ist dieser Wert einmal erhalten, so kann `gethostbyaddr` verwendet werden, um für den gegebenen Host eine `hostent`-Struktur zurückzugeben.

43.2.4 Anordnung der Bytes

Ein Problem, das besonders wichtig ist, wenn es um die Entwicklung von Anwendungen geht, die auf hybriden Netzen funktionieren sollen, ist das Thema Byte-Anordnung. Einige Systemarchitekturen sind

- ➔ big-endian (das höherwertige Bit kommt zuerst), andere
- ➔ little-endian (das höherwertige Bit kommt zuletzt).

Ein Beispiel für die erste Kategorie ist die Prozessorfamilie 68000 von Motorola. Beispiele für die zweite Kategorie sind die Prozessorfamilien von Intel und die CPUs von DEC.

Internet-Adressen (zum Beispiel Hostadressen) sind immer big-endian. Um eine korrekte Umwandlung zwischen maschinenunabhängigen IP-Nummern und ihrer maschinenabhängigen Darstellung sicherzustellen, können Sie die folgenden Funktionen einsetzen: `htonl`, `htons`, `ntohl` und `ntohs`. Diese Funktionen wandeln kurze oder lange Integer vom Netzwerk- ins Hostformat um und umgekehrt. Beachten Sie, daß diese Funktionen als Makros implementiert sein können.

Internet-Adressen sind immer big-endian

43.2.5 Kommunikation über Sockets

Unter TCP

Im Falle des verbindungsorientierten TCP-Protokolls bindet die Serveranwendung an ein bestimmtes TCP-Port an und verwendet dann die Funktion `listen`, um ihre Bereitschaft zu zeigen, ankommende Verbindungsanforderungen anzunehmen. Gleich nach dem Aufruf von `listen` ruft der Server `accept` auf, um auf ankommende Verbindungen zu warten. Wenn `accept` zurückkehrt, liefert sie die Adresse des Peer-Prozesses.

Der Client kann nach dem Erzeugen eines Socket mit dem Aufruf von `socket` eine Verbindung mit dem Aufruf von `connect` herstellen. Es ist nicht nötig, den Socket vor dem Aufruf von `connect` an einen bestimmten Port mit `bind` zu binden.

Verbindung einrichten

Nachdem die Verbindung erfolgreich hergestellt wurde, können sowohl Server als auch Client den Aufruf `send` verwenden, um Daten zu senden, und `recv`, um Daten zu empfangen. Die Semantik von `send` und `recv` ist ähnlich wie die Semantik von `read` und `write` für Dateiein-/ausgabe auf der niederen Ebene.

Daten austauschen

Tatsächlich kann unter Unix-Systemen das Funktionspaar `read/write` auch zur E/A über Sockets eingesetzt werden. Leider ist dies, wie ich bereits erwähnte, mit WinSock aufgrund der Unterschiede zwischen einem Socket-Deskriptor und einem Datei-Deskriptor nicht möglich.

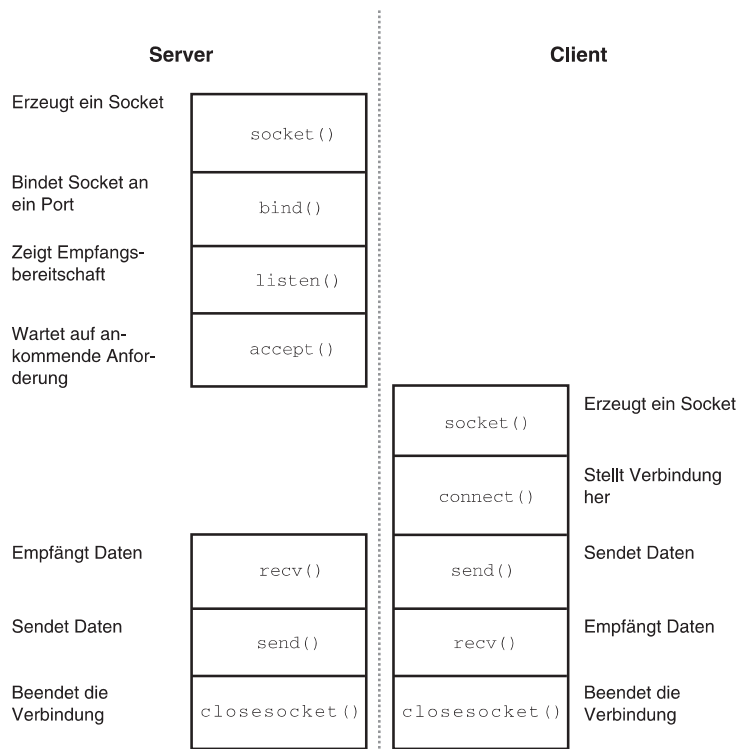


Anwendungen müssen `closesocket` aufrufen, wenn eine Socket-Verbindung beendet ist. Sowohl der Client als auch der Server können die Verbindung mit `closesocket` schließen.

Verbindung schließen

Abbildung 43.3 gibt eine Übersicht zum Aufstellen und Verwenden einer TCP-Verbindung.

Abbildung 43.3:
Kommunikation
über Sockets für
verbindungs-
orientierte
Protokolle



Unter UDP

Im Falle des verbindungslosen Protokolls UDP ist die Folge der Ereignisse etwas verschieden, die Aktivitäten von Client und Server sind symmetrischer. In diesem Fall erzeugen sowohl der Client als auch der Server ihre eigenen Sockets und binden sie an bestimmte Portnummern. Der Server ruft dann die Funktion `recvfrom` auf, die auf ankommende Daten wartet. Der Client seinerseits verwendet den Aufruf `sendto`, um Daten an eine bestimmte Adresse zu senden. Wenn diese Daten vom Server empfangen werden, kehrt `recvfrom` zurück und gibt dem Server die Adresse bekannt, von wo die Daten empfangen wurden. Er kann diese Adresse in einem nachfolgenden Aufruf von `sendto` verwenden, um so dem Client zu antworten. Abbildung 43.4 zeigt eine Übersicht dieses Vorgangs.

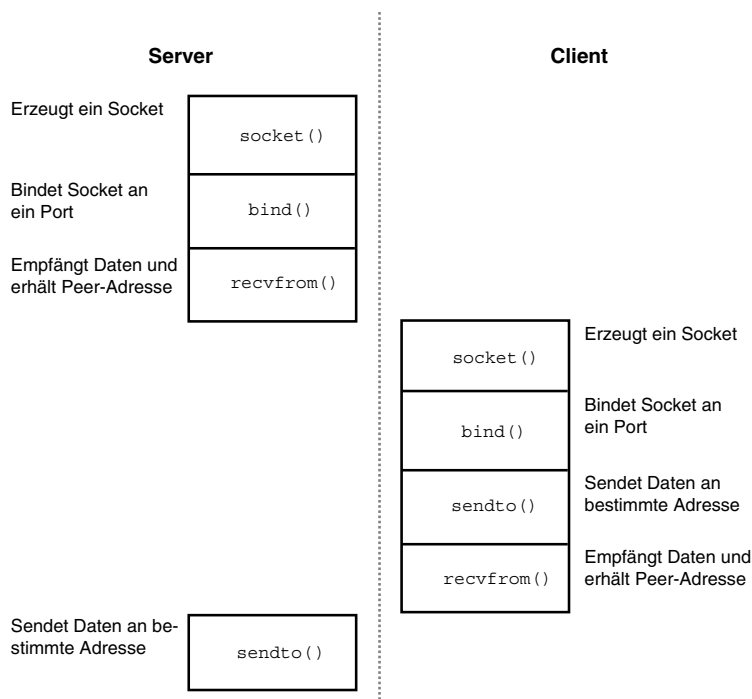


Abbildung 43.4:
Kommunikation
über Sockets für
verbindungs-
lose Protokolle

43.2.6 Das Blocking-Problem und der Aufruf von `select`

In den vereinfachten Modellen aus Abbildung 43.3 und Abbildung 43.4 verwenden sowohl der Client als auch der Server beim Warten auf Daten blockierende Aufrufe. Ein blockierender Aufruf kehrt nicht zurück zur aufrufenden Funktion, bis die angeforderten Daten verfügbar sind. Mit anderen Worten, die Anwendung, die einen solchen Aufruf macht, hängt so lange, bis der Aufruf beendet ist.

Während dieses Modell in vielen einfachen Situationen ausreichen wird, ist es offensichtlich für interaktive Anwendungen unannehmbar. Eine solche Anwendung (zum Beispiel ein Telnet-Client) kann nicht einfach eingefroren werden, bis Daten vom Server verfügbar sind.

Die Lösung, die von den meisten Unix-TCP/IP-Anwendungen verwendet wird, beruht auf dem Systemaufruf `select`. Dieser Aufruf macht es möglich, auf mehrere Deskriptoren (Datei oder Socket) zu warten. So kann ein Unix-Prozeß einfach auf Daten von einem Socket oder von der Standardeingabe warten und in Aktion treten, sobald Daten auf einem von beiden verfügbar sind.

Leider sind die Dinge bei WinSock nicht so einfach. Erinnern Sie sich, daß Socket- und Dateideskriptoren nicht austauschbar sind? Die Funktion `select` ist keine Ausnahme; sie kann nur auf mehrere Socket-Deskriptoren warten, nicht auf ein Gemisch von Datei- und Socket-Deskriptoren.

Multithreading für das Polling Obwohl es möglich ist, einen Socket während des Polling zu überwachen, ist dies keine sehr effiziente Lösung. Zum Glück kommt Rettung von der Multithread-Fähigkeit von Win32. Ein Prozeß kann einfach zusätzliche Threads starten und für jede Eingabequelle einen eigenen Thread haben. Dieser Mechanismus funktioniert sowohl für befehlszeilenorientierte als auch grafische TCP/IP-Dienstprogramme. Trotzdem bietet die WinSock-Bibliothek noch eine Funktionsfamilie, die beim Schreiben von TCP/IP-Anwendungen ohne Tricks mit Multithreads behilflich sind. Diese asynchronen Socketaufrufe können somit auch in 16-Bit-Programmen und unter Win32s eingesetzt werden.

43.2.7 Asynchrone Socket-Aufrufe

Asynchrone Socket-Aufrufe beruhen auf dem Nachrichtenverarbeitungsmechanismus unter Windows, Windows-Anwendungen über Socket-Ereignisse zu informieren.

WSAAsyncSelect Im Mittelpunkt dieses Mechanismus steht die Funktion `WSAAsyncSelect`. Durch diese Funktion kann eine Anwendung auf eine Kombination von Socket-Ereignissen warten. Anwendungen können Benachrichtigungen über die Bereitschaft zu lesen und zu schreiben, über ankommende und beendete Verbindungen sowie über das Schließen der Sockets erhalten. Die Benachrichtigung geschieht in Form einer benutzerdefinierten Nachricht, die an ein Fenster gesendet wird. Dies wird auch im Aufruf von `WSAAsyncSelect` definiert.

`WSAAsyncSelect` sendet eine einzelne Nachricht für jedes Ereignis, an dem die Anwendung Interesse angemeldet hat. Ist die Nachricht einmal gesendet, werden keine weiteren für das gleiche Ereignis gesendet, bis die Anwendung das Ereignis implizit zurücksetzt, indem sie die entsprechende Socket-Funktion aufruft. Wenn z.B. eine Benachrichtigung für ankommende Daten gesendet wurde, wird keine weitere dieser Art für das gegebene Socket gesendet, bis die Anwendung die Daten abholt, indem sie `recv` oder `recvfrom` aufruft.

Weitere asynchrone Socket-Funktionen umfassen zum Beispiel asynchrone Versionen der Standard-Berkeley-Aufrufe `gethostbyname` und `gethostbyaddr`: `WSAAsyncGetHostByName` und `WSAAsyncGetHostByAddr`. WinSock-Anwendungen können auch den Blockiermechanismus in den Standard-Aufrufen im Berkeley-Stil beeinflussen, indem sie die Funktion `WSASetBlockingHook` aufrufen.

43.3 Ein einfaches Beispiel mit WinSock

Es ist an der Zeit, diese ganze Theorie in die Praxis umzusetzen. Ich möchte mit Ihnen, ein einfaches WinSock-basiertes Befehlszeilen-Dienstprogramm besprechen. Dieses Programm nimmt Kontakt auf zu einem wohlbekannten Dienst (dem Dienst `time` auf TCP-Port 37) auf dem angegebenen Host, holt die aktuelle Zeit in maschinenlesbarer Form, formatiert sie und gibt sie aus. Mit wenigen Änderungen könnte diese Anwendung zum Setzen der Systemzeit von einem Server, der eine zuverlässige Zeit liefert, verwendet werden.

Das Programm aus Listing 43.1 kann von der Kommandozeile mit

CL GETTIME.CPP WSOCK32.LIB

kompiliert werden.

```
#include <iostream.h>
#include <time.h>
#include <winsock.h>

void main(int argc, char *argv[])
{
    time_t t;
    int s;
    struct sockaddr_in a;
    struct hostent *h;
    WSADATA wsaData;

    if (argc != 2)
    {
        cout << "Usage: " << argv[0] << " host\n";
        exit(1);
    }
    if(WSAStartup(0x101, &wsaData))
    {
        cout << "Unable to initialize WinSock library.\n";
        exit(1);
    }
    h = gethostbyname(argv[1]);
    if (h == NULL)
    {
        cout << "Cannot resolve hostname\n";
        WSACleanup();
        exit(1);
    }

    a.sin_family = AF_INET;
    a.sin_port = htons(37);
    memcpy(&(a.sin_addr.s_addr), h->h_addr, sizeof(int));
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s == 0)
    {
        cout << "Cannot establish connection: "
              << WSAGetLastError() << '\n';
    }
}
```

*Listing 43.1:
Das Dienstpro-
gramm gettime*

```

        WSACleanup();
        exit(1);
    }
    if (connect(s, (struct sockaddr *)&a, sizeof(a)))
    {
        cout << "Cannot establish connection: "
              << WSAGetLastError() << '\n';
        WSACleanup();
        exit(1);
    }
    if (recv(s, (char *)&t, 4, 0) != 4)
        cout << "Unable to obtain time.\n";
    else
    {
        t = ntohl(t) - 2208988800;
        cout << asctime(localtime(&t));
    }
    closesocket(s);
    WSACleanup();
}

```

Der Dienst `time` ist in RFC868 beschrieben. Ein Internet-Zeitserver hört auf TCP-Port 37; wenn er eine Verbindungsanforderung empfängt, sendet er die aktuelle Zeit (GMT) als 4-Byte-Zahl, die die Anzahl der seit 1. Januar 1900 vergangenen Sekunden darstellt.

Aufruf Die Anwendung `gettime` aus Listing 43.1 erwartet einen einzigen Kommandozeilenparameter, den Namen des Host, zu dem sie die Verbindung herstellen soll. Nachdem geprüft wird, ob der Parameter vorhanden ist, wird versucht, die WinSock-Bibliothek zu initialisieren.

Als nächstes wird der vom Anwender gelieferte Hostname in eine Hostadresse umgewandelt. (Es wird nicht versucht, vom User eingegebene numerische Adressen aufzulösen.) Dann wird ein Socket erzeugt, und eine Verbindung wird zu dem angegebenen Host auf TCP-Port 37 versucht.

Falls die Verbindung erfolgreich ist, versucht das Programm, genau vier Datenbytes zu empfangen. Wenn der Empfang beendet ist, werden die Daten konvertiert (`ntohl`); die Zeitbasis wird korrigiert (die Zeitfunktionen von Win32 verwenden den 1. Januar 1970 als Basisdatum für Zeitvariablen, die Anzahl der Sekunden vom 1. Januar 1900 bis 1. Januar 1970 ist 2.208.988.800); und schließlich wird die Zeit als lesbare Lokalzeit auf der Standardausgabe angezeigt:

```

C:\>gettime mud2.com
Mon Feb 24 02:48:18 1997

```

Vor dem Beenden schließt das Programm das Socket und beendet die Session mit der WinSock-Bibliothek.

43.4 Programmieren mit Sockets und die Microsoft Foundation Classes

Die Microsoft Foundation Classes enthalten die Funktionalität der Sockets in der Klasse `CAsyncSocket`. Elementfunktionen von `CAsyncSocket` entsprechen vielen der Standard- und asynchronen Funktionen aus der WinSock-Bibliothek.

CAsyncSocket

Eine Anwendung, die `CAsyncSocket` verwenden möchte, muß zuerst die Socket-Bibliothek mit einem Aufruf von `AfxSocketInit` initialisieren. Der Aufruf einer entsprechenden Aufräumfunktion ist nicht nötig.

*Initialisierung
der Socket-
Bibliothek*

Typischerweise würde eine Anwendung eine Klasse aus `CAsyncSocket` ableiten und einige Rückruffunktionen überlagern, die asynchrone Operationen durchführen.

43.4.1 Beispiel zu CAsyncSocket

Um zu sehen, wie `CAsyncSocket` funktioniert, können wir unsere Anwendung unter Verwendung von MFC umschreiben. Dieses einfache Programm kommuniziert mit dem Anwender über den in Abbildung 43.5 gezeigten Dialog.

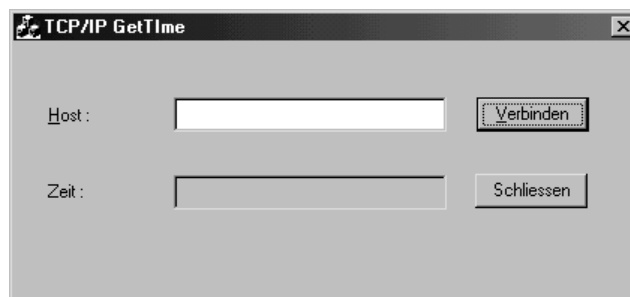


Abbildung 43.5:
MFC-Implementa-
tion des Get-
time-Pro-
gramms

Das Anwendungsgerüst

Das Gerüst dieser Anwendung kann mit dem MFC-Anwendungsassistenten erstellt werden.

1. Erzeugen Sie ein neues Projekt mit dem Namen GT.
2. Entscheiden Sie sich in Schritt 1 für eine DIALOGBASIERTE Anwendung.

3. Im 2. Schritt vergewissern Sie sich, daß Sie die Unterstützung für WINDOWS-SOCKETS anwählen. Setzen Sie hier auch den Titel des Dialogs auf einen sinnvollen String, wie *TCP/IP GetTime*.

Der Dialog

Nachdem der Anwendungsassistent das Erzeugen des Gerüsts beendet hat, ist der erste Schritt, den Dialog der Anwendung zu aktualisieren.

1. Fügen Sie die in Abbildung 43.5 gezeigten Felder hinzu.
2. Der Name des Eingabefeldes mit dem Hostnamen sollte `IDC_HOST` sein.
3. Der Name des Eingabefeldes für die Zeit sollte `IDC_TIME` sein. Für `IDC_TIME` sollte das `SCHREIBGESCHÜTZT`-Attribut gesetzt sein.
4. Der Identifizierer des Schalters VERBINDEN sollte `IDC_CONNECT` sein; der Schalter SCHLIESSEN kann den Identifizierer `IDCANCEL` behalten.
5. Als nächstes fügen Sie mit dem Klassen-Assistenten drei Elementvariablen hinzu. Zwei Stringvariablen (`m_sHost` und `m_sTime`) entsprechen den zwei Eingabefeldern; die dritte Variable, `m_cConnect`, hat den Typ `CButton` und entspricht dem Schalter VERBINDEN.

Verbindung herstellen

1. Fügen Sie unter Verwendung des Klassen-Assistenten eine Nachrichtenfunktionsfunktion für den Schalter `IDC_CONNECT` hinzu. Dieser Handler, `CGTDlg::OnConnect` (Listing 43.2), soll `BN_CLICKED`-Nachrichten verarbeiten.

Listing 43.2:
MFC *gettime*:
die Funktion
CGTDlg::OnConnect

```
void CGTDlg::OnConnect()
{
    CGTSocket *pSocket;

    pSocket = new CGTSocket(this);
    UpdateData(TRUE);
    m_cConnect.EnableWindow(FALSE);
    pSocket->Create();
    pSocket->AsyncSelect(FD_READ | FD_CLOSE);
    pSocket->Connect(m_sHost, 37);
}
```

2. Um die Zeitabfrage zu implementieren, müssen wir eine Klasse von `CAsyncSocket` ableiten und ein Objekt dieses neuen Typs erzeugen, wenn der Anwender den Schalter VERBINDEN betätigt.

Der Code, den Sie zur Datei *GTDlg.cpp*, die den Dialog implementiert, hinzufügen müssen, wird in Listing 43.3 gezeigt. Beachten Sie, daß dieser Code vor der Implementation von `CGTDlg::OnConnect` stehen muß, ansonsten scheitert das Kompilieren, da `CGTDlg::OnConnect` versucht, ein Objekt vom Typ `CGTSocket` anzulegen, das noch nicht deklariert ist.

```

class CGTSocket : public CAsyncSocket
{
public:
    CGTSocket(CGTDlg *pDlg) {m_pDlg = pDlg;};
    virtual void OnReceive(int nErrorCode);
    virtual void OnClose(int nErrorCode);
    CGTDlg *m_pDlg;
    time_t t;
};

void CGTSocket::OnReceive(int nErrorCode)
{
    Receive(&t, 4);
    t = ntohl(t) - 2208988800;
    m_pDlg->m_sTime = asctime(localtime(&t));
    m_pDlg->UpdateData(FALSE);
}

void CGTSocket::OnClose(int nErrorCode)
{
    m_pDlg->m_cConnect.EnableWindow(TRUE);
    delete this;
}
    
```

Listing 43.3:
MFC gettime: Die
Klasse CGT-
Socket

Programmablauf

In der Funktion `CGTDlg::OnConnect` wird ein neues Objekt vom Typ `CGTSocket` erzeugt. Dieser Typ ist von `CAsyncSocket` abgeleitet. Das eigentliche Socket wird durch den Aufruf der Elementfunktion `Create` erzeugt, worauf ein Aufruf von `AsyncSelect` folgt. Dieser Aufruf aktiviert die Rückruffunktionen `OnReceive` und `OnClose`. Zu diesem Zeitpunkt ist der Schalter `VERBINDEN` auch deaktiviert. Zuletzt wird die Elementfunktion `Connect` aufgerufen, um eine Verbindung zum Ziel herzustellen.

Wenn der Verbindungsversuch erfolgreich ist, sendet der Zeitserver sofort die Zeit in vier Datenbytes. Dies löst die Abarbeitung der Funktion `OnReceive` aus, die die Daten mit Hilfe von `Receive` liest und das Ergebnis zum Aktualisieren des Zeitfeldes im Dialog verwendet. Nachdem die Daten gesendet wurden, schließt der Server die Verbindung. Dies bedeutet einen Aufruf von `OnClose`, der den Schalter `VERBINDEN` im Dialog reaktiviert und das Objekt `CGTSocket` zerstört.

Beachten Sie, daß das Programm in seiner gegenwärtigen Form keinerlei Fehlerprüfung durchführt und scheitern wird, wenn ein Netzwerkfehler auftritt.

43.4.2 Synchrone Operationen und Serialisierung

Der Zweck der Klasse `CAsyncSocket` ist, daß sie eine Schnittstelle auf niedrigerer Ebene zur WinSock-Bibliothek bietet. Dahingegen bietet die Klasse `CSocket`, die von `CAsyncSocket` abgeleitet ist, Funktionen auf einer etwas höheren Ebene. *CSocket*

Anders als `CAsyncSocket` bietet `CSocket` **Blocking**. Ihre Elementfunktionen kehren nicht zurück, bis die erwünschte Operation beendet ist.



Die Rückruffunktionen `OnConnect` und `OnSend` werden niemals für `CSocket`-Objekte aufgerufen.

Ein spezielles Einsatzgebiet von `CSocket`-Objekten ist im Zusammenhang mit `CFileSocket`-Objekten das Aktivieren der MFC-Serialisierungsfunktionen, damit sie mit Sockets zusammenarbeiten. Ein `CFileSocket`-Objekt kann an ein `CArchive` und ein `CSocket` angehängt werden; danach können Daten einfach durch den Einsatz der MFC-Serialisierung gesendet und empfangen werden.

43.5 Weiterführende Informationen

TCP/IP-Programmierung ist ein sehr weites Thema, zu dem es eine überwältigende Menge an Literatur gibt. Dieses Kapitel reicht sicherlich nicht aus, um diesem komplexen Thema voll gerecht zu werden. Deshalb habe ich einige Referenzen beigefügt, die Ihnen hoffentlich helfen werden, die Informationen zu finden, die Sie brauchen.

MSDN

Für WinSock ist die maßgebende Referenz Microsofts WinSock-Spezifikation, die auf der Microsoft Developer Network Library CD-ROM veröffentlicht ist.

Bücher

Es gibt einige gute Bücher zum Thema. Ein besonders nützliches ist Uyless Black's *TCP/IP & Related Protocols* (McGraw Hill, 1994). Mehr Information zur TCP/IP-Programmierung (und vor allem zur Programmierung mit Berkeley Sockets) finden Sie in *Unix Network Programming* von W. Richard Stevens (Prentice Hall, 1990). Obwohl das Buch (wie der Titel schon sagt) für Unix-Programmierer gedacht ist, ist vieles, was darin besprochen wird, nicht Unix-spezifisch, einfach weil die entsprechenden APIs betriebssystemunabhängig sind.

Die offiziellen Definitionen der meisten Internet-Standards und -Protokolle können als RFCs (Requests For Comment) online gefunden werden. Im World Wide Web verwenden Sie folgenden URL: <http://www.cis.ohio-state.edu/htbin/rfc/rfc-index.html>.

RFCs

RFCs können auch über E-Mail vom InterNIC-Directory-and-Database-Services-Mailserver angefordert werden. Senden Sie eine Meldung an `mailserv@ds.internic.net` mit folgendem Befehl im Rumpf der Meldung:

```
document-by-name rfcNNNN
```

Um den RFC-Index zu erhalten, verwenden Sie folgenden Befehl:

```
document-by-name rfc-index
```

Es können mehrere Dokumente in einer einzigen Mail angefordert werden, wenn Sie ihre Namen durch Kommas trennen oder mehrere `document-by-name`-Befehle auf getrennten Zeilen im Rumpf der Meldung angeben.

Die RFCs wurden auch auf CD-ROM von mehreren CD-ROM-Verlegern veröffentlicht (ich verwende oft die Standards-CD-ROM von Info-Magic). Falls Sie ernsthaft TCP/IP programmieren wollen, ist es empfehlenswert, sich eine günstige CD-ROM wie diese anzuschaffen.

Internet-RFCs gehen weit zurück. Das früheste RFC auf meiner CD-ROM, RFC0003, ist vom April 1969! Sicherlich sind viele dieser RFCs mittlerweile hinfällig geworden.

Folgende RFCs definieren die vom WinSock-Programmierer am häufigsten angetroffenen Standards:

- ➡ RFC0768, »User Datagram Protocol« (J. Postel, 1980)
- ➡ RFC0791, »Internet Protocol« (J. Postel, 1981)
- ➡ RFC0792, »Internet Control Message Protocol« (J. Postel, 1981)
- ➡ RFC0793, »Transmission Control Protocol« (J. Postel, 1981)
- ➡ RFC0826, »Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware« (D. Plummer, 1982)
- ➡ RFC0903, »Reverse Address Resolution Protocol« (R. Finlayson, T. Mann, J. Mogul, M. Theimer, 1984)
- ➡ RFC1034, »Domain names – Concepts and Facilities« (P. Mockapetris, 1987)
- ➡ RFC1035, »Domain names – Implementation and Specification« (P. Mockapetris, 1987)

Zwei weitere RFCs verdienen es, erwähnt zu werden. RFC1700 (»Assigned Numbers«, J. Reynolds, J. Postel, 1994) enthält alle »wohlbekannten« Nummern wie Protokoll-Identifizierer, Portnummern und

dergleichen. RFC1800 (»Internet Official Protocol Standards,« J. Postel, 1995) ist eine Referenz von unschätzbarem Wert für alle anderen RFC-Standards.



Ein verbreiteter Irrtum ist, daß alles, was als Internet-RFC veröffentlicht ist, ein Standard sei. Dem ist nicht so. Außer den Standards können Internet-RFCs Informationsmaterial, Vorschläge oder Entwürfe zu Standards sowie Geschichtliches enthalten. Die Art eines RFCs ist gewöhnlich auf der Hauptseite angegeben.

43.6 Zusammenfassung

TCP/IP ist der Name, unter dem meistens auf die Internet-Protokollsuite Bezug genommen wird. Diese Protokollsuite, die aus Protokollen wie TCP, UDP, ICMP, IP, ARP und RARP besteht, bietet die Grundlage der Kommunikation zwischen den einzelnen Netzen des globalen Internet.

Unter Windows wird TCP/IP-Programmierung mit Hilfe der WinSock-Bibliothek durchgeführt. WinSock ist eine API, die sich stark an die Berkeley-Sockets-API, die auf zahlreichen Unix-Systemen im Einsatz ist, anlehnt. Trotzdem gibt es einige Unterschiede, die daher kommen, daß Windows und Unix sehr verschiedene Architekturen haben. Im einzelnen verwendet die Berkeley-Sockets-Bibliothek Socket-Deskriptoren, die mit Datei-Deskriptoren, die in E/A-Operationen der niederen Ebene verwendet werden, austauschbar sind. Aus diesem Grund können Standard-E/A-Funktionen auf Berkeley Sockets angewendet werden, was aber unter Windows nicht zutrifft.

Berkeley Sockets können zur Kommunikation über

- ➔ verbindungsorientierte Protokolle wie TCP oder über
- ➔ verbindungslose wie TCP/IP

verwendet werden. In beiden Fällen umfassen die Grundoperationen das Erzeugen eines Sockets, das Binden des Sockets an eine Hostadresse und Portnummer und das Senden und Empfangen der Daten.

Hostadressen können mit Hilfe der Funktion `gethostbyname`, die symbolische Systemnamen in 4-Byte-TCP/IP-Adressen umwandelt, erhalten werden. Um eine maschinenunabhängige Darstellung der Hostadressen und anderer numerischer Daten zu erreichen, gibt es einen Satz von Funktionen zur Umwandlung der Reihenfolge der Bytes vom Host zum Internet.

Um gut reagierende Anwendungen zu schreiben, dürfen diese nicht endlos hängen und auf eine Eingabe vom Socket warten. Der allgemein verwendete Berkeley-Socket-Mechanismus, nämlich die Funktion `select`, hat unter Windows nur beschränkte Verwendbarkeit wegen der Socket- und Datei-Deskriptoren, die nicht austauschbar sind. Somit kann `select` nicht eingesetzt werden, um beispielsweise auf Eingaben von `stdin` zu warten. Win32-Anwendungen können aber mehrere Threads einsetzen, um Datenaustausch auf mehreren E/A-Kanälen gleichzeitig durchzuführen.

Um die Entwicklung gut reagierender Anwendungen zu erleichtern, ohne auf die Multithread-Fähigkeiten von Windows 95 oder Windows NT angewiesen zu sein, werden asynchrone Socket-Aufrufe angeboten. Diese Aufrufe beruhen auf Windows-Meldungen, um Informationen über ankommende Daten oder andere Socket-Ereignisse an die Anwendung weiterzuleiten.

Socket-Funktionen sind in der Klasse `CAsyncSocket` in der Microsoft-Foundation-Classes-Bibliothek enthalten. Mit der Hilfe der Klasse `CSocket`, die von `CAsyncSocket` abgeleitet ist und asynchrone Socket-E/A bietet, sowie der Klasse `CSocketFile` können Anwendungen MFC-Serialisierungsfunktionen in einer Socketschnittstelle einsetzen.

Telefonie- Anwendungen mit TAPI

Kapitel

44

Die Microsoft-Telefonie-API oder TAPI bietet Dienste rund ums Telefon für Win32-Anwendungen.

TAPI wird zur Zeit auf den Windows-3.1- und Windows-95/98-Plattformen unterstützt. In diesem Kapitel besprechen wir TAPI 1.4. TAPI wird intensiv von jenen Windows-95-Anwendungen verwendet, die ein Modem brauchen, zum Beispiel der Fax-Treiber und der CompuServe-Treiber in Microsoft Exchange, die Microsoft-Netzwerk-Software oder die Windows-95/98-Wahlhilfe und die Hyperterminal-Anwendungen.

44.1 Übersicht zu TAPI

Zuerst eine wichtige Bemerkung. Wenn die meisten von uns den Begriff Telefonie im Kontext der Computerkommunikation hören, denken wir an Daten- oder Fax-Modems und Sprachtelefonleitungen – und sonst kaum an etwas. TAPI geht viel weiter über diese einfache Konzepte hinaus und bietet eine konsistente Programmierschnittstelle für eine Vielzahl von Geräten, die auf Sprachleitungen, ISDN-Leitungen und privaten Austauschleitungen funktionieren. Die Geräte umfassen Modems, Fax-Modems, Sprachmodems, rechnergesteuerte Telefonanlagen und vieles mehr.

TAPI bietet Dienste zum Plazieren ausgehender Anrufe, Akzeptieren ankommender Anrufe und zur Verwaltung von Anrufen und Geräten. Was TAPI nicht tut, ist die Behandlung der Datenströme, das heißt der Daten, die während eines Anrufes ausgetauscht werden. Zum Beispiel, wenn TAPI für Sprachanruf eingesetzt wird, handelt es sich nicht um TAPI, sondern um Sie, den Menschen, der da spricht.

Ähnlich verhält es sich bei Datenanrufen: die Anwendung bemächtigt sich des Geräts und führt Ein-/Ausgabe mit Standard-Win32-Dateifunktionen aus.

44.1.1 Unterstützte TAPI: die einfachste TAPI-Anwendung

Bevor wir tiefer in die Architektur von TAPI eintauchen, werfen Sie einen Blick auf das einfache Programm aus Listing 44.1. Es ist so einfach, wie eine TAPI-Anwendung nur sein kann. Dieses Programm akzeptiert ein einziges Kommandozeilenargument, eine Telefonnummer, und wählt die Nummer für einen Sprachanruf.

Um diese Anwendung von der Kommandozeile zu kompilieren, geben Sie folgende Zeile ein:

```
CL DIAL.C TAPI32.LIB
```

*Listing 44.1:
Die einfachste
TAPI-Anwendung*

```
#include <windows.h>
#include <stdio.h>
#include <tapi.h>

void main(int argc, char *argv[])
{
    if (argc != 2) printf("Aufruf: %s Telefonnummer\n",
                        argv[0]);
    else
    {
        printf("Wähle %s...", argv[1]);
        tapiRequestMakeCall(argv[1], NULL, NULL, NULL);
    }
}
```

Die eigentliche Anwahl wird von der Standardanwahl-Kontrollanwendung durchgeführt. Ein Beispiel für eine Anwahl-Kontrollanwendung ist die Wahlhilfe, die beispielsweise als Teil von Windows 95 geliefert wird.

Die Funktion `tapiRequestMakeCall`, die in diesem Programm eingesetzt wird, ist ideal für die Verwendung in Skripts. Zum Beispiel kann ein Aufruf dieser Funktion in Visual Basic als externe DLL aufgenommen werden.

Die Funktion `tapiRequestMakeCall` ist ein Teil der unterstützten Telefonie (Assisted Telephony)-Features von TAPI. In der aktuellen Version von TAPI gibt es nur eine weitere Funktion, `tapiGetLocationInfo`, die zum Ermitteln des Ländercodes und des Ortcodes für den Sitz des Anwenders verwendet werden kann.

44.1.2 TAPI-Konzepte

TAPI bietet eine Reihe von persönlichen Telefoniediensten. Telefonie bezieht sich in diesem Kontext im allgemeinen auf Technologien, die Rechner mit dem Telefonnetz verbinden.

TAPI-Dienste werden für alle Aspekte des Einsatzes des Telefonnetzes angeboten:

- ➔ Verbindung zum Netz,
- ➔ Anrufe tätigen und entgegennehmen,
- ➔ Anrufe verwalten (etwa Anrufe weiterleiten, Konferenzschaltungen aufbauen),
- ➔ Einsatz der Anrufernummern-Identifizierung für ankommende Gespräche
- ➔ und so weiter.

TAPI-Dienste können in Grunddienste, zusätzliche und erweiterte Dienste aufgeteilt werden.

- ➔ Grunddienste werden im allgemeinen von allen Geräten unterstützt.
- ➔ Zusätzliche Dienste können nur auf speziellen Geräten zur Verfügung stehen.
- ➔ Erweiterte Dienste sind anbieterspezifisch.

Zum Beispiel kann TAPI einen Anruf auf allen Telefonleitungen tätigen. Funktionen zur Verwaltung, wie das Weiterleiten von Anrufen, kann nur auf Geräten verfügbar sein, die eine solche Möglichkeit ausdrücklich bieten, und werden somit als zusätzlicher Dienst betrachtet.

TAPI ist nicht auf das eingeschränkt, was mit dem Kürzel POTS – *POTS* Plain Old Telephone Service – bezeichnet wird. POTS sind analoge Dienste in der LOKALEN SCHLEIFE (die Leitung vom Telefon bis zur nächsten Schaltzentrale). POTS unterstützt Gespräche mit einer Bandbreite von 3,1 kHz oder Datenübertragungen mit Geschwindigkeiten bis 33,6 Kbps mit erweiterten V.34-Modems.

Beachten Sie, daß die neuen 56-Kbps-Modems auf POTS nicht funktionieren; sie erfordern, daß nicht mehr als ein Abschnitt der gesamten Verbindung analoge Leitungen verwendet. Das heißt, Sie können Ihren Internet-Provider auf einer analogen Leitung mit 56 Kbps anrufen, doch muß dieser eine digitale Verbindung zur Schaltzentrale haben.



ISDN Hingegen unterstützt ISDN – Integrated Services Digital Network – bis zu 128 Kbps mit seinem Basic Rate Interface (BRI-ISDN). Die Geschwindigkeit auf PRI-ISDN (Primary Rate Interface) ist viel höher. TAPI unterstützt ISDN als auch andere Verbindungstypen, wie switched 56 oder T1/E1. TAPI kann auch die Features von CENTREX oder Private Branch Exchanges (PBX) nutzen.

44.1.3 TAPI-Geräte

TAPI unterscheidet zwischen Leitungsgeräten und Telefongeräten.

- Ein Leitungsgerät ist die abstrakte Darstellung eines physischen Gerätes, das Ihren Rechner mit dem Telefonnetz verbindet. Beispiele solcher Leitungsgeräte sind Modems, Faxmodems oder ISDN-Karten.
- Ein Telefongerät ist die abstrakte Darstellung eines Geräts, das die Fähigkeiten eines Telefons besitzt. Ein Telefongerät kann einen Lautsprecher haben, ein Mikrofon, Lampen, ein Display, Schalter und so weiter. Ein Telefongerät ist nicht zwangsläufig ein physisches Gerät; eine Software-Emulation, die den Lautsprecher des Rechners, Mikrofon, Soundkarte und ein Sprachmodem benutzt, eine telefonähnliche Schnittstelle auf dem Bildschirm ausgibt, kann auch als Telefongerät bezeichnet werden.

Abbildung 44.1:
TAPI-Geräte

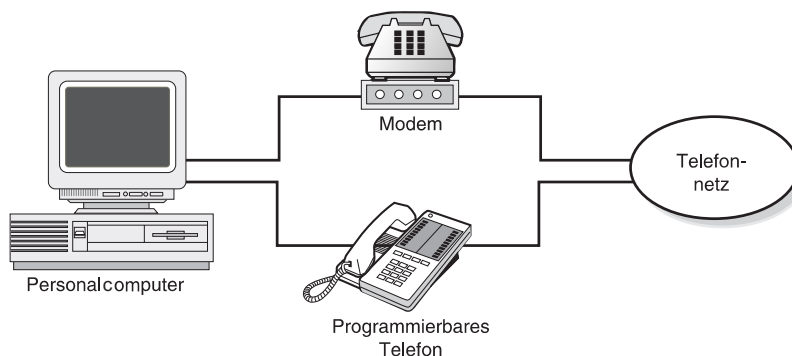


Abbildung 44.1 zeigt eine Grundkonfiguration, bestehend aus einem Leitungsgerät (ein Datenmodem) und einem Telefongerät (ein programmierbares Telefon mit TAPI-Steuerung). Beachten Sie, daß das Vorhandensein eines Telefons nicht bedeutet, daß ein TAPI-Telefongerät existiert. Falls Sie z.B. ein Telefon haben, das zusammen mit einer Wählvorrichtung benutzt wird, die den Hayes-AT-Befehlssatz unterstützt (oder ein Modem als Wählvorrichtung), so wird diese Konfi-

guration als Leitungsgerät dargestellt. Ein Telefongerät wird verwendet, wenn TAPI die Kontrolle über eines der Elemente wie Display, Glocke oder Schalter hat.

Leitungsgeräte bieten alle einen Satz von Grundfunktionen (Basis-Telefonie). Dagegen sind alle Funktionen der Telefongeräte ein Teil der zusätzlichen Telefonie. Das ist so, weil es kein Minimum an Funktionen gibt, die von einem Telefongerät erwartet werden.

44.1.4 TAPI-Adressen

TAPI unterscheidet zwischen dem Begriff einer Leitung und dem einer Adresse. Die Leitung ist eine physische Entität; die Adresse ist zum Beispiel eine Telefonnummer, die der Leitung zugewiesen ist.

Obwohl die meisten POTS-Leitungen mit einer einzigen Telefonnummer belegt sind, ist dies nicht immer der Fall. Zum Beispiel kann eine gewöhnliche Telefonleitung mit mehreren Telefonnummern konfiguriert werden, dank der Möglichkeit der Telefongesellschaft, selektiv zu läuten. Auf digitalen Leitungen ist die Verwendung mehrerer Adressen auf derselben Leitung weiter verbreitet.

*Leitung:
Telefonnummer*

Ortsabhängigkeit von Telefonnummer

Ein besonderes Merkmal der Telefonnummern ist die Tatsache, daß ihr Format ortsabhängig ist. Nehmen wir zum Beispiel eine Nummer in Ottawa, wie 613-555-1234. Wenn ich diese Nummer lokal anrufe, muß ich nur die sieben Stellen 555-1234 wählen. Rufe ich diese Nummer aus New York City an, so muß ich 1-613-555-1234 wählen. Die gleiche Nummer aus Budapest, Ungarn gewählt, erfordert 00w1-613-555-1234, wobei der Buchstabe w das Warten auf einen zweiten Wahlton bedeutet. Dieselbe Nummer von einem Gerät aus anzurufen, das mit dem PBX einer ortsansässigen Firma verbunden ist, würde vielleicht die Zahlen 8-613-555-1234 erfordern. Diese Unterschiede werden vor allem auf tragbaren Rechnern relevant.

TAPI leistet hervorragende Arbeit beim Übersetzen von Telefonnummern. Das Herzstück für diese Fähigkeit ist das KANONISCHE ADRESS-FORMAT.

Die Syntax für eine kanonische Adresse sieht folgendermaßen aus:

*Kanonische
Adresse*

```
+LandesCode Leerraum [(RegionCode) Leerraum] Nummer  
[| SubAdresse] [^ Name] CRLF
```

Zum Beispiel lautet das kanonische Format für die Nummer aus Ottawa 555-1234 folgendermaßen:

```
+1 (613) 555-1234
```

Wählbare Adresse Die kanonische Adresse unterscheidet sich von einer wählbaren Adresse. Wählbare Adressen sind jene, die nicht mit dem Zeichen + beginnen. Von diesen Nummern wird vorausgesetzt, daß sie ohne Änderungen auf einer gegebenen Leitung gewählt werden können. Eine kanonische Adresse kann mit der TAPI-Funktion `lineTranslateAddress` in eine wählbare Adresse umgewandelt werden. Die Syntax für eine wählbare Nummer lautet:

WählbareNummer [| SubAdresse] [^ Name] CRLF

Wählbare Adressen können außer Ziffern auch die DMTF-Symbole A-D, # und * enthalten sowie einige Wahlmodifizierzeichen.

Hayes-AT-Befehlssatz

Wahlmodifizierzeichen beruhen auf dem Hayes-AT-Befehlssatz und umfassen die Zeichen aus Tabelle 44.1.

Tabelle 44.1:
Wahlmodifizier-
zeichen in wähl-
baren Adressen

Wahlmodifizierzeichen	Beschreibung
!	Betätige Auflegeschalter
P oder p	Pulswahl für folgende Nummern
T oder t	Tonwahl für folgende Nummern
,	Pause
w or W	Warte auf Tonsignal
@	Warte auf stille Antwort
\$	Warte auf Abrechnungssignal
;	Gibt unvollständige Nummer an

44.2 TAPI-Software-Architektur

Das Herzstück von TAPI ist die TAPI-DLL, die dynamische Linkbibliothek, die den Anwendungen TAPI-Dienste zur Verfügung stellt. Diese DLL dient als Schicht zwischen Telefonie-Anwendungen und Anbietern von TAPI-Diensten. Ein solcher Anbieter ist der UNIMODEM-Treiber; dieser universelle Modemtreiber wird mit Windows 95 geliefert und bietet TAPI-Dienste für Modems, die zum Hayes-AT-Befehlssatz kompatibel sind.

Diese Grundarchitektur von TAPI wird in Abbildung 44.2 gezeigt. Außer der TAPI-DLL und den Anbietern von TAPI-Diensten (Treibern) gibt es eine weitere wichtige, wenn auch unsichtbare Komponente, das Programm *tapiexe.exe*. Dieses Programm spielt eine wichtige Rolle, wenn TAPI Benachrichtigungen an die anrufende Anwendung über Rückruffunktionen sendet.

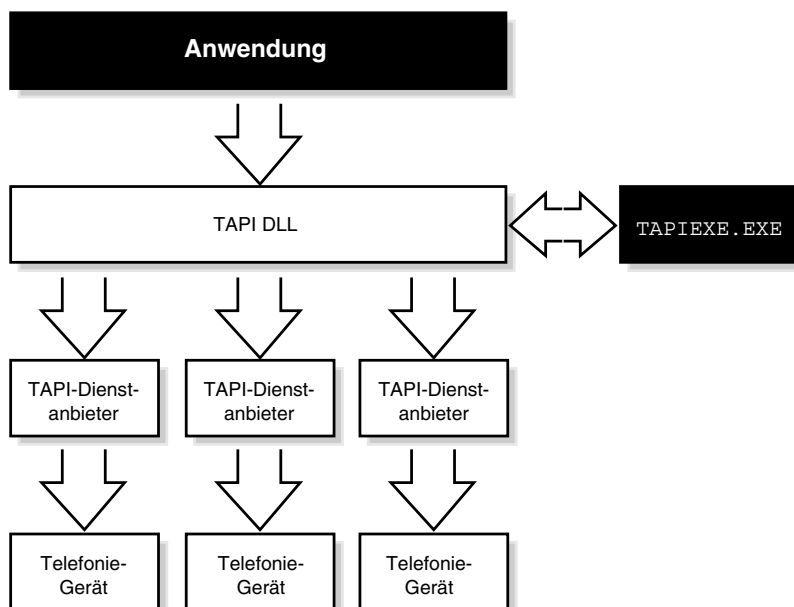


Abbildung 44.2:
Die
TAPI-Software-
Architektur

44.2.1 Synchrone und asynchrone Operationen

Viele der TAPI-Funktionen sind synchron; das heißt, wenn die TAPI-Funktion zurückkehrt, ist die Operation entweder beendet oder abgebrochen, wobei ein Fehlercode zurückgegeben wird.

Trotzdem sind einige TAPI-Funktionen asynchron; die TAPI-Funktion kehrt zurück und gibt an, ob die Operation erfolgreich angestoßen wurde, die Operation selber wird aber in einem anderen Thread ausgeführt, und die Anwendung wird über eine Rückruffunktion benachrichtigt. Die Rückruffunktion wird registriert, wenn die TAPI-Bibliothek initialisiert wird.

Der eigentliche Rückrufmechanismus verdient eine genauere Untersuchung, vor allem deshalb, weil er einige Konsequenzen auf die Funktionsweise der TAPI-Funktionen hat.

Wenn ein Dienstanbieter eine Benachrichtigung durchführen will, ruft er die TAPI-DLL auf. Tatsächlich fordert er, daß die DLL alle betroffenen Anwendungen benachrichtigt, daß ein bestimmtes Ereignis stattgefunden hat. Dieser erste Aufruf der TAPI-DLL findet im Ausführkontext des Dienstanbieters statt.

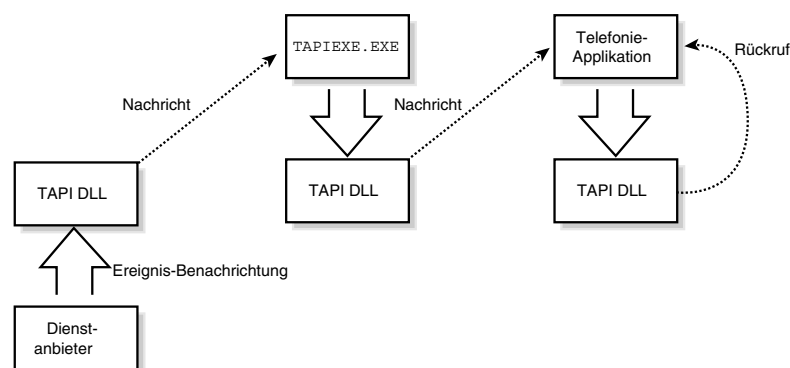
Die TAPI-DLL sendet ihrerseits eine Meldung an *tapiexe.exe*. Dieses ausführbare Programm ruft die TAPI-DLL selber auf, diesmal in sei-

nem eigenen Ausführkontext. Dieser Aufruf weist die TAPI-DLL an, eine Windows-Nachricht an die Anwendungen, die geändert werden müssen, zu schicken.

Wenn die Anwendung die Nachricht in ihrer Nachrichtenschleife erhält und verarbeitet, wird die Nachricht wieder an die TAPI-DLL verteilt, diesmal im Ausführkontext der Anwendung. Die TAPI-DLL kann daraufhin die registrierte Rückruffunktion der TAPI aufrufen, um die Anwendung über ein TAPI-Ereignis zu benachrichtigen.

Der TAPI-Benachrichtigungsmechanismus wird in Abbildung 44.3 gezeigt.

Abbildung 44.3:
Verarbeitung
von TAPI-Ereig-
nissen



Dieser Mechanismus hat wichtige Folgen für die Architektur der TAPI-Anwendungen.

- Einmal stellt das hier beschriebene Szenario klar, daß TAPI-Anwendungen eine Nachrichtenschleife haben müssen, um Benachrichtigungen korrekt zu verarbeiten. Obwohl der Einsatz einer Rückruffunktion andeuten könnte, daß eine Nachrichtenschleife überflüssig wäre, ist das nicht der Fall; die Rückruffunktion wird erst aufgerufen, nachdem die Anwendung eine Windows-Nachricht erhält, die von der TAPI-DLL verarbeitet wird. Ohne eine Nachrichtenschleife in der Anwendung wird der TAPI-Aufruf nie erfolgreich beendet.
- Eine weitere Folge betrifft den Einsatz mehrfacher Threads. Es ist wichtig zu verstehen, daß für das erwartungsgemäße Funktionieren von TAPI-Threads, die eine asynchrone TAPI-Funktion aufrufen, eine Nachrichtenschleife erforderlich ist. Die Rückruffunktion wird im Kontext des Threads aufgerufen; dies kann nur geschehen, wenn der Thread Windows-Nachrichten verarbeitet.

- ➡ Obwohl die Notwendigkeit einer Nachrichtenschleife den Einsatz von TAPI in Konsolenanwendungen nicht vollständig ausschließt, schränkt sie diesen in einigen Punkten ein. Die Konsolenanwendung muß eine Nachrichtenschleife haben, die Windows-Nachrichten verarbeitet und verteilt. Das weiter unten in diesem Kapitel folgende Beispiel (Listing 44.2) zeigt den Einsatz dieser Technik.

44.2.2 Strukturen mit variabler Länge

In TAPI werden Strukturen mit variabler Länge häufig verwendet. Diese Strukturen stellen Daten dar, deren Länge variabel ist, wie optionale Felder oder Strings.

Alle Strukturen mit variabler Länge in TAPI verwenden die Strukturelemente

- ➡ `dwTotalSize`,
- ➡ `dwNeededSize` und
- ➡ `dwUsedSize`.

Wenn eine TAPI-Funktion aufgerufen wird, von der erwartet wird, daß sie Daten in einer solchen Struktur zurückgibt, ist Ihre erste Aufgabe, die Struktur zuzuweisen und ihr Element `dwTotalSize` vor dem Aufruf zu füllen.

Die TAPI-Dokumentation bezieht sich auf Strukturen dieser Art als »flattened«. Statt über Zeiger referenziert zu werden, sind zusätzliche Datenfelder und Felder variabler Länge einfach an das Ende der Struktur angehängt. Felder variabler Länge werden in der Struktur über ein Offset und einen Längenparameter angesprochen; das Offset gibt die Anfangsposition des Feldes in Bytes relativ zum Anfang der Struktur an; die Länge ist die Länge des Feldes in Bytes.

Angenommen, eine TAPI-Funktion namens `tapiStrangeFunc` gibt Daten variabler Länge in einer Struktur `VARSTRUCT` zurück. Diese Struktur wird folgendermaßen deklariert:

Beispiel

```
typedef struct
{
    DWORD dwTotalSize;
    DWORD dwNeededSize;
    DWORD dwUsedSize;
    // hier weitere Elemente mit fester Laenge
    DWORD dwVarItem1Size;
    DWORD dwVarItem1Offset;
    // hier weitere Elemente mit fester Laenge
    DWORD dwVarItem2Size;
    DWORD dwVarItem2Offset;
} VARSTRUCT, FAR *LPVARSTRUCT;
```


dwTotalSize Bevor `tapiStrangeFunc` aufgerufen wird, müssen Sie eine `VARSTRUCT`-Struktur reservieren. Obwohl Sie die Strukturvariable als automatische Variable zuweisen könnten, ist dies nicht zu empfehlen. Verwenden Sie statt dessen folgenden Mechanismus:

```
LPVARSTRUCT pVarStruct;
pVarStruct = (LPVARSTRUCT)malloc(sizeof(pVarStruct));
pVarStruct->dwTotalSize = sizeof(VARSTRUCT);
tapiStrangeFunc(pVarStruct);
```

Die Beschreibung von `tapiStrangeFunc` sagt Ihnen, daß diese Funktion ein `DWORD`-Element zur Struktur hinzufügt. Sicher erfordert dieses zusätzliche Element zusammen mit den beiden Objekten variabler Länge, identifiziert durch die Elemente `dwVarItem1Size/dwVarItem1Offset` und `dwVarItem2Size/dwVarItem2Offset`, mehr Speicher als ursprünglich reserviert.

dwNeedSize Wenn `tapiStrangeFunc` zurückkehrt, wird diese Tatsache vom Wert des Strukturelements `dwNeedSize` angegeben. Dieses Element wird anzeigen, daß für die Rückgabe aller Werte zusätzlicher Speicher erforderlich ist. Eine mögliche Reaktion darauf wäre das Reallokieren der Struktur und ein weiterer Aufruf von `tapiStrangeFunc`:

```
pVarStruct = (LPVARSTRUCT)malloc((LPVOID)pVarStruct,
                                   pVarStruct->dwNeededSize);
pVarStruct->dwTotalSize = pVarStruct->dwNeededSize;
tapiStrangeFunc(pVarStruct);
```

Wenn dieser Aufruf von `tapiStrangeFunc` zurückkehrt, zeigt `pVarStruct` auf eine Struktur wie in Abbildung 44.4.

dwUsedSize Das Feld `dwUsedSize` ist von geringerer Bedeutung; es kommt ins Spiel, wenn TAPI nicht alle Strukturelemente ausfüllen konnte (`dwTotalSize` war kleiner als `dwNeededSize`). In diesem Fall läßt TAPI das Feld einfach leer, statt ein Feld variabler Länge zu kürzen.

44.3 TAPI-Dienste

Außer den unterstützten TAPI-Diensten, die wir bereits gesehen haben, bietet TAPI Dienste, die in drei Kategorien fallen:

- Grundlegende Telefonie,
- Zusätzliche Telefonie und
- Erweiterte Telefonie.

Grundlegende Telefonie umfaßt alle Funktionen, die von einer POTS-Leitung erwartet werden können. Dieses Minimum an Funktionen muß von allen Dienstanbietern unterstützt werden.

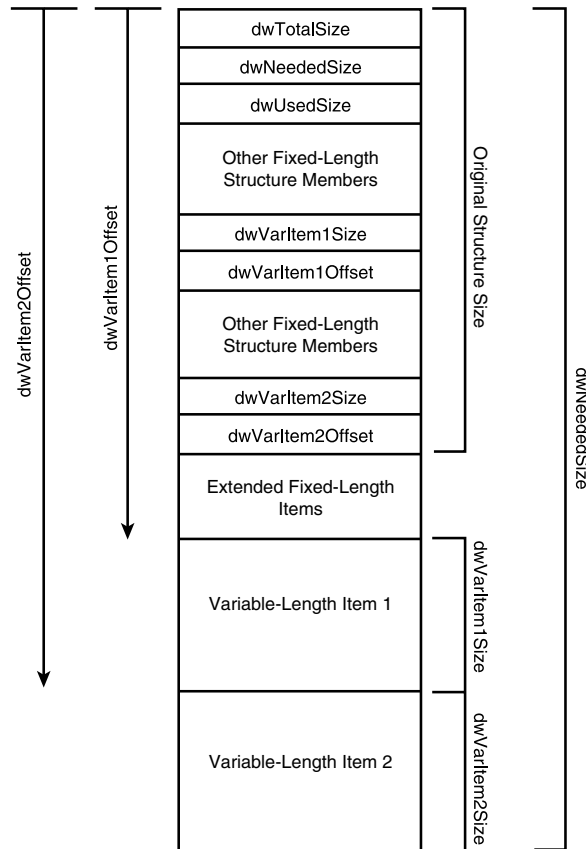


Abbildung 44.4:
Beispiel einer
TAPI-Struktur
variabler Länge

Zusätzliche Telefonie umfaßt alle Standard-TAPI-Dienste, die nicht im Funktionsumfang der Grundlegenden Telefonie enthalten sind. Das sind zusätzliche Dienste, die auf den meisten PBX anzutreffen sind, wie Zurückstellen, Weiterleiten, Konferenzschaltungen usw. Eine Anwendung kann den Satz zusätzlicher Dienste, die von einem bestimmten Gerät unterstützt werden, mit den Funktionen `lineGetDevCaps`, `lineGetAddressCaps` oder `phoneGetDevCaps` abfragen.

Erweiterte TAPI-Dienste sind anbieterspezifisch. Diese umfassen alle gerätespezifischen TAPI-Erweiterungen. TAPI bietet die nötigen Mechanismen zur Erweiterung der Dienste über Strukturen variabler Länge und Funktionen, durch die Anbieter die Anwendungen über die von ihnen unterstützten erweiterten Dienste informieren können.



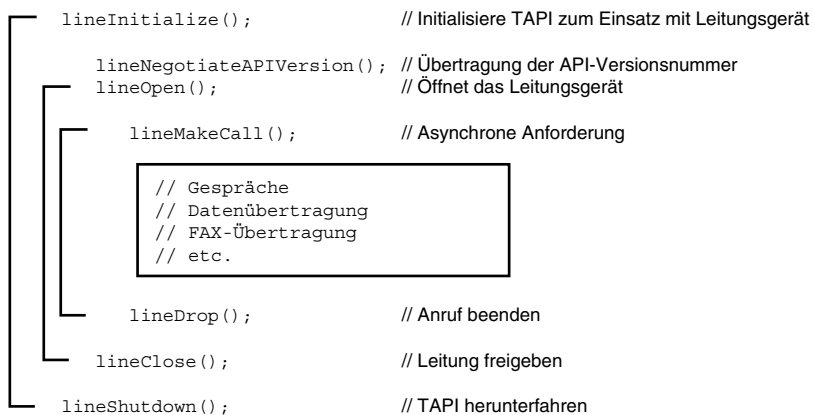
Da es kein Minimum an Diensten gibt, deren Unterstützung von Telefongeräten vorausgesetzt wird, befinden sich alle Telefongerät-Dienste in der Kategorie der zusätzlichen Telefonie.

44.3.1 Das TAPI-Programmiermodell

Das TAPI-Programmiermodell für Leitungsgeräte

Das grundlegende TAPI-Programmiermodell für Leitungsgeräte ist in Abbildung 44.5 dargestellt.

Abbildung 44.5:
Das TAPI-Pro-
grammiermo-
dell: Aufrufe in
einer typischen
TAPI-Anwen-
dung



lineInitialize Alle TAPI-Anwendungen, die Leitungsgeräte verwenden, beginnen mit einem Aufruf von `lineInitialize`. Dieser Aufruf initialisiert TAPI für den Einsatz mit Leitungsgeräten. Beachten Sie, daß dieser Aufruf nur dann erfolgen sollte, wenn die Anwendung tatsächlich die TAPI-Dienste verwenden soll; ein unnötiger Aufruf kann wertvolle TAPI-Ressourcen verbrauchen und die Anwendung zum Absturz bringen.

Einer der Parameter von `lineInitialize` ist die Adresse einer Rückruf-funktion. Durch diese Funktion wird die Anwendung über das Beenden asynchroner Funktionsaufrufe und andere TAPI-Ereignisse informiert.

lineNegotiate-APIVersion Bevor eine Anwendung ein bestimmtes Leitungsgerät öffnen kann, muß sie eine TAPI-Versionsnummer erhalten, indem sie `lineNegotiateAPIVersion` aufruft. Durch diesen Aufruf können sich die Anwendung und der Geräte-Anbieter auf eine Versionsnummer einigen, die von beiden unterstützt wird. Beachten Sie, daß die aktuelle TAPI-Versions-nummer 1.4 beträgt; nur diese und eine frühere 16-Bit-Version sind im Umlauf.

lineOpen Das Leitungsgerät wird durch den Aufruf von `lineOpen` geöffnet.

Nachher können Anwendungen eine Vielfalt von Funktionen, die das geöffnete Gerät verwenden, aufrufen. Ein Beispiel ist die Funktion `lineMakeCall`, die zum Tätigen eines Anrufs auf einem Leitungsgerät verwendet wird. Diese Funktion ist auch ein Beispiel einer asynchronen Funktion; sie kehrt sofort zurück, nachdem die Anforderung für den Anruf erfolgreich gestellt wurde. Die Anwendung wird über ihre Rückruffunktion benachrichtigt, daß die Anforderung erfolgreich gestellt wurde.

Nachdem der Aufruf erfolgt ist, kann eine Anwendung eine Vielzahl von Dingen mit der Leitung tun, je nach beabsichtigter Funktion. Eine Reihe von weiteren Funktionen können eingesetzt werden, um Informationen über die Leitung zu bekommen, sie zu konfigurieren und Adressen zu manipulieren. Zusätzliche Funktionen können zur Weiterleitung des Anrufes, zum Versetzen in den Wartezustand, zum Zustandebringen von Konferenzschaltungen und so weiter eingesetzt werden.

Ein besonderes Feature von TAPI unterstützt Datenkommunikationsanwendungen auf ganz spezielle Art. Solche Anwendungen können die Funktion `lineGetID` verwenden, um ein Handle auf das Kommunikationsgerät zu erhalten. Dieses Handle wird von TAPI für überlappende E/A geöffnet und kann von der Anwendung zum Datenaustausch mit einem entfernten Host eingesetzt werden.

Der Aufruf von `lineMakeCall` ist nicht der einzige Weg, um einen Anruf zustandezubringen. Anwendungen können auch ein Handle erhalten, indem sie ankommende Anrufe annehmen. Anwendungen, die zur Annahme ankommender Anrufe befähigt sind, werden über solche Anrufe von ihrer Rückruffunktion benachrichtigt.

Wenn die Anwendung den Anruf abbrechen will, kann sie die Funktion `lineDrop` verwenden. Ein Anruf kann auch vom anderen Ende aus abgebrochen werden; in diesem Fall wird die Anwendung von ihrer Rückruffunktion benachrichtigt.

Wenn die Anwendung das Leitungsgerät nicht mehr braucht, sollte sie die Funktion `lineClose` aufrufen, um das Gerät zu schließen.

Die Funktion `lineShutdown` kann verwendet werden, um die Sitzung der Anwendung mit TAPI zu beenden.

Programmiermodell für Telefongeräte

Das Programmiermodell für Telefongeräte ist ähnlich. Es gibt ebenso die Schritte TAPI initialisieren, Versionsnummer aushandeln und Öffnen des Geräts. Es gibt kein Äquivalent zum Plazieren eines Anrufs; Telefongerätfunktionen dienen der Manipulation der verschiedenen Komponenten eines Telefons.

Anwendungen, die anbieterspezifische Dienste der erweiterten Telefonie einsetzen möchten, müssen die Funktionen `lineNegotiateExtVersion` oder `phoneNegotiateExtVersion` verwenden, um die erweiterte Versionsnummer auszuhandeln. Gerätespezifische Funktionen können ausgeführt werden, indem sie die Escape-Funktionen `lineDevSpecific`, `lineDevSpecificFeature` (für Schaltfunktionen) und `phoneDevSpecific` aufrufen.

44.3.2 TAPI-Media-Modi

TAPI bietet zwei Konzepte, die die Qualität der Dienste einer Leitung und den Typ des Anrufs wiedergeben.

Der Träger-Modus gibt die Qualität des Dienstes an.

Der Sprachträger-Modus (`LINEBEARERMODE_VOICE`) z.B. zeigt eine POTS-Leitung mit 3,1 KHz analoger Bandbreite und keine Vorsichtsmaßnahmen für Datenintegrität. Weitere Trägermodi beschreiben ISDN oder andere Datenleitungen.

Der Medien-Modus bestimmt die Art des Anrufs. Zum Beispiel ist es auf einer Sprachleitung möglich, Sprach- oder Datenanrufe zu tätigen; diese entsprechen den Medienmodi `LINEMEDIAMODE_INTERACTIVEVOICE` oder `LINEMEDIAMODE_DATAMODEM`.

44.3.3 Mehrfachanwendungen

Die TAPI-Architektur erlaubt die Koexistenz mehrerer Anwendungen. Dies ermöglicht es zum Beispiel einer TAPI-Fax-Anwendung, eine Leitung nach ankommenden Fax-Übertragungen zu überwachen, und gleichzeitig anderen TAPI-Anwendungen, wie einer Datenkommunikationsanwendung, die gleiche Leitung für Anrufe zu nutzen.

Besitzerrechte

Anfangs wird das Besitzerrecht für einen Anruf einer Anwendung zugeordnet; es ist entweder die Anwendung, die den Anruf veranlaßt hat, oder die Anwendung, die den ankommenden Anruf empfängt.

lineHandoff Eine Anwendung kann die Besitzerrechte über einen Anruf an eine andere Anwendung durch die Funktion `lineHandoff` übertragen. Die Originalanwendung behält weiterhin den Anruf in ihrem Besitz. Sie kann dann wählen, Mitbesitzer des Anrufs zu bleiben (obwohl das nicht empfohlen wird), den Handle freizugeben und so zu signalisieren, daß sie an dem Anruf nicht weiter interessiert ist, oder `lineSetCallPrivilege` aufzurufen, um zum Anrufmonitor zu werden. Ein Anrufmonitor ist eine Anwendung, die die Existenz des Anrufs nicht beeinflussen kann, jedoch Einzelheiten über den Anruf aufnehmen kann (Logging).

Wenn ein Anruf von mehreren Anwendungen »mitbesessen« wird, läßt TAPI zu, daß sich die Anwendungen gegenseitig behindern..



Probing

Bei der Behandlung ankommender Anrufe können Anwendungen durch PROBING die Art des Anrufs bestimmen. Probing kann verwendet werden, um beispielsweise festzustellen, ob es sich um einen Daten-, Fax- oder Sprachanruf handelt. Probing wird gewöhnlich von Anwendungen ausgeführt, obwohl einige Dienstanbieter dahingehend konfiguriert werden können, daß sie Anrufe entgegennehmen und an die entsprechende Anwendung weiterleiten. Beachten Sie, daß die TAPI keine Anwendungen starten kann, um bestimmte Anruftypen zu behandeln.

Anwendungen können von laufenden Anrufen erfahren, indem sie `lineGetNewCalls` aufrufen. Über diese Funktion kann eine Anwendung Handles mit Monitor-Vorrechten für alle laufenden Anrufe erhalten.

Anwendungen können miteinander über die Funktion `lineSetAppSpecific` kommunizieren. Mit dieser Funktion können sie das Feld `dwAppSpecific` der Struktur `LINECALLINFO` setzen. Andere Anwendungen, die den Anruf besitzen oder als Monitor überwachen, werden durch eine `LINE_CALLINFO`-Meldung informiert.

44.4 Beispiel einer Datenkommunikation

Lassen Sie uns TAPI in der Praxis umzusetzen, indem wir eine früher entwickelte Konsolenanwendung umschreiben. Diese einfache Anwendung öffnet einen Kommunikations-Port und verwendet überlappende E/A-Operationen.

In der Originalversion öffnete die Anwendung einfach den Port, ohne einen Anruf zu tätigen. Es lag am Anwender, mit den entsprechenden AT-Befehlen den Anruf vorzunehmen. Der Port war in der Anwendung hardkodiert.

In der TAPI-Version aus Listing 44.2 wird der Anruf über die TAPI-Funktion `lineMakeCall` getätigt. Bevor das geschieht, wird dem Anwender die Gelegenheit gegeben, ein TAPI-Gerät zu wählen.

Um diese Anwendung von der Kommandozeile aus zu kompilieren, geben Sie folgende Zeile ein:

```
CL TTY.C TAPI32.LIB USER32.LIB
```

Die USER-Bibliothek wird wegen der Referenzen auf die Windows-Funktionen GetMessage und DispatchMessage benötigt.

Listing 44.2:
Ein einfaches
TAPI-Datenkom-
munikationspro-
gramm

```
#include <windows.h>
#include <tapi.h>
#include <stdio.h>

volatile BOOL bConnected = FALSE;
VOID FAR PASCAL lineCallback(DWORD hDevice, DWORD dwMsg,
                             DWORD dwCallbackInstance,
                             DWORD dwParam1,
                             DWORD dwParam2, DWORD dwParam3)
{
    if (dwMsg == LINE_CALLSTATE &&
        dwParam1 == LINECALLSTATE_CONNECTED)
        bConnected = TRUE;
}

LINEDEVCAPS *GetDevCaps(HLINEAPP hLineApp, DWORD dwDeviceID,
                        LPDWORD lpdwAPIVersion)
{
    LINEDEVCAPS *pLineDevCaps;
    LINEEXTENSIONID extensionID;

    lineNegotiateAPIVersion(hLineApp, dwDeviceID, 0x10004,
                           0x10004, lpdwAPIVersion, &extensionID);
    pLineDevCaps = malloc(sizeof(LINEDEVCAPS));
    pLineDevCaps->dwTotalSize = sizeof(LINEDEVCAPS);
    lineGetDevCaps(hLineApp, dwDeviceID, *lpdwAPIVersion, 0,
                  pLineDevCaps);
    if (pLineDevCaps->dwNeededSize > pLineDevCaps->dwTotalSize)
    {
        pLineDevCaps =
            realloc(pLineDevCaps, pLineDevCaps->dwNeededSize);
        pLineDevCaps->dwTotalSize = pLineDevCaps->dwNeededSize;
        lineGetDevCaps(hLineApp, dwDeviceID, *lpdwAPIVersion, 0,
                      pLineDevCaps);
    }
    return pLineDevCaps;
}

HANDLE SelectTAPIDevice(HLINEAPP hLineApp, DWORD dwNumDevs,
                       LPHLINE lphLine, LPHCALL lphCall)
{
    LINEDEVCAPS *pLineDevCaps;
    DWORD dwDeviceID;
    DWORD dwAPIVersion;
    DWORD i;
    LINECALLPARAMS lineCallParams;
    LPVARSTRING lpDeviceID;
    MSG msg;
    char szNumber[81];

    for (i = 0; i < dwNumDevs; i++)
    {
        pLineDevCaps = GetDevCaps(hLineApp, i, &dwAPIVersion);
        if (pLineDevCaps->dwMediaModes & LINEMEDIAMODE_DATAMODEM)
            printf("%d: %s\n", i,
                  (char*)pLineDevCaps + pLineDevCaps
                    - dwLineNameOffset);
        free(pLineDevCaps);
    }
}
```

```

    }
    dwDeviceID = -1;
    while (dwDeviceID < 0 || dwDeviceID >= dwNumDevs)
    {
        printf("Select device: ");
        scanf("%d", &dwDeviceID);
        if (dwDeviceID < 0 || dwDeviceID >= dwNumDevs) continue;
        pLineDevCaps =
            GetDevCaps(hLineApp, dwDeviceID, &dwAPIVersion);
        if (!(pLineDevCaps->dwMediaModes
            & LINEMEDIAMODE_DATAMODEM))
        {
            dwDeviceID = -1;
            free(pLineDevCaps);
        }
    }
    printf("Enter telephone number: ");
    scanf("%s", szNumber);
    printf("Dialing %s on %s...", szNumber,
        (char *)pLineDevCaps +
            pLineDevCaps->dwLineNameOffset);
    free(pLineDevCaps);
    lineOpen(hLineApp, dwDeviceID, lphLine, dwAPIVersion, 0, 0,
        LINECALLPRIVILEGE_NONE, LINEMEDIAMODE_DATAMODEM,
        NULL);
    memset(&lineCallParams, 0, sizeof(LINECALLPARAMS));
    lineCallParams.dwTotalSize = sizeof(LINECALLPARAMS);
    lineCallParams.dwMinRate = 2400;
    lineCallParams.dwMaxRate = 57600;
    lineCallParams.dwMediaMode = LINEMEDIAMODE_DATAMODEM;
    lineMakeCall(*lphLine, lphCall, szNumber, 0, &lineCallParams);
    while (!bConnected)
        if (GetMessage(&msg, NULL, 0, 0)) DispatchMessage(&msg);
    putchar('\n');
    lpDeviceID = malloc(sizeof(VARSTRING));
    lpDeviceID->dwTotalSize = sizeof(VARSTRING);
    lineGetID(0, 0, *lphCall, LINECALLSELECT_CALL, lpDeviceID,
        "comm/datamodem");
    if (lpDeviceID->dwNeededSize > lpDeviceID->dwTotalSize)
    {
        lpDeviceID = realloc(lpDeviceID,
            lpDeviceID->dwNeededSize);
        lpDeviceID->dwTotalSize = lpDeviceID->dwNeededSize;
        lineGetID(0, 0, *lphCall, LINECALLSELECT_CALL,
            lpDeviceID, "comm/datamodem");
    }
    return *((LPHANDLE)((char *)lpDeviceID + sizeof(VARSTRING)));
}

void main(void)
{
    HLINEAPP hLineApp;
    HLINE hLine;
    HCALL hCall;
    DWORD dwNumDevs;
    HANDLE hConIn, hConOut, hCommPort;
    HANDLE hEvents[2];
    DWORD dwCount;
    DWORD dwWait;
    COMMTIMEOUTS ctmoCommPort;
    DCB dcbCommPort;

    OVERLAPPED ov;
    INPUT_RECORD irBuffer;

```



```

BOOL fInRead;
char c;
int i;

lineInitialize(&hLineApp, GetModuleHandle(NULL),
               lineCallback,
               "Test TAPI Application", &dwNumDevs);
hCommPort =
    SelectTAPIDevice(hLineApp, dwNumDevs, &hLine, &hCall);
hConIn = CreateFile("CONIN$", GENERIC_READ | GENERIC_WRITE,
                   FILE_SHARE_READ, NULL, OPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL, 0);
SetConsoleMode(hConIn, 0);
hConOut = CreateFile("CONOUT$", GENERIC_WRITE,
                    FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL, 0);
ctmoCommPort.ReadIntervalTimeout = MAXDWORD;
ctmoCommPort.ReadTotalTimeoutMultiplier = MAXDWORD;
ctmoCommPort.ReadTotalTimeoutConstant = MAXDWORD;
ctmoCommPort.WriteTotalTimeoutMultiplier = 0;
ctmoCommPort.WriteTotalTimeoutConstant = 0;
SetCommTimeouts(hCommPort, &ctmoCommPort);
dcbCommPort.DCBlength = sizeof(DCB);
GetCommState(hCommPort, &dcbCommPort);
SetCommState(hCommPort, &dcbCommPort);
SetCommMask(hCommPort, EV_RXCHAR);
ov.Offset = 0;
ov.OffsetHigh = 0;
ov.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
hEvents[0] = ov.hEvent;
hEvents[1] = hConIn;
fInRead = FALSE;
while (1)
{
    if (!fInRead)
        while (ReadFile(hCommPort, &c, 1, &dwCount, &ov))
            if (dwCount == 1)
                WriteFile(hConOut, &c, 1, &dwCount,
                          NULL);

    fInRead = TRUE;
    dwWait =
        WaitForMultipleObjects(2, hEvents, FALSE, INFINITE);
    switch (dwWait)
    {
        case WAIT_OBJECT_0:
            if (GetOverlappedResult(hCommPort, &ov, &dwCount,
                                   FALSE))
                if (dwCount == 1)
                    WriteFile(hConOut, &c, 1, &dwCount,
                              NULL);

            fInRead = FALSE;
            break;
        case WAIT_OBJECT_0 + 1:
            ReadConsoleInput(hConIn, &irBuffer, 1, &dwCount);
            if (dwCount == 1 &&
                irBuffer.EventType == KEY_EVENT &&
                irBuffer.Event.KeyEvent.bKeyDown)
                for (i = 0;
                    i < irBuffer.Event.KeyEvent.wRepeatCount;
                    i++)
                {
                    if (irBuffer.Event.KeyEvent.uChar.AsciiChar)
                    {
                        WriteFile(hCommPort,
                                &irBuffer.Event.KeyEvent.uChar.AsciiChar,

```

```

        1, &dwCount, NULL);
    if (irBuffer.Event.KeyEvent.uChar.AsciiChar
        == 24) goto EndLoop;
    }
}
}
}
EndLoop:
    CloseHandle(ov.hEvent);
    CloseHandle(hConIn);
    CloseHandle(hConOut);
    CloseHandle(hCommPort);
    lineDrop(hCall, NULL, 0);
    lineClose(hLine);
    lineShutdown(hLineApp);
}

```

Die Tatsache, daß dies eine Konsolenanwendung ist, stellte eine besondere Herausforderung dar. Insbesondere war es nötig, eine Nachrichtenschleife zu verwenden, die den TAPI-Rückrufmechanismus zum Funktionieren bringt. Obwohl dieser Ansatz etwas unorthodox ist, veranschaulicht er erstaunlich gut das TAPI-Programmiermodell und seine Fallen.

TAPI initialisieren

Der erste Aufruf in der `main`-Funktion der Anwendung ist die TAPI-Funktion `lineInitialize`.

Als nächstes ruft `main` die Funktion `SelectTAPIDevice` auf; diese Funktion der höheren Ebene fragt den Anwender nach einem TAPI-Gerät und einer Telefonnummer, öffnet das Gerät, führt den Anruf aus und gibt ein `Win32-Handle` zurück, das die Anwendung in anschließenden E/A-Aufrufen verwenden kann.

Anruf ausführen

Die Funktion `SelectTAPIDevice` fragt zuerst alle TAPI-Leitungsgeräte nach dem Namen der Leitung ab. Diese Leitungsnamen werden dem Anwender in Form einer numerierten Liste präsentiert, und er wird aufgefordert, eine davon zu wählen. Beim Ermitteln des Leitungsnamens verwendet `SelectTAPIDevice` eine weitere Funktion, `GetDevCaps`. Beachten Sie den doppelten Aufruf der TAPI-Funktion `lineGetDevCaps`; der erste Aufruf wird verwendet, um die Größe der Struktur, die `lineGetDevCaps` zurückgibt, festzulegen. Der zweite Aufruf wird gemacht, nachdem ein genügend großer Speicherblock allokiert wurde.

Nachdem ein Leitungsgerät in `SelectTAPIDevice` gewählt wurde, wird der Anwender aufgefordert, eine Telefonnummer einzugeben. Diese Nummer wird dann, nachdem die Leitung geöffnet und die entsprechende Struktur initialisiert wurde, in einem Aufruf von `lineMakeCall` verwendet. Da `lineMakeCall` eine asynchrone TAPI-Funktion ist, zeigt

die Rückkehr von dieser Funktion nicht, daß die Anforderung beendet ist. Vor allem ist der Handle, auf das `lphCall` zeigt, noch nicht gültig. Die Anwendung muß warten, bis ihre Rückruffunktion aufgerufen wird, zum Zeichen, daß der Anruf erfolgt ist. Darüber hinaus muß sie sicherstellen, daß der Rückrufmechanismus richtig funktioniert, indem sie eine Nachrichtenschleife ausführt.

lineCallback Die Rückruffunktion `lineCallback` ist extrem einfach; sie wartet bloß auf eine Nachricht `LINE_CALLSTATE`, die angibt, daß der Anruf geschaltet wurde. Der Rest der Anwendung wird über das Beenden des Anrufs benachrichtigt, wenn die globale Variable `bConnected` von der Rückruffunktion auf `TRUE` gesetzt wird. Vor allem ist es diese Änderung, die das Beenden der Meldungsschleife in `SelectTAPIDevice` bewirkt.

lineGetID Wenn `SelectTAPIDevice` über das erfolgreiche Beenden des Anrufs durch diesen Mechanismus benachrichtigt wird, verwendet sie die Funktion `lineGetID`, um ein Handle auf den Kommunikationsport zu erhalten. Beachten Sie, wie `lineGetID` zweimal aufgerufen wird, zuerst um die GröÙer der Datenstruktur, die sie zurückgeben soll, zu ermitteln. Beachten Sie auch, wie ein extra Strukturelement vom Typ `HANDLE` erhalten wird.

Wenn `SelectTAPIDevice` zurückkehrt, gibt sie den Handle des Kommunikationsgeräts an `main` weiter. In `main` wird dieser Handle zum Konfigurieren des Kommunikationsgeräts und der Konsole für E/A und zum Umgang mit bidirektionaler Datenübertragung verwendet. Die Anwendung wird beendet, wenn der Anwender die Tastenkombination `[Strg] + [X]` betätigt. Dann schließt die Anwendung alle Handles, beendet die TAPI-Sitzung und sich selbst.

Ich habe diese Anwendung auf meinem Hauptrechner ausgeführt, der mit zwei Modems ausgestattet ist. Ein internes Faxmodem schließt meinen Rechner an meine Datenleitung an; ein altes externes Pocketmodem schließt ihn an meine Sprechleitung an. Ich verwende dieses Modem einfach als Wahlvorrichtung, doch erweist es sich als sehr nützlich, wenn ich Kommunikationsanwendungen wie diese testen muß. An meine Datenleitung ist auch ein weiterer Rechner über sein Faxmodem angeschlossen. Diese Maschine ist mein Server für Internet-Mail und TCP/IP-Verbindungen und läuft unter Linux. Dieser Server nimmt auch ankommende Anrufe an, so daß ich das Modem auf meiner Sprechleitung verwenden kann, um Anrufe darauf zu tätigen.

Eine Beispielsitzung mit dieser Konfiguration sah folgendermaßen aus:

```
C:\TTY>tty
0: SupraFAXModem 144i
1: Practical Peripherals 2400
Select device: 1
Enter telephone number: 555-1234
```

```
Dialing 555-1234 on Practical Peripherals 2400...
You have reached a private computer system. Calls to this system
are logged using calling party identification (caller ID).
Unauthorized calls violate my privacy, not to mention the law! If
you have not been specifically authorized by me to access this
system, now would be a great time to terminate your connection.
```

Viktor

Welcome to Linux 1.1.37.

```
vttl!login: vttoth
Password:
Last login: Tue Oct 17 01:57:20 on ttyS0
Linux 1.1.37. (Posix).
vttl:~$ ^X
C:\TTY>
```

Diese einfache Anwendung bietet überhaupt keine Fehlerbehandlung. Sie ist unter der Voraussetzung geschrieben, daß der Anruf immer erfolgreich ist. Es werden keine Vorsichtsmaßnahmen getroffen für den Fall, daß Anrufversuche scheitern, und die Anwendung wird in so einem Fall ihre Funktion entsprechend verweigern. Außerdem behandelt die Anwendung den Verlust der Carrier-Leitung nicht. Da sie keine Nachrichtenschleife während des Bestehens der Verbindung ausführt, wird ihre Rückruffunktion nie benachrichtigt, falls das entfernte Ende den Anruf abbricht.



44.5 Zusammenfassung

TAPI, die Microsoft API für Telefonie, bietet persönliche Telefoniedienste für Windows-Anwendungen. TAPI bietet abstrakte Modelle für Leitungsgeräte, die einen Rechner an eine Telefonleitung anschließen, sowie für Telefongeräte, die Telefone mit einigen Komponenten wie Einhängeschalter, Display, Schaltern oder Glocke darstellen und über Programme manipuliert werden können. (Beachten Sie, daß gewöhnliche Telefone nicht auf diese Art manipuliert werden können.)

Ein TAPI-Leitungsgerät stellt immer ein physisches Gerät dar. Demgegenüber kann ein Telefongerät eine Software darstellen, die das Ausgabegerät des Rechners, seine Tastatur und Sound-Hardware benützt, um die Dienste eines Telefongeräts zu unterstützen.

TAPI-Leitungsgeräte sind nicht auf die Darstellung von Geräten eingeschränkt, die an POTS-Leitungen angeschlossen sind. Leitungsgeräte können Hardware darstellen, die an ISDN-Leitungen, T1/E1-Datenleitungen, geschaltete 56-Datenleitungen und andere angeschlossen ist.

TAPI bietet Dienste zum Plazieren ausgehender Anrufe und zur Annahme ankommender Anrufe. Es ist Aufgabe der Anwendung, den Medienstrom, den effektiven Datenfluß während eines Anrufes, zu verwalten. Der Medienstrom ist ein allgemeiner Begriff, der Sprache, Daten, Faxbilder oder andere Informationen, die über eine Telefonleitung fließen, darstellt.

Die TAPI-DLL bildet eine Schicht zwischen den Anwendungen und gerätespezifischen Diensteanbietern (Treibern).

TAPI-Funktionen können synchron oder asynchron ausgeführt werden. Synchrone Funktionen kehren sofort mit einer Meldung über Erfolg oder Mißerfolg zurück. Asynchrone Funktionen andererseits kehren nur zurück, um mitzuteilen, ob eine Anforderung erfolgreich platziert wurde. Anwendungen werden über das Beenden ihrer Anforderung durch eine Rückruffunktion benachrichtigt. Damit der Rückrufmechanismus funktioniert, müssen Anwendungen eine Windows-Nachrichtenschleife ausführen.

Die TAPI-Schnittstelle wird in Begleittelefonie, Grundtelefonie, zusätzliche Telefonie und erweiterte Telefonie eingeteilt.

- ➔ Die Begleittelefonie bietet einen Satz von einfachen Funktionen zum Anrufen. Diese Funktionen sind ideal für den Einsatz in Skriptsprachen, die externe DLLs aufrufen können.
- ➔ Grundtelefonie besteht aus jenen Funktionen für Leitungsgeräte, die alle Diensteanbieter implementieren müssen. Dazu gehören Funktionen zur Annahme und Ausgabe von Anrufen sowie für das Überwachen laufender Anrufe.
- ➔ Zusätzliche Telefonie besteht aus Funktionen, die spezielle Hardware erfordern. Zum Beispiel ist spezielle Hardware für Anrufumleitung, Konferenzschaltungen und andere Verwaltungsfunktionen nötig.
- ➔ Erweiterte Telefonie stellt geräte- und anbieterspezifische Funktionen dar. Erweiterte Funktionen werden durch spezielle, von TAPI zur Verfügung gestellte Escape-Funktionen aufgerufen. Bevor die erweiterten Funktionen verwendet werden können, müssen Anwendungen ihr Vorhandensein durch Abfragen des Diensteanbieters ermitteln.

Netzwerkprogrammierung mit Pipes und Aufruf von Remote Procedures

Kapitel

45

Pipes oder Befehlsverkettungen bieten einen effizienten, einfachen Weg für zwei kooperierende Windows-Anwendungen, um über ein Netzwerk zu kommunizieren. Pipes sind leicht zu erstellen und anzuwenden; sie sind einer der bevorzugten Mechanismen zum Implementieren einer Kommunikation zwischen Server und Client, wobei die Server-Software unter Windows NT läuft. Leider ist in Windows 95 die Unterstützung von Pipes auf die Client-Seite beschränkt.

Die am öftesten verwendeten Pipes sind »Named Pipes«. Sie gehören zu den Mechanismen, die von Microsofts AUFRUFEN ENTFERNTER PROZEDUREN (REMOTE PROCEDURE CALLS ODER RPC) verwendet werden. Ein RPC ist ein Mechanismus, der Anwendungen befähigt, Prozeduren (Funktionen) aufzurufen, die Teil einer anderen Anwendung sind, die (womöglich) auf einem anderen Rechner im Netz läuft. Abgesehen von einigen Funktionen zum Initialisieren und Verwalten ist der Einsatz von RPC fast so einfach wie der Aufruf einer lokalen Funktion.

Dieses Kapitel untersucht diese beiden Kommunikationsmechanismen.

45.1 Kommunizieren mit Pipes

Pipes bieten eine Einweg- oder Zweiweg-Leitung für die Kommunikation zwischen kooperierenden Anwendungen. Die Programmierung mit Pipes beruht auf dem Client-Server Modell; eine typische Server-Anwendung erzeugt eine Pipe und wartet, daß Client-Anwendungen Zugang zur Pipe anfordern.

45.1.1 Erzeugen der Pipes

Die Win32-API unterscheidet zwischen

- ➔ Anonymen Pipes und
- ➔ Named Pipes

Anonyme Pipes

Die Anwendung von Anonymen Pipes ist etwas eingeschränkt. Wenn eine Anwendung eine Anonyme Pipe erzeugt, erhält sie zwei Handles; einer davon muß der anderen Anwendung weitergegeben werden, damit die beiden kommunizieren können. Ein möglicher Mechanismus zur Weitergabe des Handles ist die Vererbung; deshalb werden Anonyme Pipes oft zur Kommunikation zwischen Eltern- und Kindprozessen oder zwischen Kindprozessen desselben Elternprozesses eingesetzt. Da Anonyme Pipes durch Handles identifiziert werden, sind sie zwangsläufig lokal (ein Handle hat keine Gültigkeit auf einer anderen Maschine im Netzwerk). Dagegen werden Named Pipes durch einen UNC-Namen identifiziert, der im gesamten Netz gültig ist.

CreatePipe Anonyme Pipes werden mit der Funktion `CreatePipe` erzeugt. Diese Funktion gibt zwei Handles zurück:

- ➔ einer für das Lese-Ende der Pipe und
- ➔ einer für das Schreib-Ende

Named Pipes

CreateNamedPipe Named Pipes werden durch den Aufruf der Funktion `CreateNamedPipe` erzeugt. Named Pipes haben folgende Form:

```
\\hostname\\pipe\\pipename
```

Server können keine Pipe auf einem anderen Rechner erzeugen. Aus diesem Grund muß die Komponente `hostname` im Namen der Pipe, wenn sie in `CreateNamedPipe` verwendet wird, mit einem Punkt ersetzt werden, um den lokalen Host anzugeben:

```
\\.\\pipe\\pipename
```

Eine Named Pipe kann uni- oder bidirektional sein. Wenn der Server die Pipe erzeugt, gibt er `PIPE_ACCESS_DUPLEX`, `PIPE_ACCESS_INBOUND` oder `PIPE_ACCESS_OUTBOUND` an, um dies festzulegen.

Named Pipes unterstützen asynchrone (überlappende) Ein-/Ausgabe. Dieser Vorgang wird von anonymen Pipes nicht unterstützt.

Wenn eine Named Pipe erzeugt wird, gibt der Server den Typ der Pipe an – entweder Byte-Modus oder Meldungsmodus. Byte-Pipes behandeln die Daten als einen Strom von Bytes; Meldungs-Pipes behandeln Daten als einen Strom von Meldungen. Der Lesemodus der Pipe kann Byte-Lesemodus oder Meldungslesemodus sein. Meldungs-Pipes unterstützen beide Lesemodi, doch Byte-Pipes nur den Byte-Lesemodus.

Der Lesemodus der Pipe und ihr Wartemodus (je nachdem, ob die Pipe im Blocking- oder Nonblocking-Modus operiert) kann beim Erzeugen der Pipe angegeben und nachträglich mit der Funktion `SetNamedPipeHandleState` geändert werden. Der aktuelle Zustand kann über den Aufruf von `GetNamedPipeHandleState` abgefragt werden. Weitere Informationen liefert `GetNamedPipeInfo`.

45.1.2 Der Anschluß an Named Pipes

Wenn eine anonyme Pipe erzeugt wird, erhält der erzeugende Prozeß Handles für beide Enden der Pipe. Demgegenüber öffnet ein Server beim Erzeugen einer Named Pipe nur ein Ende der Pipe. Der Server muß warten, bis ein Client versucht, sich an die Pipe anzuschließen. Erst dann kann die Pipe verwendet werden. Eine Übersicht zum Erzeugen, Verwenden und Schließen der Named Pipes zeigt Abbildung 45.1.

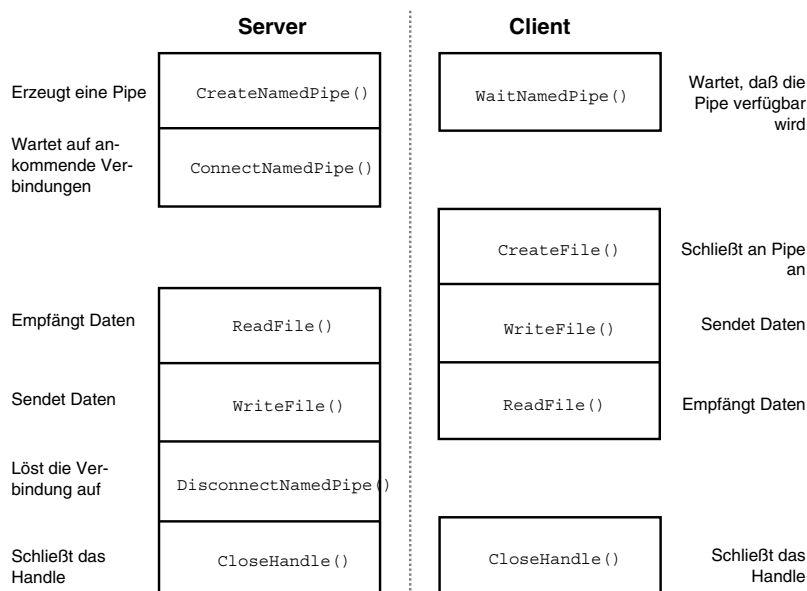


Abbildung 45.1:
Kommunizieren
mit Named
Pipes

- ConnectNamedPipe* Um auf eine Verbindung zu warten, rufen Server die Funktion `ConnectNamedPipe` auf. Diese Funktion wartet, bis ein Client sich der Pipe anschließt, die er mit ihrem Namen identifiziert (beachten Sie, daß `ConnectNamedPipe` auch asynchron verwendet werden kann, wobei der Server anderen Aufgaben nachgehen kann, während er auf einen Anschluß an die Pipe wartet).
- WaitNamedPipe* Clients können entscheiden, ob eine Named Pipe für eine Verbindung mit der Funktion `WaitNamedPipe` verfügbar ist.
- CreateFile* Ist eine Pipe verfügbar, so können Anwendungen an sie anschließen, indem sie die Funktion `CreateFile` einsetzen und den Namen der Pipe als Dateinamen angeben.
- DisconnectNamedPipe* Eine stehende Verbindung kann vom Server durch Aufrufen der Funktion `DisconnectNamedPipe` abgebrochen werden. Wenn ein Server diese Funktion aufruft, wird der clientseitige Handle der Pipe ungültig, und alle Daten, die der Client noch nicht gelesen hat, werden verworfen. Um sicherzugehen, daß der Client alle Daten vor dem Aufruf von `DisconnectNamedPipe` gelesen hat, können Server die Funktion `FlushFileBuffers` aufrufen. Nachdem `DisconnectNamedPipe` zurückkehrt, können Server entweder den Pipe-Handle schließen oder ihn für einen weiteren Aufruf von `DisconnectNamedPipe` verwenden.
- CloseHandle* Clients können die Verbindung einfach abbrechen, indem sie `CloseHandle` mit dem Handle der Pipe aufrufen, den sie durch den Aufruf von `CreateFile` erhalten haben.
- Server können einige Strategien anwenden, um mehrfache Verbindungen zu meistern, wie Starten einzelner Threads und Verwenden überlappender E/A. Ein Server kann für dieselbe Pipe mehrere Instanzen erzeugen, indem er `CreateNamedPipe` mit demselben Namen mehrmals aufruft.

45.1.3 Datentransfer durch Pipes

- ReadFile* Pipes können gelesen oder beschrieben werden mit `ReadFile` und `WriteFile`. Named Pipes können auch für überlappende E/A genutzt werden; in diesem Fall müssen die Funktionen `ReadFileEx` und `WriteFileEx` eingesetzt werden.
- TransactNamedPipe* Für Pipes im Meldungsmodus gibt es einen alternativen Mechanismus für den schnellen und effizienten Meldungstransfer. Clients können die Funktion `TransactNamedPipe` aufrufen, um in einer einzigen Netzwerkoperation eine Meldung zu lesen und zu schreiben. Die Funktion `CallNamedPipe` kombiniert in einem einzigen Vorgang die Aufrufe von `WaitNamedPipe`, `CreateFile`, `TransactNamedPipe` und `CloseHandle`. Mit anderen

Worten, `CallNamedPipe` wartet, daß die angegebene Pipe verfügbar wird, öffnet sie, tauscht Meldungen aus und schließt die Pipe in einem einzigen Funktionsaufruf.

45.2 Ein Arbeitsbeispiel

Dieses Beispiel implementiert eine Named Pipe als Client- und Server-Paar. Listing 45.1 zeigt die Server-Anwendung.

Die Anwendung kann leicht von der Kommandozeile aus kompiliert werden. Geben Sie dazu folgende Zeile ein:

```
CL PIPES.C
```

```
#include <windows.h>

void main(void)
{
    HANDLE hPipe;
    DWORD dwRead;
    char c = -1;

    hPipe = CreateNamedPipe("\\\\.\\pipe\\hello",
        PIPE_ACCESS_INBOUND, PIPE_WAIT,
        PIPE_UNLIMITED_INSTANCES,
        256, 256, 1000, NULL);
    ConnectNamedPipe(hPipe, NULL);
    while (c != '\\0')
    {
        ReadFile(hPipe, &c, 1, &dwRead, NULL);
        if (dwRead > 0 && c != 0) putchar(c);
    }
    DisconnectNamedPipe(hPipe);
    CloseHandle(hPipe);
}
```

*Listing 45.1:
Ein einfacher Na-
med Pipe Server*

Diese Anwendung erzeugt die Pipe `hello` auf dem lokalen Server. Die Pipe wird im Blocking-Modus erzeugt (`PIPE_WAIT`); Operationen auf dieser Pipe kehren nicht zurück, bevor sie nicht abgeschlossen sind. Nachdem die Pipe erzeugt wurde, ruft der Server `ConnectNamedPipe` auf und wartet auf ankommende Client-Verbindungen.

Wenn ein Client angeschlossen ist, versucht der Server, von der Pipe zu lesen. Er liest einzelne Bytes von der Pipe und schreibt sie an die Standardausgabe, bis ein Nullzeichen vorkommt; dann bricht er die Verbindung ab, schließt das Handle und beendet den Ablauf.

Der Client, in Listing 45.2 gezeigt, akzeptiert zwei Kommandozeilenparameter, der erste identifiziert den Host, an den er sich anschließt, der zweite stellt den String dar, den er an den Server auf dem Host schickt.

Die Anwendung kann leicht von der Kommandozeile aus kompiliert werden. Geben Sie dazu folgende Zeile ein:

CL PIPEC.C

*Listing 45.2:
Ein einfacher Na-
med Pipe Client*

```
#include <windows.h>
#include <string.h>
#include <stdio.h>

void main(int argc, char *argv[])
{
    HANDLE hPipe;
    DWORD dwWritten;
    char *pszPipe;

    if (argc != 3)
    {
        printf("Usage: %s hostname string-to-print\n", argv[0]);
        exit(1);
    }
    pszPipe = malloc(strlen(argv[1]) + 14);
    sprintf(pszPipe, "\\\\"%s\\"pipe\\hello", argv[1]);
    WaitNamedPipe(pszPipe, NMPWAIT_WAIT_FOREVER);
    hPipe = CreateFile(pszPipe, GENERIC_WRITE, 0, NULL,
                      OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
                      NULL);
    free(pszPipe);
    WriteFile(hPipe, argv[2], strlen(argv[2])+1, &dwWritten,
              NULL);
    CloseHandle(hPipe);
}
```

Nachdem er seine Kommandozeilenparameter verarbeitet hat, wartet der Client durch den Aufruf von `WaitNamedPipe` auf die Verfügbarkeit des Servers. Wenn `WaitNamedPipe` zurückkehrt, ruft der Client `CreateFile` auf, um die Verbindung zum Server herzustellen. Er verwendet dann `WriteFile`, um den String an den Server zu schicken, bevor er die Verbindung mit `CloseHandle` abbricht.

Sie können diese Anwendung auf einer einzigen Maschine unter Windows NT testen (Sie erinnern sich, daß Windows 95 das Server-Ende der Pipes nicht unterstützt) oder auf zwei Maschinen in einem Netzwerk. Falls Sie eine Maschine benützen, starten Sie `pipes.exe` in einem DOS-Fenster, und verwenden Sie ein weiteres Fenster für den Client. Rufen Sie den Client mit einer der folgenden ähnlichen Kommandozeile auf:

PIPEC MYHOST "HELLO, WORLD!"



Vergewissern Sie sich, daß anstelle von MYHOST der tatsächliche Name des Windows-NT-Hostrechners, auf dem pipes.exe ausgeführt wird, steht (Sie erinnern sich, daß das serverseitige Programm nur unter Windows NT läuft).

45.3 Microsoft Remote Procedure Calls

Microsofts RPC erweitern die konzeptuelle Eleganz und Einfachheit des Aufrufs einer Funktion oder Subroutine, indem sie für Aufrufe in einem Netz einen ähnlichen Mechanismus zur Verfügung stellen. RPC-Server bieten eine Reihe von Funktionen, die von RPC-Clients aufgerufen werden können.

Der RPC-Mechanismus wird in Windows sehr häufig eingesetzt. Vor allem ist der RPC-Mechanismus die Grundlage der COM- und OLE-Technologie.

45.3.1 RPC-Grundlagen

Der Schlüssel zum RPC-Mechanismus sind Stub-Funktionen.

Wenn ein RPC-Client eine RPC-Funktion aufruft, gibt er einen gewöhnlichen Funktionsaufruf aus. Die Funktion, die den Aufruf erhält, ist ein RPC-Stub; diese Stub-Funktion konvertiert die Argumente der Funktion für den Versand im Netz (ein Vorgang, der mit MARSHALING bezeichnet wird) und sendet den Aufruf mit den Argumenten zum Server.

Stub-Funktionen

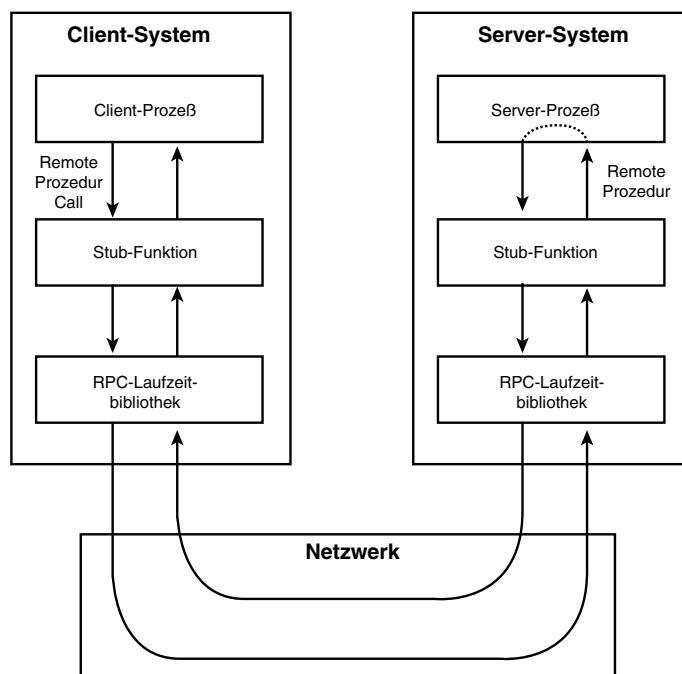


Abbildung 45.2:
Ausführung von
Remote Procedure
Calls

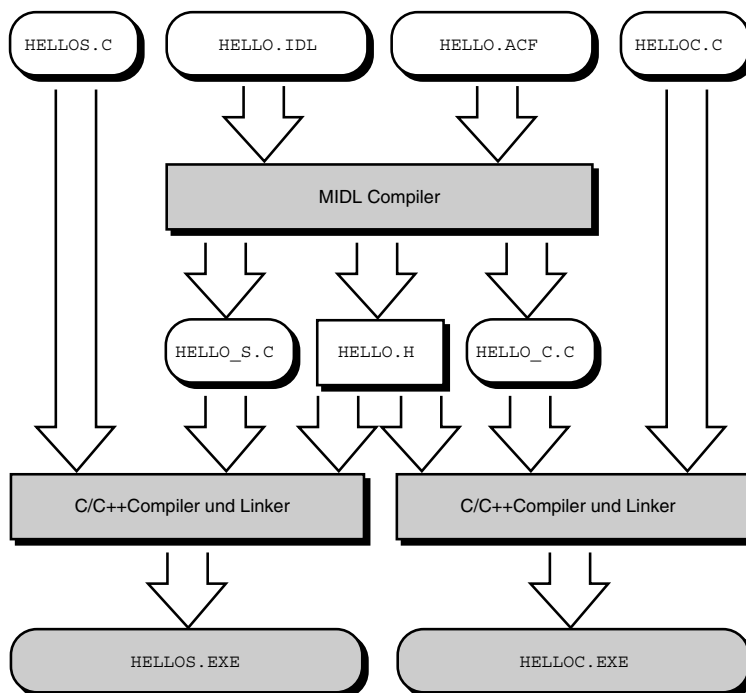
Stub-Funktionen auf der Serverseite UNMARSHALLEN die Funktionsargumente und rufen die Implementation der Funktion auf dem Server auf. Wenn die Funktion zurückkehrt, wird ihr Rückgabewert mit einem umgekehrten Mechanismus an den Client weitergereicht. Dieser Vorgang ist in Abbildung 45.2 dargestellt.

Beim Entwickeln von RPC-Anwendungen ist ein Element von zentraler Bedeutung: die Schnittstelle. Klar, daß die Stub-Funktionen auf der Client- und Server-Seite auf identischen Funktionsdefinitionen beruhen müssen, ansonsten würde der RPC-Prozeß mit Sicherheit scheitern. Das Tool, das sicherstellt, daß die Stub-Funktionen auf beiden Seiten kompatibel sind, ist der Microsoft Interface Development Language (MIDL) Compiler.

45.3.2 Ein einfaches Beispiel

Das nächste Beispiel klärt die grundlegenden Konzepte von Microsoft-RPCs. Dieses Beispiel, das RPC-Äquivalent der *Hello-World*-Anwendung, implementiert eine einfache Server-Funktion, die einen von der Client-Anwendung erhaltenen String ausgibt.

Abbildung 45.3:
RPC-Erstellung



Der erste Schritt in der Entwicklung dieses Beispiels ist die Angabe der Schnittstelle. Dies wird in Form zweier Dateien, die Eingabedateien für den MIDL-Compiler darstellen, getan.

MIDL-Compiler

Der MIDL-Compiler wird drei Dateien erzeugen:

- eine Headerdatei, die sowohl in die Server- als auch in die Client-Anwendung eingefügt werden muß, und
- zwei C-Quelldateien, die die Stub-Funktionen für Client und Server implementieren.

Diese Dateien müssen mit unserer Implementierung der Client- und Server-Anwendung gelinkt werden, um die endgültigen Programme zu erzeugen. Dieser Vorgang wird in Abbildung 45.3 veranschaulicht.

45.3.3 Angabe der Schnittstelle

Die Schnittstelle für eine RPC-Implementierung wird in Form zweier Dateien angegeben:

- die Schnittstellendefinitionsdatei (Interface Definition Language File) und
- die Anwendungskonfigurationsdatei (Application Configuration File).

Diese zwei Dateien zusammen dienen als Input für den MIDL-Compiler.

Schnittstellendefinitionsdatei

Die Schnittstellendefinitionsdatei für unsere Beispielanwendung sehen Sie in Listing 45.3.

```
[
    uuid (6fdd2ce0-0985-11cf-87c3-00403321bfac),
    version(1.0)
]
interface hello
{
    void HelloProc([in, string] const unsigned char *pszString);
    void Shutdown(void);
}
```

*Listing 45.3:
Die Schnittstellendefinitionsdatei*

Diese Datei besteht aus zwei Teilen: dem Schnittstellenheader und dem Schnittstellenrumpf.

Der Header hat folgende Syntax:

Schnittstellenheader

```
[ interface-attributes ] interface interface-name
```

Das vielleicht wichtigste der Schnittstellenattribute ist der GUID oder GLOBALLY UNIQUE IDENTIFIER der Schnittstelle. Der GUID ist ein 128-

GUID

Bit-Identifizierer, der als einzigartig in der Welt angenommen wird. Mit anderen Worten, es sollte keine zwei Anwendungen auf der Welt geben, die denselben Identifizierer haben.

Die Visual-C++-Distribution bietet ein Tool, das Programm *guidgen.exe*, das solche einzigartigen Identifizierer erzeugen soll, teils auf Informationen aus der Hardware Ihres Rechners, teils durch einen Zufallsgenerator-Algorithmus.

Der GUID wird in Form eines Strings aus 32 Hexadezimalstellen ausgedrückt. Die spezifische Form ist 8, 4, 4, 4 und 12 Stellen, durch Bindestriche getrennt. Das Programm *guidgen.exe* generiert GUID-Strings in dieser Form.

Versionsnummer Außer dem GUID gibt ein weiteres Schnittstellenattribut die Versionsnummer der Schnittstelle an. Die Rolle der Versionsnummer ist, mögliche Inkompatibilitäten der Schnittstellenversionen abzufangen.

Schnittstellenrumpf Im zweiten Teil der Schnittstellendefinitionsdatei werden Funktionsprototypen definiert. Die Syntax der Prototypen ist der der Sprache C ähnlich, doch enthält sie auch zusätzliche Elemente. Das Schlüsselwort *in* gibt dem MIDL-Compiler an, daß der folgende Parameter nur für Eingaben bestimmt ist, das heißt, er wird vom Client an den Server geschickt. Das Schlüsselwort *string* gibt an, daß die Daten als Zeichenkette gesendet werden.

Anwendungskonfigurationsdatei

Die Anwendungskonfigurationsdatei ist in Syntax und Erscheinung der Schnittstellendefinitionsdatei ähnlich. Diese Datei enthält jedoch Informationen über Daten und Attribute, die mit der eigentlichen Transmission der RPC-Daten nicht in Verbindung stehen.

Die Anwendungskonfigurationsdatei für unser Projekt wird in Listing 45.4 gezeigt.

Listing 45.4. Die Anwendungs-konfigurations-datei.

```
[
    implicit_handle(handle_t hello_IfHandle)
]
interface hello
{
}
```

In dieser Datei geben wir einen BINDUNGSHANDLE für die Schnittstelle an. Dieser Handle ist ein Datenobjekt, das die Verbindung zwischen Client und Server darstellt. Da der Handle nicht über das Netz gesendet wird, ist er in der Anwendungskonfigurationsdatei angegeben.

Das Schlüsselwort *implicit_handle* schreibt einen Handle vor, der als globale Variable geführt wird. Dieser Handle wird vom Client beim Aufruf von RPC-Laufzeitfunktionen verwendet.

Kompilation

Die beiden Dateien *hello.idl* und *hello.acf* können mit dem MIDL-Compiler in einem einzigen Befehl kompiliert werden:

```
MIDL HELLO.IDL
```

Das Ergebnis des Kompiliervorganges sind drei Dateien, die der MIDL-Compiler erzeugt: *hello_c.c*, *hello_s.c* und *hello.h*.

Die Dateien *hello_c.c* und *hello_s.c* enthalten die client- und serverseitige Implementierung der Stub-Funktionen; der Header *hello.h* enthält die nötigen Deklarationen.

Wenn man sich diese generierten Dateien ansieht, wird klar, welcher großer Anteil der Arbeit des Netzwerkprogrammierers durch den Einsatz des MIDL-Compilers automatisiert wird. Die generierten Stub-Funktionen führen alle nötigen Aufrufe in der Laufzeitbibliothek, um die Argumente zu marshallen und entmarshallern sowie im Netz zu kommunizieren. Aber die ganze Schönheit und elegante Einfachheit des RPC-Mechanismus wird erst klar, wenn wir unseren Client und Server implementieren.

45.3.4 Implementieren des Servers

Die Implementierung der Server-Anwendung steht in Listing 45.5.

Um die Server-Anwendung zu kompilieren, verwenden Sie folgende Kommandozeile:

```
CL HELLOS.C HELLO_S.C RPCRT4.LIB
```

```
#include <stdlib.h>
#include <stdio.h>
#include "hello.h"

void HelloProc(const unsigned char *pszString)
{
    printf("%s\n", pszString);
}

void Shutdown(void)
{
    RpcMgmtStopServerListening(NULL);
    RpcServerUnregisterIf(NULL, NULL, FALSE);
}

void main(int argc, char * argv[])
{
    RpcServerUseProtseqEp("ncacn_ip_tcp", 20, "8000", NULL);
    RpcServerRegisterIf(hello_v1_0_s_ifspec, NULL, NULL);
    RpcServerListen(1, 20, FALSE);
}

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
```

*Listing 45.4:
Ein einfacher
RPC-Server*


```

{
    return(malloc(1en));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
{
    free(ptr);
}

```

Die RPC-Laufzeitbibliothek wird initialisiert und der Server zum Warten auf ankommende Verbindungen durch die drei RPC-Laufzeitbibliothek-Aufrufe in der `main`-Funktion des Programms gebracht. Der erste dieser Aufrufe ist `RpcServerUseProtseqEp`; dieser Aufruf definiert das Netzwerkprotokoll und den Endpunkt, das bzw. der in der Anwendung verwendet werden. In unserem Beispiel verwende ich das TCP/IP-Protokoll (`ncacn_ip_tcp`) als ein Protokoll, das sowohl unter Windows NT als auch unter Windows 95 unterstützt wird. Der RPC-Mechanismus kann viele andere Protokolle verwenden, wie die Tabelle 45.1 zeigt.

Tabelle 45.1:
RPC-Protokolle

Protokollname	Beschreibung
<code>ncacn_ip_tcp</code>	TCP over IP
<code>ncacn_nb_tcp</code>	TCP over NetBIOS
<code>ncacn_nb_ipx</code>	IPX over NetBEUI
<code>ncacn_nb_nb</code>	NetBIOS over NetBEUI
<code>ncacn_np</code>	Named Pipes
<code>ncacn_spx</code>	SPX
<code>ncacn_dnet_nsp</code>	DECnet Transport
<code>ncadg_ip_udp</code>	UDP over IP
<code>ncadg_ipx</code>	IPX
<code>ncalrpc</code>	lokaler Prozeduraufruf

Protokolle, deren Namen mit `ncacn` beginnen, sind verbindungsorientierte Protokolle; diejenigen, deren Name mit `ncadg` beginnt, sind Datagramm-Protokolle (verbindungslos).

Da Windows 95 Named Pipes nur auf Client-Seite unterstützt, wird das Protokoll `ncacn_np` nur für RPC-Client-Anwendungen unterstützt. Windows 95 unterstützt nicht `ncacn_nb_ipx` und `ncacn_nb_tcp`. Das Protokoll `ncacn_dnet_nsp` wird nur für 16-Bit-Windows- und MS-DOS-Clients unterstützt.

Die Bedeutung des Endpunkt-Parameters hängt vom Protokoll ab. Wenn z.B. das Protokoll `ncacn_ip_tcp` verwendet wird, stellt der Endpunkt-Parameter eine RPC-Portnummer dar.

Das Starten des Servers besteht aus zwei Schritten.

*Starten des
Servers*

- ➡ Zuerst wird die Schnittstelle des Servers registriert.
- ➡ Danach geht er in einen Zustand, in dem er auf ankommende Verbindungen horcht.

Das Registrieren des Servers macht ihn für ankommende Client-Verbindungen verfügbar.

Die eigentliche entfernte Prozedur, `HelloProc`, ist genau wie eine lokale Prozedur implementiert. Eigentlich ist es möglich, die Implementierung dieser Funktion in eine getrennte Datei zu legen. Auf diese Art können Anwendungen, die die Funktion lokal aufrufen, mit ihr gelinkt werden, während Anwendungen, die sie über RPC aufrufen, statt dessen mit der clientseitigen Stub gelinkt werden.

Eine weitere Funktion, `Shutdown`, wurde bereitgestellt, um das Herunterfahren des Servers aus der Ferne zu erleichtern. Dies wird erreicht, indem der Horchzustand beendet und die Registrierung des Servers aufgehoben wird.

Zusätzlich zu diesen beiden Funktionen erfordern die Spezifikationen von Microsoft RPC das Implementieren zweier Speicherverwaltungsfunktionen. Die Funktionen

*Speicher-
verwaltung*

➡ `midl_user_allocate` und

➡ `midl_user_free`

werden zum Zuweisen und Freigeben eines Speicherblocks verwendet.

In einfachen Fällen können sie auf die C-Laufzeitfunktionen `malloc` und `free` abgebildet werden; in großen, komplexen Anwendungen ermöglichen diese Funktionen eine feinere Kontrolle über die Speicherbelegung. Beide Funktionen werden verwendet, wenn Argumente gemarshallt oder entmarshallt werden.

45.3.5 Implementieren des Client

Das Implementieren des RPC-Client ist nur ein wenig komplizierter als für eine Anwendung, die einen lokalen Funktionsaufruf enthält. Die Implementierung des Client für unser Beispiel wird in Listing 45.6 gezeigt.

Die Client-Anwendung kann mit folgender Kommandozeile kompiliert werden:

```
CL HELLOC.C HELLO_C.C RPCRT4.LIB
```

Listing 45.5:
Ein einfacher
RPC-Client

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "hello.h"

void main(int argc, char *argv[])
{
    unsigned char *pszStringBinding;

    if (argc != 3)
    {
        printf("Usage: %s hostname string-to-print\n", argv[0]);
        exit(1);
    }
    RpcStringBindingCompose(NULL, "ncacn_ip_tcp", argv[1],
                           "8000", NULL, &pszStringBinding);
    RpcBindingFromBinding(pszStringBinding,
                          &hello_IfHandle);
    if (strcmp(argv[2], "SHUTDOWN")) HelloProc(argv[2]);
    else Shutdown();
    RpcStringFree(&pszStringBinding);
    RpcBindingFree(&hello_IfHandle);
    exit(0);
}

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len)
{
    return(malloc(len));
}

void __RPC_USER midl_user_free(void __RPC_FAR * ptr)
{
    free(ptr);
}
```

Aufruf Diese Client-Implementierung erwartet zwei Kommandozeilenparameter:

- ➡ den Namen des Servers, mit dem verbunden wird, und
- ➡ den String, der an den Server gesendet werden soll.

Wegen des verwendeten Protokolls (TCP/IP) muß der Name des Servers der Internet-Name oder die IP-Adresse des Host sein. Falls Sie den Client und den Server auf demselben Host testen, können Sie den Default-Namen des lokalen Host, `localhost`, oder seine Default-IP-Adresse, `127.0.0.1`, verwenden.

Der Name des Protokolls, der Hostname und der Endpunkt werden in einem String Binding mit der Funktion `RpcStringBindingCompose` kombiniert. Zum Beispiel würde der String Binding für das Protokoll `ncacn_ip_tcp`, den lokalen Host und TCP Port 8000 folgendermaßen aussehen:

```
ncacn_ip_tcp:localhost[8000]
```

Die Funktion `RpcStringBindingCompose` ist bloß eine Erleichterung, die den Programmierer von der Aufgabe befreit, den String Binding aus seinen Komponenten von Hand zusammenzustellen. Dieser String

Binding wird verwendet, um den Binding-Handle für die Schnittstelle im Aufruf von `RpcBindingFromStringBinding` zu erhalten. Das Erhalten des Binding-Handles zeigt an, daß die Verbindung verfügbar ist.

Aufruf entfernter Prozeduren

Ist einmal die Verbindung hergestellt, so ist ihre Verwendung die Einfachheit selbst. Der Aufruf einer entfernten Prozedur ist dann identisch mit dem Aufruf einer lokalen Funktion. In unserer Client-Implementierung rufen wir die Funktion `HelloProc` mit dem zweiten Kommandozeilenargument auf, das heißt, sofern das Argument nicht der String `SHUTDOWN` ist, was einen Aufruf der Funktion `Shutdown` bewirkt, anstatt den entfernten Server herunterzufahren.

Wie der Client muß auch der Server seine Implementierung für `midl_user_allocate` und `midl_user_free` aufweisen.

Nachdem Sie Client- und Server-Programm kompiliert haben, können Sie die beiden Programme in zwei DOS-Fenstern testen. Nachdem Sie den Server in einem der Fenster gestartet haben, führen Sie den Client in dem anderen Fenster folgendermaßen aus:

```
HELLOC LOCALHOST "HELLO, WORLD!"
```

Um den Server von der Client-Seite herunterzufahren, geben Sie ein:

```
HELLOC LOCALHOST SHUTDOWN
```

45.3.6 RPC-Ausnahmebehandlung

Falls Sie versuchen, die im vorigen Abschnitt entwickelte Client-Anwendung alleine, ohne einen Server gestartet zu haben, auszuführen, so kommt ein ernster Mangel unserer Implementierung zum Vorschein. Weder der Client noch der Server in diesem Beispiel führen irgendeine Fehlerbehandlung durch. Das bedeutet im einzelnen, daß der Client nicht richtig auf Situationen reagiert, wenn kein Server verfügbar ist.

Anders als andere Fehler muß die Nichtverfügbarkeit eines Servers als sehr wahrscheinlich betrachtet und entsprechend behandelt werden. (Nicht daß ich damit andeuten will, daß die Behandlung anderer Fehler in produktiven Anwendungen vernachlässigt werden kann!) Die Implementierung von Microsoft-RPCs bietet einen speziellen Mechanismus für diesen Zweck, der in seiner Erscheinung den strukturierten Ausnahmen in Win32 oder der C++-Ausnahmenbehandlung sehr ähnlich ist.

Durch Schützen der entfernten Prozeduraufrufe durch die RFC-Ausnahmenmakros kann die elegante Behandlung der Netzwerkfehler sei-

tens einer Client-Anwendung sichergestellt werden. Zum Beispiel in unserer Anwendung *Hello, World* könnten die Aufrufe von `HelloProc` und `Shutdown` auf der Client-Seite folgendermaßen geschützt werden:

```
RpcTryExcept {
    if (strcmp(argv[2], "SHUTDOWN")) HelloProc(argv[2]);
    else Shutdown();
}
RpcExcept(1) {
    printf("RPC run-time exception %08.8X\n", RpcExceptionCode());
}
RpcEndExcept
```

Falls Sie die Client-Anwendung mit dieser Änderung rekompilieren, wird sie nicht länger abstürzen, wenn der Server nicht verfügbar ist; sie wird elegant beendet, indem die Ausnahmenummer ausgegeben wird.

45.3.7 Fortgeschrittene RPC-Feature

Obwohl diese Beispielanwendung die Einfachheit des Einsatzes von Microsoft-RPCs vorführt, läßt sie die Leistungsfähigkeit und die vielfältigen Features des RPC-Mechanismus bloß erahnen. Der folgende Abschnitt erwähnt einige der bemerkenswertesten Feature von Microsoft-RPCs.

Zeiger und Felder als Argumente

Der MIDL-Compiler kann verwendet werden, um entfernte Prozeduren, die allerlei Argumente akzeptieren, anzugeben. Dies schließt Zeiger und Felder ein; trotzdem erfordern Zeiger und Felder eine besondere Betrachtung. Da die RPC-Stub-Funktionen nicht nur die Zeigerargumente selbst marshallen, sondern auch die Daten, auf die diese zeigen, ist es nötig, in der Schnittstellenspezifikation die Größe eines Speicherblocks, auf den ein Zeiger zeigt, zu definieren. Es gibt eine Reihe von Attributen, um die Größe eines Feldes anzugeben. Betrachten Sie zum Beispiel eine entfernte Prozedur, die die Größe eines Feldes und einen Zeiger darauf als Parameter erwartet:

```
void myproc(short s, double *d);
```

Sie können die Schnittstelle für so eine Prozedur in Ihrer IDL-Datei folgendermaßen identifizieren:

```
void myproc([in] short s, [in, out, size_is(d)] double d[]);
```

Das informiert den MIDL-Compiler, einen Stub-Code zu generieren, der eine Anzahl von `s` Feldelementen marshallt.

Das Attribut `size_is` ist nicht das einzige, das bei der Angabe von Feldargumenten behilflich ist. Weitere umfassen `length_is`, `first_is`, `last_is` und `max_is`.

RPC-Namensdienst

Der RPC-Mechanismus kann den Microsoft RPC Name Service Provider unter Windows NT verwenden. Durch diesen Mechanismus ist es für Clients möglich, einen RPC-Server dem Namen nach zu lokalisieren. Der Einsatz des RPC-Namendienstes ermöglicht die Entwicklung von Clients, die nicht ein explizites Binding Handle verwenden (wie es unser Beispiel *Hello, World* tut). Solche Clients würden keine Aufrufe der RPC-Laufzeitbibliothek enthalten und würden auch nicht anders aussehen als Programme, die lokale Funktionen einsetzen, außer daß sie mit dem clientseitigen Stub gelinkt sind.

Abgeleitete Schnittstellen

Die Syntax von MIDL ermöglicht das Definieren von Schnittstellen, die von anderen Schnittstellen abgeleitet sind. Die Syntax ist ähnlich wie beim Ableiten von Klassen in C++:

```
[attributes] interface interface-name : base-interface
```

Die abgeleitete Schnittstelle erbt Elementfunktionen, Statuscodes und Attribute von der Basis-Schnittstelle.

45.4 Zusammenfassung

Pipes stellen einen einfachen, effizienten Kommunikationsmechanismus zwischen Anwendungen dar, der von Windows 95 (nur Clientseite) und Windows NT unterstützt wird.

Pipes können Namen haben oder namenlos sein. Anonyme Pipes werden typischerweise zwischen einem Eltern- und einem Kindprozeß oder zwei Zwillingsprozessen eingesetzt. Die Verwendung anonymer Pipes erfordert die Mitteilung eines Pipe-Handles von einem Prozeß zum anderen (zum Beispiel durch Vererbung des Handle). Da anonyme Pipes nur durch den Handle identifiziert werden, können sie nicht zur Kommunikation in einem Netzwerk eingesetzt werden.

Anders als anonyme Pipes, unterstützen Named Pipes überlappende Ein-/Ausgabeoperationen. Ein Server kann eine Kombination von Techniken verwenden, einschließlich überlappende E/A und separate Threads, um mehrere Clients gleichzeitig zu bedienen.

Sowohl Server als auch Clients kommunizieren durch eine Named Pipe mit Hilfe des Handles der Pipe und Standard-Win32-Ein-/Ausgabefunktionen.

Ein Microsoft RPC ist ein Mechanismus zum Aufrufen von Funktionen aus der Ferne. Er bietet eine transparente Schnittstelle, durch die Client-Anwendungen entfernte Funktionen auf eine Art aufrufen können, die dem Aufruf lokaler Funktionen sehr ähnlich ist.

Die Schlüssel dazu sind die RPC-Laufzeitbibliothek und der Microsoft-MIDL-Compiler. Die Schnittstelle zwischen einem Client und einem Server kann mit einer einfachen, C-ähnlichen Syntax beschrieben werden, wovon der MIDL-Compiler server- und clientseitige Stub-Funktionen generiert. Dadurch wird der Programmierer von der Aufgabe komplexer Netzwerkprogrammierung entlastet und muß nicht kompatible Versionen dieser Funktionen auf der Server- und Client-Seite pflegen.

Die tatsächliche Implementierung geschieht durch diese Stub-Funktionen. Wenn ein Client eine entfernte Prozedur aufruft, so wird der Aufruf von der entsprechenden clientseitigen Stub-Funktion übernommen. Die Stub-Funktion ihrerseits ruft die RPC-Laufzeitbibliothek auf, die über die Verbindung im Netzwerk Stub-Funktionen auf der Server-Seite aufruft. Die clientseitige Stub-Funktion marschallt die Funktionsargumente zum Versand über das Netzwerk. Die serverseitige Stub-Funktion entmarschallt diese Argumente und ruft die tatsächlich implementierte Funktion auf. Wenn die Funktion zurückkehrt, wird der Vorgang in umgekehrter Richtung durchgeführt, da die Rückgabewerte zur Client-Anwendung transportiert werden müssen.

Fortgeschrittene RPC-Features umfassen RPC-Namensdienste, Zeiger und Felder als Funktionsargumente, abgeleitete Schnittstellen und vieles mehr.

Multimedia

Teil **VII**

- 
- 46. Multimedia-Anwendungen**
 - 47. Die Grafikbibliothek OpenGL**
 - 48. Hochleistungsgrafik und Ton**



Multimedia- Anwendungen

Kapitel

46



Es ist einige Jahre her, daß Microsoft Windows zur Wahlplattform für Multimedia-Anwendungen wurde. Seit der Einführung der Multimedia-Fähigkeiten in Windows 3.1 sind die Hardwarepreise spürbar gesunken. Noch vor wenigen Jahren erforderte Multimedia – Video und Sound – teure Workstations; heute sind sie auf den meisten Heimcomputern verfügbar. CD-ROM-Laufwerke, mittlerweile stark verbreitet, sorgen dafür, daß Multimedia-Präsentationen auf den meisten PCs zumindest schon einmal ausprobiert wurden. Kurzum, Multimedia kann nicht länger vom Programmierer ignoriert werden.

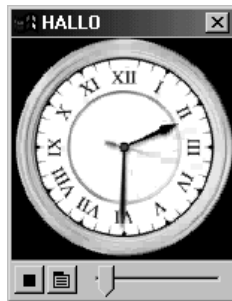
Multimedia-Anwendungen erfordern geeignete Hardware. Videos abzuspielen ist auf allen PCs möglich; die meisten PCs sind mit Soundkarten ausgerüstet, die Audioaufnahmen und Abspielen ermöglichen. Videoaufnahmen sind eine andere Sache: Hardware, die zur Aufnahme voll bewegter Bilder geeignet ist, ist noch teuer und somit wenig verbreitet. Scanner sind in letzter Zeit im Preis gesunken, doch andere Multimedia-Geräte (wie computergesteuerte VCRs) finden sich normalerweise nur in den Labors der Spezialisten.

Obwohl Multimedia-Programmierung unter Windows ziemlich komplex sein kann, können Anwendungen viele Dinge mit wenigen einfachen Funktionsaufrufen erledigen. Nichts veranschaulicht dies besser als die Funktion `MCIWndCreate`, die das Abspielen von Videos mit einem einzigen Funktionsaufruf ermöglicht. Unsere Übersicht der Multimedia-Programmierung beginnt mit einem näheren Blick auf diese Funktion; für viele einfache Anwendungen werden Sie nichts Komplizierteres brauchen.

46.1 Videos abspielen mit einem Funktionsaufruf

Das Programm aus Listing 46.1. kann vielleicht als das Multimedia-Äquivalent einer Hello-World-Anwendung angesehen werden. Dieses Programm erhält einen einzigen Kommandozeilenparameter, den Namen einer Multimedia-Datei, z.B. einer AVI-Videodatei, und spielt sie in einem Fenster ab.

Abbildung 46.1:
AVI-Abspielen
mit MCIWnd-
Create



Dieses einfache Programm kann von der Kommandozeile aus kompiliert werden:

```
CL HALLO.C USER32.LIB VFW32.LIB
```

Listing 46.1:
Eine einfache
Multimedia-
Anwendung

```
#include <windows.h>
#include <vfw.h>

void SetClientRect(HWND hwnd, HWND hwndMCI)
{
    RECT rect;

    GetWindowRect(hwndMCI, &rect);
    AdjustWindowRectEx(&rect, GetWindowLong(hwnd, GWL_STYLE),
                      FALSE, GetWindowLong(hwnd, GWL_EXSTYLE));
    MoveWindow(hwnd, rect.left, rect.top, rect.right - rect.left,
               rect.bottom - rect.top, TRUE);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case MCIWNDM_NOTIFYPOS:
        case MCIWNDM_NOTIFYSIZE:
            SetClientRect(hwnd, (HWND)wParam);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
}
```

```

        return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
        wndClass.lpfnWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "HALLO";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("HALLO", "HALLO",
                      WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU,
                      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
                      NULL, NULL, hInstance, NULL);
    SetClientRect(hwnd, MCIWndCreate(hwnd, hInstance, WS_VISIBLE
                                     | WS_CHILD | MCIWDM_SHOWALL |
                                     MCIWDM_NOTIFYSIZE |
                                     MCIWDM_NOTIFYPOS, lpCmdLine));

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    while (GetMessage(&msg, NULL, 0, 0))
        DispatchMessage(&msg);
    return msg.wParam;
}

```

Gut, ich habe geschummelt. Außer dem einzigen Funktionsaufruf haben wir da noch einige Zeilen Code, die Nachrichten verarbeiten. Durch das Beantworten der Nachrichten `MCIWDM_NOTIFYPOS` und `MCIWDM_NOTIFYSIZE` können wir sicherstellen, daß das Hauptfenster der Anwendung automatisch in der Größe angepaßt wird. So erhalten wir ein Fenster, das für die Videowiedergabe richtig dimensioniert ist. Trotzdem veranschaulicht die Anwendung die Leistungsfähigkeit von `MCIWndCreate` gut; mit diesem einzigen Funktionsaufruf waren wir in der Lage, eine vollständige Videowiedergabe-Anwendung zu erstellen.

`MCIWndCreate` ist nicht die einzige einfache Funktion, die für die Wiedergabe von Medien verwendet werden kann. Eine weitere Funktion dieser Art ist `Playback`; diese Funktion kann verwendet werden, um Waveform-Audiodateien abzuspielen.

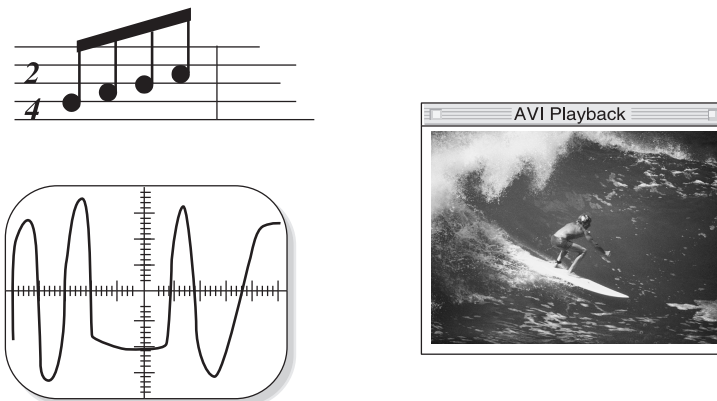
46.2 Grundlagen der Multimedia-Programmierung

Multimedia stellt die Fähigkeit des Betriebssystems dar, Video und Sound aufzunehmen und wiederzugeben. Die Programmierung von Multimedia wird über eine Reihe von Schnittstellen oder APIs gemacht, die von den Anwendungen zu diesem Zweck verwendet werden können.

46.2.1 Multimedia-Datenformate

Windows erkennt drei grundlegende Multimedia-Formate: Waveform Audio, MIDI-Sequenzen und Video (Abbildung 46.2).

Abbildung 46.2:
Multimedia in
Windows



WAV	Bei Waveform handelt es sich um gesampelte, digitalisierte Audiodaten. Waveform Audio wird typischerweise in Dateien mit der Erweiterung <code>.wav</code> gespeichert. Windows erkennt Waveform-Audiodateien mit Mono- und Stereo-Daten, einer Vielzahl von Samplingraten und -tiefen. Es gibt auch einige verschiedene Kompressionsmethoden für die effiziente Speicherung der Waveform-Daten.
MID	MIDI ist das Akronym für Musical Instrument Digital Interface. Dieser internationale Standard spezifiziert ein Protokoll für die Verbindung zwischen Computern und elektronischen Musikinstrumenten. MIDI-Sequenzen sind Daten, die auf MIDI-kompatiblen Instrumenten abgespielt werden können. MIDI-Daten werden unter Windows in Dateien mit der Erweiterung <code>.mid</code> gespeichert. MIDI-Da-

teien können auf externen Geräten, die mit dem Computer verbunden sind, abgespielt werden, oder auf eingebauten Synthesizern, die MIDI unterstützen.

AVI Das am meisten verwendete Videoformat unter Windows ist das Dateiformat Audio Video Interleaved (AVI). AVI-Dateien können eingesetzt werden, um einen bewegten Video-Stream und einen oder mehrere Audiokanäle zu speichern. Windows erkennt Videodaten mit verschiedenen Auflösungen, Farbtiefen und Refreshraten. Es gibt auch einige Kompressionsformate, die weit verbreitet sind. Andere Videoformate, wie MPEG oder Quicktime, werden ebenfalls durch Treiber von Microsoft oder Drittanbietern unterstützt.

Windows Multimedia-Funktionen können auch zum Abspielen von Audio-CDs eingesetzt werden. Mit Treibern von Drittanbietern ist auch das Aufnehmen und Abspielen anderer Multimedia-Formate möglich.

46.2.2 Multimedia-Schnittstellen

Sie können, Ihren Programmanforderungen entsprechend, zwischen drei Schnittstellenebenen wählen, um mit dem Multimedia-Subsystem in Windows zu kommunizieren.

- ➡ Die Schnittstelle der höchsten Ebene basiert auf der Fensterklasse *MCIWnd*. Im Beispielprogramm aus Listing 46.1 ermöglicht die Funktion *MCIWnd Videowiedergabe* mit nur einem Funktionsaufruf. *Schnittstellenebenen*
- ➡ Die Schnittstelle der mittleren Ebene ist das *MEDIA CONTROL INTERFACE* oder *MCI*. *MCI* bietet eine geräteunabhängige Schnittstelle mit Befehlsmeldungen und Befehlsstrings für die Aufnahme und Wiedergabe visueller und Audiodaten.
- ➡ Auf der niedrigsten Ebene gibt es einige Schnittstellen für Waveform-Audio, Video und MIDI-Aufnahme und -Wiedergabe. Zusätzliche Schnittstellen bieten Mixen für Audio, gepufferte Dateien ein-/ausgabe und Steuerung für Joystick und Timer.

Alle diese Schnittstellen dienen einem Zweck: eine Programmschnittstelle zwischen Bedieneranwendungen einerseits und Treiber für Multimediahardware andererseits zu bieten. Durch diese Treiber wird die Geräteunabhängigkeit unter Windows erreicht.

Welche Schnittstelle sollten Sie für Ihre Anwendung wählen? Das hängt von den Anforderungen Ihres Projekts ab.

Welche Schnittstelle zu welchem Zweck?

- ➔ Anwendungen, die nur einfache Wiedergabe erfordern, sind gute Kandidaten für MCIWnd-Dienste. Zum Beispiel ein Lexikon, das Videoclips zu den Artikeln bietet, könnte ein MCIWnd-Fenster für die Videowiedergabe verwenden.
- ➔ Ein Beispiel einer Multimedia-Anwendung mit höheren Ansprüchen ist eine Anwendung mit Audioaufnahme und Wiedergabe. Ein solches Programm sollte wohl auf die Dienste der MCI-Schnittstelle setzen, um seine Funktionalität zu implementieren.
- ➔ Eine noch anspruchsvollere Multimedia-Anwendung, wie Videoaufnahme und Mixen, würde wohl die Dienste der niederen Ebene beanspruchen.

46.3 Programmieren mit MCIWnd

Die Klasse MCIWnd ist die einfachste Multimedia-Programmschnittstelle auf höchster Ebene in Windows. Anwendungen, die einfache Wiedergabefähigkeiten erfordern, können MCIWnd-Fenster zu diesem Zweck einsetzen; ein solches Fenster kann mit einem einzigen Funktionsaufruf erzeugt werden, wie das Programm in Listing 46.1 zeigt.

46.3.1 Die Klasse MCIWnd

Fenster der Klasse MCIWnd bieten eine Anwenderschnittstelle mit bis zu vier Schaltern, einer Schieberleiste und einer optionalen Wiedergabefläche (Abbildung 46.3).

Abbildung 46.3:
MCIWnd-
Fensterkontrol-
elemente



- Play/Stop** Der Play- und Stop-Schalter kann verwendet werden, um die Wiedergabe zu starten und anzuhalten. Die Wiedergabe startet an der Position, die durch die Schieberleiste angegeben wird. Spezielle Wiedergabeeffekte können aufgerufen werden, indem die - oder **[Strg]**-Taste gedrückt wird, während der Schalter **Play** betätigt wird; beim Drücken der **[Strg]**-Taste wird eine Vollbild-Wiedergabe erzielt, das Drücken der -Taste bewirkt eine Rückwärts-wiedergabe.
- Menu** Der **Menu**-Schalter kann zum Aufruf eines Popup-Menüs verwendet werden. Wenn zum Beispiel eine AVI-Datei geladen wird, enthält das Popup-Menü Optionen zum Setzen der Wiedergabegeschwindigkeit, der Lautstärke, Zoom und Videokonfiguration. Andere Befehle ermöglichen das Kopieren der aktuellen Dateien in die Windows-Zwischenablage und das Öffnen einer weiteren Datei. Ein weiterer Menübefehl ermöglicht das Senden eines MCI-Befehlsstrings direkt zum gerade aktiven Multimediagerät.
- Record** Der Schalter **Record** kann Geräten, die Aufnahmefähigkeit besitzen, zugewiesen werden.
- Schieber** Die Schieberleiste wird verwendet, um die aktuelle Wiedergabe- oder Aufnahmeposition relativ zur Dateigröße anzuzeigen. Die Schieberleiste kann auch zur Positionierung an verschiedenen Stellen in der Datei während der Wiedergabe verwendet werden.

Optimale Leistung in der Videowiedergabe erfordert, daß das Wiedergabefenster auf einer Vier-Pixel-Grenze ausgerichtet ist. Normalerweise richtet Windows das Wiedergabefenster automatisch aus.

46.3.2 MCIWnd-Funktionen

Ein **MCIWnd**-Fenster wird durch einen Aufruf von **MCIWndCreate** erzeugt. *MCIWndCreate*
Der Aufruf dieser Funktion registriert die Klasse **MCIWnd** und erzeugt ein **MCIWnd**-Fenster.

Außer der Angabe des Handles des Elternfensters und eines Instanz-Handles geben die Parameter dieser Funktion auch einen Satz von Fensterstilen und einen optionalen Dateinamen an.

Die Fensterstile bestimmen, welche Elemente des **MCIWnd**-Fensters sichtbar sind und wie es mit dem Anwender einerseits und dem Code der Anwendung andererseits zusammenwirkt. *Fensterstile*

- ➔ Zum Beispiel kann durch Angabe des Stils `MCIWND_RECORD` ein `MCIWnd`-Fenster mit einem sichtbaren Aufnahme-Schalter erzeugt werden.
- ➔ Wird `MCIWND_NOTIFYSIZE` angegeben, so sendet das `MCIWnd`-Fenster Nachrichten an sein Elternfenster, wann immer die Fenstergröße geändert wird.

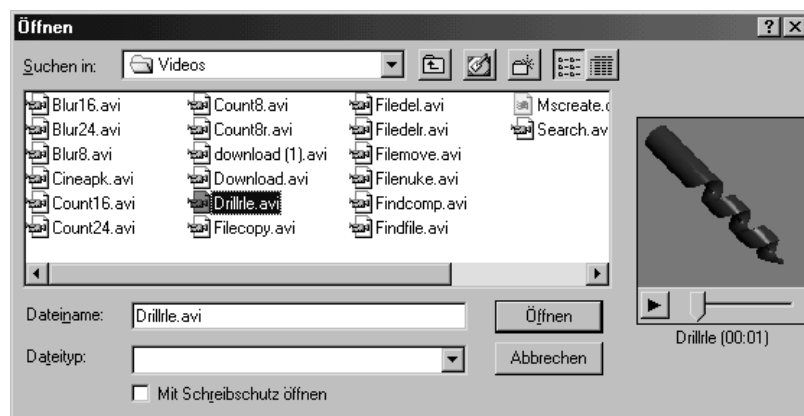
`MCIWnd`-Fenster können als Kindfenster oder überlappende Fenster erzeugt werden. Falls sie als überlappende Fenster erzeugt wurden, haben sie einen Titelbalken, deren Inhalt dem jeweiligen Stil entspricht (`MCIWND_SHOWMODE`, `MCIWND_SHOWNAME` oder `MCIWND_SHOWPOS`).

`MCIWnd`-Fenster können auch über Aufrufe von `CreateWindow` oder `CreateWindowEx` erzeugt werden. Bevor Sie das tun können, müssen Sie `MCIWndRegisterClass` aufrufen. Diese Funktion registriert die Fensterklasse, die durch die Konstante `MCIWND_WINDOW_CLASS` angegeben wird.

GetOpenFileNamePreview / GetSaveFileNamePreview

Es gibt zwei weitere Funktionen, die Fenster der Klasse `MCIWnd` verwenden. Die Funktionen `GetOpenFileNamePreview` und `GetSaveFileNamePreview` erweitern die Standardfunktionen `GetOpenFileName` und `GetSaveFileName`, indem sie zum Standarddialog Datei öffnen ein Multimedia-Vorschaufenster hinzufügen (Abbildung 46.4).

Abbildung 46.4:
Datei-öffnen-
Dialog mit
einem Vor-
schaufenster,
von `GetOpenFi-
leNamePreview`
erzeugt



Das Programm in Listing 46.2 zeigt den Einsatz von `GetOpenFileNamePreview`. Um dieses Programm zu kompilieren, geben Sie folgende Zeile ein:

```
CL OFNP.C VFW32.LIB
```

```
#include <windows.h>
#include <vfw.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpCmdLine, int nCmdShow)
{
    OPENFILENAME ofn;
    memset(&ofn, 0, sizeof(ofn));
    ofn.lStructSize = sizeof(ofn);
    GetOpenFileNamePreview(&ofn);
}
```

*Listing 46.2:
Der Einsatz von
GetOpenFileNamePreview*

46.3.3 MCIWnd-Makros

Anwendungen können mit einem MCIWnd-Fenster kommunizieren, indem sie mit der Windows-Funktion `SendMessage`-Nachrichten senden. Es gibt eine Vielzahl von Hilfsmakros, die das Senden dieser Nachrichten vereinfachen.

Durch diese Nachrichten können Anwendungen das Aussehen und Verhalten des MCIWnd-Fensters steuern, Aufnahme und Wiedergabe starten und stoppen, das MCI-Gerät oder die Datei schließen oder eine neue Datei oder ein neues Gerät für Aufnahme oder Wiedergabe öffnen, bestimmte Aufnahme- oder Wiedergabepositionen suchen, Informationen über Fähigkeiten der Geräte und aktuelle Einstellungen abfragen, MCI-Attribute setzen und das MCI-Gerät steuern.

Tabelle 46.1 faßt die MCIWnd-Nachrichten und Hilfsmakros zusammen.

Nachricht	Makro	Beschreibung
MCI_CLOSE	MCIWndClose	Schließe MCI-Gerät oder Datei
MCI_OPEN	MCIWndOpenDialog	Öffne Daten-Datei
MCI_PAUSE	MCIWndPause	Pause bei Wiedergabe oder Aufnahme
MCI_PLAY	MCIWndPlay	Start Wiedergabe
MCI_RECORD	MCIWndRecord	Start Aufnahme
MCI_RESUME	MCIWndResume	Beende Wiedergabe oder Aufnahme
MCI_SAVE	MCIWndSave	Speichere Inhalt
MCI_SAVE	MCIWndSaveDialog	Speichere Inhalt
MCI_SEEK	MCIWndEnd	Bewege zum Ende

*Tabelle 46.1:
MCIWnd-Nachrichten
und Makros*

Nachricht	Makro	Beschreibung
MCI_SEEK	MCIWndHome	Bewege zum Anfang
MCI_SEEK	MCIWndSeek	Bewege zur Position
MCI_STEP	MCIWndStep	Bewege Position
MCI_STOP	MCIWndStop	Stop Wiedergabe oder Aufnahme
MCIWNDM_CAN_CONFIG	MCIWndCanConfig	Kann Gerät konfiguriert werden?
MCIWNDM_CAN_EJECT	MCIWndCanEject	Kann Medium auswerfen?
MCIWNDM_CAN_PLAY	MCIWndCanPlay	Kann Gerät abspielen?
MCIWNDM_CAN_RECORD	MCIWndCanRecord	Kann Gerät aufnehmen?
MCIWNDM_CAN_SAVE	MCIWndCanSave	Kann in Datei gespeichert werden?
MCIWNDM_CAN_WINDOW	MCIWndCanWindow	Window-Befehle unterstützt?
MCIWNDM_CHANGEYLES	MCIWndChangeStyles	Ändere Fensterstil
MCIWNDM_EJECT	MCIWndEject	Wirf Medium aus
MCIWNDM_GETACTIVETIMER	MCIWndGetActiveTimer	Gib Aktualisierungsperiode zurück
MCIWNDM_GETALIAS	MCIWndGetAlias	Gib Geräte-Alias zurück
MCIWNDM_GET_DEST	MCIWndGetDest	Gib Wiedergaberechteck zurück
MCIWNDM_GETDEVICE	MCIWndGetDevice	Gib Gerätenamen zurück
MCIWNDM_GETDEVICEID	MCIWndGetDeviceID	Gib Gerätebezeichner zurück
MCIWNDM_GETEND	MCIWndGetEnd	Gib Endeposition zurück
MCIWNDM_GETERROR	MCIWndGetError	Gib letzten MCI-Fehler zurück
MCIWNDM_GETFILENAME	MCIWndGetFileName	Gib aktuellen Dateinamen zurück

Nachricht	Makro	Beschreibung
MCIWNDM_GETINACTIVETIMER	MCIWndGet-InactiveTimer	Gib Aktualisierungsperiode zurück
MCIWNDM_GETLENGTH	MCIWndGet-Length	Gib Länge des Inhalts zurück
MCIWNDM_GETMODE	MCIWndGet-Mode	Gib aktuellen Modus zurück
MCIWNDM_GETPALETTE	MCIWndGet-Palette	Gib MCI-Palettenhandle zurück
MCIWNDM_GETPOSITION	MCIWndGet-Position	Gib Position zurück
MCIWNDM_GETPOSITION	MCIWndGet-PositionString	Gib Position zurück
MCIWNDM_GETREPEAT	MCIWndGet-Repeat	Ständige Wiedergabe?
MCIWNDM_GETSOURCE	MCIWnd-GetSource	Gib Arbeits-Rechteck zurück
MCIWNDM_GETSPEED	MCIWndGet-Speed	Gib Wiedergabegeschwindigkeit zurück
MCIWNDM_GETSTART	MCIWndGet-Start	Gib Startposition zurück
MCIWNDM_GETSTYLES	MCIWndGet-Styles	Gib Fensterstil zurück
MCIWNDM_GETTIMEFORMAT	MCIWndGet-TimeFormat	Gib Zeitformat zurück
MCIWNDM_GETVOLUME	MCIWndGet-Volume	Gib Lautstärkeeinstellung zurück
MCIWNDM_GETZOOM	MCIWnd-GetZoom	Gib Zoomeinstellung zurück
MCIWNDM_NEW	MCIWndNew	Erzeuge neue Datei
MCIWNDM_OPEN	MCIWndOpen	Öffne MCI-Gerät und Datei
MCIWNDM_OPENINTERFACE	MCIWndOpenInterface	Öffne IAVI-Schnittstelle
MCIWNDM_PLAYFROM	MCIWndPlay-From	Wiedergabe ab Position
MCIWNDM_PLAYFROMTO	MCIWndPlay-FromTo	Wiedergabe-Bereich

Nachricht	Makro	Beschreibung
MCIWNDM_PLAYREVERSE	MCIWndPlayReverse	Wiedergabe umgekehrt
MCIWNDM_PLAYTO	MCIWndPlayTo	Wiedergabe bis Position
MCIWNDM_PUT_DEST	MCIWndPutDest	Ändere Wiedergabe-Rechteck
MCIWNDM_PUT_SOURCE	MCIWndPutSource	Ändere Arbeits-Rechteck
MCIWNDM_REALIZE	MCIWndRealize	Realisiere MCI-Palette
MCIWNDM_RETURNSTRING	MCIWndReturnString	Gib MCI-Antwort zurück
MCIWNDM_SENDSTRING	MCIWndSendString	Sende MCI-Befehl
MCIWNDM_SETACTIVETIMER	MCIWndSetActiveTimer	Setze Aktualisierungsperiode
MCIWNDM_SETINACTIVETIMER	MCIWndSetInactiveTimer	Setze Aktualisierungsperiode
MCIWNDM_SETOWNER	MCIWndSetOwner	Setze Besitzer-Fenster
MCIWNDM_SETPALETTE	MCIWndSetPalette	Setze MCI-Palette
MCIWNDM_SETREPEAT	MCIWndSetRepeat	Setze Wiederholmodus
MCIWNDM_SETSPEED	MCIWndSetSpeed	Setze Wiedergabegeschwindigkeit
MCIWNDM_SETTIMEFORMAT	MCIWndSetTimeFormat	Setze Zeitformat
MCIWNDM_SETTIMERS	MCIWndSetTimers	Setze Aktualisierungsperiode
MCIWNDM_SETVOLUME	MCIWndSetVolume	Setze Lautstärke
MCIWNDM_SETZOOM	MCIWndSetZoom	Setze Video-Zoom
MCIWNDM_SETTIMEFORMAT	MCIWndUseFrames	Setze Zeitformat
MCIWNDM_SETTIMEFORMAT	MCIWndUseTime	Setze Zeitformat

Nachricht	Makro	Beschreibung
MCIWNDM_VALIDATEMEDIA	MCIWndValidateMedia	Aktualisiert Positionen
WM_CLOSE	MCIWndDestroy	Schließe MCIWnd-Fenster

46.3.4 MCIWnd-Benachrichtigungen

MCIWnd-Fenster können, wenn sie dazu befähigt werden, Benachrichtigungen an ihre Elternfenster senden. Es gibt fünf Typen solcher Benachrichtigungen. Alle fünf können aktiviert werden, indem man beim Erzeugen des Fensters den Fensterstil `MCIWNDF_NOTIFYALL` angibt. Alternativ dazu können Benachrichtigungen einzeln aktiviert werden.

- Die Nachricht `MCIWNDM_NOTIFYERROR` wird an das Elternfenster gesendet, um es von MCI-Fehlern in Kenntnis zu setzen. Diese Benachrichtigung kann mit dem Fensterstil `MCIWNDF_NOTIFYERROR` aktiviert werden.
- Die Nachricht `MCIWNDM_NOTIFYMEDIA` setzt das Elternfenster über einen stattgefundenen Medienwechsel in Kenntnis. Diese Nachricht kann mit dem Fensterstil `MCIWNDF_NOTIFYMEDIA` aktiviert werden.
- Das Elternfenster wird von Änderungen in der Position und Größe des MCIWnd-Fensters über die Nachrichten `MCIWNDM_NOTIFYPOS` und `MCIWNDM_NOTIFYSIZE` in Kenntnis gesetzt. Diese Nachrichten können mit den Fensterstilen `MCIWNDF_NOTIFYPOS` und `MCIWNDF_NOTIFYSIZE` aktiviert werden.
- Schließlich wird das Elternfenster über alle Änderungen im Ablauf (zum Beispiel Änderung vom Abspielmodus zum Stop-Modus) durch die Nachricht `MCIWNDM_NOTIFYMODE` unterrichtet. Diese Nachrichten werden durch den Fensterstil `MCIWNDF_NOTIFYMODE` aktiviert.

46.4 Die Mediensteuerschnittstelle

Die Mediensteuerschnittstelle bietet eine Reihe von geräteunabhängigen Befehlsnachrichten und Befehlsstrings zur Steuerung der Multimediaeräte. Befehlsnachrichten und Befehlsstrings können abwechselnd eingesetzt werden.

MCI erkennt eine Vielzahl von verschiedenen Multimediageräten. Diese Geräte sind in Tabelle 46.2 aufgezählt.

Tabelle 46.2:
Multimedia-
geräte

Gerätename	Beschreibung
animation	Animationsfähiges Gerät
cdaudio	Audio CD-Player
dat	Digital-Audio Tape Player
digitalvideo	Digitales Video im Fenster (nicht GDI)
other	Undefiniertes Gerät
overlay	Analoges Video im Fenster
scanner	Bildscanner
sequencer	MIDI-Sequencer
vcr	Videocassettenrecorder oder Player
videodisc	Video Disc Player
waveaudio	Waveform-Audiogerät

Einsatz von Befehlsnachrichten und -strings

Sowohl Befehlsnachrichten als auch Befehlsstrings können eingesetzt werden, um Geräte anzusteuern und Informationen vom Gerät abzurufen.

Befehlsnachrichten	fragen Informationen als Strukturen ab, die in C-Programmen leicht zu interpretieren sind.
Befehlsstrings	fragen Informationen als Strings ab, die von der Anwendung analysiert und interpretiert werden müssen.

Alle MCI-Geräte unterstützen einen Kern von MCI-Befehlen und -Nachrichten. Viele Geräte unterstützen zusätzliche, gerätespezifische Befehle.

Befehlsstrings

mciSendString Befehlsstrings werden an die Geräte mit der Funktion `mciSendString` gesendet. Eine einfache Einsatzmöglichkeit dieser Funktion wird in Listing 46.3 gezeigt; diese Anwendung, die einen einzigen Dateinamen als Kommandozeilenargument annimmt, spielt eine Videodatei ab, indem Sie den Pfadnamen der Videodatei per Befehlsstrings an MCI schickt.

Um diese Anwendung zu kompilieren, geben Sie folgende Zeile ein:

```
CL MCISTR.C USER32.LIB WINMM.LIB
```

```
#include <windows.h>
#include <stdlib.h>

#define CMDSTR "OPEN %s ALIAS MOVIE"

int WINAPI
WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR lpCmdLine, int d4)
{
    char *pBuf;

    pBuf = malloc(sizeof(CMDSTR) + strlen(lpCmdLine) - 2);
    if (!pBuf) return -1;
    wsprintf(pBuf, CMDSTR, lpCmdLine);
    mciSendString(pBuf, NULL, 0, NULL);
    free(pBuf);
    mciSendString("PLAY MOVIE WAIT", NULL, 0, NULL);
    mciSendString("CLOSE MOVIE", NULL, 0, NULL);
    return 0;
}
```

*Listing 46.3:
MCI-Wiedergabe
mit Befehls-
strings*

Befehlsnachrichten

Befehlsnachrichten werden mit der Funktion `mciSendCommand` an Geräte gesendet. Diese Funktion akzeptiert einige Parameter, von denen einer eine befehlspezifische Struktur ist. Die Struktur kann entweder eine allgemeine sein, wie `MCI_OPEN_PARMS`, oder eine gerätespezifische Erweiterung der allgemeinen Version. Anwendungen füllen diese Struktur in geeigneter Weise aus, bevor sie den Aufruf von `mciSendCommand` durchführen. Befehle, die Informationen zurückgeben, tun dies, indem sie Elemente dieser Struktur ändern.

*mciSend-
Command*

Das Programm in Listing 46.4 zeigt eine einfache Wiedergabe mit Hilfe der MCI-Befehlsschnittstelle. Dieses Programm kann von der Kommandozeile aus kompiliert werden:

```
CL MCIMSG.C WINMM.LIB
```

Wie sein Befehlsstring-Gegenstück nimmt auch dieses Programm den Namen einer Multimediadatei auf der Kommandozeile an und spielt sie ab.

```
#include <windows.h>
#include <stdlib.h>

int WINAPI
WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR lpCmdLine, int d4)
{
    MCI_OPEN_PARMS mciOpen;
    MCI_PLAY_PARMS mciPlay;
    MCI_GENERIC_PARMS mciClose;

    mciOpen.dwCallback = 0;
    mciOpen.lpstrElementName = lpCmdLine;
```

*Listing 46.4:
MCI-Wiedergabe
mit Befehlsmel-
dungen*


```

mciOpen.lpstrAlias = "MOVIE";
mciSendCommand(0, MCI_OPEN, MCI_OPEN_ALIAS
| MCI_OPEN_ELEMENT, (DWORD)&mciOpen);

mciPlay.dwCallback = 0;
mciSendCommand(mciOpen.wDeviceID, MCI_PLAY, MCI_WAIT,
(DWORD)&mciPlay);

mciClose.dwCallback = 0;
mciSendCommand(mciOpen.wDeviceID, MCI_CLOSE, 0,
(DWORD)&mciClose);

return 0;
}

```

46.4.1 Die Syntax der Befehlsstrings

Die allgemeine Syntax für MCI-Befehlsstrings lautet:

```
command identifier [argument [, argument]]
```

- Der Abschnitt `command` spezifiziert einen MCI-Befehl wie `PLAY`, `OPEN` oder `CLOSE`.
- Der `identifier` identifiziert ein MCI-Gerät. Das kann ein MCI-Gerätename oder ein Aliasname sein. Der Identifizierer stellt eine Instanz des entsprechenden MCI-Treibers dar, die erzeugt wurde, als der Treiber geöffnet wurde.
- Befehlsargumente werden eingesetzt, um Flags und Parameter für jeden MCI-Befehl anzugeben.

Betrachten Sie zum Beispiel folgenden MCI-Befehlsstring:

```
play music from 0 to 100 wait
```

In diesem String ist `play` ein MCI-Befehl; `music` ist (wahrscheinlich) ein Alias, der erzeugt wurde, als das Gerät geöffnet wurde; `from 0 to 100` ist eine Reihe von Argumenten, die auf den Befehl `play` anwendbar sind.

Das Flag `wait` in diesem Befehl gibt an, daß der Aufruf von `mciSendString` nicht zurückkehren soll, bevor der Befehl beendet ist. Normalerweise kehren Aufrufe sofort zurück, und die Befehle werden im Hintergrund abgearbeitet.

Ein weiteres oft eingesetztes Flag ist `notify`. Die Angabe dieses Flags bewirkt, daß MCI eine Multimedia-Benachrichtigung (`MM_MCNOTIFY`) der Anwendung sendet, sobald ein Befehl ausgeführt ist.

Einige Geräte (zum Beispiel digitale Videogeräte) unterstützen das `test`-Flag. Ein Befehl, der mit diesem Flag gegeben wird, wird nicht ausgeführt; MCI testet immerhin, ob der Befehl vom angegebenen Gerät ausgeführt werden könnte, und gibt eine Fehlermeldung zurück, wenn das nicht zutrifft.

Diese Flags können auch für Befehle, die als Befehlsnachrichten ausgegeben werden, angegeben werden.

46.4.2 MCI-Befehlssatz

MCI-Befehle fallen in verschiedene Kategorien. Diese umfassen

- ➔ Systembefehle,
- ➔ erforderliche Befehle und
- ➔ optionale Befehle.

Systembefehle Die beiden Systembefehle `break` und `sysinfo` werden von MCI selbst erkannt und ausgeführt. Der `break`-Befehl wird verwendet, um eine virtuelle Taste zu belegen, die andere MCI-Befehle abbricht; der `sysinfo`-Befehl gibt Informationen zu MCI-Diensten und Geräten zurück.

Erforderliche Befehle sind jene, die jedes MCI-Gerät implementieren muß. Dieser Satz umfaßt die folgenden fünf Befehle: `capability`, `close`, `info`, `open` und `status`. Die Befehle `capability`, `info` und `status` beziehen Informationen über den Zustand und die Fähigkeiten des Geräts. Die Befehle `open` und `close` werden verwendet, um die Geräte zu öffnen und zu schließen.

Optionale Befehle können in zwei weitere Kategorien eingeteilt werden: grundlegende Befehle und erweiterte Befehle. Grundlegende Befehle umfassen `load` und `save`, `play`, `record`, `stop`, `pause` und `resume`, `seek` und `set` sowie bestimmte Formen des `status`-Befehls. Für die meisten Geräte ist es vernünftig anzunehmen, daß ein Teil dieser Befehle, die auf das Gerät zutreffen, unterstützt werden. Zum Beispiel kann von einem Wiedergabegerät erwartet werden, daß es zumindest die Befehle `play` und `stop` kennt; ein Aufnahmegerät sollte `record` und `stop` unterstützen.

Erweiterte Befehle umfassen einige zusätzliche Konfigurations- und Editierbefehle. Ihre Anwendung sollte nicht erwarten, daß einer dieser Befehle unterstützt wird; statt dessen sollten Sie den Befehl `capability` einsetzen, um die verfügbaren Befehle herauszubekommen.

46.4.3 MCI-Funktionen und -Makros

Wir haben bereits zwei MCI-Funktionen kennengelernt: `mciSendCommand` wird verwendet, um eine MCI-Befehlsnachricht zu versenden, während `mciSendString` zum Senden eines MCI-Befehlsstrings verwendet wird.

MCI-Gerät abfragen

Ein weiterer Satz von drei Funktionen kann verwendet werden, um Informationen über ein MCI-Gerät zu gewinnen.

- Die Funktion `mciGetCreatorTask` gibt den Handle der Task zurück, die ein bestimmtes MCI-Gerät erzeugt hat.
- Die Funktion `mciGetDeviceID` gibt den Identifizierer des MCI-Geräts zurück, das über einen Namen geöffnet wurde.
- Die Funktion `mciGetErrorString` gibt den Fehlerstring zurück, der einem gegebenen Fehlercode entspricht.

Abläufe überwachen

Zwei zusätzliche Funktionen, `mciGetYieldProc` und `mciSetYieldProc`, können verwendet werden, um Prozeduren zu überwachen. Diese Funktionen werden regelmäßig von der MCI aufgerufen, während sie auf das Ende eines Befehls wartet, der mit dem `wait`-Flag ausgegeben wurde.

Zeitformate

MCI bietet auch eine Reihe von Makros, die sich mit Zeitformaten befassen. In den MCI-Befehlen werden verschiedene Zeitformate verwendet, um die Aufnahme- oder Wiedergabeposition zu setzen oder abzufragen. Positionen können als Zeitwerte ausgedrückt werden, als Spuren usw. Zeitformatmakros werden eingesetzt, um Positionswerte zu erzeugen und individuelle Elemente eines Positionswerts abzufragen. Zum Beispiel extrahiert `MCI_HMS_HOUR` die Stundenkomponente einer Position, ausgedrückt als Stunden, Minuten und Sekunden (HMS); `MCI_MAKE_HMS` erzeugt einen Zeitwert aus Parametern, die Stunden, Minuten und Sekunden angeben.

46.4.4 MCI-Benachrichtigungen

MCI-Geräte können zwei Arten von Nachrichten zur Anwendung senden.

- Die Nachricht `MM_MCINOTIFY` wird verwendet, um die Anwendung über das Ende eines Befehls zu unterrichten. Die Parameter dieser Nachricht identifizieren den Befehl und geben an, ob der Befehl er-

folgreich ausgeführt oder ob er wegen eines Fehlers oder einer anderen Bedingung abgebrochen wurde.

- ➔ Die Nachricht `MM_MCISIGNAL` wird als Antwort auf den erweiterten MCI-Befehl `signal` eingesetzt. Durch diesen Befehl können Anwendungen erreichen, daß MCI eine `MM_MCISIGNAL`-Meldung sendet, wenn eine bestimmte Stelle im Kontext erreicht ist.

46.5 Fortgeschrittene Schnittstellen

Windows bietet einige Schnittstellen auf der untersten Ebene zum Umgang mit Multimedia.

46.5.1 Die Funktionen `AVIFile` und `AVIStream`

Die Funktionen und Makros `AVIFile` bieten Zugriff auf Dateien, die Daten im Resource Information File Format (RIFF) enthalten; Beispiele für solche Dateien umfassen digitales Video und Waveform-Audiodateien.

`AVIFile`-Funktionen beruhen auf dem OLE Component Object Model. `AVIFile`-Funktionen können verwendet werden, um Dateien zu öffnen oder zu schließen, Dateien in die Windows-Zwischenablage zu legen und Dateieigenschaften abzufragen oder zu manipulieren.

Die `AVIStream`-Funktionen werden verwendet, um die eigentlichen Audio- oder Videodaten zu lesen oder zu schreiben.

46.5.2 Kundenspezifische Dateien- und Streamhandler

Für Video- und Audio-Datenquellen, die nicht AVI oder Waveform entsprechen, können Sie kundenspezifische Dateien- und Streamhandler erstellen. Kundenspezifische Dateien- und Streamhandler sind installierbare Treiber, die Zugriff auf Daten aus verschiedenen Quellen mit Hilfe des OLE Component Object Model ermöglichen.

Kundenspezifische Handler sind dynamische Linkbibliotheken (»inproc« OLE Server), die die `IAVIFile` und `IAVIStream`-Schnittstellen implementieren. Diese Schnittstellen werden von der Funktionsfamilie `AVIFile` und `AVIStream` benutzt.

46.5.3 DrawDib-Funktionen

DrawDib-Funktionen bieten leistungsfähige Methoden zum Zeichnen geräteunabhängiger Bitmaps (DIBs) in den Grafikmodi mit 8, 16, 24 und 32 Bits.

DrawDib-Funktionen beruhen nicht auf der GDI, sondern schreiben direkt in den Videospeicher. Sie bieten eine Vielzahl von Diensten, vom Strecken und Dithering der Bilder bis hin zum Komprimieren und Entkomprimieren vieler bekannter Formate.

46.5.4 Der Video Compression Manager

Der Video Compression Manager oder VCM bietet Zugriff auf installierbare Kompressoren, die Videodaten in Echtzeit bearbeiten.

Anwendungen können VCM zum Komprimieren und Entkomprimieren der Videodaten nutzen und die Wechselwirkung zwischen Videodaten, kundenspezifischen Daten und Renderern steuern.

46.5.5 Videoaufnahme

Videoaufnahme kann mit der Fensterklasse `AVICap` und einer Reihe von Funktionen und Makros bewältigt werden.

Im einfachsten Ablauf erzeugen Anwendungen ein `AVICap`-Fenster mit der Funktion `capCreateCaptureWindow` und senden Nachrichten an dieses Fenster, um die Aufnahme zu kontrollieren. Es gibt eine große Anzahl an Makros, die das Senden von Nachrichten an `AVICap`-Fenster und das Verarbeiten der Ergebnisse der Nachrichten vereinfachen.

46.5.6 Waveform-Audio aufnehmen und wiedergeben

Windows bietet eine Reihe von Funktionen für die Aufnahme und Wiedergabe von Waveform-Audio. Die einfachste ist `PlaySound`, die das Abspielen von Audiodateien ermöglicht, die in den vorhandenen Speicher passen.

Andere Waveform-Audio-Funktionen können zur Kontrolle über einzelne Waveform-Ein- und Ausgabegeräte verwendet werden.

Weitere Waveform-Audio-Funktionen können zur Kontrolle einzelner Waveform-Ein- und Ausgabegeräte eingesetzt werden.

46.5.7 Der Audio Compression Manager

Windows bietet auch eine Programmschnittstelle zum Audio Compression Manager oder ACM. ACM ermöglicht eine unkomplizierte Kom-

pression und Entkompression von Waveform-Audiodaten während der Aufnahme oder Wiedergabe.

ACM ist als »mapper«, also als »Abbilder«, installiert. Das bedeutet, daß ACM die Anfragen nach Audioaufnahmen oder Wiedergaben abfangen kann und Daten nach Bedarf kodieren oder dekodieren kann. ACM kann auch nach einem Waveform-Gerät oder einem ACM-Kompressor oder Dekompressor, der ein bestimmtes Format behandeln kann, suchen.

46.5.8 MIDI-Aufnahme und Wiedergabe

Das Abspielen von MIDI-Dateien wird vom MIDI-Sequencer verrichtet. MIDI-Funktionen der niederen Ebene ermöglichen Anwendungen die Kontrolle über MIDI-Aufnahme und Wiedergabe sowie die Bearbeitung von MIDI-Daten.

46.5.9 Audio-Mixer

Audiostränge können durch Mixergeräte gesteuert werden. Windows bietet eine Reihe von Funktionen zum Öffnen und zum Einsatz von Mixern.

46.5.10 Verschiedene Multimedia-Dienste

Weitere Multimedia-Dienste umfassen eine Reihe von Funktionen für gepufferte Ein-/Ausgabe auf der niederen Ebene, die für Aufnahme und Wiedergabe auf Medien optimiert sind, Dateidienste für das RIFF-Format, Funktionen und Benachrichtigungen für Joysticks und Funktionen zum Erzeugen und Verwalten hochauflösender Timer.

46.6 Zusammenfassung

Mit der raschen Entwicklung leistungsstarker Personalcomputer, die mit CD-ROM-Laufwerken ausgerüstet sind, wurde Multimedia den Massen nahegebracht. Der Windows-Programmierer kann dieses Gebiet der Programmierung nicht länger ignorieren. Multimedia, wenn auch nur in der Form von Soundeffekten oder Videoclips für Lernzwecke, wird rasch ein Teil jeder neuen Anwendung.

Multimedia unter Windows besteht aus der Fähigkeit, Video- und Audiodateien aufzunehmen und wiederzugeben. Windows erkennt ein Videoformat (AVI-Dateien) und zwei Audioformate (MIDI und Waveform).

Windows bietet Multimediaschnittstellen auf drei verschiedenen Ebenen. Auf der höchsten Ebene ist die Klasse `MCIWnd`. Durch diese Fensterklasse ist es möglich, Videowiedergabe mit einem einzigen Funktionsaufruf durchzuführen. Die API von `MCIWnd` bietet auch eine Vielzahl von Makros, die die Funktion `SendMessage` zum Kommunizieren mit einem `MCIWnd`-Fenster verwenden. Diese Fenster können auch ihrem Elternfenster Benachrichtigungen senden.

Die Programmierschnittstelle der mittleren Ebene ist die Media Control Interface oder MCI. MCI bietet Befehlsstrings und Befehlsnachrichten zur Steuerung der Multimediageräte; diese können abwechselnd eingesetzt werden. Alle MCI-Geräte erkennen einen Kern von Befehlen; viele Geräte kennen zusätzliche, gerätespezifische Befehle. Obwohl die Verwendung der Befehlsstring-Schnittstelle einfacher ist, kann es in manchen Fällen vorteilhafter sein, für Befehle, die Informationen von dem Gerät abfragen, statt dessen Befehlsnachrichten einzusetzen. Anders als Befehlsstrings, die Antworten als Strings zurückgeben, die von der Anwendung analysiert und interpretiert werden müssen, geben Befehlsnachrichten Informationen als Strukturen zurück.

Videodienste der niederen Ebene umfassen die Funktionsfamilie `AVIFile`, Schnittstellen zu Videokompression und Videoaufnahme sowie leistungsfähige Funktionen zum Zeichnen geräteunabhängiger Bitmaps. Für Daten aus nichtstandardisierten Quellen können kundenspezifische Dateien- und Streamhandler entwickelt werden.

Die Funktionsfamilie `AVIFile` sowie kundenspezifische Dateien- und Streamhandler sind auch auf Audio anwendbar. Andere Audiodienste der niederen Ebene umfassen Funktionen für Aufnahme und Wiedergabe von Waveform- und MIDI-Dateien. Zusätzliche Schnittstellen werden für den Zugang zum Audio Compression Manager und Mixergeräten zur Verfügung gestellt.

Weitere Multimedia-Schnittstellen der niederen Ebene umfassen leistungsfähige gepufferte Dateien-Ein-/Ausgabe, Joystick-Steuerung und Multimedia-Timer.

Die Grafikbibliothek OpenGL

Kapitel

47

OpenGL ist eine geräte- und betriebssystemunabhängige Bibliothek für dreidimensionale Grafik und grafisches Rendering. OpenGL wurde ursprünglich von Silicon Graphics Inc. (SGI) für den Einsatz auf ihren hochwertigen grafischen Workstations entwickelt. Seitdem wurde OpenGL zu einem weitverbreiteten Standard mit Implementierungen auf zahlreichen Betriebssystemen und Hardwareplattformen, einschließlich der Betriebssysteme Windows NT und Windows 95.

Windows 95 Service Release 2 enthält nun OpenGL-Unterstützung; dies war in der ursprünglichen Ausgabe von Windows 95 nicht der Fall. Deshalb ist es oft sinnvoll, die OpenGL-Laufzeitdateien gemeinsam mit Ihrer OpenGL-Anwendung zu verteilen. Diese DLLs befinden sich auf der Ausgabe der Microsoft-Developer-Network-Subscription-CD vom Oktober 1995 oder später. Sie können sie auch von Microsofts Website herunterladen.



Zusätzlich zu den Implementierungen der Standard-OpenGL-Bibliothek bietet Windows eine Reihe von Funktionen, die OpenGL in das Betriebssystem integrieren. Im einzelnen werden Funktionen bereitgestellt, die OpenGL-Rendering-Kontexte mit GDI-Gerätekontexten verbinden. Diese Windows-Erweiterungen der OpenGL-Bibliothek werden mit Namen bezeichnet, die mit `wgl` beginnen. Zu diesen OpenGL-Erweiterungen kommen noch einige Win32-API-Funktionen hinzu, die definiert wurden, um bestimmte Aspekte der OpenGL-Programmierung zu erleichtern.

Die OpenGL-Bibliothek ist umfangreich und komplex. Falls Sie Zugang zu einem umfangreichen Satz Handbücher benötigen, sollten Sie die Anschaffung von The OpenGL Reference Manual vom OpenGL

Architecture Review Board oder The OpenGL Programming Guide von Jackie Neider, Tom Davis und Mason Woo in Betracht ziehen. Beide erschienen im Verlag Addison-Wesley.

In diesem Kapitel werde ich mich auf den Einsatz von OpenGL in Windows und MFC-Anwendungen konzentrieren und einen kurzen (von Vollständigkeit weit entfernten) Überblick über die OpenGL-Bibliothek geben.

47.1 Übersicht zu OpenGL

Der Zweck der OpenGL-Bibliothek ist, zwei- und dreidimensionale Gegenstände in einen Rahmenpuffer wie den Pixelspeicher der Grafikkarte Ihres Computers zu rendern.

Die OpenGL-Bibliothek ist grundsätzlich prozedural. Das bedeutet, daß Sie in Ihrer Anwendung nicht beschreiben, wie ein Gegenstand aussieht; statt dessen geben Sie an, wie ein Gegenstand gezeichnet wird. Komplexe geometrische Gegenstände werden anhand einfacher Elemente, die Ihre Anwendung definiert, beschrieben.

Die Implementierung der OpenGL-Bibliothek folgt dem Client-Server-Modell. OpenGL-Clients und Server müssen nicht einmal auf derselben Maschine sein.

47.1.1 Grundlegende Konzepte von OpenGL

Vertex Auf der Basisebene befaßt sich OpenGL mit Vertices. Ein Vertex ist ein Punkt, zum Beispiel der Endpunkt einer Linie oder die Ecke eines Vielecks. Vertices können zwei- oder dreidimensional sein.

Primitive Auf der nächsten Ebene finden wir Primitive. Primitive bestehen aus einer Gruppe von einem oder mehreren Vertices. Zum Beispiel ein Rechteck, als ein Satz von vier Vertices beschrieben, ist eine Primitive.

Eine Reihe von Einstellungen kontrolliert die Art, wie Vertices zu Primitiven zusammengeschlossen und wie Primitive in einen Rahmenpuffer gezeichnet werden. Zum Beispiel können Anwendungen eine dreidimensionale Transformationsmatrix angeben, die definiert, wie die Koordinaten eines Gegenstandes in Koordinaten der Zeichenfläche umgewandelt werden.

Zusätzlich zu ihrer Fähigkeit, Punkte und Linien zu zeichnen, kann OpenGL auch

- ➡ Flächen zeichnen,
 - ➡ Beleuchtungsangaben anwenden und
 - ➡ Textur-Bitmaps
- verwenden.

Ein weiterer Satz von Features ermöglicht Anwendungen das selektive Anwenden oder Löschen von Pixeln. Zum Beispiel kann das Zeichnen eines Pixels von Eigenschaften wie der Pixeltiefe oder -transparenz abhängig gemacht werden.

Eine sehr vereinfachte Ansicht der Arbeitsweise von OpenGL wird in Abbildung 47.1 dargestellt.

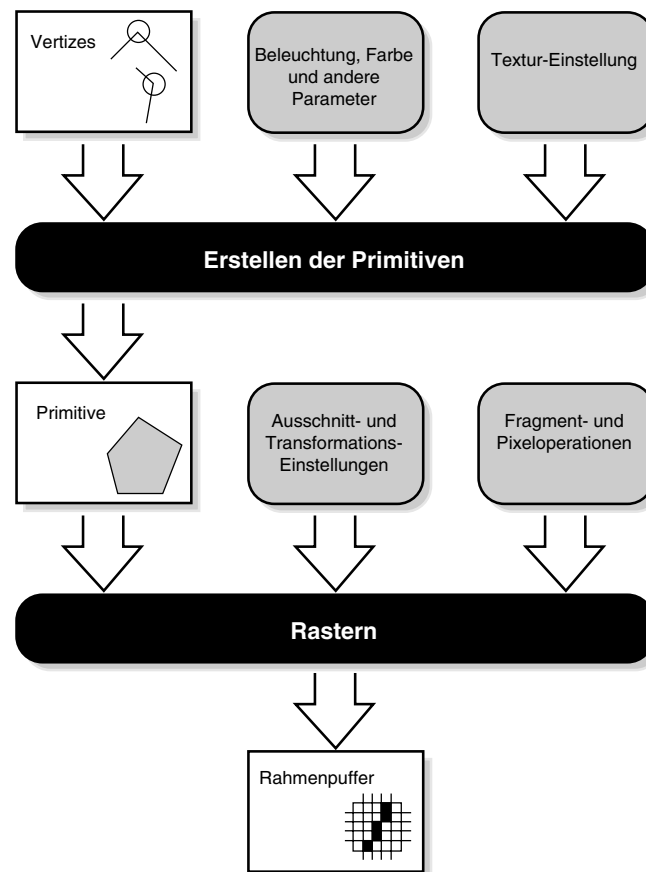


Abbildung 47.1:
Vereinfachte
Ansicht der
OpenGL-Opera-
tionen

47.1.2 Initialisierung

Bevor die OpenGL-Bibliothek eingesetzt werden kann, müssen eine Reihe von Initialisierungsschritten durchgeführt werden.

Rendering-Kontext Jede Windows-OpenGL-Anwendung muß einen Rendering-Kontext mit einem Gerätekontext verbinden. Der Gerätekontext muß der Kontext eines Ausgabegerätes oder eines Speichergerätes, das mit dem Ausgabegerätekontext kompatibel ist, sein. Um einen Rendering-Kontext zu setzen, müssen Anwendungen zuerst die Win32-Funktion `SetPixelFormat` verwenden, um ein Pixelformat für das Gerät zu setzen; als nächstes müssen Sie `wglCreateContext` aufrufen, mit dem Gerätekontext-Handle als Parameter. Wenn erfolgreich, gibt `wglCreateContext` einen Renderingkontext-Handle vom Typ `HGLRC` zurück.



Windows unterstützt das Zeichnen in einen Druckergerätekontext mit der OpenGL-Bibliothek nicht. Falls Sie ein mit OpenGL erstelltes Bild ausdrucken wollen, gibt es die Möglichkeit, in einen mit dem Ausgabegerät kompatiblen Speichergerätekontext zu zeichnen und dann die resultierende Bitmap an den Drucker zu transferieren.

Pixeldaten-Modi OpenGL erkennt unter Windows zwei Arten von Pixeldaten-Modi:

- ➔ RGBA-Formate und
- ➔ farbindexbasierte Formate.

Wenn der RGBA-Modus gewählt ist, werden Pixelfarben als RGB-Farbwerte angegeben. Ist der Farbindexmodus gewählt, so werden die Pixelfarben mit einem Indexwert aus der Systempalette selektiert. Diese beiden Modi werden relevant auf palettenbasierten Geräten mit 256 Farben (zahlreiche VGA-kompatible Videokarten). Wenn Ihre Anwendung den RGBA-Modus auf einem solchen Gerät verwendet, muß sie ihre eigene Palette verwalten und auf palettenbezogene Meldungen von Windows antworten.

Anforderungen an Fenster Es gibt spezifische Anforderungen, die von einem Fenster erfüllt werden müssen, wenn es für OpenGL-Operationen verwendet werden soll. Ein solches Fenster kann nicht mit einer Fensterklasse, die den Stil `CS_PARENTDC` gesetzt hat, erzeugt werden. Das Fenster selbst muß die Stilparameter `WS_CLIPCHILDREN` und `WS_CLIPSIBLINGS` haben, um mit OpenGL kompatibel zu sein.

Beachten Sie, daß Sie zur Verbesserung der Leistung Ihrer Anwendung eine Fensterklasse verwenden können, die einen Null-Hintergrundpinsel verwendet; der Hintergrund des Fensters wird von der OpenGL-Bibliothek sowieso gelöscht.

Bevor ein Rendering-Kontext verwendet werden kann, muß er als aktueller Kontext mit der Funktion `wglMakeCurrent` gesetzt werden. Diese Funktion erwartet zwei Parameter, der eine ist ein Gerätekontext-Handle. Interessanterweise muß dieser Handle nicht mit dem identisch sein, der in `wglCreateContext` verwendet wird, doch muß er sich auf dasselbe Gerät beziehen. Somit ist es zum Beispiel möglich, einen OpenGL-Renderingkontext mit einem Gerätekontext-Handle, der von `GetDC` zurückgegeben wurde, zu setzen, `wglMakeCurrent` hingegen mit einem von `BeginPaint` zurückgegebenen Handle.

Sobald ein Rendering-Kontext bereit ist, Befehle anzunehmen, können Sie zusätzliche Initialisierungsbefehle geben; zum Beispiel könnten Sie den Framepuffer vor dem Zeichnen löschen, Koordinatentransformationen durchführen, Lichtquellen konfigurieren oder andere Optionen de- oder aktivieren.

Ein Schritt, der in der Initialisierung nicht unterlassen werden darf, ist der Aufruf der Funktion `glViewport`. Mit dieser Funktion können Sie die Größe des Rendering-Viewports setzen oder ändern. Typischerweise sollten Sie diese Funktion aufrufen, nachdem der Rendering-Kontext initialisiert ist, und weiterhin jedesmal, wenn Ihre Anwendung eine `WM_SIZE`-Nachricht empfängt, zum Zeichen, daß sich die Fenstergröße geändert hat.

*Rendering-
Viewport setzen*

47.1.3 Zeichnen mit OpenGL

Die meisten OpenGL-Zeichnungen bestehen aus einer Reihe von Vertex-Operationen, die zwischen Aufrufen von `glBegin` und `glEnd` eingeschlossen sind.

*glBegin
glEnd*

➡ Der `glBegin`-Aufruf identifiziert den Typ der Primitiven, die von der folgenden Vertexoperation definiert wird.

➡ `glEnd` markiert das Ende der Konstruktion der Primitiven.

Zum Beispiel bilden die folgenden Aufrufe ein Fünfeck:

Fünfeck zeichnen

```
glBegin(GL_POLYGON);
glVertex2d(0.0, 1.0);
glVertex2d(-0.951057, 0.309017);
glVertex2d(-0.587785, -0.809017);
glVertex2d(0.587785, -0.809017);
glVertex2d(0.951057, 0.309017);
glEnd();
```

Die Funktion `glBegin` kann zum Definieren einer Vielzahl von Primitiven eingesetzt werden. Tabelle 47.1 zählt die erlaubten Parameter dieser Funktion auf.

Tabelle 47.1:
Mit `glBegin`
aufgebaute
Primitive

glBegin-Parameter	Beschreibung
GL_POINTS	Eine Reihe von Punkten
GL_LINES	Eine Reihe von Linien
GL_LINE_STRIP	Eine verbundene Gruppe von Linien
GL_LINE_LOOP	Eine verbundene, geschlossene Gruppe von Linien
GL_TRIANGLES	Ein Satz von Dreiecken
GL_TRIANGLE_STRIP	Ein Satz von verbundenen Dreiecken
GL_TRIANGLE_FAN	Ein Satz von verbundenen Dreiecken
GL_QUADS	Ein Satz von Vierecken
GL_QUAD_STRIP	Ein Satz von verbundenen Vierecken
GL_POLYGON	Ein Vieleck

Wenn `glBegin` einen Satz von verbundenen Primitiven definiert, gibt es bestimmte Regeln, wie Vertices einer Primitiven als Vertices der folgenden Primitiven wiederverwendet werden. Zum Beispiel, wenn `GL_LINE_STRIP` angegeben ist, wird der Vertex, der den Endpunkt eines Linienabschnittes darstellt, zum Anfangspunkt des nächsten Linienabschnittes.

47.1.4 Zusätzliche Bibliotheken

Zusätzlich zu den Grundfunktionen von OpenGL bietet Microsofts Implementierung von OpenGL zwei weitere OpenGL-Bibliotheken.

- ➡ Die OPENGL UTILITY LIBRARY (GLU) enthält eine Reihe von Funktionen, die Texturen, Koordinatentransformation, Rendering von Kugeln, Scheiben und Zylindern, B-Spline-Kurven und Flächen sowie Fehlerbehandlung unterstützen. Außerdem bietet die GLU-Bibliothek Funktionen zur Polygonzerlegung; diese Funktionen können eingesetzt werden, um konvexe oder konkave Vielecke in einfache, konvexe Polygone zu zerlegen (die einzigen, die OpenGL behandeln kann).
- ➡ Die OPENGL PROGRAMMING GUIDE AUXILIARY LIBRARY (GLAUX) bietet außer einigen Funktionen zum Umgang mit dreidimensionalen Objekten auch Funktionen zum Verwalten und Ausführen einer OpenGL-Anwendung. Diese Funktionen sind sehr nützlich beim schnellen Portieren der OpenGL-Anwendungen aus anderen Umgebungen. Vor allem bieten diese Funktionen Fensterverwaltung, implementieren eine einfache Nachrichtenschleife und stellen eine

Fensterprozedur zur Behandlung von Nachrichten bereit. Wie auch immer sind diese Funktionen nicht für den Einsatz in produktiven Anwendungen gedacht.

47.2 Erstellen von OpenGL-Windows-Anwendungen in C

Werfen wir einen Blick auf eine ganz einfache OpenGL-Anwendung. Diese Anwendung, in Listing 47.1 gezeigt, gibt einen Würfel aus. Der Würfel ist leicht gedreht, um dreidimensional zu erscheinen, und wird seitlich beleuchtet. In ihrer Einfachheit ist diese Anwendung das Gegenstück zur Anwendung `Hello, world` unter Windows.

Diese Anwendung kann einfach von der Kommandozeile aus kompiliert werden. Ich habe die Quelldatei `cube.c` genannt; um sie zu kompilieren, geben Sie folgendes ein:

```
CL CUBE.C USER32.LIB GDI32.LIB OPENG32.LIB
```

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>

HGLRC hglrc;

void DrawHello(HWND hwnd)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT clientRect;
    GLfloat lightPos[4] = {-1.0F, 2.0F, 0.6F, 0.0F};

    hDC = BeginPaint(hwnd, &paintStruct);
    if (hDC != NULL)
    {
        GetClientRect(hwnd, &clientRect);
        wglMakeCurrent(hDC, hglrc);
        glViewport(0, 0, clientRect.right, clientRect.bottom);
        glLoadIdentity();
        glClear(GL_COLOR_BUFFER_BIT);
        glColor4d(1.0, 1.0, 1.0, 1.0);
        glRotated(30.0, 0.0, 1.0, 0.0);
        glRotated(15.0, 1.0, 0.0, 0.0);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

        glBegin(GL_QUADS);

        glNormal3d(0.0, -1.0, 0.0);
        glVertex3d(0.5, -0.5, 0.5);
        glVertex3d(-0.5, -0.5, 0.5);
        glVertex3d(-0.5, -0.5, -0.5);
        glVertex3d(0.5, -0.5, -0.5);
```

*Listing 47.1:
Eine einfache
OpenGL-Anwen-
dung*

```

        glVertex3d(0.0, 0.0, -1.0);
        glVertex3d(-0.5, -0.5, -0.5);
        glVertex3d(-0.5, 0.5, -0.5);
        glVertex3d(0.5, 0.5, -0.5);
        glVertex3d(0.5, -0.5, -0.5);

        glVertex3d(1.0, 0.0, 0.0);
        glVertex3d(0.5, -0.5, -0.5);
        glVertex3d(0.5, 0.5, -0.5);
        glVertex3d(0.5, 0.5, 0.5);
        glVertex3d(0.5, -0.5, 0.5);

        glVertex3d(0.0, 0.0, 1.0);
        glVertex3d(-0.5, -0.5, 0.5);
        glVertex3d(-0.5, 0.5, 0.5);
        glVertex3d(0.5, 0.5, 0.5);
        glVertex3d(0.5, -0.5, 0.5);

        glVertex3d(-1.0, 0.0, 0.0);
        glVertex3d(-0.5, -0.5, 0.5);
        glVertex3d(-0.5, 0.5, 0.5);
        glVertex3d(-0.5, 0.5, -0.5);
        glVertex3d(-0.5, -0.5, -0.5);

        glVertex3d(0.0, 1.0, 0.0);
        glVertex3d(-0.5, 0.5, 0.5);
        glVertex3d(0.5, 0.5, 0.5);
        glVertex3d(0.5, 0.5, -0.5);
        glVertex3d(-0.5, 0.5, -0.5);

        glEnd();
        glFlush();
        wglMakeCurrent(NULL, NULL);
        EndPaint(hwnd, &paintStruct);
    }
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                        WPARAM wParam, LPARAM lParam)
{
    switch(uMsg)
    {
        case WM_PAINT:
            DrawHello(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;
    HDC hDC;
    PIXELFORMATDESCRIPTOR pfd;
    int iPixelFormat;

    if (hPrevInstance == NULL)

```

```

{
    memset(&wndClass, 0, sizeof(wndClass));
    wndClass.style = CS_HREDRAW | CS_VREDRAW;
    wndClass.lpfnWndProc = WndProc;
    wndClass.hInstance = hInstance;
    wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndClass.lpszClassName = "HALLO";
    if (!RegisterClass(&wndClass)) return FALSE;
}

hwnd = CreateWindow("HALLO", "HALLO",
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN |
    WS_CLIPSIBLINGS,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
    NULL, NULL, hInstance, NULL);

hDC = GetDC(hwnd);
memset(&pfd, 0, sizeof(pfd));
pfd.nSize = sizeof(pfd);
pfd.nVersion = 1;
pfd.dwFlags = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL;
pfd.iPixelFormat = PFD_TYPE_RGBA;
pfd.iLayerType = PFD_MAIN_PLANE;
pfd.cDepthBits = 16;
iPixelFormat = ChoosePixelFormat(hDC, &pfd);
SetPixelFormat(hDC, iPixelFormat, &pfd);
hglrc = wglCreateContext(hDC);
ReleaseDC(hwnd, hDC);
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
wglMakeCurrent(NULL, NULL);
wglDeleteContext(hglrc);
return msg.wParam;
}

```

Die folgenden Abschnitte erklären die Funktionsweise dieser Anwendung.

47.2.1 Initialisieren von OpenGL

Die erste Serie von OpenGL-Aufrufen in dieser Anwendung beginnt in *WinMain* WinMain, gleich nachdem das Fenster der Anwendung erzeugt wurde. Nach dem Erhalten eines Gerätekontexthandles für den Client-Bereich dieses Fensters wird das Pixelformat des Gerätekontexts auf einen Wert gesetzt, der durch `ChoosePixelFormat` erhalten wurde.

Die Funktion `ChoosePixelFormat` kann verwendet werden, um Pixelformate für ein bestimmtes Gerät zu identifizieren, so daß es am besten einen Satz erforderter Charakteristika erfüllt. *ChoosePixelFormat*

Beachten Sie, daß, obwohl wir den RGBA-Datenmodus verwenden, diese Anwendung keine Palettennachrichten verarbeitet. Folglich kann sie den Würfel nicht richtig in 256-farbigem Palettenmodus darstellen. Dies ist so gemacht, um die Anwendung möglichst einfach zu halten; in einer produktiven Anwendung würden Sie sicherlich nicht versäumen wollen, eine für Ihre Anwendung passende Palette zu erzeugen und zu verwalten.

wglCreateContext Nachdem das Pixelformat festgelegt wurde, wird ein Rendering-Kontext mit einem Aufruf von `wglCreateContext` erzeugt. Der Handle des Rendering-Kontexts wird in einer globalen Variablen abgelegt, auf die aus anderen Funktionen heraus zugegriffen wird.

Wenn alle Initialisierungen abgeschlossen sind, geht die Anwendung in ihre Nachrichtenschleife.

wglMakeCurrent Nach Beenden der Nachrichtenschleife wird vor Beenden der Anwendung durch den Aufruf von `wglMakeCurrent` und `wglDeleteContext` aufgeräumt.
wglDeleteContext

47.2.2 Die Fensterfunktion

Die einfache Fensterfunktion der Anwendung verarbeitet nur zwei Nachrichten: `WM_PAINT` und `WM_DESTROY`. Wenn eine `WM_PAINT`-Meldung empfangen wird, ruft die Fensterfunktion die Funktion `DrawHello` auf. In dieser Funktion finden die Zeichenvorgänge von OpenGL statt.

Der erste Schritt in `DrawHello` ist das Wählen des Rendering-Kontexts als aktuellen Kontext und Setzen der Viewport-Größe durch den Aufruf von `glViewport`. Die Größe des Viewports wurde durch einen Aufruf der Win32-Funktion `GetClientRect` erhalten. Als nächstes wird der Rahmenpuffer gelöscht und eine Identitäts-Transformationsmatrix geladen.

Die Transformationsmatrix wird durch zwei aufeinanderfolgende Drehungen, angegeben in den Aufrufen von `glRotated`, geändert. Der erste Aufruf dreht die Ansicht um die senkrechte Achse. Der zweite Aufruf kippt die Ansicht nach vorne, indem sie eine Drehung um die waagerechte Achse macht. Als Ergebnis werden wir den Würfel aus einem Sichtpunkt sehen, der etwas oberhalb und links vom Würfel liegt.

Die Drehungen werden von Aufrufen gefolgt, die den Beleuchtungsmodus aktivieren und die Lichtquellen festlegen. Der Code gibt eine einzige Lichtquelle an, die den Würfel von links oben beleuchtet.

Nachdem die Initialisierung abgeschlossen ist, kann das eigentliche Zeichnen beginnen. Eine Reihe von sechs Vierecken wird gezeichnet, um die sechs Seiten des Würfels darzustellen. Für jedes der Vierecke wird der Normalvektor durch einen Extraaufruf von `glNormal3d` definiert. Wenn der Aufbau der sechs Primitiven abgeschlossen ist, wird `glFlush` aufgerufen, um sicherzustellen, daß alle OpenGL-Vorgänge abgeschlossen sind. Dann wird der Gerätekontext freigegeben, und die Funktion kehrt zurück.

47.2.3 Kompilieren und Ausführen der Anwendung

Beachten Sie, daß Sie für Anwendungen, die die Bibliothek GLU oder GLAUX verwenden, auch *glu32.lib* oder *glaux.lib* in der Kommandozeile angeben müssen. Da OpenGL rechenintensiv ist, wäre es eine gute Idee, mit entsprechend gesetzten Optimierungsflags zu kompilieren.

Die Anwendung sollte ein Fenster mit dem dreidimensionalen Bild eines Würfels ausgeben, wie in Abbildung 47.2.

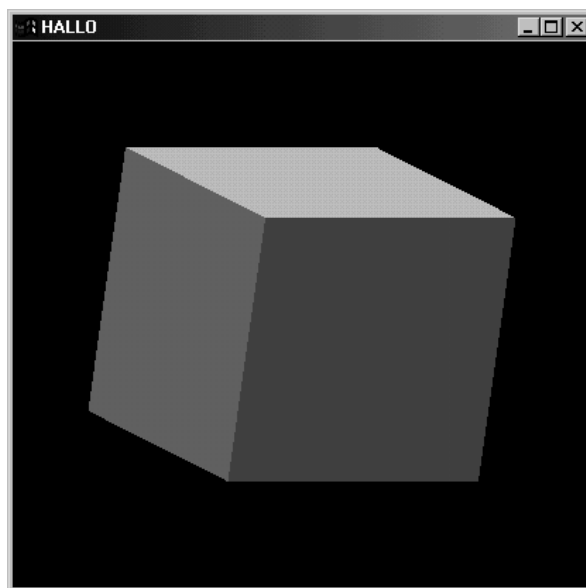


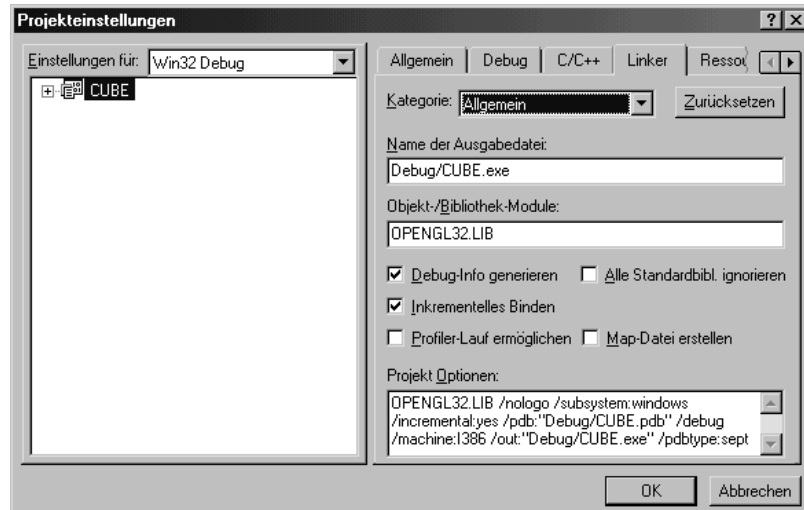
Abbildung 47.2:
Ausführen der
Windows-
Anwendung
cube.exe

47.3 OpenGL in MFC-Anwendungen

Die OpenGL-Bibliothek kann in MFC-Anwendungen ebenso leicht eingesetzt werden. Um die OpenGL-Bibliotheken zu aktivieren, fügen Sie die Namen der entsprechenden Bibliotheken Ihrem Projekt hinzu (Abbildung 47.3).

Wenn in einer MFC-Anwendung die OpenGL-Bibliothek initialisiert wird, ist es wichtig sich zu merken, welches Fenster für einen Rendering-Kontext verwendet werden soll. Zum Beispiel falls es ein Fenster ist, das als Rendering-Kontext dienen soll, muß dieses Fenster beim Erzeugen des Rendering-Kontexts verwendet werden.

Abbildung 47.3:
Hinzufügen der
OpenGL-Biblio-
theken zu einem
MFC-Projekt



47.3.1 Initialisierung von OpenGL

Die in diesem Abschnitt vorgestellte MFC-OpenGL-Anwendung beruht auf einem vom Anwendungsassistenten generierten Anwendungsgerüst mit Single-Dokument-Schnittstelle.

In dieser Anwendung zeichnen wir einen mit dem in der vorher besprochenen C-Anwendung identischen Würfel. Der Würfel wird in das View-Fenster der Anwendung gezeichnet.

PreCreate-Window Folglich ist die erste Aufgabe nach dem Erzeugen des Gerüsts der Anwendung das Ändern der Elementfunktion `PreCreateWindow` der View-Klasse, um sicherzustellen, daß das View-Fenster mit den entsprechenden Flags erzeugt wird.

Die modifizierte Version dieser Funktion wird in Listing 47.2 gezeigt.

Listing 47.2:
Modifizierte Version von CCube-View: : PreCreateWindow

```
BOOL CCubeView::PreCreateWindow(CREATESTRUCT& cs)
{
    cs.style |= WS_CLIPSIBLINGS | WS_CLIPCHILDREN;
    return CView::PreCreateWindow(cs);
}
```

Wie Sie sehen, ist die Änderung an dieser Funktion einfach; sie besteht nur aus dem Hinzufügen der Flags `WS_CLIPSIBLINGS` und `WS_CLIPCHILDREN` zum Fensterstil, um das richtige Funktionieren der OpenGL-Bibliotheken sicherzustellen.

OnCreate Wesentlich mehr Initialisierungsarbeit wird in der Elementfunktion `OnCreate` der View-Klasse durchgeführt. Diese Elementfunktion muß mit dem Klassen-Assistenten oder der Assistentenleiste als Bearbeitungs-

funktion für WM_CREATE-Nachrichten hinzugefügt werden. Die Implementierung dieser Funktion, gezeigt in Listing 47.3, erzeugt einen Rendering-Kontext nach dem Setzen eines Pixelformats für den Gerätekontext des View-Fensters.

```
int CCubeView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    PIXELFORMATDESCRIPTOR pfd;
    int iPixelFormat;
    CDC *pDC;

    pDC = GetDC();
    memset(&pfd, 0, sizeof(pfd));
    pfd.nSize = sizeof(pfd);
    pfd.nVersion = 1;
    pfd.dwFlags = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL;
    pfd.iPixelFormat = PFD_TYPE_RGBA;
    pfd.iLayerType = PFD_MAIN_PLANE;
    pfd.cDepthBits = 16;
    iPixelFormat = ChoosePixelFormat(pDC->m_hDC, &pfd);
    SetPixelFormat(pDC->m_hDC, iPixelFormat, &pfd);
    m_hglrc = wglCreateContext(pDC->m_hDC);
    ReleaseDC(pDC);

    return 0;
}
```

*Listing 47.3:
Implementierung von CCubeView: OnCreate*

Der Handle des Rendering-Kontexts ist in der Elementvariablen `m_hglrc` abgelegt. Diese Elementvariable sollte zur Deklaration der View-Klasse im Abschnitt `Attributes` wie folgt hinzugenommen werden:

```
class CCubeView : public CView
{
    ...
// Attribute
public:
    CCUBEDoc* GetDocument();
    HGLRC m_hglrc;
    ...
}
```

47.3.2 Zeichnen des Würfels

Das eigentliche Zeichnen des Würfels geschieht in der Elementfunktion `OnDraw` der View-Klasse. Diese Elementfunktion, gezeigt in Listing 47.4, ist der Funktion `DrawHello` aus der C-Anwendung von vorher in diesem Kapitel sehr ähnlich. Nachdem der Rendering-Kontext zum aktuellen gemacht wird, macht die Funktion einige Initialisierungen, einschließlich Setzen der Viewport-Größe, Koordinatentransformationen und Festlegen der Beleuchtung. Nachher werden vier Vierecke, die zusammen den Würfel darstellen, gezeichnet.

Listing 47.4:
Implementie-
rung von *CCube-*
View: : OnDraw

```
void CCubeView::OnDraw(CDC* pDC)
{
    CMFCCubeDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CRect clientRect;
    GLfloat lightPos[4] = {-1.0F, 2.0F, 0.6F, 0.0F};

    GetClientRect(&clientRect);
    wglMakeCurrent(pDC->m_hDC, m_hglrc);
    glViewport(0, 0, clientRect.right, clientRect.bottom);
    glLoadIdentity();
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4d(1.0, 1.0, 1.0, 1.0);
    glRotated(30.0, 0.0, 1.0, 0.0);
    glRotated(15.0, 1.0, 0.0, 0.0);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    glBegin(GL_QUADS);

    glNormal3d(0.0, -1.0, 0.0);
    glVertex3d(0.5, -0.5, 0.5);
    glVertex3d(-0.5, -0.5, 0.5);
    glVertex3d(-0.5, -0.5, -0.5);
    glVertex3d(0.5, -0.5, -0.5);

    glNormal3d(0.0, 0.0, -1.0);
    glVertex3d(-0.5, -0.5, -0.5);
    glVertex3d(-0.5, 0.5, -0.5);
    glVertex3d(0.5, 0.5, -0.5);
    glVertex3d(0.5, -0.5, -0.5);

    glNormal3d(1.0, 0.0, 0.0);
    glVertex3d(0.5, -0.5, -0.5);
    glVertex3d(0.5, 0.5, -0.5);
    glVertex3d(0.5, 0.5, 0.5);
    glVertex3d(0.5, -0.5, 0.5);

    glNormal3d(0.0, 0.0, 1.0);
    glVertex3d(-0.5, -0.5, 0.5);
    glVertex3d(-0.5, 0.5, 0.5);
    glVertex3d(0.5, 0.5, 0.5);
    glVertex3d(0.5, -0.5, 0.5);

    glNormal3d(-1.0, 0.0, 0.0);
    glVertex3d(-0.5, -0.5, 0.5);
    glVertex3d(-0.5, 0.5, 0.5);
    glVertex3d(-0.5, 0.5, -0.5);
    glVertex3d(-0.5, -0.5, -0.5);

    glNormal3d(0.0, 1.0, 0.0);
    glVertex3d(-0.5, 0.5, 0.5);
    glVertex3d(0.5, 0.5, 0.5);
    glVertex3d(0.5, 0.5, -0.5);
    glVertex3d(-0.5, 0.5, -0.5);

    glEnd();
    glFlush();
    wglMakeCurrent(NULL, NULL);
}
```

Beachten Sie, daß diese Implementierung nicht berücksichtigt, daß das MFC-Anwendungsgerüst die Funktion `OnDraw` der View-Klasse aufruft, wenn in einen Druckergerätekontext gezeichnet wird. In diesem Zustand würden Versuche, die Anwendung zum Drucken einzusetzen, scheitern.

47.3.3 Ausführen der Anwendung

Bevor Sie die Anwendung kompilieren können, müssen Sie in Ihre Klassendeklarationsdatei `CUBEView.cpp` zwei Header-Dateien von OpenGL einbinden. Fügen Sie folgende zwei Zeilen gleich nach `#include "stdafx.h"` hinzu:

```
#include <GL/gl.h>
#include <GL/glu.h>
```

Vergessen Sie auch nicht, die OpenGL-Bibliothekssdatei zum Projekt hinzuzufügen, falls Sie es noch nicht getan haben (siehe Abbildung 47.3).

Um die Anwendung aufzurufen, kompilieren Sie sie und führen sie aus dem ERSTELLEN-Menü aus.

Das Fenster der Anwendung sollte ähnlich dem aus Abbildung 47.4 erscheinen. Wie in unserem vorherigen OpenGL-Beispiel verarbeitet auch dieses Programm die Palettenmeldungen nicht in geeigneter Weise, was zur Folge hat, daß die Ausgabe in 256-farbigen Paletten-Video-modi unvollständig oder korruptiert erscheint.

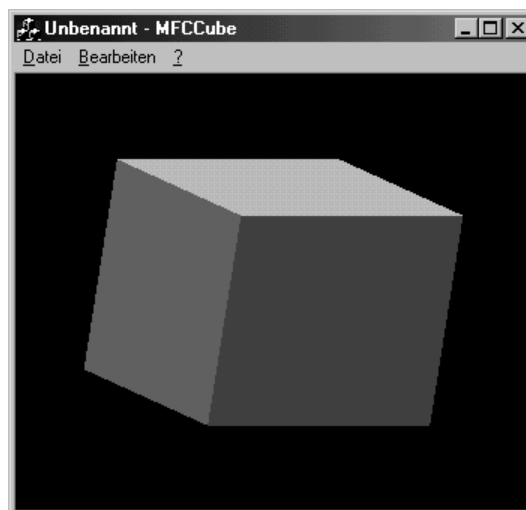


Abbildung 47.4:
Ausführen der
`cube.exe-MFC-`
Anwendung

47.4 Zusammenfassung

OpenGL ist eine Bibliothek von Funktionen hoher Qualität für dreidimensionale Grafik und Rendering. Die Geräte- und Plattformunabhängigkeit der Bibliothek läßt sie als beste Wahl bei der Entwicklung portierbarer Grafikanwendungen erscheinen.

OpenGL-Zeichnungen werden aus Primitiven aufgebaut; Primitive sind einfache Dinge wie Linien oder Vielecke, die ihrerseits aus Vertices bestehen.

Die OpenGL-Bibliothek bildet Primitive aus Vertices unter Berücksichtigung einer Vielfalt von Einstellungen wie Farbe, Beleuchtung und Textur. Die Primitiven werden dann je nach Transformationen, Clipping-Einstellungen und anderen Parametern verarbeitet; am Ende des Rasterungsvorgangs stehen Pixeldaten in einem Rahmenpuffer.

Die Windows-Implementierung der OpenGL-Bibliothek besteht aus der Kernbibliothek, Utility-Funktionen (GLU) und zusätzlichen Funktionen (GLAUX). Die zusätzliche Bibliothek kann eingesetzt werden, um einfache eigenständige OpenGL-Anwendungen zu erstellen, da sie Meldungsschleifen und Fensterprozeduren intern erzeugt. Trotzdem sollte diese Bibliothek nicht in produktiven Anwendungen eingesetzt werden, da die Implementierung sehr einfach ist.

Windows bietet auch eine Reihe von Erweiterungsfunktionen (WGL), die den Einsatz der OpenGL-Funktionen im Kontext der Windows-GDI vereinfachen. Außerdem wurden der Win32-API neue Funktionen hinzugefügt, die Pixelformate und doppelte OpenGL-Pufferung unterstützen.

Die wichtigsten Schritte in der Erstellung einer Windows-OpenGL-Anwendung sind folgende:

- ➔ Versichern Sie sich, daß Ihre Fensterklasse nicht mit dem Stil `CS_PARENTDC` erzeugt wurde. Versichern Sie sich, daß Ihr Fenster mit den Stilparametern `WM_CLIPCHILDREN` und `WM_CLIPSIBLINGS` erzeugt wird.
- ➔ Erzeugen Sie einen OpenGL-Rendering-Kontext. Eine geeignete Stelle, dies zu tun, befindet sich in der Handler-Funktion von `WM_CREATE` für das Fenster, das Sie mit OpenGL verwenden wollen.
- ➔ Fügen Sie passende Aufrufe in Ihre `WM_PAINT`-Bearbeitungsfunktion ein, um das OpenGL-Bild zu zeichnen.

- ➔ Optional können Sie eine Bearbeitungsfunktion für `WM_SIZE`-Meldungen einbauen, um Änderungen in der Größe des Viewports zu berücksichtigen. Verwenden Sie `glViewport`, um die Größe des Viewports zu setzen.
- ➔ Falls Sie beabsichtigen, Ihre Anwendung auf 256farbigen Geräten zu betreiben, fügen Sie die Behandlung kundenspezifischer Paletten hinzu.

Hochleistungsgrafik und Ton

Kapitel 48

Am Anfang meiner Karriere hatte ich die Gelegenheit, fast ein Jahr als Mitglied in einem Programmiererteam mitzuarbeiten, das Spiele für den Commodore-64-Heimcomputer entwickelte. Obwohl wir kaum fürs Essen genug verdienten, habe ich die schönsten Erinnerungen an diese Zeit. Ich glaube nicht, daß ich jemals soviel Spaß mit Computern hatte wie damals beim Entwerfen grüner Drachen, beim Schreiben von Echtzeitprogrammen, beim Kodieren von Bartoks Allegro Barbaro auf dem primitiven Soundsynthesizer des C64 und beim Erlernen der Hochleistungs-Zeichenalgorithmen. Spieleprogrammieren ist wohl die größte technische Herausforderung beim Programmieren von Desktop-Systemen, und die Freude zu sehen, wie das eigene Werk zum Leben erwacht, der Lohn selbst.

Spiele unter Windows sind so alt wie Windows selbst; ich kann mich vague erinnern, auf einer Maschine unter Windows 1.0 Reversi gespielt zu haben. Trotzdem blieb das Ziel, Spiele im Arkadenstil mit Echtzeitanimation und Ton unter Windows zu entwickeln, bis unlängst schwer erreichbar. Und das trotz der Tatsache, daß die Vorteile der Windows-Plattform für die Spieleentwicklung enorm sind. Unter Windows muß man sich nicht länger mit der Vielfalt von Grafikbeschleunigern, Soundkarten, TSR-Treibern und dem Rest der Paraphernalien herum-schlagen, die das Entwickeln der einfachsten Grafikanwendung unter DOS zum Alptraum machten. Die Geräteunabhängigkeit von Windows erledigt das alles.

Ein Grund, weshalb die Spieleentwicklung unter Windows so lange hinter der DOS-Plattform zurückblieb, ist die Geschwindigkeit. Das GDI, obwohl großartig bezüglich der Geräteunabhängigkeit, ist nicht sehr effizient; das Neuzeichnen eines Fensters mit einer relativ großen Fläche erfordert eine bemerkenswerte Zeit.

In der Vergangenheit versuchte Microsoft, den Spieleentwicklern leistungsfähige Grafikbibliotheken und andere Tools zur Verfügung zu stellen. Trotzdem kann nichts davon mit ihrem neuesten Produkt, dem DirectX SDK (früher als Windows 95 Game SDK herausgegeben), verglichen werden. Dieses Produkt ist eine Familie von Bibliotheken für Grafik, Sound, Joystick-Steuerung und Kommunikation (für Spiele mit mehreren Spielern). Diese Bibliotheken basieren auf Microsofts Common-Object-Modell, die Grundlage der OLE-Technik von Microsoft.

Die letzte Version des DirectX SDK und seiner Komponenten kann von Microsofts Website geladen werden. Es wird auch den Abonnenten der Microsoft Developer Network Professional ausgeteilt.

Die Unterstützung von DirectX ist nicht mehr auf Windows 95/98 eingeschränkt. Windows NT 4.0 bietet auch Unterstützung für viele Features von DirectX.

48.1 Die APIs von DirectX

Das DirectX SDK besteht aus einer Reihe von APIs. Die Namen dieser APIs beginnen mit dem Wort `Direct` und werden deshalb als DirectX APIs bezeichnet.

DirectDraw	Die erste dieser APIs, <code>DIRECTDRAW</code> , bietet eine Low-level-Hochgeschwindigkeitsschnittstelle zur Grafik-Hardware Ihres Computers. Durch DirectDraw ist es möglich, Spiele zu entwickeln, die Echtzeitanimation in einem Fenster verwenden. Die API bietet auch die Fähigkeit, Anwendungen über den gesamten Bildschirm zu schreiben.
Direct3D	Das letzte Mitglied der DirectX-Familie, <code>DIRECT3D</code> , bietet Unterstützung für 3D-Grafik und Animation.
DirectSound	Die API <code>DIRECTSOUND</code> bietet Low-level-Zugriff auf die Sound-Hardware Ihres Computers. Die wichtigsten Features dieser API sind die Möglichkeit, Sounds laufend in einer Schleife abzuspielen, sowie die Möglichkeit, sie zu mixen.
DirectPlay	Die API <code>DIRECTPLAY</code> bietet eine spieleorientierte Kommunikationsschnittstelle, um mehrere Spieler über ein Modem, Netzwerk oder einen Online-Dienst zu verbinden.
DirectInput	Die API <code>DIRECTINPUT</code> bietet einen verbesserten, zuverlässigeren Zugriff auf die Funktionen des Joystick.

DirectSetup Schließlich bietet die API `DIRECTSETUP` eine Möglichkeit, die Laufzeitkomponenten von DirectX über eine einzige Funktion einzustellen.

48.1.1 DirectX und das Common-Object-Modell

Die Dienste der API von DirectX werden durch das Common-Object-Modell (COM) angeboten. Ein COM-Objekt ist der Außenwelt durch eine oder mehrere Schnittstellen ausgesetzt. Jede der Schnittstellen ist im Grunde eine Tabelle von Zeigern auf die Funktionen, die die `METHODEN` des Objekts darstellen. Somit ähnelt die Schnittstelle der Implementierung einer reinen virtuellen Klasse in C++; die tatsächlichen Implementierungen dieser Funktionen sind im Objekt selber enthalten.

Die Ähnlichkeit mit C++-Klassen endet bei der Vererbung. Obwohl es möglich ist, eine neue Schnittstelle aus einer alten zu erzeugen, wird die neue die Methoden der alten nicht erben. Ein Beispiel: Obwohl alle COM-Schnittstellen von der Basisschnittstelle `IUnknown` abgeleitet sind, müssen Sie die nötigen Methoden von `IUnknown` einzeln implementieren.

COM-Schnittstellen sind kompatibel mit der Implementierung von C++-Klassen. Die Methodentabelle einer COM-Schnittstelle kann leicht in virtuelle Elementfunktionen der C++-Klasse, die die Schnittstelle darstellt, umgesetzt werden. Wenn Anwendungen COM-Schnittstellen aus der Sprache C heraus verwenden, müssen sie sich explizit auf die virtuelle Funktionstabelle (`vtable`) der Schnittstelle beziehen. Die Headerdateien von DirectX API stellen bequeme Makros für alle DirectX-Methoden zur Verfügung.

Wie ich bereits erwähnte, sind alle COM-Schnittstellen von `IUnknown` abgeleitet. Die Schnittstelle von `IUnknown` kennt drei Methoden.

- ➡ Die erste dieser Methoden, `QueryInterface`, kann eingesetzt werden, um zu bestimmen, ob eine bestimmte Schnittstelle vorhanden ist.
- ➡ `AddRef` und `Release` implementieren das Zählen der Referenzen der Objekte, während sie erzeugt, von anderen Objekten referenziert oder freigegeben werden.

Diese Methoden müssen von allen COM-Schnittstellen, die von `IUnknown` abgeleitet sind, implementiert werden.

Laut Vereinbarung müssen die Namen der COM-Schnittstellen mit dem Großbuchstaben `I` beginnen. Ebenfalls laut Vereinbarung werden die Methoden mit einer C++-Syntax referenziert, obwohl sie auch von C aus mit einer expliziten Referenz auf `vtable` aufgerufen werden können.

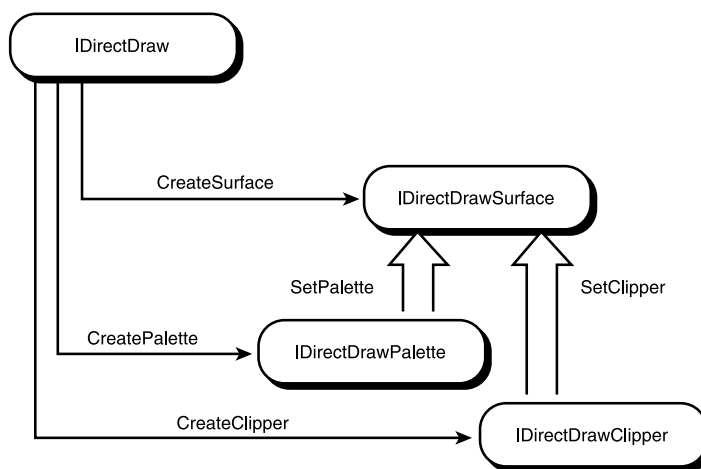
48.1.2 DirectDraw

DirectDraw ist ein Client für die Dienste der HAL (Hardware Abstraction Layer) und HEL (Hardware Emulation Layer) von DirectDraw. DirectDraw ist in der dynamischen Linkbibliothek *ddraw.dll* implementiert. Die HAL von DirectDraw implementiert geräteabhängige Funktionen; sie implementiert nur Funktionen, die vom Gerät unterstützt werden. Für nicht unterstützte Funktionen meldet die HAL einfach, daß sie nicht verfügbar sind. Die HAL führt keine Parametervalidierung durch; die gesamte Parametervalidierung wird von DirectDraw durchgeführt.

Die HEL von DirectDraw bietet eine Software-Emulation für Funktionen, die in der Hardware nicht implementiert sind. Die HAL und HEL von DirectDraw sind als Treiberbibliotheken implementiert wie *ati.vxd* und *atim32.drv*.

Die DirectDraw-API bietet Zugang zu den Low-level-Grafikdiensten über eine Reihe von COM-(Common-Object-Model-)Schnittstellen. Diese Schnittstellen und ihre Beziehungen sind in Abbildung 48.1 schematisch dargestellt.

Abbildung 48.1:
DirectDraw-
COM-Schnitt-
stellen



IDirectDraw2 Die erste dieser Schnittstellen, *IDirectDraw2*, stellt die Grafikhardware (Beschleunigerkarte) in Ihrem Computer dar. Anwendungen, die Dienste von DirectDraw verwenden, beginnen ihren Ablauf mit dem Erzeugen eines DirectDraw-Objekts durch die Funktion *DirectDrawCreate*. *DirectDrawCreate* gibt einen Zeiger auf eine *IDirectDraw*-Schnittstelle zurück. Bevor sie etwas anderes tun, müssen Anwendungen auch *IDirectDraw::SetCooperativeLevel* aufrufen. Anwendungen, die einen ausschließlichen Zugriff auf die Grafikhardware des Computers über

`IDirectDraw::SetCooperativeLevel` erfordern, können den Videomodus ändern und Vollbild-Grafik und Animation implementieren. Andere Anwendungen sind auf den aktuellen Videomodus beschränkt und sollten ihre grafischen Aktivitäten dem Fenster widmen, das sie besitzen.

Die Methode `IDirectDraw::CreateSurface` erzeugt ein Objekt des Typs `DirectDrawSurface` und gibt einen Zeiger auf eine `IDirectDrawSurface`-Schnittstelle zurück. `DirectDrawSurface`-Objekte stellen rechteckige Bereiche im Speicher (typischerweise im Videospeicher) dar, in die Anwendungen hineinzeichnen können. Die primäre Zeichenfläche ist die sichtbare Anzeigefläche; sekundäre Zeichenflächen können im Videospeicher oder im Hauptspeicher des Computers existieren und werden für das Zeichnen außerhalb des sichtbaren Bereichs verwendet.

Die Schnittstelle `IDirectDraw2` ersetzt die Schnittstelle `IDirectDraw`. Aus Gründen der Kompatibilität gibt `DirectDrawCreate` einen Zeiger auf eine `IDirectDraw`-Schnittstelle zurück. Falls Sie die Fähigkeiten der Schnittstelle `IDirectDraw2` benötigen, so kann diese Schnittstelle durch den Aufruf von `IDirectDraw::QueryInterface` erzeugt werden. Ähnlich gibt der Aufruf von `IDirectDraw::CreateSurface` einen `IDirectDrawSurface`-Zeiger zurück. Um eine `IDirectDrawSurface2` zu erzeugen, rufen Sie `IDirectDraw::QueryInterface` auf.



Das Zeichnen auf eine Fläche ist durch eine Reihe von Bit-Blt-Methoden möglich, die von der `IDirectDrawSurface`-Schnittstelle verfügbar gemacht werden, oder durch Verwenden gewöhnlicher GDI-Funktionen. Die Methode `IDirectDrawSurface::GetDC` kann verwendet werden, um einen Gerätekontext-Handle auf die Fläche zu erhalten. GDI wird diesen Handle als einen Handle auf einen Speicher-Gerätekontext behandeln, auch wenn die Fläche die primäre Zeichenfläche ist.

`IDirectDrawSurface` unterstützt glatte Animation mit Back-Buffer-Flächen. Die Methode `IDirectDrawSurface::Flip` kann verwendet werden, um den Inhalt der primären Fläche und der Back-Buffer-Fläche auszutauschen, im Hintergrund zu zeichnen, die beiden Flächen umzuschalten, wenn das Zeichnen beendet ist, und das nächste Bild im Hintergrundpuffer zu zeichnen. Sicherlich ist dies eine von mehreren Techniken, die zur glatten Animation führen. Falls zu einem gegebenen Zeitpunkt nur kleine Ausschnitte der Ausgabefläche aktualisiert werden, kann eine andere Technik (wie das Verwenden von mehreren Puffern, wie sie ein Beispiel weiter unten in diesem Kapitel implementiert sind) von größerem Vorteil sein.

Eine andere Schnittstelle, `IDirectDrawPalette`, bietet Zugriff auf Palettendienste. Diese Schnittstelle stellt die Hardware-Palette dar und um-

IDirectDraw-Palette

geht die Windows-Palette. Deshalb erfordert der Einsatz von `IDirectDrawPalette` einen ausschließlichen Zugriff auf die Video-Hardware. `IDirectDrawPalette` kann für zahlreiche Effekte eingesetzt werden, einschließlich Paletten-Animation. Beachten Sie, daß diese Schnittstelle auf Systemen, die eine Hardware-Palette nicht unterstützen, nicht verfügbar sein kann.

Eine `IDirectDrawPalette`-Schnittstelle wird durch einen Aufruf von `IDirectDraw::CreatePalette` erzeugt. Die Palette muß über `IDirectDrawSurface::SetPalette` einer Ausgabefläche zugeordnet werden.

IDirectDraw-Clipper Die Schnittstelle `IDirectDrawClipper` stellt eine Clip-Liste dar. Eine Clip-Liste kann einem `DirectDrawSurface`-Objekt hinzugefügt und während der Bit-Blt-Operationen verwendet werden. Ein Fenster-Handle kann auch einer Clip-Liste hinzugefügt werden; in diesem Fall stellt die Clip-Liste die Clipping-Rechtecke des Fensters dar.

Die Schnittstelle `IDirectDrawClipper` wird erzeugt, wenn `IDirectDraw::CreateClipper` aufgerufen wird, und wird einer Zeichenfläche durch den Aufruf von `IDirectDrawSurface::SetClipper` hinzugefügt.

Die API von `DirectDraw` kann nicht nur eingesetzt werden, um die primäre Video-Hardware in Ihrem Computer darzustellen, sondern auch für sekundäre Ausgabegeräte, die normalerweise von Windows nicht erkannt werden. Betrachten wir zum Beispiel ein Entwicklungssystem, das eine sekundäre Videokarte und einen Monitor für Testzwecke hat. Eine `IDirectDraw`-Schnittstelle, die dieses sekundäre Gerät darstellt, kann durch einen Aufruf von `DirectDrawCreate` erzeugt werden. Der erste Parameter dieser Funktion stellt den GUID (globally unique identifier) eines Ausgabetreibers dar; wenn er auf `NULL` gesetzt ist, erzeugt `DirectDrawCreate` eine Schnittstelle, die den aktiven Ausgabetreiber darstellt. Trotzdem können Sie einen sekundären GUID angeben, der jedwedes sekundäre Gerät in Ihrem System darstellt.

48.1.3 Direct3D

`Direct3D` besteht aus einer Reihe von COM-Objekttypen, die, mit `DirectDraw` eng verbunden, eine Unterstützung für dreidimensionale Grafik und Animation auf der niederen Ebene bieten.

`Direct3D` selbst besteht aus zwei APIs.

RETAINED MODE ist eine High-level-API zum Verwalten von dreidimensionalen Objekten und Erzeugen von dreidimensionalen Szenen.

Der Retained Mode von Direct3D unterstützt Objekte wie Animationen und Animations-Sets, Geräte, visuelle Objekte, beschrieben als Meshes von Polygonflächen, Rahmen, Lichtquellen, Materialien, Schatten, Texturen und mehr.

IMMEDIATE MODE ist eine Low-level-API zur Kommunikation mit der Beschleuniger-Hardware.

Der Immediate Mode von Direct3D erkennt eine Familie von acht Objekttypen: Schnittstellen, Geräte, Texturen, Materialien, Lichtquellen, Viewports, Matrizen und Ausführungspuffer. Die letzteren enthalten Informationen über Vertices und Operationscodes, die zusammen das Rendering definieren.

48.1.4 DirectSound

Die DirectSound-API bietet Zugriff auf die Waveform Soundhardware Ihres Computers. Das Soundgerät wird durch die Schnittstelle `IDirectSound` dargestellt; einzelne Puffer sind durch `IDirectSoundBuffer` dargestellt.

DirectSound bietet keine MIDI-Funktionalität. Um die MIDI-Fähigkeiten Ihrer Soundhardware zu verwenden, setzen Sie die Standard Win32 Multimedia APIs für MIDI ein.



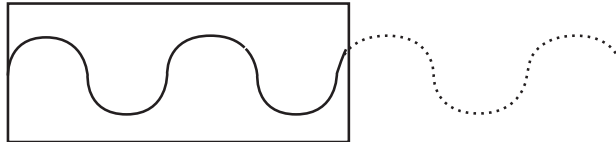
Die wichtigste Fähigkeit von DirectSound ist das Mixen der Waves. Dies wird durch die Verwendung einer Reihe von primären und sekundären Sound-Puffern erreicht.

Ein primärer Puffer stellt den Hardware-Puffer des Soundgeräts dar, also den Puffer, der gerade spielt. Sekundäre Puffer können verschiedene Audioströme darstellen, die zum Abspielen im primären Puffer zusammengemixt werden. Dieser Mechanismus wird schematisch in Abbildung 48.2 dargestellt.

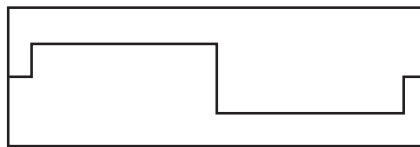
Eine `IDirectSound`-Schnittstelle wird durch den Aufruf von `DirectSoundCreate` erzeugt. Bevor die Schnittstelle verwendet werden kann, müssen Sie auch `DirectSoundCreate::SetCooperativeLevel` aufrufen, um die erforderliche Zugriffsebene auf die Soundkarte anzugeben. Für die meisten Anwendungen sollte dies `DSSCL_NORMAL` sein. Diese Zugriffsebene sichert ein glattes Zusammenspiel zwischen Anwendungen, die um dieselben Hardware-Ressourcen konkurrieren.

Abbildung 48.2:
Mischen der
Waves in
DirectSound

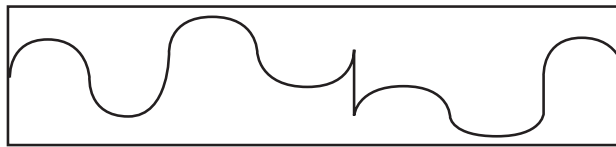
Sekundärer Puffer 1 (Schleife)



Sekundärer Puffer 2



Primärer Puffer



Ein Sound-Puffer wird durch den Aufruf von `IDirectSound::CreateSoundBuffer` zugewiesen. Eine Schnittstelle zum Soundpuffer wird in der Form eines `IDirectSoundBuffer`-Zeigers zurückgegeben. Anwendungen müssen normalerweise keinen primären Soundpuffer allokalieren; dieser Puffer wird implizit zugewiesen, wenn die Inhalte der sekundären Puffer abgespielt werden.

Der Inhalt eines Puffers wird mit der Methode `IDirectSoundBuffer::Play` abgespielt. Der Aufruf dieser Funktion während des Abspielens eines Puffers wird die Playback-Flags aktualisieren, aber den Ablauf sonst nicht beeinflussen (zum Beispiel wird er keinen Neustart des Abspielens am Pufferanfang bewirken). Um die Position im Abspielen zu ändern, verwenden Sie `IDirectSoundBuffer::SetCurrentPosition`.

`IDirectSoundBuffer::Play` kann auch für ständiges Abspielen eingesetzt werden. Diese Möglichkeit ist ideal für Hintergrundsounds, wie das Geräusch der Motoren in einem Flugsimulationsspiel.

48.1.5 DirectPlay

Spieleorientierte Kommunikationsdienste sind durch die API von `DirectPlay` verfügbar.

Die DirectPlay-API besteht aus zwei Komponenten:

- der IDirectPlay-Schnittstelle und
- dem DirectPlay-Server.

Microsoft bietet DirectPlay-Server für Modem und Netzwerkanschlüsse; andere Server (zum Beispiel Server für Online-Dienste) werden von Drittanbietern zur Verfügung gestellt. Um herauszubekommen, welche Server auf einem Computer installiert sind, setzen Sie die Funktion `DirectPlayEnumerate` ein.

Ein DirectPlay-Objekt wird von der Funktion `DirectPlayCreate` erzeugt. Sie müssen den GUID des gewählten DirectPlay-Servers dieser Funktion übergeben. Dann gibt `DirectPlayCreate` einen Zeiger auf eine IDirectPlay-Schnittstelle zurück.

Sie können vorhandene DirectPlay-Sitzungen aufzählen, indem Sie `IDirectPlay::EnumSessions` aufrufen. Diese Methode muß aufgerufen werden, nachdem das DirectPlay-Objekt erzeugt wurde.

Sie können mit der Methode `IDirectPlay::Open` eine DirectPlay-Sitzung erzeugen oder sie an eine bereits vorhandene anschließen. Diese Methode stellt die eigentliche Verbindung her. DirectPlay ruft die nötige Anwenderschnittstelle auf, um das Kommunikationsprotokoll zu konfigurieren. Ist zum Beispiel eine Modemverbindung angefordert, so wird DirectPlay einen Dialog öffnen, der die Telefonnummer und andere Anwahlinformationen abfragt.

Spiele, die DirectPlay verwenden, müssen durch einen globalen eindeutigen Identifizierer GUID identifiziert werden. Ein GUID kann mit der Utility `guidgen.exe`, die in Visual C++ enthalten ist, generiert werden.

Nachdem Sie sich an eine Sitzung angeschlossen haben, müssen Sie Spieler erzeugen. Ein Spieler wird mit `IDirectPlay::CreatePlayer` erzeugt. Eine Liste der Spieler in der Session wird durch den Aufruf von `IDirectPlay::EnumPlayers` erstellt.

Um Meldungen zwischen den Spielern auszutauschen, können Sie `IDirectPlay::Send` und `IDirectPlay::Receive` einsetzen.

Es gibt weitere Methoden zum Verwalten der Sitzungen, Spieler, Spielergruppen und Meldungen. Spieler werden zerstört mit `IDirectPlay::DestroyPlayer`, eine Session mit `IDirectPlay::Close` beendet.

48.1.6 DirectInput

Die API von DirectInput besteht aus Diensten für Joystick, Maus und Tastatur. Die Unterstützung des Joysticks wird durch die Funktion `joyGetPosEx` gewährleistet, die Informationen zur Position und Schalterstatus für Joysticks mit sechs Freiheitsgraden (Achsen) und 32 Schaltern zurückgeben kann, sowie durch Funktionen zur Abfrage der Fähigkeiten: `joyGetDevCaps`, `joyGetNumDevs` und `joyConfigChanged`. Die Unterstützung für Tastatur und Maus wird durch die COM-Objekttypen `DirectInput` und `DirectInputDevice` erreicht.

48.1.7 DirectSetup

Die DirectSetup-API bietet eine einzelne Funktion zur Einstellung von DirectX-Komponenten, die im Rahmen von Anwendungen weitergegeben werden.

Wenn Sie Anwendungen, die APIs von DirectX verwenden, weitergeben, müssen Sie sie mit den weiterverteilbaren Dateien von DirectX weitergeben. Diese Dateien müssen, laut Lizenzvereinbarung von DirectX, in unveränderter Form weitergegeben werden. Unter den DirectX-Dateien befinden sich auch die DirectSetup-DLLs, die die Funktion `DirectXSetup` implementieren.

`DirectXSetup` wird mit Parametern aufgerufen, die das Stammverzeichnis der Installation festlegen, sowie mit Flags, die angeben, welche Komponenten von DirectX installiert werden sollen. Obwohl es möglich ist, nur ausgewählte Komponenten zu installieren, empfiehlt Microsoft, dies nicht zu tun; Einsparungen in der Festplattenkapazität wären minimal aufgrund der Abhängigkeiten zwischen den Komponenten.

Das Vorhandensein von DirectX ist mehr als nur eine Annehmlichkeit. Wegen der Komplexität der Installation von DirectX sollten Entwickler nicht versuchen, von Hand zu installieren.

48.2 Ein einfaches Beispiel

Die Anwendung in Listing 48.1 verwendet `DirectDraw` und `DirectSound`-Dienste, um einen lauten hüpfenden Ball zu implementieren.

Die Anwendung kann von der Befehlszeile aus mit dem C/C++-Compiler von Visual C++ kompiliert werden:

```
RC BOUNCE.RC
```

```
CL BOUNCE.CPP BOUNCE.RES USER32.LIB GDI32.LIB DDRAW.LIB
DSOUND.LIB
```

Die fertige Anwendung sieht aus wie in Abbildung 48.3. Um die Anwendung zum Laufen zu bringen, müssen Sie die Laufzeitbibliotheken von DirectX installiert haben.

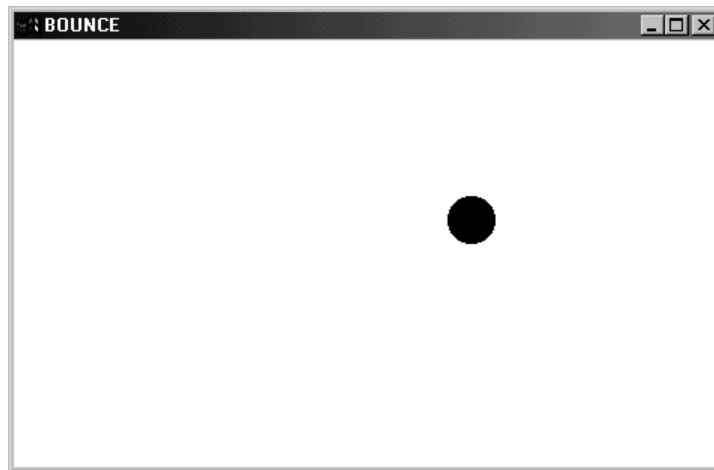


Abbildung 48.3:
Die Anwendung
hüpfender Ball

```
#include <windows.h>
#include <ddraw.h>
#include <dsound.h>

IDirectDraw *dd;
IDirectDrawSurface *dds0, *dds1, *dds2, *dds3;
IDirectDrawClipper *ddc;

IDirectSound *ds;
IDirectSoundBuffer *dsb1, *dsb2;

int x = 20, y = 20;
int vx = 5, vy = 3;

void MoveBall(HWND hwnd, BOOL bMove)
{
    BOOL bBounce = FALSE;
    RECT rectSrc, rectDest;
    int ox, oy, nx, ny;

    GetClientRect(hwnd, &rectDest);
    ClientToScreen(hwnd, (POINT *)&rectDest.left);
    ClientToScreen(hwnd, (POINT *)&rectDest.right);
    if (bMove)
    {
        ox = rectDest.left +
            MulDiv(rectDest.right - rectDest.left - 32, x, 500);
        oy = rectDest.top +
            MulDiv(rectDest.bottom - rectDest.top - 32, y, 500);
        x += vx;
        y += vy;
        if (x < 0) { x = 0; vx = -vx; bBounce = TRUE; }
    }
}
```

Listing 48.1:
Einfache DirectX-
Anwendung

```

        if (x >= 500) { x = 1000 - x; vx = -vx; bBounce = TRUE; }
        if (y < 0) { y = -y; vy = -vy; bBounce = TRUE; }
        if (y >= 500) { y = 1000 - y; vy = -vy; bBounce = TRUE; }
        if (bBounce)
        {
            dsbl->SetCurrentPosition(0);
            dsbl->Play(0, 0, 0);
        }
    }
    nx = rectDest.left +
        MulDiv(rectDest.right - rectDest.left - 32, x, 500);
    ny = rectDest.top +
        MulDiv(rectDest.bottom - rectDest.top - 32, y, 500);

    rectSrc.left = rectSrc.top = 0;
    rectSrc.right = rectSrc.bottom = 32;

    if (bMove)
    {
        rectDest.left = rectDest.top = 0;
        rectDest.right = rectDest.bottom = 32;
        dds2->Blt(&rectDest, dds3, &rectSrc, DDBLT_WAIT, NULL);

        if (abs(nx - ox) < 32 && abs(ny - oy) < 32)
        {
            if (nx < ox)
            {
                rectSrc.left = ox - nx;
                rectSrc.right = 32;
                rectDest.left = 0;
                rectDest.right = 32 - rectSrc.left;
            }
            else
            {
                rectDest.left = nx - ox;
                rectDest.right = 32;
                rectSrc.left = 0;
                rectSrc.right = 32 - rectDest.left;
            }
            if (ny < oy)
            {
                rectSrc.top = oy - ny;
                rectSrc.bottom = 32;
                rectDest.top = 0;
                rectDest.bottom = 32 - rectSrc.top;
            }
            else
            {
                rectDest.top = ny - oy;
                rectDest.bottom = 32;
                rectSrc.top = 0;
                rectSrc.bottom = 32 - rectDest.top;
            }
            dds2->Blt(&rectDest, dds1, &rectSrc, DDBLT_WAIT,
                NULL);
        }

        rectSrc.left = rectSrc.top = 0;
        rectSrc.right = rectSrc.bottom = 32;
        rectDest.left = ox;
        rectDest.top = oy;
        rectDest.right = rectDest.left + 32;
        rectDest.bottom = rectDest.top + 32;
        dds0->Blt(&rectDest, dds2, &rectSrc, DDBLT_WAIT, NULL);
    }
}

```

```

    rectDest.left = nx;
    rectDest.top = ny;
    rectDest.right = rectDest.left + 32;
    rectDest.bottom = rectDest.top + 32;
    dds0->Blt(&rectDest, dds1, &rectSrc, DBLT_WAIT, NULL);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg,
                          WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;

    switch(uMsg)
    {
        case WM_PAINT:
            hDC = BeginPaint(hwnd, &paintStruct);
            if (hDC != NULL)
            {
                MoveBall(hwnd, FALSE);
                EndPaint(hwnd, &paintStruct);
            }
            break;
        case WM_TIMER:
            MoveBall(hwnd, TRUE);
            break;
        case WM_KEYDOWN:
            switch (wParam)
            {
                case VK_LEFT: vx--; break;
                case VK_UP: vy--; break;
                case VK_RIGHT: vx++; break;
                case VK_DOWN: vy++; break;
                case VK_ESCAPE: PostMessage(hwnd, WM_CLOSE, 0, 0);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR d3, int nCmdShow)
{
    MSG msg;
    HWND hwnd;
    WNDCLASS wndClass;
    DDSURFACEDESC ddsd;
    DSBUFFERDESC dsbd;
    HDC hddDC;
    RECT rect;
    HRSRC hrsrc;
    HGLOBAL hrData;
    DWORD *pRData;
    LPBYTE pMem1, pMem2;
    DWORD dwSize1, dwSize2;

    if (hPrevInstance == NULL)
    {
        memset(&wndClass, 0, sizeof(wndClass));
        wndClass.style = CS_HREDRAW | CS_VREDRAW;
    }

```

```

        wndClass.lpfWndProc = WndProc;
        wndClass.hInstance = hInstance;
        wndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wndClass.lpszClassName = "BOUNCE";
        if (!RegisterClass(&wndClass)) return FALSE;
    }
    hwnd = CreateWindow("BOUNCE", "BOUNCE",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
        NULL, NULL, hInstance, NULL);

    DirectDrawCreate(NULL, &dd, NULL);
    dd->SetCooperativeLevel(hwnd,
        DDSCL_NORMAL |
        DDSCL_NOWINDOWCHANGES);
    memset(&ddsd, 0, sizeof(DDSURFDESC));
    ddsd.dwSize = sizeof(DDSURFDESC);
    ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;
    ddsd.dwFlags = DDSCL_CAPS;
    dd->CreateSurface(&ddsd, &dds0, NULL);

    dd->CreateClipper(0, &ddc, NULL);
    dds0->SetClipper(ddc);
    ddc->SetHWND(0, hwnd);

    ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
    ddsd.dwHeight = 32;
    ddsd.dwWidth = 32;
    ddsd.dwFlags = DDSCL_CAPS | DDSCL_HEIGHT | DDSCL_WIDTH;
    dd->CreateSurface(&ddsd, &dds1, NULL);
    dd->CreateSurface(&ddsd, &dds2, NULL);
    dd->CreateSurface(&ddsd, &dds3, NULL);

    dds1->GetDC(&hddDC);
    SaveDC(hddDC);
    rect.left = rect.top = 0;
    rect.right = rect.bottom = 32;
    FillRect(hddDC, &rect, (HBRUSH)(COLOR_WINDOW + 1));
    SelectObject(hddDC, GetStockObject(BLACK_BRUSH));
    SelectObject(hddDC, GetStockObject(BLACK_PEN));
    Ellipse(hddDC, 0, 0, 32, 32);
    RestoreDC(hddDC, -1);
    dds1->ReleaseDC(hddDC);

    dds3->GetDC(&hddDC);
    FillRect(hddDC, &rect, (HBRUSH)(COLOR_WINDOW + 1));
    dds3->ReleaseDC(hddDC);

    DirectSoundCreate(NULL, &ds, NULL);
    ds->SetCooperativeLevel(hwnd, DSSCL_NORMAL);

    memset(&dsbd, 0, sizeof(DSBUFFERDESC));
    dsbd.dwSize = sizeof(DSBUFFERDESC);
    dsbd.dwFlags = DSBCAPS_STATIC | DSBCAPS_CTRLDEFAULT;

    hrsrc = FindResource(hInstance, "BOUNCE.WAV", "WAVE");
    hrData = LoadResource(hInstance, hrsrc);
    pRData = (DWORD *)LockResource(hrData);
    dsbd.dwBufferBytes = *(pRData + 10);
    dsbd.lpwfxFormat = (LPWAVEFORMATEX)(pRData + 5);

    ds->CreateSoundBuffer(&dsbd, &dsb1, NULL);

```

```

dsb1->Lock(0, dsbd.dwBufferBytes, &pMem1, &dwSize1,
           &pMem2, &dwSize2, 0);
memcpy(pMem1, (LPBYTE)(pRData + 11), dwSize1);
if (dwSize2 != 0)
    memcpy(pMem2, (LPBYTE)(pRData + 11) + dwSize1, dwSize2);
dsb1->Unlock(pMem1, dwSize1, pMem2, dwSize2);

hrsrc = FindResource(hInstance, "HUM.WAV", "WAVE");
hRData = LoadResource(hInstance, hrsrc);
pRData = (DWORD *)LockResource(hRData);
dsbd.dwBufferBytes = *(pRData + 10);
dsbd.lpwfxFormat = (LPWAVEFORMATEX)(pRData + 5);

ds->CreateSoundBuffer(&dsbd, &dsb2, NULL);

dsb2->Lock(0, dsbd.dwBufferBytes, &pMem1, &dwSize1,
           &pMem2, &dwSize2, 0);
memcpy(pMem1, (LPBYTE)(pRData + 11), dwSize1);
if (dwSize2 != 0)
    memcpy(pMem2, (LPBYTE)(pRData + 11) + dwSize1, dwSize2);
dsb2->Unlock(pMem1, dwSize1, pMem2, dwSize2);

dsb2->Play(0, 0, DSBPLAY_LOOPING);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

SetTimer(hwnd, 1, 100, NULL);
while (GetMessage(&msg, NULL, 0, 0))
    DispatchMessage(&msg);
KillTimer(hwnd, 1);
return msg.wParam;
}

```

Diese Anwendung verwendet die Dienste von DirectDraw und DirectSound. Der Ablauf beginnt in `WinMain`, wo zuerst eine ganz gewöhnliche Fensterklasse registriert und das Fenster der Anwendung erstellt wird.

WinMain

Als nächstes wird `DirectDrawCreate` aufgerufen, um eine `IDirectDraw`-Schnittstelle zur Display-Hardware zu erzeugen.

IDirectDraw-Schnittstelle erzeugen

Ausgabeflächen für die Animation

Diese Schnittstelle wird verwendet, um eine primäre Ausgabefläche zu erzeugen. Dann wird mit `IDirectDraw::CreateClipper` eine Cliquenliste erzeugt und an diese primäre Ausgabefläche angehängt. Der Handle des Fensters der Anwendung wird dieser Cliquenliste angehängt, um das richtige Funktionieren der Bit-Blit-Operationen in der primären Fläche zu sichern, wenn das Fenster teilweise bedeckt ist.

Als nächstes werden einige sekundäre Ausgabeflächen erzeugt. Diese sekundären Flächen werden in einem speziellen Algorithmus verwendet, um ein glattes Neuzeichnen des hüpfenden Balls zu erreichen. Zwei dieser Flächen sind initialisiert, eine mit dem Bild des Balls (erzeugt durch die `Ellipse`-Funktion der GDI), die andere mit der Hintergrundfarbe.

Schnittstelle zur Sound-Hardware

Nachdem die Initialisierung der Zeichenflächen beendet ist, wird eine Schnittstelle zur Sound-Hardware des Computers mit `DirectSoundCreate` erzeugt. Danach werden zwei sekundäre Soundpuffer mit `IDirectSound::CreateBuffer` erzeugt. Der eine wird verwendet, um den Audio-Stream des Hintergrundgeräusches zu speichern; der andere speichert den Stream für das Hüpfgeräusch. Durch Aufrufen von `IDirectSoundBuffer::Play` mit dem Parameter `DSBPLAY_LOOPING` wird das Hintergrundgeräusch kontinuierlich abgespielt.

Timer

Bevor wir in die Nachrichtenschleife der Anwendung gelangen, wird `SetTimer` aufgerufen, um einen Timer auf 100 Millisekunden zu setzen. Dieser Timer wird verwendet, um periodisch die Position des hüpfenden Balls zu aktualisieren und die Ausgabe neu zu zeichnen.

Wenn ein `WM_TIMER`-Ereignis empfangen wird, ruft die Fensterfunktion der Anwendung die Funktion `MoveBall` auf. Diese Funktion berechnet die neue Position des Balls aufgrund der globalen Geschwindigkeitswerte `vx` und `vy`. Die Position des Balls wird als Integer im Bereich 0 bis 500 ausgedrückt; diese werden dann in Bildschirmkoordinaten mit der `MulDiv`-Funktion umgesetzt.

Die Animation

Das Neuzeichnen des Balls ist ein komplexer Vorgang, der drei sekundäre Puffer benötigt. Weshalb ist diese Komplexität nötig?

Offensichtlich müssen wir zwei Dinge tun, wenn wir die Position des Balls auf dem Bildschirm aktualisieren wollen: den Ball an der alten Position löschen und ihn an der neuen zeichnen. Da der Ball sich sehr schnell bewegt, besteht die Möglichkeit, daß seine alte und neue Position zusammenfallen. Den Ball einfach an seiner alten Position zu löschen, würde unerwünschtes Flickern verursachen, da die vom Ball bedeckten Pixel sowohl an der alten als auch der neuen Position kurzzeitig weiß werden.

Um dies zu vermeiden, erzeugen wir, statt einfach den Ball an seiner alten Position zu löschen, einen sekundären Puffer, der die alte Position des Balls darstellt. In diesen Puffer zeichnen wir Teile des Balls, die sichtbar werden, wenn der Ball in der neuen Position ist. Wir kopieren diesen Puffer an die alte Position des Balls; wir kopieren auch den gesamten Ball an seine neue Position. Abbildung 48.4 zeigt eine schematische Darstellung dieses Mehrschritt-Vorgangs. Wenn Ihre Anwen-

nung nur kleine Ausschnitte ihres Fensters aktualisiert, kann diese Prozedur bessere Performance aufweisen als Aktualisieren des Puffers im Hintergrund und Einsatz der Funktion `IDirectDrawBuffer::Flip`.

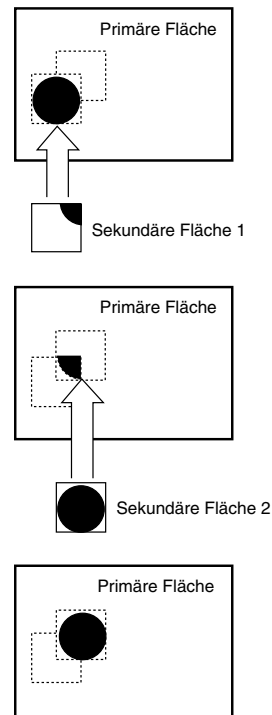


Abbildung 48.4:
Glatte Bild-
schirmaktuali-
sierung mit
mehreren Puf-
fern

Die Anwendung antwortet auch auf Tastaturereignisse. Die Pfeiltasten können verwendet werden, um den Ball in waage- oder senkrechter Richtung zu beschleunigen. Das Drücken der `[Esc]`-Taste beendet die Anwendung.

*Tastaturunter-
stützung*

Die beiden Sounds, die die Anwendung verwendet, befinden sich in den Waveform-Dateien *hum.wav* und *bounce.wav*. Diese Dateien werden in der kleinen Ressourcendatei der Anwendung aus Listing 48.2 referenziert.

```
BOUNCE.WAV      WAVE      DISCARDABLE      "bounce.wav"
HUM.WAV         WAVE      DISCARDABLE      "hum.wav"
```

Listing 48.2:
Ressourcendatei
der einfachen
DirectX-Anwen-
dung.

Beachten Sie, daß der Einfachheit der Anwendung halber einige implizite Annahmen zum Inhalt und der Struktur dieser Dateien gemacht werden. Falls Sie eine andere Datei verwenden möchten, sind Sie gut beraten, einige der DirectX-Beispiele anzusehen, um eine Idee zu bekommen, wie eine allgemeine Parsing-Fähigkeit erreicht wird.

48.3 Zusammenfassung

Microsoft hat die Notwendigkeit für eine leistungsfähige Schnittstelle für Spiele unter Windows als Echtzeitanwendungen erkannt und das SDK DirectX entwickelt. Dieses SDK bietet eine Familie von APIs, kollektiv als DirectX-APIs bezeichnet, zum Zugriff auf die Video-, Sound-, Kommunikations- und Joystick-Hardware.

Die Basis vieler Elemente in den APIs von DirectX ist Microsofts Common-Object-Modell. Einige der Schnittstellen der DirectX-APIs sind von der COM-IUnknown-Schnittstelle abgeleitet.

Die Video-Hardware wird durch DirectDraw-Objekte dargestellt, der Zugriff erfolgt über die IDirectDraw-Schnittstelle. Durch ein DirectDraw-Objekt können Anwendungen Zeichenflächen, Paletten und Cliplisten erzeugen. Die Schnittstelle zu Zeichenflächen, IDirectDrawSurface, bietet leistungsfähige Bit-Blt-Fähigkeiten. Die Schnittstelle IDirectDraw kann verwendet werden, um die Ausgabehardware in verschiedenen Arten zu manipulieren; dies schließt ausschließlichen Zugriff auf die Ausgabehardware für Spiele mit Vollbild und Animation ein. Eine spezielle Anwendung von IDirectDraw ist die Möglichkeit des Zugriffs auf die sekundäre Videohardware, die von Windows selbst nicht erkannt wird.

Basierend auf den Diensten von DirectDraw bietet die Direct3D-API eine Reihe von fortgeschrittenen Schnittstellen für dreidimensionale Grafik und Animation. Diese API ist in zwei Komponenten unterteilt. Der Direct3D Immediate Mode bietet Low-level-Zugriff auf die Hardware des Videobeschleunigers; Direct3D Retained Mode bietet high-level dreidimensionale Rendering-Konzepte.

Der Zugang zur Sound-Hardware wird durch die Schnittstelle IDirectSound erreicht. Anwendungen verwenden die Sound-Hardware typischerweise durch Erzeugen einer Reihe von sekundären Sound-Puffern; die Inhalte dieser Puffer werden in den primären Puffer gemixt und von DirectSound abgespielt. Diese Fähigkeit zum Mixen der Waves sowie die Möglichkeit, im Schleifenmodus abzuspielen, sind die spielebezogenen Haupteigenschaften von DirectSound.

DirectPlay bietet Kommunikationsdienste für Anwendungen mit mehreren Spielern. Solche Anwendungen können über Netzwerke, Modems oder Online-Dienste kommunizieren. DirectPlay bietet eine einfache Sende-/Empfangsmöglichkeit für Programme, die an Spiele-Sessions teilnehmen.

Die API DirectInput bietet Dienste für Joystick, Tastatur und Mauseingabe. Joystick-Unterstützung wird über Funktionen wie joyGetPosEx er-

reicht, wobei die Position und andere Informationen über Kontrollgeräte mit mehreren Schaltern und Achsen erhalten werden. Die Unterstützung für Tastatur und Maus wird durch COM-Objekte erzielt.

Einstellen der DirectX-API ist eine komplexe Aufgabe. Eine Schnittstelle mit einer einzigen Funktion, die ein Setup von DirectX von den Standardverzeichnissen aus durchführt, wird in der Form der Funktion `DirectXSetup` angeboten.

Anhänge

Teil VIII

- 
- A. Erzeugen eigener
Anwendungsassistenten
 - B. Übersicht zu C/C++
 - C. Die Standard-Laufzeitbibliothek
 - D. Die Standard-C++-Bibliothek
 - E. Literatur
 - F. Zur CD-ROM

Erzeugen eigener Anwendungsassistenten



Eines der besten Feature des Visual-C++-Entwicklungssystems war schon immer die Möglichkeit, stark angepaßte Anwendungsgerüste zu erstellen. Das Werkzeug, das diese Fähigkeit umsetzt, ist der Visual C++-Anwendungsassistent. Durch eine Reihe von Dialogen leitet der Anwendungsassistent den Anwender durch den Vorgang des Einstellens der Anfangsparameter des Projekts. Er verwendet dann diese vom Anwender eingegebenen Parameter, um eine Reihe von Vorlagen, die die Grundlage des neuen Projekts bilden, anzupassen.

Wenn der Anwendungsassistent ein Codegenerator ist, dann muß die Fähigkeit, angepaßte Anwendungsassistenten zu erzeugen, als Codegenerator-Generator bezeichnet werden, oder? Um die potentiellen Quellen der Verwirrung zu meiden, die durch dieses, zugegeben uninspirierte, Wortspiel veranschaulicht werden, sollten wir uns im voraus auf einige Termini einigen.

Für den Zweck dieses Kapitels bezeichnet der Begriff ENTWICKLER Sie, den Autor eines angepaßten Anwendungsassistenten. Der ANWENDUNGSASSISTENTENANWENDER ist der Programmierer, der diesen angepaßten Anwendungsassistenten zum Erzeugen eines Anwendungsgerüsts einsetzt. Der Begriff ENDANWENDER bezieht sich auf Anwender der Anwendung, die vom Anwendungsassistentenanwender erstellt wurde.

A.1 Wie funktioniert der Anwendungsassistent?

Bevor wir darangehen können, einen angepaßten Anwendungsassistenten zu erzeugen, ist es angebracht, die Funktionsweise des Anwendungsassistenten genau zu verstehen.

Was geschieht, wenn Sie den Anwendungsassistenten aufrufen? Das erste, was Sie sehen, ist eine Reihe von Dialogen, die der Anwendungsassistent präsentiert. Durch diese Dialoge können Sie zahlreiche Parameter des Projekts bestimmen.

Diese Parameter werden intern vom Anwendungsassistenten als eine Reihe von Substitutionsmakros gespeichert. Zu diesen kommen weitere intern definierte Makros hinzu.

Wenn der Anwendungsassistent mit dem Generieren eines neuen Projekts beginnt, bildet er Projektdateien aus einem Satz von Vorlagen. Die Vorlagen werden verarbeitet und alle Makroreferenzen darin mit den intern definierten oder vom Anwender angegebenen Werten expandiert.

Abbildung A.1 zeigt schematisiert, wie ein Anwendungsassistent ein Projekt erzeugt.

Hier wird die Verallgemeinerungsmethode des Anwendungsassistenten offensichtlich. Wenn Sie nun Ihre eigenen Dialoge, Vorlagen und Makros anstatt (oder zusätzlich zu) denen des Anwendungsassistenten einbringen könnten, würde eine an Ihre Anforderungen angepaßte Anwendung entstehen. Das ist genau die Art, wie angepaßte Anwendungsassistenten aufgebaut sind.

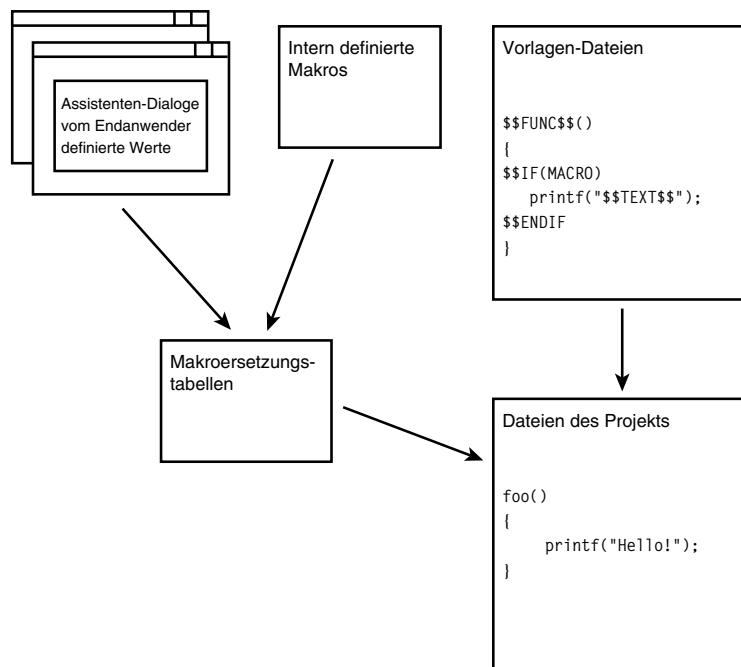


Abbildung A.1:
Wie der Anwendungsassistent
ein neues
Projekt erzeugt

A.2 Ein Beispiel: der HelloWizard

Ich weiß, ich bin ein Langweiler. Könnte ich nicht etwas Unterhaltssameres finden als ein weiteres Hello, World-Projekt? Sie haben aber dieses Buch nicht seinem Unterhaltungswert zuliebe gekauft, oder? Somit kann ich vielleicht für die Verwendung von Beispielen, die nicht soviel Phantasie zeigen, entschuldigt werden, wenn sie Ihnen dafür helfen, sich auf das zu veranschaulichende Thema zu konzentrieren. Je einfacher ein Beispiel (sofern es nicht übermäßig vereinfacht ist), desto besser kommt es auf den Punkt, ohne Sie mit unnötigen und unwichtigen Details zu belästigen.

A.2.1 Übersicht zum HelloWizard

Unser Anwendungsassistentenprojekt ist ein wirklich sehr einfaches. Als Grundlage dient das Programm aus Listing A.1.

Listing A.1:
Ein einfaches
Hello-World-
Programm in
Windows

```
#include <afxwin.h>

int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
    MessageBox(NULL, "Hello, World!", "HELLO", MB_OK);
    return 0;
}
```

Der neue Anwendungsassistent, HelloWizard, wird dem Anwendungsassistenten-Anwender helfen, eine Anwendung zu erzeugen, die einen beliebigen Text ausgibt, nicht nur diesen langweiligen String »Hello, World!«. Wir möchten auch, daß diese revolutionäre Anwendung die richtigen Quellkommentare enthält, beispielsweise den Namen der Datei.

A.2.2 Erzeugen eines angepaßten Anwendungsassistenten

Angepaßte Anwendungsassistenten werden mit dem Anwendungsassistenten erzeugt.

1. Als ersten Schritt zum Erzeugen des neuen Projekts rufen Sie den Befehl DATEI/NEU auf.
2. Im Dialog NEU auf der Registerseite *Projekte*, geben Sie den Namen des neuen Projekts ein (zum Beispiel, HELLO) und wählen BENUTZERDEFINIERTER ANWENDUNGS-ASSISTENTEN als Projekttyp aus. (Geben Sie auch das richtige Verzeichnis an, in dem Sie das Projekt haben möchten.) Klicken Sie auf OK.

Es erscheint der erste von zwei Assistenten-Dialogen (der einzige, den wir für HelloWizard verwenden), wie Abbildung A.2 zeigt.

3. Zuerst müssen Sie den Startpunkt wählen, von dem aus Ihr neuer benutzerdefinierter Anwendungsassistent erstellt wird.
 - Sie können den neuen benutzerdefinierten Anwendungsassistenten auf einem existierenden Projekt aufbauen. Mit Hilfe des neuen benutzerdefinierten Anwendungsassistenten wird der Anwendungsassistenten-Anwender dann in der Lage sein, Projekte zu erstellen, die eine identische Kopie des Originals sind – bis auf den Namen des Projekts, der mit einem Namen seiner Wahl über das ganze Projekt ersetzt wird.
 - Sie können einen benutzerdefinierten Anwendungsassistenten erzeugen, der das Verhalten des Standard-MFC-Anwendungsassistenten nachahmt. Das ist sehr nützlich, wenn es Ihr Ziel ist, den Standard-Anwendungsassistenten mit kundenspezifischen Erweiterungen auszustatten.

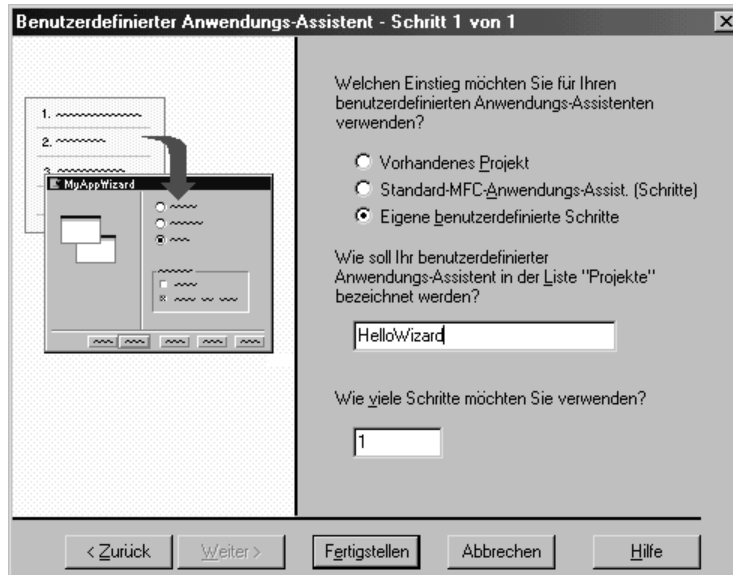


Abbildung A.2:
Erzeugen eines
benutzerdefinierten Anwen-
dungsassistentenprojekts mit
dem Anwen-
dungsassistenten

- Schließlich können Sie einen neuen benutzerdefinierten Anwendungsassistenten erzeugen, indem Sie Ihre eigenen Schritte einbauen und so das Default-Verhalten des Anwendungsassistenten umgehen.
4. Die ersten beiden Schritte sind am besten für Anwendungen auf der Basis des MFC-Anwendungsgerüsts geeignet. Da unser HelloWorld-Programm keine solche Anwendung ist, sind wir auf die dritte Wahlmöglichkeit beschränkt, die Option EIGENE BENUTZERDEFINIERTESCHRITTE.

Wenn Sie auf diese Option klicken, wird das Eingabefeld im unteren Teil des Dialogs aktiviert. In diesem Feld können Sie die Anzahl der benutzerdefinierten Schritte angeben (Anzahl der Assistenten-Dialoge), die Ihr Assistent erfordern wird. HelloWizard wird nur einen solchen Schritt erfordern, somit können Sie dieses Feld bei seiner Default-Einstellung belassen.

Eine weitere Änderung, die beim Anklicken der Option EIGENE BENUTZERDEFINIERTESCHRITTE eingetreten ist, ist der Titel, der nun »Schritt 1 von 1« lautet, und der Schalter WEITER, der deaktiviert wurde. Der zweite Schritt ist für benutzerdefinierte Anwendungsassistenten, die auf angegebenen Schritten basieren, nicht relevant.



Ebenfalls in diesem Dialog befindet sich ein weiteres Feld, in dem Sie den Namen angeben können, unter dem der benutzerdefinierte Anwendungsassistent erscheinen wird. Der Standardname ist <Projektname> AppWizard (zum Beispiel HALLO AppWizard).

5. Ändern Sie diesen Namen auf den eleganteren HelloWizard.

Dies ist eine rein kosmetische Änderung; dieser Name wird zu keinem anderen Zweck verwendet als zur Identifizierung des benutzerdefinierten Anwendungsassistenten, wenn sein Anwender eine neue Arbeitsfläche für sein Projekt erstellt.

6. Wenn Sie den Dialog schließen, indem Sie den Schalter FERTIGSTELLEN betätigen, gibt der Anwendungsassistent eine Zusammenfassung des neuen benutzerdefinierten Anwendungsassistentenprojekts im Dialog NEUE INFORMATIONEN ZUM NEUEN PROJEKT aus. Klicken Sie den Schalter OK, und das neue Projekt wird erstellt.

A.2.3 Vorlagen für benutzerdefinierte Anwendungsassistenten

Neue, von einem benutzerdefinierten Anwendungsassistenten erstellte Projekte werden mit Hilfe von Vorlagen definiert. Als der Anwendungsassistent das neue benutzerdefinierte Anwendungsassistentenprojekt erstellt, hat er zwei spezielle Vorlagendateien im Unterverzeichnis `Template` des Verzeichnisses des neuen benutzerdefinierten Anwendungsassistentenprojektes erzeugt:

■ ➔ `confirm.inf` und

■ ➔ `newproj.inf`

Confirm.inf

Die erste dieser Dateien, `confirm.inf`, dient der Beschreibung des Projekts.

newproj.inf

Die andere Vorlagendatei, `newproj.inf`, ist die vielleicht wichtigste Datei in Ihrem benutzerdefinierten Anwendungsassistentenprojekt. Diese Datei sagt dem Anwendungsassistenten, welche andere Dateien das Projektgerüst enthält.

Die Quelldatei

In der Tat, was sind denn für andere Dateien vorhanden? Wir haben einen benutzerdefinierten Anwendungsassistenten gewählt, der ganz aus selbstdefinierten Schritten besteht, somit sind sowohl `confirm.inf`

als auch *newproj.inf* leer erzeugt worden. Ihren Inhalt müssen wir liefern. Doch bevor wir dies tun, müssen wir einen Blick auf die einzige Vorlagen-Quelldatei unseres Projekts werfen, *HELLO.CPP*.

Diese Datei gibt es natürlich noch nicht, doch Sie können sie leicht erstellen, indem Sie die paar Zeilen aus Listing A.1 eintippen. Doch wie kann man dieses Programm in eine allgemeine Vorlage verwandeln, die als Startpunkt für viele Projekte dienen kann?

Die Antwort ist die Verwendung von Makros an verschiedenen Stellen in dieser Datei. Eigentlich benützen wir zwei solche Makros.

*Einsatz der
Makros*

- ➡ Das erste Makro wird automatisch vom Anwendungsassistenten geliefert und gibt den Namen der Anwendung an, die der Anwender gerade erstellt.
- ➡ Das zweite Makro werden wir später erstellen, und es wird den Text enthalten, den der Anwender des Anwendungsassistenten in die neue Anwendung einfügen will.

Anwendungsassistenten-Makros werden gekennzeichnet, indem ihren Namen zwei Dollarzeichen vorne und hinten angehängt werden. Zum Beispiel wird auf das Makro `ROOT`, das den Projektnamen definiert, in Vorlagendateien als `$$ROOT$$` Bezug genommen. Dasselbe gilt für selbstdefinierte Makros – in unserem Fall das Makro `$$HELLOTEXT$$`, das den Wert referenziert, den der Anwender des Anwendungsassistenten mit dem neuen Programm ausgeben will.

Listing A.2 zeigt *HELLO.CPP* in seiner Vorlagenversion. Sie können diese Datei mit dem integrierten Editor erzeugen und im Unterverzeichnis `Template` speichern.

```

////////////////////////////////////
// $$ROOT$$$.CPP

#include <afxwin.h>

int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
    MessageBox(NULL, "$$HELLOTEXT$$", "$$ROOT$$", MB_OK);
    return 0;
}

```

*Listing A.2:
Das Programm
Hello, World als
benutzerdefinier-
te Anwendungs-
assistenten-
vorlage*

Beachten Sie, daß die Makrosubstitution unabhängig davon stattfindet, wo ein Makro eingesetzt wird (innerhalb eines Kommentars, zwischen Anführungszeichen).



Ein benutzerdefinierter Anwendungsassistent beherrscht einerseits Textvorlagen wie unser *HELLO.CPP*, andererseits aber auch binäre Vorlagen (zum Beispiel eine Bitmap-Datei). Anders als Textvorlagen werden binäre Vorlagen als solche, also ohne Makrosubstitution, kopiert.

newproj.inf bearbeiten

Wie ich bereits erwähnte, ist der Zweck der Vorlage *newproj.inf*, alle Vorlagen aufzulisten, aus denen ein neues Projekt besteht. Entsprechend müssen wir die Vorlage *newproj.inf* von HelloWizard ändern und *HELLO.CPP* einfügen. Diese neue Version von *newproj.inf* wird in Listing A.3 gezeigt.

```

Listing A.3:  $$$ newproj.inf = Vorlage für eine Liste von Vorlagedateien
Die Vorlage  $$$/ Das Format ist 'Quellressourcenname' \t 'Zielfeldname'
newproj.inf  $$$/ Dem Namen der Quellressource kann eine beliebige Kombination von
für          $$$/ '=' , '-' , '!' , '?' , ':' , '#' und/oder '*' vorangestellt sein.
HelloWizard $$$/ '=' => Die Ressource ist eine Binärdatei.
            $$$/ '-' => Die Datei soll dem Projekt nicht hinzugefügt werden
            $$$/      (standardmäßig werden alle Dateien hinzugefügt).
            $$$/ '!' => Die Datei soll von der Programmerstellung ausgenommen werden.
            $$$/ '?' => Die Datei soll als Hilfedatei behandelt werden.
            $$$/ ':' => Die Datei soll als Ressource behandelt werden.
            $$$/ '#' => Die Datei soll als Vorlage behandelt werden ('!' vorausgesetzt).
            $$$/ '*' => Beim Laden sollen die Ressourcen des benutzerdefinierten
            $$$/      Anwendungs-Assistenten nicht berücksichtigt werden.
            $$$/ Der Name beginnt mit / => es wird ein neues Unterverzeichnis angelegt.

HELLO.CPP  $$$ROOT$$$CPP

```

Wie der etwas kryptische (vom Anwendungsassistenten generierte) Header von *newproj.inf* andeutet, kann der Name jeder in dieser Liste geführten Datei eine Kombination aus drei Flags vorangestellt bekommen.

Die Bedeutung des Flags * ist etwas verschleiert; es gibt an, daß die Ressourcen des benutzerdefinierten Anwendungsassistenten nicht nach einer Ressource mit diesem Namen durchsucht werden sollen, sondern statt dessen die Default-Ressource eingesetzt werden soll. Das ist sinnvoll, wenn Ihr benutzerdefinierter Anwendungsassistent eine Vorlage enthält, die eine Default-Vorlage mit gleichem Namen ersetzt und Sie, je nach Bedingung, die eine oder die andere verwenden möchten.

In unserem Fall verwenden wir keines der Flags, da wir möchten, daß die aus der Vorlage *HELLO.CPP* erzeugte Datei automatisch dem Projekt des Anwenders des Anwendungsassistenten hinzugefügt wird. Der Name der Vorlage wird von einem Tabulator gefolgt (jajwohl, es muß ein einzelnes Tab-Zeichen sein; andere Whitespace-Zeichen sind nicht erlaubt), wonach der Name folgt, den diese Datei im Verzeichnis des Projekts des Anwendungsassistentenanwenders haben wird. Si-

cherlich können auch hier Makros eingesetzt werden; wir verwendeten das Makro `$$ROOT$$`, um den Dateinamen anzugeben, der mit dem vom Anwender angegebenen Projektnamen gleich ist.

confirm.inf bearbeiten

Das Ändern von *confirm.inf* dient bloß kosmetischen Zwecken. Trotzdem gibt es mir die Gelegenheit, einige Anweisungen für Vorlagen des Anwendungsassistenten zu zeigen (vor allem die Anweisungen `$$IF` und `$$ENDIF`), die sonst für unseren einfachen HelloWizard nicht benötigt würden. Listing A.4 zeigt diese neue Version von *confirm.inf*.

Der Anwendungsassistent erstellt nun ein neues Projekt mit den folgenden Einstellungen

Projektname: `$$ROOT$$\MAK`

Begrüßungstext: `$$HELLOTEXT$$`

`$$IF(TARGET_INTEL)`

Die erstellte Konsolenanwendung kann in den DOS-Fenster von Windows 95 und Windows NT ausgeführt werden.

`$$ENDIF`

Wenn ein Makro in einer Anwendungsassistentenvorlagenanweisung wie `$$IF` referenziert wird, ist es nicht nötig, die Dollarzeichen an seinen Namen vorne und hinten anzuhängen.

Listing A.4:
Die Vorlage confirm.inf für HelloWizard

A.2.4 Benutzerdefinierte Anwendungsassistentenressourcen

Wir haben *newproj.inf* und *confirm.inf* geändert und eine neue Vorlage, *HELLO.CPP*, hinzugefügt. Doch wie finden diese Dateien ihren Weg in den endgültigen benutzerdefinierten Anwendungsassistenten? Welche Form nimmt ein benutzerdefinierter Anwendungsassistent an?

Das Ziel des benutzerdefinierten Anwendungsassistentenprojekts ist eine einzelne Datei mit der Erweiterung `.awx` (in unserem Fall *HELLO.awx*). Trotz der eigenartigen Erweiterung ist diese Datei tatsächlich eine DLL, die von Visual Studio aufgerufen wird, wenn ein neues Projekt dieser Art erzeugt wird. Woher weiß Visual Studio, daß diese neue Bibliothek da ist? Der benutzerdefinierte Anwendungsassistent muß, wenn er fertig ist, in ein spezielles Verzeichnis gelegt werden (normalerweise *Programme\Microsoft Visual Studio\Common\MSDev98\Template*), wo Visual Studio ihn finden kann.

Wie ich sagte, ist das benutzerdefinierte Anwendungsassistentenprojekt tatsächlich ein DLL-Projekt. Als solches hat es auch eine Ressourcendatei. Wenn Sie die Ressourcen-Ansicht des benutzerdefinierten Anwendungsassistentenprojekts öffnen, werden Sie einen Abschnitt

namens "TEMPLATE" bemerken. In diesem Abschnitt sind zwei Dateien angegeben, die Ihnen bereits vertraut sind: *confirm.inf* und *newproj.inf*.

Warum haben wir nicht einfach die Ressourcen-Ansicht beim Editieren dieser Dateien verwendet? Das Problem ist folgendes: Da diese Dateien kundenspezifische Ressourcendateien sind, stellt Visual Studio keine Bedingungen hinsichtlich ihres Inhalts. Falls Sie versuchen, sie in der Ressourcen-Ansicht zu öffnen, werden sie mit dem binären Editor geöffnet. Um sie als Textdateien zu öffnen, müssen Sie den Befehl DATEI/ÖFFNEN von Visual Studio verwenden.

Die Datei, die offensichtlich in dem Abschnitt "TEMPLATE" fehlt, ist unsere neu erstellte Datei *HELLO.CPP*.

1. Um sie hinzuzufügen, drücken Sie in der Ressourcen-Ansicht die rechte Maustaste über dem Ordner »TEMPLATE« und wählen den Befehl IMPORTIEREN aus dem Popup-Menü.
2. Wählen Sie *HELLO.CPP* im Verzeichnis *Template*, und klicken Sie auf IMPORTIEREN.
3. In dem Ressourcentyp-Dialog, der geöffnet wird, wählen Sie »TEMPLATE« als Ressourcentyp und klicken OK.

Visual Studio öffnet die Datei *HELLO.CPP* für binäres Editieren; Sie können das Fenster gleich wieder schließen. Es ist aber nötig, den Bezeichner dieser neu hinzugefügten Ressource zu ändern.

4. Die Ressource wurde mit dem symbolischen Bezeichner `IDR_TEMPLATE1` hinzugefügt; ändern Sie diesen auf den Textbezeichner "HELLO.CPP". Dazu klicken Sie einfach auf den Eintrag `IDR_TEMPLATE1` und geben an Stelle der früheren ID, `IDR_TEMPLATE1`, in die Ressourceneigenschaften "HELLO.CPP" ein – einschließlich der Anführungszeichen.



Dadurch wird etwas über die Datei NEWPROJ.INF aufgedeckt: Als wir eine Referenz auf HELLO.CPP hinzugefügt haben, haben wir uns auf den symbolischen Namen dieser Ressource in der Ressourcendatei unseres Projekts bezogen, nicht auf den eigentlichen Dateinamen. Sicherlich ist es sinnvoll, die beiden Namen identisch zu halten, sofern möglich.

A.2.5 Benutzerdefinierte Anwendungsassistenten-dialoge

Nachdem wir unsere Vorlagen zusammengestellt haben, ist zweifellos die bohrende Frage, woher die anwenderdefinierten Makros ihre Werte nehmen? Es ist leicht einzusehen, wie der Anwendungsassistent durch irgendeine interne Zauberei die Werte für die Default-Makros wie `$$ROOT$$` oder `$$TARGET_INTEL$$` weiterreicht. Doch woher bekommt unser kleines Makro `$$HELLOTEXT$$` seine Werte?

Um diese Frage zu beantworten, wenden wir unsere Aufmerksamkeit den Dialogen unseres benutzerdefinierten Anwendungsassistenten zu. Wenn Sie in der Ressourcen-Ansicht den Abschnitt `Dialog` öffnen, werden Sie dort genau einen Dialog vorfinden: `IDD_CUSTOM1`. Das ist so, weil wir beim Erzeugen unseres benutzerdefinierten Anwendungsassistentenprojekts angegeben haben, daß unser benutzerdefinierter Anwendungsassistent aus nur einem Schritt besteht.

Öffnen Sie diesen Dialog durch Doppelklicken.

1. In dem Dialog steht ein einziges Feld: eine »TODO«-Marke. Diese muß ersetzt werden. HelloWizard fordert den Anwender zur Eingabe eines Textparameters auf. Folglich sollte die neue Version dieses Dialogs eine statische Marke und ein Eingabfeld, symbolisch durch `IDC_HELLOTEXT` bezeichnet, enthalten. Dieser neue, zugegeben etwas spartanisch aussehende, Dialog wird in Abbildung A.3 gezeigt.

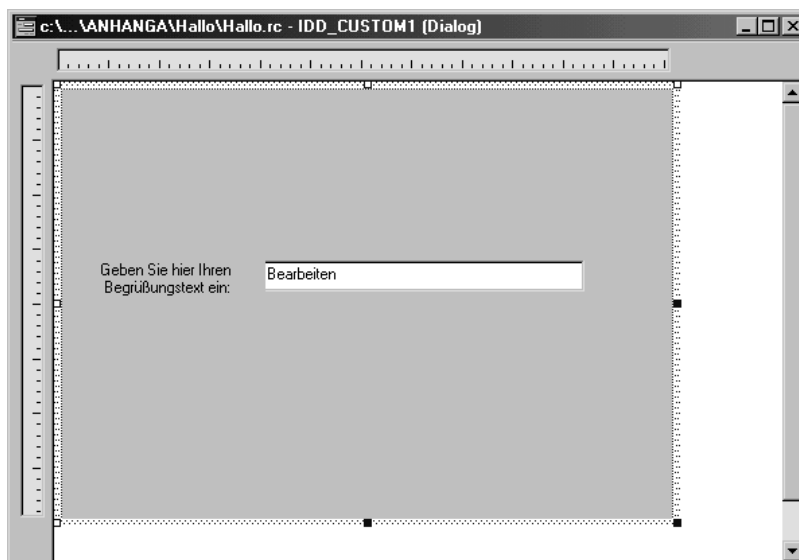


Abbildung A.3:
Der einzige Dialog von HelloWizard

Die Werte eines benutzerdefinierten Anwendungsassistentendialogs werden Elementvariablen durch denselben Dialog-Datenaustauschvorgang wie auch in anderen MFC-Dialogen überwiesen.

2. Um eine Elementvariable mit dem neuen Feld `IDC_HELLOTEXT` zu verbinden, rufen Sie den Klassen-Assistenten auf. (Drücken der `[Strg]`-Taste und Doppelklick auf das Steuerelement.)
3. Der Name der neuen Variablen sollte `m_sHelloText` sein. Sie hat die Kategorie Wert und den Typ `CString`.
4. Nachdem Sie diese Variable hinzugefügt haben, können Sie den Klassen-Assistenten-Dialog schließen und auch alle Ressourcen.

A.2.6 Das Makro-Wörterbuch

Die neue Elementvariable `m_sHelloText` wurde der Dialogklasse `CCustomDlg` entsprechend dem Dialog `IDD_CUSTOM1` hinzugefügt. Durch diese Klasse wird auch der Wert, der `m_sHelloText` zugewiesen wurde, in das Makro-Wörterbuch des benutzerdefinierten Anwendungsassistenten übertragen.

Werfen Sie mal einen Blick auf die Klasse `CHalloAppWiz`. Diese Klasse wird von `CCustomAppWiz` abgeleitet. `CCustomAppWiz` hat eine Elementvariable `m_Dictionary`, die vom Typ `CMapStringToString` ist und Abbildungen von Makronamen auf Makrowerte enthält. Dieser Sammlung von Makronamen und Makrowert-Paaren müssen wir unser neues Makro hinzufügen.

Am Ende der Datei *HalloAw.cpp* ist ein einzelnes Objekt vom Typ `CHalloAppWiz` deklariert. Durch dieses globale Objekt haben wir Zugang auf das Makro-Wörterbuch an jeder Stelle in unserem benutzerdefinierten Anwendungsassistenten.

Getan wird es in der Elementfunktion `OnDismiss` der Klasse `CCustomDlg`. Diese Funktion wird immer dann aufgerufen, wenn der Anwender einen Anwendungsassistentendialog schließt. Wir müssen nur eine Zeile Code hinzufügen, die das Makro-Wörterbuch mit dem neuen Makrowert aktualisiert. Die geänderte Funktion `OnDismiss` wird in Listing A.5 gezeigt.

```

Listing A.5: // Diese Funktion wird immer dann aufgerufen, wenn der Benutzer
Geänderte   // bei diesem Schritt "Weiter", "Zurück" oder "Fertigstellen"
Version von // drückt.
CCustomDlg: // \< Nehmen Sie alle Gültigkeitsprüfungen und den
:           // Datenaustausch für das Dialogfeld in dieser Funktion vor.
OnDismiss   BOOL CCustomDlg::OnDismiss()
{
    if (!UpdateData(TRUE))
        return FALSE;
}

```

```
// ZU ERLEDIGEN: Variablen der Vorlage auf Grundlage der
// Dialogfelddaten festlegen.
Halloow.m_Dictionary["HELLOTEXT"] = m_sHelloText;

return TRUE; // FALSE zurückgeben, falls das Dialogfeld
// nicht geschlossen werden soll
}
```

Ob Sie es glauben oder nicht, das war's. Der neue HelloWizard ist einsatzbereit, sobald er kompiliert und im Verzeichnis MSDEV/TEMPLATE installiert ist. Sie müssen ihn dort nicht selbst installieren; dies wird automatisch als letzter Schritt im Erstellen des Projekts durchgeführt.

A.2.7 Testen des neuen benutzerdefinierten Anwendungsassistenten

Funktioniert unser neuer HelloWizard wie beabsichtigt? Nichts einfacher als das zu testen: Wir erstellen mit seiner Hilfe ein neues Projekt.

1. Wählen Sie NEU im DATEI-Menü, und wechseln Sie zur Seite PROJEKTE.

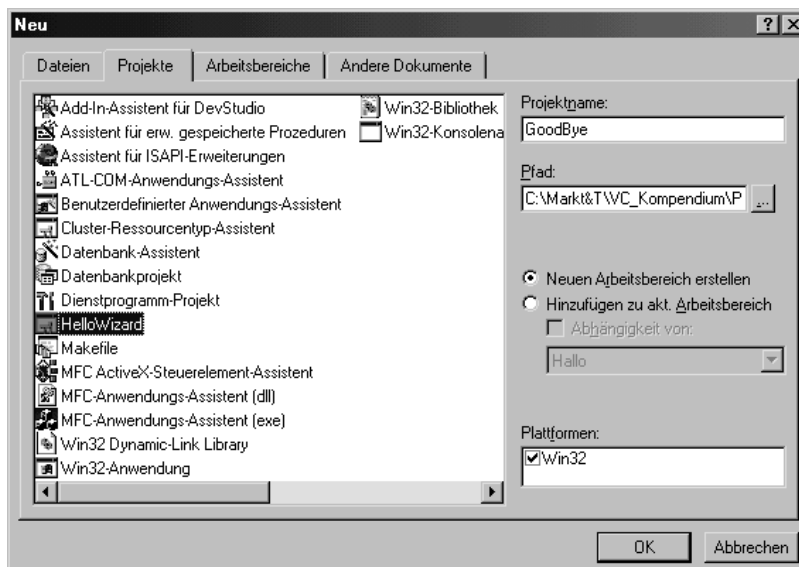
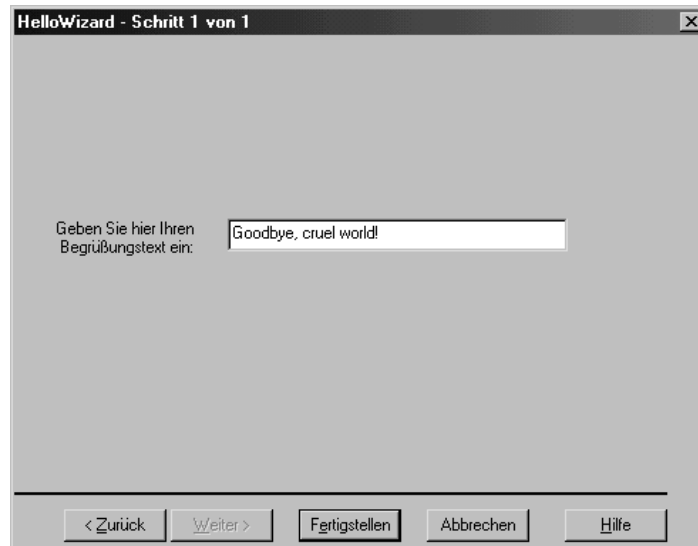


Abbildung A.4:
Auswahl des
neuen Assisten-
ten

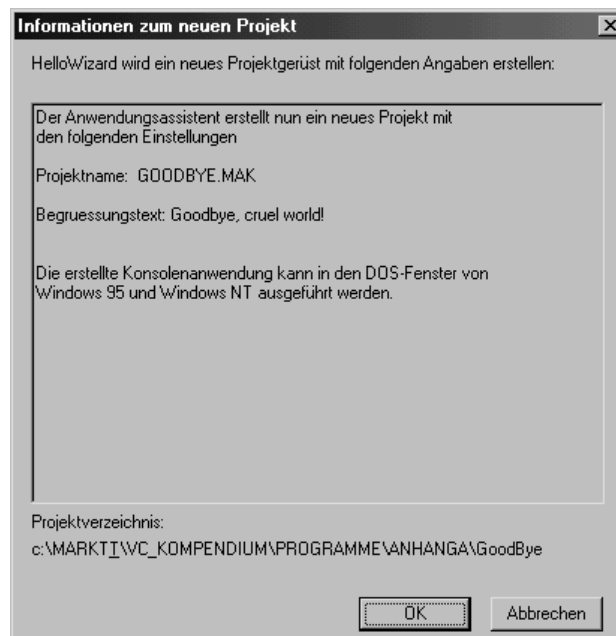
2. In der Liste der Projekttypen werden Sie jetzt auch Ihren HelloWizard finden. Wählen Sie Ihren Assistenten aus, und geben Sie dem Projekt einen Namen (beispielsweise GOODBYE).
3. Sobald Sie OK klicken, erscheint die erste (und im Fall von HelloWizard einzige) Seite des Anwendungsassistenten. Geben Sie den Text ein, den die neue Anwendung ausgeben soll (Abbildung A.5)

Abbildung A.5:
Erstellen einer
Anwendung mit
HelloWizard



4. Wenn Sie HelloWizard mit dem Schalter FERTIGSTELLEN schließen, erscheint der Dialog INFORMATIONEN ZUM NEUEN PROJEKT; darin erscheint der verarbeitete Inhalt von *confirm.inf* (Abbildung A.6).

Abbildung A.6:
Bestätigungs-
dialog von
HelloWizard



5. Wenn Sie den OK-Schalter klicken, wird das neue Projekt erstellt.

Werfen Sie einen Blick auf die neu erzeugte Quelldatei `GOODBYE.CPP` (gezeigt in Listing A.6). Wie Sie sehen, wurden alle Makro-Referenzen richtig ersetzt.

```
////////////////////////////////////// GOODBYE.CPP

#include <afxwin.h>

int WINAPI WinMain(HINSTANCE d1, HINSTANCE d2, LPSTR d3, int d4)
{
    MessageBox(NULL, "Goodbye, cruel world!", "GOODBYE", MB_OK);
    return 0;
}
```

*Listing A.6:
GOODBYE.CPP,
erstellt von
HelloWizard*

A.3 Weitere Eigenschaften des Anwendungsassistenten

Obwohl HelloWizard alle wichtigen Schritte beim Erzeugen eines neuen benutzerdefinierten Anwendungsassistenten zeigt, gibt es noch einige Bereiche in der Entwicklung von benutzerdefinierten Anwendungsassistenten, die wir noch nicht berührt haben.

A.3.1 Anwendungsassistentenklassen

Die meisten der Funktionen der benutzerdefinierten Anwendungsassistenten werden durch die Klasse `CCustomAppWiz` implementiert. Beim Entwickeln von HelloWizard haben wir das Datenmitglied `m_Dictionary` dieser Klasse geändert. In komplexeren benutzerdefinierten Anwendungsassistenten können wir bessere Kontrolle über die Bearbeitung der Vorlagen haben, indem wir die Elementfunktionen dieser Klasse überlagern. Zum Beispiel durch Überlagerung der Elementfunktion `ProcessTemplate` können wir das Verhalten der Anwendungsassistentenmakroexpansion ändern oder völlig ersetzen.

`CAppWizStepDlg` ist eine von `CDialog` abgeleitete Klasse, die das Verhalten der Anwendungsassistentendialoge implementiert. Jeder Schritt in einem Anwendungsassistentendialog besitzt ein entsprechendes `CAppWizStepDlg`-Objekt. Anwendungen überschreiben gewöhnlich die Elementfunktion `OnDismiss` dieser Klasse (wie HelloWizard es tat), um ein kundenspezifisches Verhalten zu implementieren, wie das Aktualisieren des Inhalts des Wörterbuchs des Objekts `CCustomAppWiz`.

Wenn ein benutzerdefiniertes Anwendungsassistentenprojekt erzeugt wird, hat es üblicherweise eine weitere Klasse, `CDialogChooser`, definiert. Diese Klasse ist nicht aus irgendeiner MFC-Basisklasse abgeleitet; statt dessen dient sie als ein bequemes Mittel zum Zugriff auf das

Feld von `CAppWizStepDlg`-Objekten, die die Anwenderschnittstelle des benutzerdefinierten Anwendungsassistenten umfassen.

A.3.2 Funktionen der *MFCAPWZ.DLL*

Die DLL des Anwendungsassistenten, *MFCAPWZ.DLL*, exportiert eine Reihe von Funktionen, die benutzerdefinierte Anwendungsassistenten aufrufen können. Normalerweise brauchen Sie diese Funktionen nicht direkt aufzurufen; ein Code, der sie aufruft, wird beim Erzeugen Ihres benutzerdefinierten Anwendungsassistentenprojekts generiert.

Diese Funktionen von *MFCAPWZ.DLL* umfassen `GetDialog` (erhält einen Zeiger auf einen Standard-Anwendungsassistentenschritt), `SetCustomAppWizClass` (gibt einen Zeiger auf Ihre benutzerdefinierte Anwendungsassistentenklasse), `SetNumberOfSteps`, `ScanForAvailableLanguages` und `SetSupportedLanguages`.

A.3.3 Kontextbezogene Hilfe

Wir haben es vernachlässigt, eine Hilfedatei für HelloWizard zu erstellen, obwohl unser benutzerdefiniertes Anwendungsassistentenprojekt bereits mit einem Gerüst einer Hilfedatei ausgestattet war.

Die Hilfedatei muß einen Eintrag für jeden Schritt des benutzerdefinierten Anwendungsassistenten aufweisen. Sie muß die Form einer `Win-Help`-Datei haben. Zur Zeit gibt es keine Möglichkeit, die Hilfedatei für einen benutzerdefinierten Anwendungsassistenten mit dem Visual-C++-Hilfesystem zu integrieren. Der Basisname der Hilfedatei (das heißt, der Dateiname ohne Erweiterung) muß mit dem Basisnamen Ihres benutzerdefinierten Anwendungsassistenten identisch sein (der `AWX`-Datei). Darüber hinaus muß die Hilfedatei in dem gleichen Verzeichnis wie der benutzerdefinierte Anwendungsassistent stehen.

A.3.4 Debuggen eines benutzerdefinierten Anwendungsassistenten

Unser HelloWizard wurde in einem Zug durch Editieren, Kompilieren und Ausführen erstellt – nicht sehr schwierig, da dem vom Anwendungsassistenten generierten benutzerdefinierten Anwendungsassistentenprojekt nur eine Zeile hinzugefügt wurde. Unglücklicherweise tendieren Projekte im richtigen Leben dazu, komplex und fehlerhaft zu sein, sonst brauchten wir ja keine symbolischen Debugger!

Wie debuggt man ein benutzerdefiniertes Anwendungsassistentenprojekt, das eine DLL ist, die von der Umgebung selbst aufgerufen wird,

die normalerweise für das Debuggen eingesetzt wird (das heißt Visual Studio selbst)? Die Antwort liegt auf der Hand: durch Aufrufen einer zweiten Kopie von Visual Studio. Geben Sie auf der Seite `Debug` im Dialog PROJEKTEINSTELLUNGEN (Aufruf über PROJEKT/EINSTELLUNGEN) im Feld AUSFÜHRBARES PROGRAMM FÜR DEBUG-SITZUNG den Pfad des Programms Visual Studio (*msdev.exe*) an. Probieren Sie die Features Ihres benutzerdefinierten Anwendungsassistenten in dieser zweiten Kopie aus, während Sie die erste Kopie von Visual Studio als Debug-Umgebung zum Setzen von Haltepunkten, Ansehen der Werte der Variablen oder Verfolgen der Ausführung nutzen.

Es gibt einen wesentlichen Unterschied zwischen der Debug-Version eines benutzerdefinierten Anwendungsassistenten und der eines typischen MFC-Projekts. Wie Sie wohl bemerkt haben, ist die Debug-Version des benutzerdefinierten Anwendungsassistenten als Pseudo-Debug gekennzeichnet. Wozu diese Unterscheidung von normalen Debug-Versionen?

Der Grund ist, daß normale Debug-Ziele eine Speicherzuweisung verwenden, die verschieden von (und inkompatibel mit) dem Mechanismus der Speicherzuweisung der Lieferversion ist. Da Sie keinen Zugriff auf die Binärdateien des Debuggers von Visual C++ haben, ist es notwendig, diese Pseudo-Debug-Ziele einzusetzen, die eigentlich Lieferziele mit deaktivierten Optimierungen und hinzugefügter Debug-Information sind.

Welche Nachteile haben diese Zieltypen im Vergleich zu normalen Debug-Zielen? Mir fallen zwei Nachteile ein.

1. Durch Aussetzen des Speicherzuweisungsmechanismus wird es unwahrscheinlich, daß Sie Speicherprobleme und Fehler in der Speicherzuweisung Ihres benutzerdefinierten Anwendungsassistenten ausführen können.
2. Die Makros `ASSERT`, `VERIFY` und andere aus den Lieferversionen der MFC-Bibliothek tun nichts; wenn eine Bedingung angetroffen wird, die diese Makros normalerweise aktivieren würde, können Sie das Ergebnis nicht sehen. Dies ist kein Problem im Code des benutzerdefinierten Anwendungsassistenten selbst, da das vom Anwendungsassistenten generierte Gerüst des benutzerdefinierten Anwendungsassistenten die Pseudo-Debug-Versionen der entsprechenden Debug-Funktionen definiert.

Sicher, wenn es darauf ankommt, können Sie immer Ihre eigene Version der MFC mit ähnlich definierten Debug-Funktionen kompilieren, um diese Fähigkeiten in einem Pseudo-Debug-Ziel zu aktivieren.

A.3.5 Anweisungen und Makros in benutzerdefinierten Anwendungsassistenten

Es gibt zahlreiche Anweisungen und Makros in benutzerdefinierten Anwendungsassistenten, die ich nicht besprochen habe.

In der Datei *confirm.inf* von HelloWizard machten wir von den Anweisungen `$$IF` und `$$ENDIF` Gebrauch, um einen für die Intel-Plattform spezifischen Kommentar hinzuzufügen. Weitere benutzerdefinierte Anwendungsassistentenanweisungen sind `$$ELIF`, `$$ELSE`, `$$INCLUDE`, `$$BEGINLOOP`, `$$ENDLOOP`, `$$SET_DEFAULT_LANG`, und `$$//`.

Es gibt Standard-Anwendungsassistentenmakros, die jedem der Standard-Anwendungsassistentenschritte entsprechen. Weitere Standard-Makros beschreiben das neue Projekt, helfen beim Lokalisieren (internationale Unterstützung) und bieten verschiedene Funktionalitäten.

A.3.6 Einschränkungen

Obwohl benutzerdefinierte Anwendungsassistenten sehr leistungsfähig sind, gibt es Einschränkungen.

Die Anwendungsassistententechnologie wurde ursprünglich entwickelt, um auf die Anforderung nach angepaßten Gerüsten für MFC-Anwendungen zu reagieren. Somit ist diese Technologie zwangsläufig auf die Erstellung von MFC-Anwendungen abgestimmt.

Es gibt keinen einfachen Weg (das heißt, ich konnte keinen Weg finden, den ich nicht als Krampf bezeichnen würde), um in der neuen Projektdatei die Kompilierflags direkt zu beeinflussen. Somit, ob es Ihnen gefällt oder nicht, sind Sie mit einer Projektdatei festgefahren, die annimmt, daß Ihr Projekt eine MFC-Anwendung oder DLL ist. Obwohl es keinen Grund dafür gibt, warum die Anwendungsassistententechnologie nicht auch für Konsolenanwendungen eingesetzt werden könnte, erfordert dies manuelle Änderungen in der generierten Projektdatei, nachdem der Anwendungsassistent seine Aufgabe beendet hat.

Das Fehlen der Fähigkeit, Einstellungen der Projektdatei direkt zu beeinflussen, führt zu weiteren Einschränkungen, die sogar MFC-Projekte betreffen. Wenn sich Ihr Projektgerüst zum Beispiel auf eine nicht-standardisierte Bibliothek stützt, muß ihr Name manuell zum generierten Projekt hinzugefügt werden, ansonsten scheitert das Kompilieren des neuen Projekts.

Diese Einschränkungen sind eine Tatsache; trotzdem mindern sie nicht den Wert der benutzerdefinierten Anwendungsassistenten, vor allem in großen Organisationen.

A.4 Zusammenfassung

Benutzerdefinierte Anwendungsassistenten erweitern die Fähigkeit, Projektgerüste für anwenderdefinierte Projekttypen zu erstellen. Die wichtigsten Schritte zur Erstellung eines neuen benutzerdefinierten Anwendungsassistenten können folgendermaßen zusammengefaßt werden:

- Erzeugen eines benutzerdefinierten Anwendungsassistenten mit dem Anwendungsassistenten. Angabe, ob der neue Anwendungsassistent auf Standard-Anwendungsassistentenschritten, einem vorhandenen Projekt oder kundenspezifischen Schritten beruht.
- Ändern und Hinzufügen von Vorlagendateien nach Bedarf in dem Template-Verzeichnis des Anwendungsassistentenprojekts. Fügen Sie Referenzen auf neue Vorlagen in dem Abschnitt »TEMPLATE« der Ressourcendatei Ihres benutzerdefinierten Anwendungsassistenten hinzu.
- Erzeugen der Anwendungsassistentendialoge für das benutzerdefinierte Anwendungsassistentenprojekt. Zuweisen von Elementvariablen an Dialogfelder wie bei jedem Dialog, der auf der Klasse `CDialog` beruht.
- Ändern der Elementfunktion `OnDismiss` für jeden der Anwendungsassistentendialoge, um das Makro-Wörterbuch des benutzerdefinierten Anwendungsassistenten zu aktualisieren.
- Weitere Änderungen des Codes nach Bedarf. Rekompilieren des Projekts. Der neue benutzerdefinierte Anwendungsassistent wird automatisch in das Template-Verzeichnis des Visual Studio kopiert, damit Visual Studio ihn dort findet.
- Falls nötig, Debuggen des benutzerdefinierten Anwendungsassistenten durch Starten einer zweiten Kopie von Visual Studio.
- Erstellen einer Hilfedatei für den benutzerdefinierten Anwendungsassistenten mit dem Gerüst für Hilfedateien, das für das benutzerdefinierte Anwendungsassistentenprojekt erstellt wurde.

Übersicht zu C/C++

Anhang

B

Es gibt da draußen eine Menge guter C/C++-Lehrbücher und -Nachschlagwerke; dieser Anhang soll auf keinen Fall mit ihnen in Konkurrenz treten. Trotzdem ist es vielleicht nützlich, gleich hier am Ende des Buches eine Referenz zu finden. Es folgt daher eine (sehr) kurze Übersicht über die Programmiersprache C/C++.

B.1 Der Präprozessor

Während der ersten Kompilierungsphase werden alle C- und C++-Programme vom Präprozessor analysiert. Dieses Werkzeug erkennt bestimmte Anweisungen und wertet einige spezifische Operatoren und andere konstante Ausdrücke aus.

B.1.1 Präprozessoranweisungen

Präprozessoranweisungen erscheinen in der Form von Schlüsselwörtern, denen ein Nummernzeichen (#) vorangestellt ist, einzeln in einer Zeile des Quellcodes. Die vom Visual-C++-Compiler erkannten Anweisungen werden in Tabelle B.1 aufgelistet.

Makroname	Beschreibung
#define	Definiert ein Symbol oder Makro
#undef	Entfernt eine vorangegangene Definition
#if	Kompiliert, wenn ein konstanter Ausdruck wahr ist
#ifdef	Kompiliert, wenn ein Symbol definiert ist

Tabelle B.1:
Visual-C++-
Präprozessor-
anweisungen

Makroname	Beschreibung
<code>#if defined(X)</code>	Dasselbe wie <code>#ifdef X</code>
<code>#ifndef</code>	Kompiliert, wenn ein Symbol nicht definiert ist
<code>#elif</code>	»else if«-Konstrukt für bedingtes Kompilieren
<code>#else</code>	»else«-Konstrukt für bedingtes Kompilieren
<code>#endif</code>	Ende des bedingt kompilierten Blocks
<code>#include</code>	Einfügen einer weiteren Datei zum Kompilieren
<code>#import</code>	Importieren einer COM-Bibliothek
<code>#error</code>	Ausgabe eines Kompilierfehlers
<code>#line</code>	Ändern des Dateinamens und der Zeilennummer für Fehlermeldung
<code>#pragma</code>	Implementationsspezifische Anweisungen



Bedingte Kompilierung bedeutet, daß der angegebene Block, wenn die Bedingung nicht erfüllt ist, nach der Bearbeitung durch den Präprozessor vom Compiler vollständig ignoriert wird. Mit anderen Worten, er kann Syntaxfehler enthalten, nichtexistierende Symbole referenzieren oder gar Texte enthalten, ohne daß der Compiler Fehler meldet.

B.1.2 Ausdrücke für den Präprozessor

Der Präprozessor wertet konstante Ausdrücke zur Kompilierzeit statt zur Laufzeit aus, wodurch der Code merklich effizienter wird.

Konstante Ausdrücke

Ein konstanter Ausdruck besteht ausschließlich aus Konstanten oder Symbolen, die für den Präprozessor definiert sind und in konstante Ausdrücke aufgelöst werden.

Konstanten können numerisch sein, Zeichen, Zeichenketten oder Boolesche Konstanten. Einige Beispiele:

```
1
-845
3.141592654
6.054e23
'A'
"Hello, World!"
true
false
```

Konstanten und konstante Ausdrücke werden vom Präprozessor zur Kompilierzeit ausgewertet. Deshalb ist es manchmal von Vorteil, Konstante Ausdrücke statt berechneter Werte für bessere Lesbarkeit zu verwenden, ohne die Effizienz zu opfern. Vergleichen Sie

```
#define PI 3.141592653589
#define cube(x) (x*x*x)
V = 4.0 * PI / 3.0 * cube(r);
```

mit der Zeile

```
V = 4.188790204785 * r * r * r;
```

Was ist leichter als die Formel zur Berechnung des Volumens einer Kugel zu erkennen?

Operatoren für den Präprozessor

Der Präprozessor erkennt drei spezielle Operatoren. Der Stringoperator (`#`) kann zum Konvertieren eines Makroarguments in einen String verwendet werden. Zum Beispiel:

```
#define PRINTVAR(x) cout << "Der Wert von " << #x << " ist " << x;
```

Wird dies mit einer Variablen `vVar` als Argument aufgerufen, erweitert der Präprozessor den Aufruf folgendermaßen:

```
cout << "Der Wert von " << "vVar" << " ist " << vVar;
```

Der Zeichenoperator (`#@`) macht dasselbe, bloß daß das Makroargument in eine Zeichenkonstante in einfachen Anführungszeichen umgewandelt wird.

Der Verkettungsoperator (`##`) wird in Makros zum Verbinden einzelner Elemente eingesetzt. Betrachten Sie zum Beispiel folgende Makrodefinition:

```
#define VARI(i) (var##i)
```

Beim Aufruf als `VAR(5)` wird das Makro nach `var5` erweitert.

B.1.3 Vordefinierte Symbole

Während des Kompilierens sind auch einige vordefinierte Symbole verfügbar. Einige davon sind im C++-Standard definiert, während andere von Microsoft-spezifisch sind. Diese Symbole bestimmen Datum und Zeit des Kompilierens, den Namen der Quelldatei, den Sprachtyp (zum Beispiel ANSI C, C++), den Zielprozessor, die Compiler-Version, das Betriebssystem und einige Compiler-Optionen. Sie werden in Tabelle B.2 zusammengefaßt.

Tabelle B.2:
Vordefinierte
Symbole von
Visual C++

Symbolname	Beschreibung
ANSI Makros	
<code>__DATE__</code>	Ergibt den String »Mmm dd yyyy«, der das Kompilierdatum darstellt
<code>__FILE__</code>	Der Name der aktuellen Datei als String
<code>__LINE__</code>	Die aktuelle Zeilennummer als Integer-Konstante
<code>__STDC__</code>	Gibt »Standard C« an; nur definiert, wenn mit der Option /Za kompiliert wurde
<code>__TIME__</code>	Ergibt den String »hh:mm:ss« mit der Kompilierzeit
<code>__TIMESTAMP__</code>	Datum und Zeit der letzten Änderung der Datei als String »WoTag Mmm Datum Dd hh:mm:ss yyyy«
Visual-C++-spezifische Symbole	
<code>_CHAR_UNSIGNED</code>	Definiert, falls char-Typ unsigned (/J Compileroption)
<code>__cplusplus</code>	Definiert für C++-Programme
<code>_CPPRTTI</code>	Definiert, falls Laufzeitinformation aktiviert ist (/GR Compileroption)
<code>_CPPUNWIND</code>	Definiert, falls C++-Ausnahmebehandlung aktiviert ist (/GX Compileroption)
<code>_DLL</code>	Definiert, wenn eine DLL kompiliert wird (/MD)
<code>_M_ALPHA</code>	Definiert als 1, wenn die CPU eine DEC Alpha ist
<code>_M_IX86</code>	Definiert, wenn die CPU eine Intel x86 ist; der Wert stellt den Prozessortyp dar (zum Beispiel 500 = Pentium)
<code>_M_MPPC</code>	Definiert, wenn die Zielplattform ein Power Macintosh ist; der Wert stellt den Prozessortyp dar (zum Beispiel 601)
<code>_M_MRX000</code>	Definiert, wenn die Zielplattform MIPS ist; der Wert stellt den Prozessortyp dar (zum Beispiel 4000)
<code>_M_PPC</code>	Definiert, wenn die Zielplattform ein Power PC ist; der Wert stellt den Prozessortyp dar (zum Beispiel 604)
<code>_MFC_VER</code>	Definiert die MFC-Version (zum Beispiel 0x0421)
<code>_MSC_VER</code>	Definiert die Compilerversion (1100)
<code>_MT</code>	Definiert, wenn /MD oder /MT verwendet werden
<code>_WIN32</code>	Definiert die Win32-Plattform; immer definiert

B.1.4 Kommentare

Die Sprache C/C++ kennt zwei Arten von Kommentaren.

- ➔ Mehrzeilige Kommentare beginnen mit einem Schrägstrich und einem Asterisk (/); alles nach diesem Symbolpaar wird als Kommentar betrachtet und vom Compiler nicht interpretiert, bis das Ende des Kommentars (*/) angetroffen wird.
- ➔ Einzeilige Kommentare beginnen mit zwei Schrägstrichen (//).

B.2 Compiler

Die Sprache C/C++ ist eine deklarative Sprache. Das bedeutet, daß jedes im Programm verwendete Symbol vorher explizit deklariert sein muß. Ein Symbol kann als Typ oder Name aus einem Namensraum, als Marke, Variable oder Funktion deklariert sein.

Ein C++-Programm besteht aus Anweisungen. Anweisungen können Deklarationen, Definitionen, Ausdrücke oder andere C/C++-Anweisungen mit Schlüsselwörtern sein.

B.2.1 Deklarationen und Definitionen

Eine Deklaration unterscheidet sich von einer Definition dadurch, daß die Deklaration den Typ eines Symbols festlegt, während die Definition tatsächlich Speicherplatz dafür zuweist. Zum Beispiel ist folgende Zeile eine Deklaration:

```
extern int i;
```

Diese Anweisung bezeichnet für den Compiler den Typ der Variablen `i`, doch weil sie mit dem Schlüsselwort `extern` deklariert ist, wird angenommen, daß sie in einem anderen Modul definiert ist; somit wird ihr kein Speicher zugewiesen. Ähnlich ist eine typische C/C++-Headerdatei voll mit Funktionsdeklarationen wie folgende:

```
void MyFunc(int mypar);
```

Diese Zeile bezeichnet für den Compiler den Rückgabotyp und die Parameterliste der Funktion, doch da es keinen Funktionsrumpf gibt, wird die Funktion in diesem Stadium nicht definiert.

Eine Definition kann auch als Deklaration dienen; zum Beispiel, wenn Sie die globale Variable `int i` definieren, muß sie nicht noch einmal in einer Headerdatei deklariert sein, es sei denn, sie wird von mehr als einem Modul referenziert.

B.2.2 Gültigkeitsbereich und Bindung

Der Gültigkeitsbereich einer Deklaration bestimmt die Stellen im Programm, an denen ein Symbol sichtbar ist. Deklarationen können in einem Codeblock, einer Funktion oder Klasse lokal sein. Sie können auch global, das heißt, überall in einem Modul sichtbar sein.

Mit dem Gültigkeitsbereich ist auch der Begriff »Bindung« geknüpft. Ein Symbol hat interne Bindung, wenn es nicht mehreren Modulen gemeinsam ist, ansonsten heißt es, es hat externe Bindung.

B.2.3 Typen

Eine Variable kann in C/C++ von einfachem, abgeleitetem oder zusammengesetztem Typ sein.

B.2.4 Einfache Typen

Die einfachen oder fundamentalen Typen sind in Tabelle B.3 aufgezählt.

Tabelle B.3:
Typen in
Visual C++

Typ	Beschreibung
bool	Integraltyp; ein boolescher Wert wahr oder falsch:
char	Integraltyp, ein Zeichen (1 Byte)
short	Zwei-Byte-Integer
int	Vier-Byte-Integer
long	Vier-Byte-Integer
float	Vier-Byte-IEEE-Fließkommazahl
double	Acht-Byte-IEEE-Fließkommazahl
long double	Dasselbe wie double
void	Ein Pseudotyp, der auf einen beliebigen Typ oder eine Funktion ohne Rückgabewert zeigt
enum	Aufzählungstyp
__intn	Integraltyp, ist n Bits breit, wobei n = 8, 16, 32 oder 64

Die Ganzzahlentypen sind alle vorzeichenbehaftet, das heißt, sie können sowohl positive als auch negative Zahlen darstellen. Dieses Verhalten kann mit dem Schlüsselwort `unsigned` verändert werden. Zum Beispiel stellt `char` normalerweise einen Wert zwischen -128 und 127 dar; `unsigned char` stellt einen Wert zwischen 0 und 255 dar.

Kommandozeilenflags können dieses Verhalten ändern. In diesem Fall kann mit dem Schlüsselwort `signed` ein vorzeichenbehafteter Typ erzwungen werden.

B.2.5 Spezifizierer

Die Lebensdauer und der Gültigkeitsbereich einer Variablen kann mit den Spezifizierern für die Speicherklasse beeinflusst werden. Diese sind in Tabelle B.4 aufgezählt.

Spezifizierer	Beschreibung
<code>auto</code>	Die Variable wird automatisch auf dem Stapel angelegt, wenn der Block ausgeführt wird, in dem sie definiert ist. Dies ist das normale Verhalten für Variablen, die in dem Rumpf einer Funktion definiert sind.
<code>register</code>	Die Variable wird in den Registern des Prozessors angelegt, wenn möglich. Optimierungen durch den Compiler können das ändern. Eine so deklarierte Variable hat keine Adresse.
<code>static</code>	Die Variable wird auf der Halde angelegt und behält ihren Wert, solange das Programm läuft. Dieses Verhalten zeigen globale Variablen. Eine globale Variable, die explizit statisch deklariert wird, wird von anderen Modulen nicht gesehen (interner Linktyp).
<code>extern</code>	Die Variable wird deklariert, bekommt aber keinen Speicherplatz zugewiesen. Es wird angenommen, daß sie sich in einem anderen Modul befindet.

*Tabelle B.4:
Spezifizierer
für die Speicher-
klasse*

Deklarationen können auch Typenbezeichner enthalten. Diese Schlüsselwörter sind in Tabelle B.5 gezeigt.

Bezeichner	Beschreibung
<code>const</code>	Die Variable kann initialisiert, doch ihr Wert nachträglich nicht geändert werden. Der Einsatz einer solchen Variablen links von einer Zuweisung führt zu einem Fehler.
<code>volatile</code>	Dieses Schlüsselwort sagt dem Compiler, daß der Wert dieser Variablen sich außerhalb seiner Kontrolle verändern kann, um extreme Optimierungen zu vermeiden. Eine typische Anwendung wären Variablen, die von mehreren Threads verändert werden.

*Tabelle B.5:
Typenbezeichner*

B.2.6 Abgeleitete Typen

Diese umfassen Felder, Zeiger und Referenzen.

Felder

Felder werden mit eckigen Klammern, die die Dimension des Feldes angeben, deklariert. Mehrdimensionale Felder sind erlaubt:

```
int n[3][3][3];
```

Zeiger

Zeiger stellen die Adresse eines Objekts vom gegebenen Typ dar. Zum Beispiel stellt eine als `char *` deklarierte Variable die Adresse eines Zeichens dar.

Referenzen

Eine Referenz ist ein anderer Name für ein Objekt. Referenzen werden hauptsächlich für Funktionsargumente und Rückgabewerte verwendet, wenn die Übergabe eines Objekts über seinen Wert unpraktisch, über die Adresse jedoch ungünstig ist.

Betrachten wir zum Beispiel eine Funktion `funk`, die ein Argument `bigtyp` bekommt. Der selbstdefinierte `bigtyp` bezieht sich auf einen großen zugewiesenen Speicherbereich. Deshalb wäre eine als `funk(bigtyp x)` deklarierte Funktion ungünstig, da das Argument vollständig kopiert würde. Man könnte einen Zeiger verwenden, wie in `funk(bigtyp *px)`, doch dann müßte die Funktion umgeschrieben werden, um überall `x` durch `*px` zu ersetzen. Die Alternative ist, eine Referenz als Argument zu verwenden: `funk(bigtyp &x)`. Das ist die beste Lösung: Das Argument wird nicht kopiert, es wird bloß die Adresse an `funk` übergeben, und innerhalb der Funktion kann `x` verwendet werden statt eines Zeigers auf diesen Typ.

Beachten Sie beim Deklarieren von Zeigern oder Referenzen, daß die Operatoren `*` und `&` an die Variable gebunden sind, nicht an den Typ. Deshalb ist folgende Deklaration irreführend, wenn auch zulässig:

```
int* p, q;
```

Durch das Leerzeichen würde man meinen, beide Variablen seien als Zeiger deklariert. Dem ist aber nicht so, die Deklaration ist nämlich mit folgender gleichbedeutend:

```
int *p, q;
```

oder

```
int *p;  
int q;
```

Leider wird diese irreführende Syntax oft in von Microsoft geliefertem Code verwendet, was manchem Programmierer Ärger verursacht.

B.2.7 Typendefinitionen

Das Schlüsselwort `typedef` kann zum Erzeugen selbstdefinierter Typen verwendet werden. Diese Typendefinitionen funktionieren ähnlich wie Präprozessormakros. Betrachten Sie folgende Zeilen:

```
typedef unsigned char uchar;  
#define uchar (unsigned char)
```

Der Hauptunterschied besteht darin, daß Makros vom Präprozessor interpretiert werden, `typedef` jedoch vom Compiler. Somit sind Definitionen mit `typedef` den Gültigkeitsregeln des Compilers unterworfen. Zum Beispiel ist ein `typedef` innerhalb einer Funktion nur im Funktionsrumpf gültig, ein Makro an der gleichen Stelle würde bis ans Ende der Datei gültig bleiben.

B.2.8 Zusammengesetzte Typen

Zusammengesetzte Typen umfassen Klassen, Strukturen und Varianten.

Klassen

Eine Klasse ist eine zusammengesetzte Gruppe von Datenobjekten und Funktionen, die diese Objekte manipulieren. Betrachten Sie folgende Klassendefinition:

```
class number {  
    int n;  
    int sq() { return n * n; };  
};
```

In diesem Fall wird `n` als Elementvariable bezeichnet, `sq` als Elementfunktion.

Die obige Definition veranlaßt den Compiler nicht, Speicher zuzuweisen. Das geschieht nur, wenn ein Objekt vom Typ `number` deklariert wird (die Klasse wird instantiiert). Zum Beispiel

```
int foo(void) {  
    number x;  
    return x.sq(); // sq berechnet nun x.n * x.n  
}
```

Per Definition sind Elemente einer Klasse `privat`, das heißt, sie sind nur für Elementfunktionen sichtbar und können nicht vererbt werden. Elemente können explizit als `private`, `protected` und `public` deklariert werden. `Protected`-Elemente sind nur für Elementfunktionen sichtbar, können jedoch vererbt werden. `Public`-Elemente sind sowohl sichtbar

als auch vererbbar. Eine Struktur, die mit dem Schlüsselwort `struct` deklariert ist, verhält sich wie eine Klasse, nur daß ihre Elemente `public` sind.

Das durch `protected` und `private` bestimmte Verhalten kann mit dem Schlüsselwort `friend` geändert werden. Eine Klasse oder Funktion, die als `friend` zu einer anderen Klasse deklariert ist, kann auf die `protected`- und `private`-Elemente der Klasse zugreifen:

```
class myclass {
    friend int f1(myclass &);
private:
    int i;
};

int f1(myclass &o)
{
    return o.i;
}

int f2(myclass &o)
{
    return o.i;    // Error: cannot access private member
}
```

Ein Datenelement kann als `static` deklariert werden, wobei es von den Objekten der Klasse gemeinsam genutzt wird. Ein typisches Beispiel für den Einsatz solcher Datenelemente wäre ein Instanzzähler:

```
class CMyObject
{
public:
    CMyObject() { nInstance++; };
    ~CMyObject() { nInstance--; };
private:
    static int nInstance;
};
```

Klassen können von anderen Klassen abgeleitet werden. Zum Beispiel

```
class twonumbers : public number {
    int m;
    int pr() { return n * m; };
};
```

Damit dieses Beispiel funktioniert, müssen die Elementvariablen der Klasse `number` als `protected` oder `public` deklariert sein.

Strukturen

Eine Struktur, mit dem Schlüsselwort `struct` definiert, ist wie eine Klasse, mit der Ausnahme, daß ihre Elemente `public` sind.

Elemente einer Klasse oder Struktur können außerhalb der Klasse explizit referenziert werden, indem dem Bezeichner der Name der Klasse, gefolgt von zwei Doppelpunkten, vorangestellt wird, zum Beispiel `myclass::myclassvar`.

Varianten

Varianten, mit dem Schlüsselwort `union` deklariert, sind im Aussehen den Strukturen ähnlich, die nur Datenelemente enthalten – doch mit einem entscheidenden Unterschied: Die Elemente überlappen sich im Speicher. Dies erlaubt Programmen, sich auf die gleichen Speicherstellen mit verschiedenen Namen zu beziehen, wie in folgendem Beispiel:

```
union {
    unsigned int i;
    float f;
} u;
```

Wenn Sie nun `u.f` den Wert 3.1415 zuweisen, wird `u.i` 0x40490FD0 sein, die binäre Darstellung der IEEE-Fließkommazahl.

B.2.9 Namensräume

Um das Problem der Namenskonflikte in verschiedenen Bibliotheken zu lösen, bietet C++ das Konzept des Namensraums. Namensräume werden mit dem Schlüsselwort `namespace` definiert. In einem Namensraum definierte Symbole sind eindeutig in Bezug auf gleichnamige Symbole in anderen Namensräumen.

Das Schlüsselwort `using` wird im Zusammenhang mit Namensräumen verwendet, um den Namensraum zu bezeichnen, der unqualifiziert verwendet werden kann, das heißt ohne den Namensraum explizit anzugeben.

Namensräume können explizit referenziert werden, indem dem Bezeichner der Name des Namensraums mit zwei Doppelpunkten (`::`) vorangestellt wird.

B.2.10 Templates

Der Einsatz von Templates erlaubt dem C++-Programmierer das Definieren allgemeiner Funktionen oder Klassen, die auf eine Familie von Typen anwendbar sind.

Obwohl Vorlagen typischerweise im Zusammenhang mit Klassen eingesetzt werden, zeigt das folgende einfache Beispiel anhand einer Funktion das Verhalten am besten:

```
#include <iostream.h>

template <class T> T square(T x) {return x * x;};

void main(void)
{
    int i = 3;
    double x = 3.141592654;

    cout << "Das Quadrat von " << i << " ist " << square(i)
```

```
<< endl;
cout << "Das Quadrat von " << x << " ist " << square(x)
<< endl;
}
```

Das Schlüsselwort `typename` kann in Vorlagendefinitionen verwendet werden, um anzuzeigen, daß ein unbekannter Bezeichner ein Typ ist.

B.2.11 Funktionsdeklarationen

Eine Funktionsdefinition besteht aus dem Namen der Funktion, der Parameterliste und dem Rückgabewert. Der Rückgabewert wird durch die Anweisung `return` bestimmt, wie in folgendem Beispiel:

```
double cube(double x)
{
    return x * x * x;
}
```

Der Typ einer Funktion zeigt den Typ des zurückgegebenen Wertes an. Dieser kann auch `void` sein, wenn eine Funktion keinen Wert zurückgibt.

Die Deklaration und Definition einer Funktion kann an verschiedenen Stellen im Code sein. Typischerweise werden Funktionsdeklarationen in Headerdateien gemacht, wo sie allen Quelldateien zugänglich sind. Eine Funktionsdeklaration besteht aus dem Namen der Funktion, der Parameterliste und dem Rückgabewert, nicht jedoch aus dem Rumpf der Funktion. Die Parameterliste muß keine Namen für die Parameter enthalten, der Typ allein genügt. Zum Beispiel

```
double cube(double);
```

Das Schlüsselwort `inline` in der Deklaration einer Funktion weist den Compiler an, Funktionsaufrufe durch Kopieren des Funktionscodes an der Aufrufstelle zu implementieren, im Gegensatz zum Aufruf der Funktion als Subroutine. Dies vermeidet Laufzeitverluste, die durch den Aufruf entstehen, auf Kosten der Programmgröße. Beachten Sie, daß der Compiler nicht immer das Schlüsselwort `inline` beachtet, zum Beispiel bei Funktionen, die sich selber rekursiv aufrufen. Diese können nicht als `inline`-Funktionen implementiert werden.

B.2.12 Funktionsüberladung

Funktionen können überladen werden, es können also mehrere Funktionen mit demselben Namen, doch mit verschiedenen Parameterlisten definiert werden, wie zum Beispiel:

```
double cube(double);
int cube(int);
```

Der zur Überladung von Funktionen verwendete Mechanismus kann auch auf Operatoren angewandt werden. Dies wird durch das Schlüsselwort `operator` ermöglicht, wie das folgende Beispiel zeigt:

```
class complex {
    complex(double r0, double i0) {r = r0; i = i0;};
    double r;
    double i;
    complex operator+(complex &a) {return complex(r+a.r,
                                                    i+a.i);};
}
```

Funktionsparameter können auch Defaultwerte haben. Betrachten Sie zum Beispiel folgende Funktionsdeklaration aus der MFC-Bibliothek:

```
int AfxMessageBox(LPCTSTR lpszText, UINT nType = MB_OK,
                  UINT nIDHelp = 0);
```

Diese Form von Deklaration ist gleichbedeutend mit der Deklaration dreier überladener Versionen der Funktion mit 1, 2, und 3 Parametern.

B.2.13 Elementfunktionen

C++ erlaubt die Verwendung von Funktionen innerhalb von Klassen. Solche Funktionen werden Elementfunktionen genannt. Elementfunktionen können Elementdaten der Klasse referenzieren. Elementfunktionen werden immer an ein bestimmtes Objekt gebunden aufgerufen, mit Ausnahme der statischen Elementfunktionen, die nur auf statische Elementdaten zugreifen können.

Nichtstatische Elementfunktionen können auf den Zeiger `this` zugreifen, ein spezieller Zeiger, der auf das aktuelle Objekt zeigt. Betrachten Sie zum Beispiel die Klassendeklaration:

```
class CMyClass {
    int x;
};
```

In einer Elementfunktion dieser Klasse wären `x` und `this->x` gleichbedeutend.

Klassen haben zwei spezielle Elementfunktionen:

- Der Konstruktor wird aufgerufen, wenn die Klasse erzeugt wird.
- Der Destruktor wird aufgerufen, wenn die Klasse zerstört wird.

Diese Funktionen werden typischerweise für Initialisierung und Aufräumaufgaben eingesetzt. Die Konstruktorfunktion wird mit dem Klassennamen bezeichnet, die Destruktorfunktion mit dem Klassennamen und einer vorangestellten Tilde.

Wenn ein Konstruktor mit dem Schlüsselwort `explicit` deklariert wird, nimmt der Compiler keine impliziten Typenumwandlungen daran vor.

Elementfunktionen können als `virtual` deklariert sein. Referenzen auf solche Funktionen werden mit Hilfe einer Tabelle (`vtable`) gemacht, um sicherzustellen, daß die richtige Elementfunktion aufgerufen wird, auch wenn das Objekt über einen Zeiger auf die Elternklasse referenziert wird.

Elementfunktionen können auch mit dem Schlüsselwort `const` deklariert werden. Diese Elementfunktionen können den Wert einer Elementvariablen nicht ändern, es sei denn, sie werden mit dem Schlüsselwort `mutable` versehen.

B.2.14 Ausführbare Anweisungen

Deklarationen und Definitionen wurden in den vorherigen Abschnitten besprochen. Die folgenden Abschnitte enthalten eine Zusammenfassung der restlichen Anweisungstypen, die ein C/C++-Programm enthalten kann.

Zusammengesetzte Anweisungen

Zusammengesetzte Anweisungen sind einfach eine Reihe von Anweisungen, in geschweiften Klammern eingeschlossen. Wo immer eine Anweisung vorkommen kann, kann auch eine zusammengesetzte Anweisung sein. Deklarationen innerhalb einer zusammengesetzten Anweisung sind im allgemeinen außerhalb ungültig:

```
int i = 1;
{
    int i = 2;
    cout << i << endl; // Ausgabe: '2'
}
cout << i << endl;    // Ausgabe: '1'
```

Ausdrücke und Operatoren

Ein Ausdruck in C/C++ besteht aus Konstanten, Variablen oder Funktionsaufrufen, die durch Operatoren verbunden sind.

Die Operatoren der Sprache C/C++ sind in Tabelle B.6 aufgezählt.

Tabelle B.6:
Operatoren

Operator	Beschreibung
Speicherzuweisungsoperatoren	
<code>new</code>	Speicher allokieren
<code>delete</code>	Speicher freigeben

Operator	Beschreibung
Typenbezogene Operatoren	
dynamic_cast	Sichere Typenumwandlung
static_cast	Sichere Typenumwandlung
const_cast	Sichere Typenumwandlung
reinterpret_cast	Sichere Typenumwandlung
typeid	Sichere Typenumwandlung
Unäre Präfix-Operatoren	
+	Plus-Zeichen (kein Effekt)
-	Negation
*	Zeigerdereferenzierung
&	Adresse von
!	Logisches NICHT
~	Arithmetisches NICHT
++	Prä-Inkrement
--	Prä-Dekrement
sizeof	Speichergröße
Unäre Postfix-Operatoren	
++	Post-Inkrement
--	Post-Dekrement
Binäre Operatoren	
+	Addition
[ms]	Subtraktion
*	Multiplikation
/	Division
%	Modulo
^	Exklusives ODER
&	Arithmetisches UND
&&	Logisches UND
	Arithmetisches ODER
	Logisches ODER
>>	Rechtsschieben
<<	Linksschieben

Operator	Beschreibung
=	Zuweisung
+=	Zuweisung
-=	Zuweisung
*=	Zuweisung
/=	Zuweisung
%=	Zuweisung
^=	Zuweisung
&=	Zuweisung
=	Zuweisung
>>=	Zuweisung
<<=	Zuweisung
==	Vergleich auf Gleichheit
!=	Vergleich auf Ungleichheit
<	Kleiner als
<=	Kleiner oder gleich
>	Größer als
>=	Größer oder gleich
.	Element von
->	Dereferenzierung und Element von
.*	Zeiger auf Element
->*	Zeiger auf Element
,	Liste
[]	Index
Tertiäre Operatoren	
? :	Bedingter Ausdruck

Ein sehr wichtiges Konzept in C/C++ ist der L-Wert oder Linkswert. Ein L-Wert ist einfach gesagt ein Ausdruck, der eine Speicherstelle referenziert. L-Wert-Ausdrücke können als Operanden des Adreßoperators (&) auftreten. L-Werte, die kein `const`-Objekt referenzieren, können auch als linker Operand einer Zuweisung auftreten.

Inline-Assembleranweisungen

Der C/C++-Compiler kann auch Assembleranweisungen übersetzen. Das Schlüsselwort `__asm` kann von einer einzelnen oder von mehreren

in geschweiften Klammern eingeschlossenen Assembleranweisungen werden. Zum Beispiel wird auf Intel-Prozessoren die Zeile `__asm int 3` oft zum Einfügen eines Haltepunkts in dem Code verwendet.

Beachten Sie, daß Visual C++ nur das Schlüsselwort `__asm` unterstützt, dessen Syntax sich vom AT&T-Schlüsselwort `asm` leicht unterscheidet.

Ausnahmebehandlung

Die Sprache C++ unterstützt Ausnahmen. Ausnahmen sind hauptsächlich unbedingte Übergaben der Programmkontrolle, die bis jenseits der Funktionsgrenzen reichen können. Sie werden hauptsächlich zur Fehlermeldung eingesetzt.

Eine Ausnahme wird mit dem Schlüsselwort `throw` gesetzt und mit dem Konstrukt `try/catch` aufgefangen. Ausdrücke sind in C++ typisiert, somit unterscheidet `catch` die Ausnahmen aufgrund des Objekttyps, mit dem sie gesetzt wurden.

Ausnahmen werden oft zur Fehlerbehandlung eingesetzt. Zum Beispiel:

```
double divide(double dividend, double divisor)
{
    try
    {
        if (divisor == 0.0) throw 1;
        if (divisor < 1E-10) throw 2;
        return dividend / divisor;
    }
    catch(int e)
    {
        switch (e)
        {
            case 1: cerr << "Division by zero." << endl; break;
            case 2: cerr << "Divisor too small." << endl; break;
            default: throw;
        }
        return 0.0;
    }
}
```

Steht das Schlüsselwort `throw` allein, löst es die zuletzt aufgefangene Ausnahme erneut aus.

Obwohl das vorherige Beispiel einen einfachen Integertyp zur Bezeichnung der Ausnahme verwendet, wird gewöhnlich eine Klassenhierarchie zu diesem Zweck definiert.

Der Operator `new` kann eine Ausnahme vom Typ `xalloc` setzen. Dazu muß das Programm mit dem Befehlszeilenflag `/GX` kompiliert und die Funktion `_set_new_handler` mit `_standard_new_handler` als Argument aufgerufen werden.

Programmsteuerung

C/C++ bietet zwei Schlüsselwörter zur unbedingten Übergabe der Kontrolle an einen anderen Punkt im Programm.

Die Anweisung `goto` kann die Ausführung an eine Marke übergeben. Beachten Sie, daß die Marke sich innerhalb der gleichen Funktion wie die `goto`-Anweisung befinden muß. Eine Marke ist ein Symbol, das von einem Doppelpunkt (:) gefolgt wird. Die Ausführung wird an die gleich darauffolgende Anweisung übergeben.

Die Anweisung `return` verursacht das Beenden der Ausführung einer Funktion und Übergabe der Kontrolle an die aufrufende Funktion. Beachten Sie, daß am Ende jeder Funktion ein implizites `return` angenommen wird.

Bedingte Ausführung

Die Anweisung `if` erlaubt bedingte Ausführung. Die Syntax der `if`-Anweisung lautet:

```
if (Ausdruck)
    Anweisung;
else
    Anweisung;
```

Das Schlüsselwort `else` und die darauffolgende Anweisung sind optional.

Die Anweisung `switch` erlaubt das Ausführen bestimmter Programmzweige aufgrund des Wertes einer ganzzahligen Variablen. Die typische `switch`-Anweisung hat die Form:

```
switch ( integral_expression ) {
    case1 (constant-expression): statement(s)
    case2 (constant-expression): statement(s)
    ...
    default: statement(s)
}
```

Die letzte Anweisung in einem `case`-Block ist typischerweise, jedoch nicht notwendigerweise, eine `break`-Anweisung. Diese weist den Compiler an, die Ausführung an einen Punkt nach der `switch`-Anweisung zu übergeben. Wird `break` weggelassen, so werden die Befehle sequentiell in folgenden `case`-Blöcken ausgeführt.

Schleifen

Schleifenanweisungen bieten einen leicht einzusetzenden Mechanismus zum wiederholten Ausführen von Befehlen. Strenggenommen sind Schleifenanweisungen überflüssig, `if` und `goto` würden reichen. Schleifenanweisungen ergeben aber besser lesbaren Code.

Die Syntax der `for`-Anweisung lautet:

```
for ( Ausdruck1 ; Ausdruck2 ; Ausdruck3 )  
    Befehl;
```

Wenn diese Anweisung angetroffen wird, werden zuerst `Ausdruck1` und dann `Ausdruck2` ausgewertet; ergibt `Ausdruck2` `Null`, wird die Ausführung an einen Punkt nach der `for`-Anweisung übergeben. Ansonsten werden nacheinander der Rumpf der `for`-Anweisung ausgeführt, der `Ausdruck3` ausgewertet und `Ausdruck2` erneut getestet, so lange bis `Ausdruck2` `Null` ergibt.

Mit anderen Worten könnte eine `for`-Anweisung mit `if` und `goto` folgendermaßen umgeschrieben werden:

```
Ausdruck1;  
LOOP:  
    if (Ausdruck2) goto END;  
Anweisung;  
Ausdruck3;  
goto LOOP;  
END;
```

Die `while`-Anweisung wird zum wiederholten Ausführen von Befehlen verwendet, bis der Ausdruck `Null` wird. Die `while`-Anweisung in ihrer grundlegenden Form `while (Ausdruck) Befehl` kann folgendermaßen umgeschrieben werden:

```
LOOP:  
    if (!Ausdruck) goto END;  
    Anweisung;  
    goto LOOP;  
END;
```

Eine `do`-Anweisung ist `while` ähnlich, ausgenommen daß der Rumpf mindestens einmal ausgeführt wird, bevor der Ausdruck ausgewertet wird. Eine `do`-Anweisung hat folgende Syntax:

```
do  
    Befehl;  
while (Ausdruck)
```

Unter Verwendung von `if` und `goto` kann dies folgendermaßen umformuliert werden:

```
LOOP:  
    Anweisung;  
    if (Ausdruck) goto LOOP;
```

Zwei Schlüsselwörter von C/C++ haben in Schleifen eine besondere Bedeutung. Das Schlüsselwort `break` kann verwendet werden, um jederzeit aus der Schleife auszusteigen. Im Endeffekt ist `break` gleichbedeutend mit einer `goto`-Anweisung zu einer Marke gleich hinter der Schleife. Die Anweisung `continue` ist gleichbedeutend mit einem `goto` zu einer Marke am Anfang der Schleife.

Speicherzuweisung und Freigabe

Objekte können in C++ mit dem Operator `new` Speicher zugewiesen bekommen, der mit `delete` freigegeben wird. Das Ergebnis des `new`-Operators ist ein Zeiger auf den entsprechenden Typ. Der Operator `delete` wirkt auf einen Zeiger auf den entsprechenden Typ.

Eine Variante der Operatoren, `new[]` und `delete[]`, wird zur Allokation und Freigabe eines Feldes von Objekten verwendet, wie in folgendem Beispiel:

```
int *p;
p = new int[100];
...
delete[] p;
```

Casting

Der Compiler kann eine implizite Typenumwandlung vornehmen, wo so etwas sinnvoll ist. Führt die Umwandlung zu einem Verlust von Stellen, wird eine Warnung ausgegeben. Diese Warnung kann unterbunden werden, indem die Umwandlung explizit gemacht wird:

```
int i;
char c;
...
c = i;           // Warnung
c = (char)i;     // Keine Warnung
```

Umwandlungen von Zeigern auf verschiedene Typen ergeben normalerweise eine Fehlermeldung, explizite Umwandlungen sind aber zugelassen.

Eine neuere Methode zur expliziten Typenumwandlung verwendet die Operatoren `dynamic_cast`, `static_cast`, `const_cast`, `reinterpret_cast`.

Typeninformation zur Laufzeit

Der Visual-C++-Compiler unterstützt Typeninformation zur Laufzeit. Der Typ eines Ausdrucks wird vom Operator `typeid` in der Form einer `const type_info&` zurückgegeben.

Wenn der Operator `typeid` scheitert, erzeugt er eine Ausnahme vom Typ `bad_typeid`.

B.3 Kompilieren und Programmausführung

Bevor ein C/C++-Programm ausgeführt werden kann, muß der Quellcode zuerst kompiliert werden. Es folgt eine kurze Übersicht über den Kompilervorgang und den Ablauf und Endes eines C/C++-Programms.

B.3.1 Übersicht zum Kompilieren

Das Kompilieren eines Programms hat mehrere eigenständige Phasen.

Während der Bearbeitung durch den Präprozessor werden spezifische Anweisungen ausgeführt, Makros ersetzt und konstante Ausdrücke berechnet. Obwohl dieser Vorgang typischerweise ein fester Bestandteil des Kompilierens ist, kann der Compiler mit der Befehlszeilenoption `/P` angewiesen werden, das Zwischenergebnis in eine Datei abzuspeichern.

Während der Code erzeugt wird, wird die C- oder C++-Quelldatei in Objektcode übersetzt. Ein typisches C/C++-Programm besteht aus mehreren Quelldateien. Symbole, die von den Modulen gemeinsam benutzt werden, sind in Headerdateien deklariert. Der Visual-C++-Compiler bietet die Option, vorkompilierte Header zu benutzen, was die Effizienz des Kompilervorgangs merklich verbessert, indem das wiederholte Bearbeiten der identischen Teile am Anfang der Quelldateien vermieden wird. Wenn umfangreiche Headerdateien verwendet werden, beispielsweise in MFC-Anwendungen, kann dieser Teil Zehntausende von Programmzeilen enthalten.

Während des Linkens werden symbolische Referenzen in numerische Adressen aufgelöst. Module der Objektdaten und Bibliotheksdateien werden zu einer ladbaren, ausführbaren Abbildung zusammengefaßt. Die Gültigkeit der Symbole aus den kompilierten Modulen wird von den Regeln des Linkers bestimmt. Trifft der Linker auf ein unbekanntes Symbol, gibt er eine Fehlermeldung aus.

B.3.2 Ausführen eines C/C++-Programms

Das Ausführen eines C-Programms beginnt mit einem Aufruf der Funktion `main`. Unter Windows kann diese durch `wmain` für Unicode-Anwendungen oder `WinMain` für interaktive Windows-Programme ersetzt werden.

Bevor die Hauptfunktion des Programms aufgerufen wird, finden einige Initialisierungsschritte statt. Einige davon sind spezifisch für das Betriebssystem (zum Beispiel das Setzen der Standardein- und -ausgabe), während andere der Programmiersprache eigen sind (wie das Setzen der Funktionsparameter für `main` oder des für automatische Variablen verwendeten Stapels).

In C-Programmen ist die erste Programmzeile, die tatsächlich ausgeführt wird, die erste Zeile der `main`-Funktion. In C++ ist dies nicht immer der Fall. Gibt es global deklarierte Variablen von Typen, die Konstruktoren haben, so werden diese Konstruktoren zuerst aufgerufen. Die Reihenfolge, in der Konstruktoren, Destruktoren und die `main`-Funktion aufgerufen werden, demonstriert das folgende einfache Beispiel:

```
#include <iostream.h>

class MyFirstType
{
public:
    MyFirstType() { cout << "In MyFirstType::MyFirstType\n"; };
    ~MyFirstType() { cout << "In MyFirstType::~MyFirstType\n"; };
};

class MySecondType
{
public:
    MySecondType() { cout << "In MySecondType::MySecondType\n"; };
    ~MySecondType() { cout << "In MySecondType::~MySecondType\n"; };
};

MyFirstType MyGlobalVar;

void main(void)
{
    cout << "Erste Zeile von main\n";

    MySecondType MyLocalVar;

    cout << "main wird ausgefuehrt\n";
}
```

Die Ausgabe dieses Programms sieht folgendermaßen aus:

```
In MyFirstType::MyFirstType
Erste Zeile von main
In MySecondType::MySecondType
main wird ausgefuehrt
In MySecondType::~MySecondType
In MyFirstType::~MyFirstType
```

Die Standard-Laufzeitbibliothek

Anhang C

Die Standard-Laufzeitbibliothek ist eine Sammlung von Funktionen und Klassen, die für grundlegende Systemaufgaben verwendet werden.

Die Namen der Funktionen aus der Laufzeitbibliothek sind standardisiert. Alle Erweiterungen des Standards seitens Microsoft haben Namen, denen ein Unterstrich vorangestellt ist.

Die Funktionen der Standard-Laufzeitbibliothek befinden sich in der Datei *LIBC.LIB* (oder der Multithread-Version *LIBCMT.LIB*). Eine DLL-Version ist unter dem Namen *MSVCRT.LIB* verfügbar.

Dieser Anhang bietet eine Zusammenfassung der wichtigsten Kategorien der Laufzeitbibliothek.

C.1 Zugriff auf Argumente

Die Makros `va_arg`, `va_end` und `va_start` werden zum Implementieren von Funktionen mit variabler Anzahl von Argumenten verwendet. Diese Makros sind in *stdarg.h* definiert. Eine Version dieser Makros, die mit dem Makros für Unix System V kompatibel ist, kann in *vararg.h* gefunden werden.

C.2 Manipulieren von Puffern

Funktionen zum Manipulieren von Puffern wirken auf Speicherbereiche. Sie umfassen `memchr`, `memcmp`, `memcpy`, `memmove` und `memset`. Die Implementation von Microsoft enthält auch die Funktionen `_memcpy`, `_memicmp` und `_swab`. Diese Funktionen sind in *memory.h* oder *string.h* deklariert, mit Ausnahme von `_swab`, das in *stdlib.h* steht.

C.3 Klassifizieren der Bytes

Funktionen zum Klassifizieren der Bytes werden zur Bestimmung des Typs eines Byte- oder Multibyte-Zeichens verwendet. Sie umfassen die Funktion `isleadbyte` und eine Reihe von Microsoft-spezifischen Funktionen, deklariert in `ctype.h`.

C.4 Klassifizieren der Zeichen

Funktionen zum Klassifizieren der Zeichen werden verwendet, um den Typ eines Zeichens zu ermitteln. Folgende Standardfunktionen sind in `ctype.h` deklariert: `isalnum`, `isalpha`, `isctrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit` und `mblen`. Die meisten dieser Funktionen haben Unicode-Versionen, einige haben auch die Microsoft-spezifische Multibyte-Implementation.

C.5 Datenumwandlung

Diese Funktionen wandeln Daten aus einer Form in die andere um. Sie umfassen: `abs`, `atof`, `atoi`, `atol`, `labs`, `mbctowcs`, `mbtowc`, `strtod` und `wcs-
tod`, `strtol` und `wcstol`, `strtoul` und `wcstoul`, `strxfrm` und `wcsxfrm`, `tolower` und `towlower`, `toupper` und `towupper`, `wctombs` und `wctomb`. Die meisten sind in `stdlib.h` oder `math.h` deklariert. Es gibt auch einige Microsoft-spezifische Umwandlungsfunktionen.

C.6 Debug-Unterstützung

Die Unterstützung des Debugging-Vorganges geschieht mit einer Reihe von Microsoft-spezifischen Routinen. Diese umfassen `Debug`-Versionen der Speicherzuweisungsroutinen, Funktionen zur Prüfung der Integrität von zugewiesenen Speicherblöcken und Steuerung des Ablaufs.

C.7 Verzeichniskontrolle

Funktionen zur Verzeichniskontrolle werden verwendet, um den Verzeichnisbaum zu durchqueren, Verzeichnisse zu erstellen, zu löschen und zu manipulieren.

Da Verzeichnisse auf grund der verschiedenen Dateisysteme ein betriebssystemspezifisches Feature darstellen, sind diese Funktionen nicht

standardisiert. Funktionen wie `_chdir`, `_chdrive`, `_getcwd`, `_getdrive`, `_mkdir` und `_rmdir` sind Erweiterungen von Microsoft. Die Deklarationen dieser Funktionen können in der Headerdatei *direct.h* gefunden werden.

C.8 Ausnahmebehandlung

Trifft ein Programm auf eine nichtbehandelte Ausnahme, so wird die Funktion `terminate` aufgerufen. Diese kann durch eine kundenspezifische Version mit Hilfe von `set_terminate` ersetzt werden.

Microsoft C/C++ verwendet nicht die Funktion `unexpected`. Diese Funktion, deren Default auch `terminate` darstellt, kann ebenso durch `set_terminate` ersetzt werden.

Die Funktion `set_se_translator`, eine Microsoft-Erweiterung, kann zur Behandlung von Win32-Ausnahmen in C++-Programmen als C++-Ausnahmen eingesetzt werden.

Die Funktionen zur VC-Ausnahmebehandlung sind in *eh.h* deklariert.

C.9 Dateibehandlung

Die Routinen zur Dateibehandlung werden zum Umgang mit Dateien eines Dateisystems benötigt. Außer den Standardfunktionen `remove` und `rename`, die in *stdio.h* deklariert sind, definiert Microsoft einige zusätzliche Funktionen zur Dateibehandlung wie `_chsize` oder `_fstat`. Diese zusätzlichen Funktionen sind in *io.h* deklariert.

C.10 Unterstützung für Fließkommazahlen

Fließkommazahlen werden durch eine Reihe von Funktionen wie `abs`, `acos`, `asin`, `atan`, `atan2`, `atof`, `ceil`, `cos`, `cosh`, `difftime`, `div`, `exp`, `fabs`, `floor`, `fmod`, `frexp`, `labs`, `ldexp`, `ldiv`, `log`, `log10`, `modf`, `pow`, `rand`, `sin`, `sinh`, `sqrt`, `srand`, `strtod`, `tan` und `tanh` unterstützt. Sie sind in *math.h* deklariert.

Die E/A-Funktionen `printf` und `scanf` unterstützen ebenfalls das Fließkommaformat, um Fließkommazahlen als Text darzustellen oder umgekehrt.

Die Fließkommafunktionen verwenden per default 64-Bit-Genauigkeit. Dieses Verhalten kann durch Linken der Objektdatei *FP10.OBJ* mit Ihrem Programm geändert werden. Die Datei muß auf der Befehlszeile vor der Standard-Laufzeitbibliothek stehen.

C.11 Eingabe und Ausgabe

Eingabe und Ausgabe werden auf zahlreichen Ebenen unterstützt. Die Standard-Laufzeitbibliothek unterstützt Stream-E/A. Die E/A der niederen Ebene wird durch Microsoft-spezifische Funktionen erreicht. Zusätzliche Microsoft-spezifische Funktionen bieten Konsolen- und Port-E/A.

C++-Programme können auch die *iostream*-Bibliothek nutzen, die ein Teil der Standard-C++-Bibliothek ist.

C.11.1 Stream-E/A

Stream-E/A ist die am meisten genutzte Form der E/A in C-Programmen. Die E/A kann auf den vordefinierten Streams *stdin*, *stdout* und *stderr* stattfinden oder auf Streams, die Sie selber zu einer Datei oder einem Gerät öffnen. Die Bibliothek unterstützt folgende Funktionen: *clearerr*, *fclose*, *fEOF*, *ferror*, *fflush*, *fgetc*, *fgetpos*, *fgets*, *fopen*, *fprintf*, *fputc*, *fputs*, *fread*, *freopen*, *fscanf*, *fseek*, *fsetpos*, *ftell*, *fwrite*, *getc*, *getchar*, *gets*, *printf*, *putc*, *putchar*, *puts*, *rewind*, *scanf*, *setbuf*, *setvbuf*, *sprintf*, *sscanf*, *tmpfile*, *tmpnam*, *ungetc*, *vfprintf*, *vprintf* und *vsprintf*. Einige dieser Funktionen haben auch Unicode-Äquivalente. Es gibt auch Microsoft-spezifische Erweiterungen, die betriebssystemspezifische Funktionen bieten. Stream-E/A-Funktionen sind in *stdio.h* deklariert.

C.11.2 Konsolen-E/A und Low-Level-E/A

E/A-Funktionen bieten einen direkteren Zugang zu den E/A-Fähigkeiten des Betriebssystems – zumindest theoretisch. Auf Win32-Systemen sind die »wahren« E/A-Funktionen der niederen Ebene *CreateFile*, *ReadFile* und *WriteFile*. Die Funktionen *open*, *_read* und *_write* werden nur aus Kompatibilitätsgründen mitgeliefert und sollten in neuen Entwicklungen nicht eingesetzt werden.

C.12 Internationalisierung

Windows NT und Windows 95/98 erkennen Zeichen in drei verschiedenen Zeichensätzen – nun, eigentlich zwei, da der dritte eine Unter-
menge der anderen ist.

Der wohlbekannte ASCII-Zeichensatz mit seiner Erweiterung auf bis zu 128 internationalen Zeichen bildet einen Einzelbyte-Zeichensatz (SBCS). Darin werden den Bytes im Speicher eins zu eins die anzuzeigenden Zeichen zugeordnet.

Der ASCII-SBCS ist ein Multibyte-Zeichensatz (MBCS). MBCS verwenden Zeichen mit variabler Länge. Wenn ein Byte in einem bestimmten Bereich liegt, wird angenommen, es sei ein führendes Byte, das noch ein Folgebyte hat; zusammen definieren sie ein Zeichen. Die aktuellen Bereiche für den Zeichensatz und das führende Byte werden von der lokalen Einstellung bestimmt. Diese wird mit der Funktion `setlocale` vorgenommen.

Unicode bietet eine andere Lösung, mit Zeichen, die immer zwei Bytes lang sind. Während Strings, die MCBS verwenden, von den lokalen Einstellungen abhängen, ist das mit Unicode nicht der Fall. Unicode kann gleichzeitig Zeichen aus verschiedenen Zeichensätzen darstellen.

C.13 Speicherzuweisung

Funktionen zur Speicherzuweisung verwalten den Speicher dynamisch. `free`, `malloc` und `realloc`, sind in `stdlib.h` deklariert. Zudem gibt es ein paar Microsoft-spezifische Funktionen, die vor allem beim Debuggen Verwendung finden.

C.14 Steuerung der Prozesse und der Umgebung

Die Funktionen zur Prozeßkontrolle steuern das Ausführen und Beenden eines Prozesses, behandeln Signale und stoßen andere Prozesse an. Diese Funktionen umfassen `abort`, `assert`, `atexit`, `exit`, `perror`, `raise`, `setjmp`, `signal` und `system`. Davon ist `assert` in `assert.h` deklariert, `setjmp` in `setjmp.h`, `raise` und `signal` in `signal.h` und der Rest in `process.h` oder `stdlib.h`.

Die Funktion `getenv` wird verwendet, um den Wert einer Umgebungsvariablen abzufragen. Es gibt keine Standardfunktion zum Setzen eines Wertes, die Funktion `_putenv` ist Microsoft-spezifisch. Diese Funktionen sind in *stdlib.h* deklariert.

C.15 Suchen und Sortieren

Die Funktion `bsearch` implementiert einen binären Suchalgorithmus, `qsort` den Quicksort-Algorithmus. Beide Funktionen verwenden eine benutzerdefinierte Vergleichsfunktion. Sie sind in *search.h* deklariert. Microsoft bietet auch zwei lineare Suchfunktionen, `lfind` und `lsearch`.

C.16 Strings manipulieren

Funktionen zum Manipulieren von Strings werden zum Kopieren, Vergleichen oder Ändern nullterminierter Strings eingesetzt. Sie umfassen `strcat`, `strchr`, `strcmp`, `strcoll`, `strcpy`, `strcspn`, `strerror`, `strftime`, `strlen`, `strncat`, `strncmp`, `strncpy`, `strpbrk`, `strrchr`, `strspn`, `strstr`, `strtok` und `strxfrm`. Deklariert sind sie in *string.h*.

C.17 Systemaufrufe

Unter MS-DOS umfaßte diese Kategorie eine Reihe von Funktionen zum Zugang auf niederer Ebene zu Systemfunktionen von MS-DOS und BIOS. Wegen der Unabhängigkeit vom Betriebssystem des Win32-Programmiersmodells sind die meisten dieser Funktionen verschwunden. Was übrigbleibt, sind drei Funktionen (und ihre Unicode- und 64-Bit-Äquivalente) zum Iterieren durch Dateien, die einem angegebenen Dateinamen entsprechen: `_findfirst`, `_findnext` und `_findclose`, deklariert in *io.h*.

C.18 Zeitverwaltung

Funktionen zur Zeitverwaltung werden eingesetzt, um die Systemzeit zu setzen oder abzufragen, den Wert in ein anderes Format umzuwandeln und die Zeitzonen zu verwalten. Die in *time.h* deklarierten Funktionen umfassen `asctime`, `clock`, `ctime`, `difftime`, `gmtime`, `localtime`, `mktime`, `strftime` und `time`.

Die Zeit in Sekunden wird vom Typ `time_t` dargestellt, der als vorzeichenbehafteter 4-Byte-Langwert definiert ist. Der Wert stellt die An-

zahl der vergangenen Sekunden seit Mitternacht des 1. Januars 1970. Der größte Wert bei diesem Typ ist 2.147.483.647, was einer Zeit von 68 Jahren, 18 Tagen, 3 Stunden, 14 Minuten und 7 Sekunden entspricht. Die Frage ist, wie viele Maschinen mit einem Win32-Betriebssystem am Morgen des 19. Januars 2038 noch in Betrieb sein werden.

C.19 Die ANSI-C-Laufzeitbibliothek

Im folgenden werden die Funktionen der ANSI-C_Bibliothek nach Header-Dateien geordnet vorgestellt, wobei die Namen der C-Konvention verwendet wurden.

Nach dem neuen C++-Standard wird den Dateinamen der C-Header-Dateien jeweils ein »c« vorangestellt, und die Extension ».h« fällt weg (`<stdlib.h>` würde also beispielsweise zu `<cstdlib>`). *C versus C++*

`<assert.h>`

<code>assert</code>	Prüft eine Bedingung und bricht bei Nichterfüllung das Programm ab.

`<ctype.h>`

Enthält die Funktionen zur Klassifizierung und Konvertierung einfacher Zeichen.

<code>isalnum</code>	Testet, ob Zeichen eine Ziffer oder ein Buchstabe ist.
<code>isalpha</code>	Testet, ob Zeichen ein Buchstabe ist.
<code>iscntrl</code>	Testet, ob Zeichen ein Kontrollzeichen ist.
<code>isdigit</code>	Testet, ob Zeichen eine Zahl ist.
<code>isgraph</code>	Testet, ob Zeichen druckbar ist (ohne Leerzeichen)
<code>islower</code>	Testet, ob Zeichen ein Kleinbuchstabe ist.
<code>isprint</code>	Testet, ob Zeichen druckbar ist (mit Leerzeichen)
<code>ispunct</code>	Testet, ob Zeichen ein Interpunktionszeichen ist.
<code>isspace</code>	Testet, ob Zeichen ein Zwischenraumzeichen (White-space) ist.
<code>isupper</code>	Testet, ob Zeichen ein Großbuchstabe ist.
<code>isxdigit</code>	Testet, ob Zeichen eine Hexadezimalziffer ist.

<code>tolower</code>	Wandelt Groß- in Kleinbuchstaben um.
<code>toupper</code>	Wandelt Klein- in Großbuchstaben um.

<errno.h>

Enthält verschiedene Makros, die zur Benachrichtigung über Fehler verwendet werden.

<code>EDOM</code>	Wird im Falle eines ungültigen Funktionsarguments von den mathematischen Funktionen in <i>errno</i> kopiert.
<code>ERANGE</code>	Wird im Falle eines ungültigen Ergebniswertes von den mathematischen Funktionen in <i>errno</i> kopiert.
<code>errno</code>	Wird im Fehlerfall von den mathematischen Funktionen gesetzt.

<float.h>

Enthält verschiedene Makros, die zu verschiedenen implementierungsspezifischen Konstanten expandieren. Aufgeführt sind nur die Konstanten für den Datentyp `float`. Die Konstanten für `double` und `long double` beginnen mit den Suffixen `DBL` beziehungsweise `LDBL`. (`FLT_ROUNDS` gilt für alle drei Fließkommatypen).

<code>FLT_DIG</code>	Anzahl der signifikanten Ziffern in der Mantisse, wenn dargestellt zur Basis 10
<code>FLT_EPSILON</code>	Differenz zwischen 1 und der nächstgrößeren darstellbaren Zahl.
<code>FLT_MANT_DIG</code>	Anzahl der Ziffern in der Mantisse, wenn die Basis <code>FLT_RADIX</code> zur Darstellung verwendet wird
<code>FLT_MAX</code>	Größte positive Zahl.
<code>FLT_MAX_10_EXP</code>	Größte positive Zahl für den Exponenten zur Basis 10.
<code>FLT_MAX_EXP</code>	Größte positive Zahl für den Exponenten zur Basis <code>FLT_RADIX</code>
<code>FLT_MIN</code>	Kleinste positive Zahl (Wie nah kommt man an 0 heran?)
<code>FLT_MIN_10_EXP</code>	Kleinste negative Zahl für den Exponenten zur Basis 10.
<code>FLT_MIN_EXP</code>	Kleinste negative Zahl für den Exponenten zur Basis <code>FLT_RADIX</code>

FLT_RADIX	Basis des Exponenten
FLT_ROUNDS	Modus, nach dem Zahlen gerundet werden:
	-1 unbestimmt
	0 zu Null
	1 zum nächsten Wert
	2 zu +unendlich
	3 zu -unendlich
	Gilt auch für double- und long-double-Zahlen.

<iso646.h>

Synonyme für bestimmte Operatoren.

and	&&
and_eq	&=
bitand	&
bitor	
compl	~
not	!
not_eq	!=
or	
or_eq	=
xor	^
xor_eq	^=

<limits.h>

Enthält Konstanten, die Größe und Wertebereiche der elementaren Datentypen festlegen (die Werte sind implementierungsspezifisch und werden vom Compiler festgelegt).

CHAR_BIT	Anzahl der Bits in kleinstem Datentyp
CHAR_MAX	kleinster Wert für Objekte vom Typ char
CHAR_MIN	größter Wert für Objekte vom Typ char
INT_MAX	kleinster Wert für Objekte vom Typ int
INT_MIN	größter Wert für Objekte vom Typ int
LONG_MAX	kleinster Wert für Objekte vom Typ long int
LONG_MIN	größter Wert für Objekte vom Typ long int
MB_LEN_MAX	maximale Anzahl von Bytes in Multibyte-Zeichen

SCHAR_MAX	kleinster Wert für Objekte vom Typ signed char
SCHAR_MIN	größter Wert für Objekte vom Typ signed char
SHRT_MAX	kleinster Wert für Objekte vom Typ short int
SHRT_MIN	größter Wert für Objekte vom Typ short int
UCHAR_MAX	größter Wert für Objekte vom Typ unsigned char
UINT_MAX	größter Wert für Objekte vom Typ unsigned int
ULONG_MAX	größter Wert für Objekte vom Typ unsigned long int
USHRT_MAX	größter Wert für Objekte vom Typ unsigned short int

<locale.h>

Enthält Funktionen und Makros, um landesspezifische Eigenheiten festzulegen und definiert dazu die Struktur `lconv`.

localeconv	Die Struktur <code>lconv</code> initialisieren.
setlocale	Landesspezifische Werte abfragen und verändern.

<math.h>

Deklariert die mathematischen Funktionen. Während unter C nur die `double`-Versionen der Funktionen verfügbar sind, sind sie unter C++ zusätzlich für die Datentypen `float` und `long double` überladen.

acos	Arkuskosinus berechnen.
asin	Arkussinus berechnen.
atan	Arkustangens berechnen.
atan2	Arkustangens berechnen.
ceil	Rundet auf nächste Ganzzahl auf.
cos	Kosinus berechnen.
cosh	Kosinus hyperbolicus berechnen.
exp	Exponentialfunktion berechnen.
fabs	Ermittelt Absolutwert.
floor	Größtmögliche Ganzzahl ermitteln, die kleiner oder gleich dem Argument ist.
fmod	Rest einer Fließkommazahl berechnen.
frexp	Exponentialwert berechnen.
ldexp	Fließkommazahl multiplizieren.
log	Natürlichen Logarithmus berechnen.

log10	Logarithmus zur Basis 10 berechnen.
modf	Argument in ganzzahligen und gebrochenen Teil zergliedern.
pow	Zahl potenzieren.
sin	Sinus berechnen.
sinh	Sinus hyperbolicus berechnen.
sqrt	Quadratwurzel berechnen.
tan	Tangens berechnen.
tanh	Tangens hyperbolicus berechnen.

<setjmp.h>

Definiert den Datentyp `jmp_buf`, der von dem Makro `setjmp()` und der Funktion `longjmp()` benutzt wird. Mit Hilfe von `setjmp()` und `longjmp()` kann man Sprünge zwischen Funktionen ausführen (im Gegensatz zu `goto`, mit dem man nur innerhalb von Funktionen springen kann).

longjmp	Umgebung wiederherstellen nach nicht lokalem Sprung.
setjmp	Umgebung sichern. Umgebung kann mittels <code>longjmp()</code> wiederhergestellt werden.

<signal.h>

Definiert Konstanten und Funktionen, die für die Behandlung von Sonderbedingungen (Signalbearbeitung) erforderlich sind.

raise	Sendet ein Signal an das Programm.
signal	Legt die Verarbeitung eines Signals fest.

<stdarg.h>

Definiert die Makros, mit denen Funktionen mit einer variablen Anzahl von Argumenten verarbeitet werden können.

va_list	<code>va_list</code> ist der Array-Typ, der die von <code>va_arg</code> und <code>va_end</code> benötigten Informationen speichert. Wenn eine aufgerufene Funktion eine variable Argumentliste übernimmt, deklariert sie eine Variable <code>ap</code> vom Typ <code>va_list</code>

va_start	va_start ist ein Makro, das ap auf den ersten variablen Parameter setzt, der der Funktion übergeben wird. va_start muß vor va_arg und va_end aufgerufen werden.
va_arg	Das Makro va_arg erwartet einen Zeiger ap auf einen der variablen Parameter und den Datentyp des Parameters. Der erste Aufruf von va_arg liefert den Wert des ersten variablen Parameters zurück und erhöht ap entsprechend, so daß ap danach auf den nächsten Parameter zeigt. Weitere Aufrufe von va_arg – wieder mit ap – liefern die jeweils nachfolgenden Parameter zurück.
va_end	Das Makro va_end hilft der aufgerufenen Funktion, einen normalen Rücksprung durchzuführen. va_end sollte aufgerufen werden, nachdem va_arg alle Argumente abgearbeitet hat.

<stddef.h>

Definiert die folgenden Typen und Makros

NULL	Wert des Null-Zeigers.
offsetof	Ermittelt den Offset eines Elements in einer Struktur.
ptrdiff_t	Ganzzahliger, vorzeichenbehafteter Datentyp, der das Ergebnis der Subtraktion zweier Zeiger repräsentieren kann.
size_t	Vorzeichenloser Ganzzahltyp, der von sizeof als Ergebnistyp verwendet wird.
wchar_t	Zeichentyp für umfangreichere Zeichensätze.

<stdio.h>

Enthält die Funktionen zur Ein- und Ausgabe, sowie zur Arbeit mit Dateien.

clearerr	Fehlerbedingung aus Datenstrom entfernen.
fclose	Datenstrom schließen.
feof	Testet Datenstrom auf Dateiende.
ferror	Testet Datenstrom auf Fehler.
fflush	Leert Puffer eines Datenstroms.
fgetc	Liest Zeichen von einem Datenstrom.
fgetpos	Ermittelt Positionszeiger eines Datenstroms.

fgetc	Liest eine Zeile von einem Datenstrom.
fopen	Öffnet einen Datenstrom.
fprintf	Formatierte Ausgabe an einen Datenstrom.
fputc	Schreibt ein Zeichen an Datenstrom.
fputs	Schreibt einen String an Datenstrom.
fread	Liest Daten von Datenstrom.
freopen	Öffnet einen Datenstrom erneut.
fscanf	Liest formatiert von Datenstrom.
fseek	Positioniert Dateizeiger neu.
fsetpos	Positioniert Dateizeiger neu.
ftell	Liest Position des Dateizeigers.
fwrite	Schreibt Daten unformatiert an Datenstrom.
getc	Liest Zeichen aus Datenstrom.
getchar	Liest Zeichen aus Datenstrom stdin.
gets	Liest String von stdin.
perror	Schreibt Fehlermeldung an stdout.
printf	Formatierte Ausgabe an stdout.
putc	Schreibt Zeichen in Datenstrom.
putchar	Schreibt Zeichen an Datenstrom.
puts	Schreibt Zeichenkette an Datenstrom stdout.
remove	Löscht eine Datei.
rename	Benennt eine Datei um.
rewind	Positioniert Dateizeiger auf Anfang des Datenstroms.
scanf	Formatierte Eingabe von stdin.
setbuf	Datenstrompufferung verändern.
setvbuf	Datenstrompufferung und Puffergröße verändern.
sprintf	Formatierte Ausgabe an einen String.
sscanf	Formatierte Eingabe von einem String.
tmpfile	Temporäre Datei erzeugen.
tmpnam	Temporären Dateinamen erzeugen.
ungetc	Zeichen wieder zurück an Datenstrom schicken.
vfprintf	Formatierte Ausgabe an Datenstrom.
vprintf	Formatierte Ausgabe an stdout.
vsprintf	Formatierte Ausgabe in String.

<stdlib.h>

Enthält die Standard-Bibliotheksfunktionen, zu denen u.a. Routinen zur dynamischen Speicherverwaltung und zur Prozeßkontrolle gehören.

abort	Beendet Programm unverzüglich.
abs	Ermittelt Absolutwert einer Integerzahl (In C++ für long int, float und long double überladen).
atexit	Funktion registrieren, die beim Programmende ausgeführt werden soll.
atof	Wandelt String in Fließkommazahl um.
atoi	Wandelt String in Integerzahl um.
atol	Wandelt String in long int um.
bsearch	Binäre Suche durchführen.
calloc	Speicher allokieren und initialisieren.
div	Ganzzahldivision durchführen (In C++ für long überladen).
exit	Programm beenden.
free	Dynamisch angeforderten Speicher freigeben.
getenv	Eintrag in Programmumgebung lesen.
labs	Absolutwert einer Zahl vom Typ long int ermitteln.
ldiv	Ganzzahldivision mit Zahlen vom Typ long int (In C++ für long überladen).
malloc	Speicher anfordern.
mblen	Länge eines Multibyte-Zeichens ermitteln.
mbstowcs	Zeichenkette mit Zeichen, die aus mehreren Bytes bestehen, in Wide Character-String umwandeln.
mbtowc	Zeichen, das aus mehreren Bytes besteht, in Wide Character umwandeln
qsort	QuickSort durchführen.
rand	Pseudo-Zufallszahl erzeugen.
realloc	Speicherblock neu zuordnen.
srand	Pseudo-Zufallszahl erzeugen.
strtod	String in Fließkommazahl umwandeln.
strtol	String in long int umwandeln.
strtoul	String in unsigned long umwandeln.
system	Programm unterbrechen und anderes ausführen.

wcstombs	Zeichenkette mit Wide Characters in Multibyte-String umwandeln.
wctomb	Wide Character in Multibyte-Zeichen umwandeln.

<string.h>

Enthält Funktionen zur Manipulation von Zeichenketten.

memchr	Sucht in einem Speicherbereich nach erstem Vorkommen eines Zeichens.
memcmp	Vergleicht zwei Speicherbereiche.
memcpy	Kopiert Anzahl Zeichen von einem Speicherbereich in einen anderen.
memmove	Kopiert Anzahl Zeichen von einem Speicherbereich in einen anderen, der sich mit dem Ursprungsbereich überlappen darf.
memset	Initialisiert einen Speicherbereich mit einem Zeichen.
strcat	Fügt Zeichenkette an eine andere an.
strchr	Ermittelt Zeiger auf erstes Vorkommen eines Zeichens in String.
strcmp	Vergleicht zwei Zeichenketten.
strcoll	Vergleicht zwei Zeichenketten.
strcpy	Kopiert einen String in einen anderen.
strcspn	Sucht in Zeichenkette nach Zeichen, die in anderer Zeichenkette nicht enthalten sind.
strerror	Übersetzt eine Fehlernummer in eine Zeichenkette.
strlen	Ermittelt die Länge einer Zeichenkette.
strncat	Fügt n Zeichen einer Zeichenkette an andere an.
strncmp	Vergleicht einen String mit einem Teil eines anderen Strings.
strncpy	Kopiert n Zeichen eines Strings in einen anderen.
strpbrk	Sucht nach erstem Vorkommen eines beliebigen Zeichens aus einer Zeichenkette in einer anderen Zeichenkette.
strrchr	Ermittelt Zeiger auf letztes Vorkommen eines Zeichens in einer Zeichenkette.
strspn	Ermittelt Index des ersten Zeichens in einer Zeichenkette, das nicht zum Zeichensatz einer anderen Zeichenkette gehört.

strstr	Ermittelt erstes Vorkommen einer Zeichenkette in einer anderen.
strtok	Sucht nach nächstem Grundsymbol in Zeichenkette.
strxfrm	Wandelt String unter Berücksichtigung landesspezifischer Gegebenheiten um.

<time.h>

Enthält die Datentypen `time_t` und `clock_t` sowie die Datenstruktur `tm` und die allgemeinen Zeitfunktionen.

asctime	Wandelt eine als Struktur gespeicherte Zeit in eine Zeichenkette um.
clock	Ermittelt, wieviel Ticks seit dem Start des aufrufenden Prozesses verbraucht wurden.
ctime	Wandelt Zahl vom Typ <code>time_t</code> in Zeichenkette um.
difftime	Ermittelt die Differenz zwischen zwei Zeiten.
gmtime	Berechnet aus lokaler Zeit und definierter Zeitdifferenz die Greenwich Mean Time (GMT).
localtime	Wandelt einen Zeitwert vom Typ <code>time_t</code> in die Lokalzeit um.
mktime	Wandelt Zeit aus Struktur in Kalenderzeit um.
strftime	Formatiert Zeit nach landesspezifischen Bedingungen.
time	Gibt aktuelle Zeit an Variable vom Typ <code>time_t</code> zurück.

<wchar.h>

Enthält die Funktionen zur Bearbeitung von Wide Character-Strings (inklusive Ein- und Ausgabe).

btowc	Wandelt ein 1-MultiByte-Zeichen in ein Wide Character-Zeichen um.
fgetwc	Liest Zeichen von einem Datenstrom.
fgetws	Liest eine Zeile von einem Datenstrom.
fputwc	Schreibt ein Zeichen in Datenstrom.
fputws	Schreibt einen String in Datenstrom.
fwprintf	Formatierte Ausgabe an stdout.
fwscanf	Liest formatiert von Datenstrom.
getwc	Liest Zeichen aus Datenstrom.
getwchar	Liest Zeichen aus Datenstrom stdin.

mbrlen	Länge eines Multibyte-Zeichens ermitteln.
mbrtowc	Zeichen, das aus mehreren Bytes besteht, in Wide Character umwandeln.
mbsinit	Ermittelt, ob sein Argument einen anfänglichen Konvertierungsmodus anzeigt.
mbsrtowcs	MultiByte-String in Wide Character-String umwandeln.
putwc	Schreibt Zeichen in Datenstrom.
putwchar	Schreibt Zeichen an Datenstrom stdout.
swprintf	Formatierte Ausgabe in einen String.
swscanf	Formatierte Eingabe von einem String.
ungetwc	Zeichen wieder zurück an Datenstrom schicken.
vfwprintf	Formatierte Ausgabe an Datenstrom.
vswprintf	Formatierte Ausgabe in String.
vwprintf	Formatierte Ausgabe an stdout.
wcrtomb	Wide Character in Multibyte-Zeichen umwandeln.
wcscat	Fügt Zeichenkette an eine andere an.
wcschr	Ermittelt Zeiger auf erstes Vorkommen eines Zeichens in String.
wcscmp	Vergleicht zwei Zeichenketten.
wscoll	Vergleicht zwei Zeichenketten.
wscpy	Kopiert einen String in einen anderen.
wscspn	Sucht in Zeichenkette nach Zeichen, die in anderer Zeichenkette nicht enthalten sind.
wcsftime	Formatiert Zeit nach landesspezifischen Bedingungen.
wcslen	Ermittelt die Länge einer Zeichenkette.
wcsncat	Fügt n Zeichen einer Zeichenkette an andere an.
wcsncmp	Vergleicht einen String mit einem Teil eines anderen Strings.
wcsncpy	Kopiert n Zeichen eines Strings in einen anderen.
wcspbrk	Sucht nach erstem Vorkommen eines beliebigen Zeichens aus einer Zeichenkette in einer anderen Zeichenkette.
wcsrchr	Ermittelt Zeiger auf letztes Vorkommen eines Zeichens in einer Zeichenkette.
wcrtombs	Zeichenkette mit Wide Characters in Multibyte-String umwandeln.

wcsspn	Ermittelt Index des ersten Zeichens in einer Zeichenkette, das nicht zum Zeichensatz einer anderen Zeichenkette gehört.
wcsstr	Ermittelt erstes Vorkommen einer Zeichenkette in einer anderen.
wctod	String in Fließkommazahl umwandeln.
wcstok	Sucht nach nächstem Grundsymbol in Zeichenkette.
wcstol	String in long int umwandeln.
wcstoul	String in unsigned long int umwandeln.
wcsxfrm	Wandelt String unter Berücksichtigung landesspezifischer Gegebenheiten um.
wctob	Wandelt Zeichen in ein 1-Multibyte-Zeichen um.
wmemchr	Sucht in einem Speicherbereich nach erstem Vorkommen eines Zeichens.
wmemcmp	Vergleicht zwei Speicherbereiche.
wmemcpy	Kopiert Anzahl Zeichen von einem Speicherbereich in einen anderen.
wmemmove	Kopiert Anzahl Zeichen von einem Speicherbereich in einen anderen, der sich mit dem Ursprungsbereich überlappen darf.
wmemset	Initialisiert einen Speicherbereich mit einem Zeichen.
wprintf	Formatierte Ausgabe an stdout.
wscanf	Formatierte Eingabe von stdin.

<wctype.h>

Enthält die Funktionen zur Klassifizierung und Konvertierung von Wide-Character-Zeichen, sowie die Hilfstypen: `wint_t`, `wctrans_t` und `wctype_t` plus dem Makro `WEOF`.

iswalnum	Testet, ob Zeichen eine Ziffer oder ein Buchstabe ist.
iswalpha	Testet, ob Zeichen ein Buchstabe ist.
iswcntrl	Testet, ob Zeichen ein Kontrollzeichen ist.
iswctype	Testet, ob Zeichen einer bestimmten Klasse angehört. Die Klasse wird durch einen Aufruf von <code>wctype</code> spezifiziert: <code>iswctype(wc, wctype(>digit<)) // = iswcdigit(wc)</code>
iswdigit	Testet, ob Zeichen eine Zahl ist.

iswgraph	Testet, ob Zeichen druckbar ist (Leerzeichen ausgenommen).
iswlower	Testet, ob Zeichen ein Kleinbuchstabe ist.
iswprint	Testet, ob Zeichen druckbar ist.
iswpunct	Testet, ob Zeichen ein Interpunktionszeichen ist.
iswspace	Testet, ob Zeichen ein Zwischenraumzeichen ist.
iswupper	Testet, ob Zeichen ein Großbuchstabe ist.
iswxdigit	Testet, ob Zeichen eine Hexadezimalziffer ist.
towctrans	Konvertiert ein Wide Character-Zeichen.
towlower	Wandelt Groß- in Kleinbuchstaben um.
towupper	Wandelt Klein- in Großbuchstaben um.
wctrans	Ermittelt eine Abbildung von einem Wide Character-Zeichensatz in einen anderen (nach LC_TYPE)
wctype	Wird zusammen mit iswctype() verwendet und erlaubt die Identifizierung einer Klasse anhand eines Strings.

Die Standard-C++-Bibliothek

Anhang D

Die Standard-C++-Bibliothek enthält eine große Anzahl von Funktionen, Funktionstemplates, Klassen und Klassentemplates, die in C++-Programmen nützlich sind. Viele davon überlagern ähnliche Komponenten aus der Standard-Laufzeitbibliothek, andere ersetzen oder treten in Konkurrenz zu compilerspezifischen Implementierungen (beispielsweise die Stream-Klassen), wieder andere sind gänzlich neu (beispielsweise die STL, Standard Template Library).

Der empfohlene Weg, Unterstützung für bestimmte Funktionen der Standard-C++-Bibliothek in Ihre Anwendungen einzufügen, ist die Verwendung der Headerdateien in neuem Stil (ohne Erweiterung). Zum Beispiel, wenn Sie die `iostream`-Bibliothek einsetzen wollen, fügen Sie folgende Zeile ein:

```
#include <iostream>
```

Diese neuen Headerdateien funktionieren mit Namensräumen und stellen die Standard-Deklarationen in den Namensraum `std`.

D.1 Die C++-Laufzeitbibliothek

Die C++-Laufzeitbibliothek umfaßt

- Die STL (Standard Template Library) – eine Sammlung von Klassen und Funktionen zur Datenverwaltung. Die STL selbst besteht aus:
 - Containern. Höhere Datenstrukturen, die der Verwaltung und Speicherung beliebiger Daten dienen und jede eine eigene Schnittstelle für den Zugriff auf die Daten definieren. Unter

den Container finden Sie Datenstrukturen wie Vektoren (entsprechen Arrays), Warteschlangen, Mengen oder Stacks.

- Hinzu kommen die Iteratoren. Iteratoren sind den Zeigern verwandt und dienen dem Zugriff auf die Elemente, die in Containern abgespeichert wurden.
- Drittens stehen eine Reihe von Algorithmen (Funktionen) zur Verfügung, die mit Hilfe von Iteratoren auf den Elementen der verschiedenen Container operieren können. Sie stellen also eine allgemeine Erweiterung der Funktionalität der Container-Klassen dar.
- Schließlich gibt es noch eine Reihe von Hilfsklassen und -funktionen, wie Funktionsobjekte, Adapter, etc.

- ➡ Die Streamklassen für die Ein- und Ausgabe
- ➡ Die String-Klassen für die Arbeit mit Zeichenketten
- ➡ Die Klassen zur Internationalisierung
- ➡ Einer Reihe weiterer nützlicher Klassen (von `complex` bis `auto_ptr`).

D.1.1 Die Container-Klassen

`<bitset>`

```
TEMPLATE <SIZE_T N> BITSET
```

Die Klasse `bitset` dient der Verwaltung von Bit-Werten, also Folgen von Nullen und Einsen. Instanzen der Klasse `bitset` können zur übersichtlichen Verwaltung von Diagnose- und Schaltervariablen (Flags) herangezogen werden (definiert in `<bitset>`).

- ➡ Die Anzahl von Bits im Bitset wird durch den Platzhalter `N` bei der Instanziierung des Templates festgelegt und kann später nicht mehr geändert werden.
- ➡ Die Klasse verfügt über einen Konstruktor, dem ein String aus Nullen und Einsen übergeben werden kann. Das Bitset enthält dann so viele Elemente, wie sich Zeichen in dem String befinden und wird mit den Werten aus dem String initialisiert.
- ➡ Die Methoden `to_ulong()` und `to_string()` wandeln ein Bitset in einen Integer-Wert oder ein String-Objekt um.
- ➡ Die Methoden `set()`, `reset()` und `flip()` können zum Setzen einzelner Bits verwendet werden.

- ➡ Zur Abfrage einzelner Bits dienen die Methoden `test()`, `any()` und `none()`.
- ➡ Zur Manipulation einzelner Bits ist zudem der Index-Operator `[]` überladen.
- ➡ Für Mengenoperationen stehen die üblichen Bit-Operatoren zur Verfügung.

`<deque>`

```
TEMPLATE <CLASS T, CLASS ALLOCATOR = ALLOCATOR> CLASS DEQUE
```

Die Abkürzung `deque` steht für »double ended queue«, also eine Warteschlange, die Operationen auf beiden Enden erlaubt. Im Grunde ist `deque` ein dynamisches Array, das für Pop- und Push-Zugriffe am Anfang und Ende der Schlange optimiert ist, aber auch indizierte Zugriffe auf beliebige Positionen erlaubt (`deque` stellt damit gewissermaßen eine Kombination aus `vector` und `list` dar).

Der Klasse `deque` sollte stets dann der Vorzug gegeben werden, wenn Sie auf alle Elemente im Container direkten Zugriff haben möchten, der überwiegende Teil der Zugriffe aber auf die Containerelemente am Containeranfang oder -ende erfolgt. Ein weiterer Grund wäre die flexiblere Speicherverwaltung, da eine `deque` nicht wie ein `vector` einen einzigen Speicherblock allokiert, sondern eine Reihe miteinander verketteter kleinerer Speicherblöcke. Es ist allerdings zu beachten, daß Einfüge- und Löschoptionen dazu führen können, daß bestehende Iteratoren und Referenzen auf Container-Elemente ungültig werden. (definiert in `<deque>`)

- ➡ Die überladenen Formen der Konstruktoren erlauben die Einrichtung leerer Container oder Container, die mehrere Kopien eines Elementes beziehungsweise die Kopie einer Sequenz von Elementen aus einem anderen Container enthalten.
- ➡ Die Größe des Containers kann nach seiner Einrichtung durch Löschen und Einfügen von Elementen verändert werden, wobei die Klasse `deque` eine Reihe von Methoden speziell für Operationen auf den Enden des Containers zur Verfügung stellt.
- ➡ Neue Elemente können mittels `insert()`, `push_front()`, `push_back()`, `resize()` oder `assign()` eingefügt werden, wobei letztere Methode den Container zuvor leert.
- ➡ Bestehende Elemente können mittels `erase()`, `pop_front()`, `pop_back()`, oder `resize()` gelöscht werden.

- ➡ Zugriff auf einzelne Elemente oder Elementfolgen kann mit Hilfe zurückgelieferter Iteratoren geschehen (`begin()`, `end()`, `rbegin()`, `rend()` sowie `insert()`) oder über zurückgegebene Referenzen (Operator `[]`, `at()`, `front()`, `back()`).

<list>

```
TEMPLATE <CLASS T, CLASS ALLOCATOR = ALLOCATOR > CLASS LIST
```

Sequentielle Datenstruktur, die Zugriffe auf beliebige Positionen zuläßt. Der Zugriff ist stets effizient, allerdings muß auf indizierten Zugriff verzichtet werden.

Die Klasse `list` implementiert wie `vector` und `deque` einen sequentiellen Container. Der Klasse `list` sollte stets dann der Vorzug gegeben werden, wenn die Größe des Containers schwer vorherzusehen ist und unter Umständen zur Laufzeit stark schwankt. Von Vorteil ist auch, daß Einfüge- und Löschoperationen nicht die Gültigkeit bestehender Iteratoren und Referenzen auf die Container-Elemente beeinträchtigen (definiert in <list>).

- ➡ Die überladenen Formen der Konstruktoren erlauben die Einrichtung leerer Container oder Container, die mehrere Kopien eines Elementes beziehungsweise die Kopie einer Sequenz von Elementen aus einem anderen Container enthalten.
- ➡ Die Größe des Containers kann nach seiner Einrichtung durch Löschen und Einfügen von Elementen verändert werden, wobei die Klasse `list` eine Reihe von Methoden speziell für schnelle Operationen im Innern des Containers zur Verfügung stellt.
- ➡ Neue Elemente können mittels `insert()`, `push_front()`, `push_back()`, `resize()`, `merge()` oder `splice()` eingefügt werden.
- ➡ Bestehende Elemente können mittels `erase()`, `pop_front()`, `pop_back()`, `resize()`, `unique()` oder `remove()`, `remove_if()` gelöscht werden (wobei Sie beachten sollten, daß der Speicherplatz für Zeigerobjekte von Ihnen selbst wieder freigegeben werden muß).
- ➡ Der Zugriff auf einzelne Elemente kann mit Hilfe zurückgelieferter Iteratoren geschehen (`begin()`, `end()`, `rbegin()`, `rend()` sowie `insert()`) oder über zurückgegebene Referenzen (`front()`, `back()`).
- ➡ Mittels der Methode `sort()` können die Elemente sortiert werden, wobei die Art der Sortierung durch den <-Operator oder eine selbstdefinierte Vergleichsfunktion bestimmt werden kann. Durch Aufruf der Methode `reverse()` kann die Reihenfolge umgekehrt werden.

<map>

```

TEMPLATE <CLASS KEY, CLASS T,
          CLASS COMPARE = LESS<T>
          CLASS ALLOCATOR = ALLOCATOR> CLASS MAP

```

Datenstruktur, die mit Hilfe von eindeutigen Schlüsseln auf ihre Elemente zugreift (ideales Hashing). Die Idee dabei ist, daß den zu verwaltenden Objekten sogenannte Schlüssel beigeordnet sind (man kennt dies beispielsweise von Arrays, bei denen die einzelnen Elemente mit Indizes verbunden sind; Schlüssel können aber im Gegensatz zu Indizes jedem geordneten Datentyp angehören). Ähnlich den Indizes eines Arrays soll aus den Schlüsseln auf die Lokation des Objektes geschlossen werden können.

Maps werden in Situationen eingesetzt, wo Implementierungen mit Vektoren größere Speicherverluste bedingen würden. In diesen Fällen sucht die Verwendung von Schlüsseln einen Mittelweg zwischen effizientem Zugriff und kostengünstigem Speicherbedarf (definiert in <map>).

- ➡ Map-Containern wird bei der Instanzbildung neben dem Datentyp der zu verwaltenden Objekte auch ein Datentyp für den Schlüssel (Key) und ein Vergleichsoperator (Compare) übergeben, der die Ordnung der Schlüssel festlegt (siehe Funktionsobjekte).
- ➡ Die überladenen Formen der Konstruktoren erlauben die Einrichtung leerer Container ebenso wie von Containern, die eine Sequenz von Elementen aus einem anderen Container erhalten.
- ➡ Neue Elemente können mittels `insert()` oder `[]` eingefügt werden.
- ➡ Bestehende Elemente können mittels `erase()` gelöscht werden.
- ➡ Zugriff auf einzelne Elemente kann mit Hilfe zurückgelieferter Iteratoren geschehen (`begin()`, `end()`, `rbegin()`, `rend()`, `find()`, `lower_bound()`, `upper_bound()` sowie `insert()`).

<map>

```

TEMPLATE <CLASS KEY, CLASS T,
          CLASS COMPARE = LESS<T>
          CLASS ALLOCATOR = ALLOCATOR> CLASS MULTIMAP

```

Datenstruktur, die mit Hilfe von nicht unbedingt eindeutigen Schlüsseln auf ihre Elemente zugreift (Hashing).

Während die Klasse `map` nur eindeutige Schlüssel erlaubt, können in `multimap` mehrere Schlüssel bezüglich der Vergleichsfunktion identisch sein. Dies bedeutet, daß ein Schlüssel auf mehrere Objekte verweist,

die dann noch extra durchsucht werden müssen, um ein bestimmtes Objekt herauszufinden. Dies bedeutet meist verlängerte Laufzeiten bei geringerem Speicherbedarf, hat aber den Vorteil, daß auch Elemente mit Schlüsseln, die bereits vorhanden sind, in den Container aufgenommen werden (definiert in `<map>`).

➡ Die Klasse `multimap` verfügt über keinen Indexoperator `[]`.

`<set>`

```
TEMPLATE <CLASS T, CLASS COMPARE = LESS<T>
          CLASS ALLOCATOR = ALLOCATOR> CLASS MULTISSET
```

Datenstruktur, in der die Elemente automatisch sortiert abgespeichert werden. In einem `multiset`-Container können – im Gegensatz zu `set` – mehrere Elemente gleichen Werts abgelegt werden (definiert in `<set>`).

`<queue>`

```
TEMPLATE <CLASS T, CLASS CONTAINER = VECTOR<T>,
          CLASS COMPARE = LESS<CONTAINER::VALUE_TYPE>
>
          CLASS PRIORITY_QUEUE
```

Datenstruktur, die das Einfügen neuer Elemente erlaubt und bei Pop-Zugriffen stets das Element höchster Priorität (beispielsweise das größte Element) löscht.

Warteschlangen werden üblicherweise zum differenzierten Abarbeiten zeitlich geordneter Eingaben/Befehlen eingesetzt. Die Eingaben werden aufgrund bestimmter Eigenschaften (Größe, Wert einer Variablen etc.) mit einer Priorität versehen. Bei der Abarbeitung überspringen dann Eingaben höherer Priorität alle anderen Eingaben, während Eingaben gleicher Priorität in der Reihenfolge ihres zeitlichen Auftretens bearbeitet werden (definiert in `<queue>`).

➡ Bei der Spezialisierung der Template-Klasse wird dem Platzhalter Container die Klasse einer sequentiellen Datenstruktur übergeben, die die folgenden Methoden implementiert haben muß: `empty()`, `size()`, `front()`, `back()`, `push_back()`, `pop_back()`, also `vector` oder `deque`, aber nicht `list`.

Der Grund hierfür ist, daß die Klasse `priority_queue` keinen eigenen Container darstellt, sondern nur die Schnittstelle zu den Daten eines »echten« Containers. Sie enthält daher eine `protected`-Instanz des ihr zugrundeliegenden sequentiellen Containers und macht nur diejenigen Methoden dieser Containerklasse zugänglich (durch Definition eigener `public`-Methoden, die lediglich die entsprechenden Methoden der Containerklasse aufrufen), die für die Implementie-

rung einer Prioritätswarteschlange nötig sind (darum die Forderung, daß der im Template spezifizierte Container-Typ diese Methoden auch zur Verfügung stellt).

- ➡ Dem zweiten Platzhalter wird eine Funktionsklasse übergeben, deren Instanz zur Prioritätsbestimmung herangezogen wird. Auf diese Weise kann der Programmierer selbst bestimmen, wie die Priorität ermittelt wird.

<queue>

```
TEMPLATE <CLASS T, CLASS CONTAINER = DEQUE<T> >
    CLASS PRIORITY_QUEUE
```

Einfache Warteschlange. Datenstruktur, die das Einfügen von Elementen nur am Ende und Entfernen von Elementen nur am Anfang zuläßt (FIFO = First in, First out).

Eine `queue` ist eine sequentielle Datenstruktur, die das Einfügen neuer Elemente nur an ihrem Ende, und das Löschen von Elementen nur am Anfang erlaubt – eben ganz so wie eine Warteschlange vor einem Kassenschalter. Warteschlangen werden üblicherweise zum Abarbeiten zeitlich geordneter Eingaben/Befehlen eingesetzt, wie zum Beispiel bei der Botschaftsverarbeitung unter Windows, etc. (definiert in <queue>)

- ➡ Bei der Spezialisierung der Template-Klasse wird dem Platzhalter Container der Containertyp einer sequentiellen Datenstruktur übergeben, die die folgenden Methoden implementiert haben muß: `empty()`, `size()`, `front()`, `back()`, `push_back()` und `pop_front()`, also `list` oder `deque`, aber nicht `vector` (wegen fehlendem `pop_front()`).

Der Grund hierfür ist, daß die Klasse `queue` keinen eigenen Container darstellt, sondern nur die Schnittstelle zu den Daten eines »echten« Containers. Sie enthält daher eine `protected`-Instanz des ihr zugrundeliegenden sequentiellen Containers und macht nur diejenigen Methoden dieser Containerklasse zugänglich (durch Definition eigener `public`-Methoden, die lediglich die entsprechenden Methoden der Containerklasse aufrufen), die für die Implementierung einer Warteschlange nötig sind (darum die Forderung, daß der im Template spezifizierte Container-Typ diese Methoden auch zur Verfügung stellt).

<set>

```
TEMPLATE <CLASS T, CLASS COMPARE = LESS<T>
    CLASS ALLOCATOR = ALLOCATOR> CLASS SET
```

Datenstruktur, in der die Elemente automatisch sortiert abgespeichert werden.

In einem `set`-Container können – im Gegensatz zu `multiset` – keine Elemente mit gleichem Sortierwert abgelegt werden (definiert in `<set>`).

Die Klassen `set` und `multiset` sind praktisch identisch zu `map` und `multimap`, mit dem Unterschied, daß `map` (`multimap`) ihre Elemente nach einem Schlüssel und `set` (`multiset`) die Elemente nach ihrem Wert sortieren.

Sets sollten in Fällen eingesetzt werden, wo häufig auf beliebige Positionen im Container zum Einfügen und Entfernen zugegriffen wird, da der Zugriff schneller als bei anderen Containern (`deque`, `vector`, `list`) erfolgt.

- ➔ Set-Containern werden bei der Instanzbildung neben dem Datentyp der zu verwaltenden Objekte (`key`) noch ein Vergleichsoperator (`compare`) übergeben, der die Ordnung der Objekte festlegt (siehe Funktionsobjekte).
- ➔ Die überladenen Formen der Konstruktoren erlauben die Einrichtung leerer Container ebenso wie von Containern, die eine Sequenz von Elementen aus einem anderen Container erhalten.
- ➔ Neue Elemente können nur mittels `insert()` eingefügt werden.
- ➔ Bestehende Elemente können mittels `erase()` gelöscht werden.
- ➔ Zugriff auf einzelne Elemente kann mit Hilfe zurückgelieferter Iteratoren geschehen (`begin()`, `end()`, `rbegin()`, `rend()`, `find()`, `lower_bound()`, `upper_bound()` sowie `insert()`).
- ➔ Der Index-Operator `[]` ist nicht definiert.

`<stack>`

```
TEMPLATE <CLASS T, CLASS CONTAINER = DEQUE<T>> CLASS STACK
```

Keller oder Stapel. Datenstruktur, die Einfügen und Löschen jeweils nur am oberen Ende erlaubt (LIFO = Last in, First out).

Ein `stack` ist eine sequentielle Datenstruktur, die das Einfügen und Löschen von Elementen nur am oberen Ende erlaubt – ähnlich wie man Bücher übereinander stapelt (definiert in `<stack>`).

- ➔ Bei der Spezialisierung der Template-Klasse wird dem Platzhalter `Container` der Containertyp einer sequentiellen Datenstruktur übergeben, die die folgenden Methoden implementiert haben muß: `empty()`, `size()`, `back()`, `push_back()` und `pop_back()`, also `list`, `vector` oder `queue`).

Der Grund hierfür ist, daß die Klasse `stack` keinen eigenen Container darstellt, sondern nur die Schnittstelle zu den Daten eines »ech-

ten« Containers. Sie enthält daher eine `protected`-Instanz des ihr zugrundeliegenden sequentiellen Containers und macht nur diejenigen Methoden dieser Containerklasse zugänglich (durch Definition eigener `public`-Methoden, die lediglich die entsprechenden Methoden der Containerklasse aufrufen), die für die Implementierung eines Stapels nötig sind (darum die Forderung, daß der im Template spezifizierte Container-Typ diese Methoden auch zur Verfügung stellt).

`<vector>`

```
TEMPLATE <CLASS T, CLASS ALLOCATOR = ALLOCATOR> CLASS VECTOR
```

Datenstruktur, die Einfügen und Löschen auf beliebigen Positionen unterstützt, wobei Operationen auf dem Ende der Datenstruktur am schnellsten sind. Den Container `vector` kann man sich am einfachsten als Verallgemeinerung von Arrays vorstellen. Seine größten Vorteile sind sicherlich der schnelle indizierte Zugriff und die dynamische Speicherverwaltung. Nachteilig sind das relativ ineffiziente Einfügen und Löschen (beide Operationen sind um so aufwendiger, je weiter vorne das zu löschende oder einzufügende Element liegt, da die nachfolgenden Elemente dann jeweils umkopiert werden müssen; das Suchen und Lesen von Daten in sortierten Vektoren ist dagegen wieder sehr effizient) und das Problem, daß Einfüge- und Löschoperationen dazu führen können, daß bestehende Iteratoren und Referenzen auf Container-Elemente ungültig werden. Beide Nachteile hat ein `list`-Container nicht (definiert in `<vector>`).

Als Spezialisierung des Templates `vector` ist die Klasse `vector<bool>` definiert, die eine optimierte Speicherverwaltung für Bitwerte garantiert und zusätzlich die Funktion `flip()` zum Invertieren aller Bits bereitstellt.

- ➡ Die überladenen Formen der Konstruktoren erlauben die Einrichtung leerer Container oder Container, die mehrere Kopien eines Elements beziehungsweise die Kopie einer Sequenz von Elementen aus einem anderen Container enthalten.
- ➡ Die Größe des Containers kann nach seiner Einrichtung durch Löschen und Einfügen von Elementen verändert werden.
- ➡ Neue Elemente können mittels `insert()`, `push_back()`, `resize()` oder `assign()` eingefügt werden, wobei letztere Funktion den Container zuvor leert.
- ➡ Bestehende Elemente können mittels `erase()`, `pop_back()` oder `resize()` gelöscht werden.

- ➡ Der Zugriff auf einzelne Elemente kann mit Hilfe zurückgelieferter Iteratoren geschehen (`begin()`, `end()`, `rbegin()`, `rend()` sowie `insert()`) oder über zurückgegebene Referenzen (Operator `[]`, `at()`, `front()`, `back()`).



Wahrscheinlich werden Sie in dieser Aufzählung einige Datenstrukturen vermissen, beispielsweise Implementierungen für binäre Suchbäume oder Hash-Tabellen. Dies ist aber nicht unbedingt ein Manko, denn diese und andere wichtige Container-Typen können aus den zur Verfügung stehenden Containern abgeleitet werden (so kann der Container `set` beispielsweise als Ersatz für einen Binärbaum dienen und mehrdimensionale Arrays können aus Vektoren, die Vektoren als Elemente enthalten, aufgebaut werden).

Beispiel

Das folgende Programm verwendet einen `deque`-Container zur Aufbewahrung einer Sammlung von ungeraden Zahlen. Die Zahlen selbst werden von der Funktion `get_ungerade()` erzeugt.

```
/* STL_CONT.CPP Verwendung von STL-Containern */
#include <cstdlib>
#include <iostream>
#include <deque>
using namespace std;

// Generator fuer ungerade Zufallszahlen
int get_ungerade()
{
    int zufallszahl;
    zufallszahl = rand() % 10000;
    if (!(zufallszahl & 1L))
        return zufallszahl + 1;
    else
        return zufallszahl;
}

int main()
{
    deque<int> zahlen_container(5);
    int i;

    // Werte initialisieren
    for(i = 0; i < zahlen_container.size(); i++)
    {
        zahlen_container[i] = get_ungerade();
    }

    // Werte hinzufuegen
    for(i=0; i < 6; i++)
    {
        zahlen_container.push_front(get_ungerade());
    }

    // Werte loeschen
    zahlen_container.pop_front();
}
```

```
zahlen_container.pop_back();

//Werte ausgeben
for(i = 0; i < zahlen_container.size(); i++)
{
    cout << i << ": " << zahlen_container[i] << endl;
}

return 0;
}
```

Ausgabe



Falls Sie die graue Theorie zu den Containern etwas verschreckt haben *Analyse* sollte, sind Sie jetzt sicherlich erstaunt, wie einfach doch das Programm aussieht. Aber seien Sie deswegen nicht enttäuscht – das nächste Programm zur STL wird schon etwas komplizierter sein.

Bevor Sie einen STL-Container verwenden können, müssen Sie

- ➡ seine Header-Datei aufnehmen (Zeile 4), wobei Sie im Falle von `stack`, `queue` und `priority_queue` daran denken müssen, auch für den bei der Instanzbildung spezifizierten Containertyp die entsprechende Header-Datei anzugeben;
- ➡ den Namensbereich `std` einschalten, in dem alle STL-Container deklariert sind (Zeile 5).

In Zeile 18 sehen Sie dann, wie eine Template-Spezialisierung für den Datentyp `int` definiert und eine entsprechende Klasseninstanz `zahlen_container` mit fünf Elementen direkt erstellt wird.

Danach kommen verschiedene Methoden von `deque` zum Einsatz, mit denen Container-Elemente initialisiert, hinzugefügt, gelöscht und aus-

gegeben werden. Für den Zugriff auf einzelne Elemente im Container wird dabei vornehmlich der Index-Operator `[]` verwendet.

Im nachfolgenden Abschnitt werden wir sehen, wie man über Iteratoren auf Container-Elemente zugreift und welche neuen Möglichkeiten uns dies bietet.

D.1.2 Der Gebrauch von Iteratoren

<iterator>

Container verwenden üblicherweise Iteratoren, mit denen ein Container durchlaufen und auf die Elemente im Container zugegriffen werden kann. Auch die STL arbeitet intensiv mit Iteratoren zusammen, wobei die verschiedenen ihrer Funktionalität nach in unterschiedliche Kategorien gefaßt werden können:

Input-Iterator

Mit Input-Iteratoren kann man auf Container-Elemente zugreifen, um ihre Werte auszulesen, aber man kann die Elemente nicht verändern. Außerdem können Sie sich mit einem Input-Iterator von der Position seiner Initialisierung an nur in Richtung des Container-Endes bewegen, das heißt, Sie können einen Input-Iterator inkrementieren (`++`), aber nicht dekrementieren (`--`).

Output-Iterator

Output-Iteratoren stellen das Pendant zu den Input-Iteratoren dar. Mit ihnen können Sie Werte speichern (nur überschreiben), aber nicht auslesen. Sie können ebenfalls nur inkrementiert werden.

Zum Einfügen neuer Werte in einen Container dienen spezielle Adaptionen – die sogenannten Insert-Iteratoren, die Zuweisungen an das Element des Iterators in eine Einfüge-Operation umsetzen.

Forward-Iterator

Mit Forward-Iteratoren kann man einen Container ebenfalls nur in Richtung Ende durchlaufen, man kann sie aber zum Lesen und Schreiben verwenden (Ein Forward-Iterator ist eine Kombination aus Input- und Output-Iterator).

Bidirektionaler Iterator

Bidirektionale Iteratoren können wie Vorwärtsiteratoren zum Lesen und Schreiben eingesetzt werden, erlauben es aber zusätzlich, sich vorwärts wie rückwärts in einem Container zu bewegen, das heißt, sie

können inkrementiert und dekrementiert werden. Bidirektionale Iteratoren werden beispielsweise von `list`, `set`, `multiset`, `map` und `multimap` verwendet.

Random-Access-Iterator

Die Random-Access-Iteratoren können wie bidirektionale Iteratoren verwendet werden, haben aber den Vorteil, daß sie nicht nur inkrementiert oder dekrementiert werden können, sondern daß sie auch beliebige Sprünge erlauben (Plus- und Minus-Operatoren sind definiert). Sie entsprechen damit praktisch normalen Zeigern. Iteratoren für wahlfreien Zugriff werden von `vector` und `deque` verwendet.

Beispiel

Solange Sie mit STL-Containern arbeiten, brauchen Sie sich um die Definition geeigneter Iteratoren keine Gedanken zu machen, da dies bereits in der Implementation der jeweiligen Container-Klassen geschieht. Sie müssen lediglich nach Bedarf Iteratoren einrichten und korrekt initialisieren, wozu Sie entsprechende Methoden der Container-Klassen nutzen können. Ein Blick in die Definition der Container-Klasse zeigt Ihnen, welche Methoden Iteratoren zurückliefern und welche Methoden mit Iteratoren als Argumenten aufgerufen werden. Jetzt müssen Sie nur noch daran denken, daß bei den meisten Containern bestehende Iteratoren nach Einfüge- oder Löschoperationen ungültig werden.

```
/* STL_ALGO.CPP Verwendung von STL-Containern */
#include <cstdlib>
#include <iostream>
#include <algorithm>
#include <functional>
#include <deque>
using namespace std;

// Generator fuer ungerade Zufallszahlen
int get_ungerade()
{
    int zufallszahl;
    zufallszahl = rand() % 10000;
    if (! (zufallszahl & 1L))
        return zufallszahl + 1;
    else
        return zufallszahl;
}

typedef deque<int>::iterator iter;

int main(int argc, char **argv)
{
    deque<int> container(5);

    // Werte initialisieren
    generate(container.begin(), container.end(), get_ungerade);
}
```

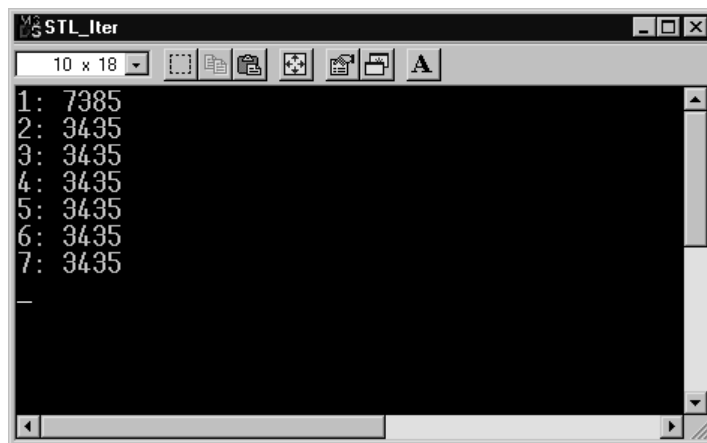
```
// Werte hinzufuegen
container.insert(container.end(),6,get_ungerade());

// Werte sortieren
stable_sort(container.begin(), container.end(), greater<int>());

//Werte ausgeben
int count = 1;
for(iter i = container.begin(); i != container.end(); i++)
{
    cout << count++ << ": " << *i << endl;
}

return 0;
}
```

Ausgabe



Analyse Gegenüber der vorigen Version wurden die `for`-Schleifen auf die Verwendung von Iteratoren umgestellt. Die meisten `for`-Schleifen schreiben geradezu nach dem Einsatz von Iteratoren.

Iteratoren und Schleifen. Um einen Iterator als Schleifenvariable verwenden zu können, muß er zuerst erzeugt und dann initialisiert werden. Bei der Erzeugung beziehungsweise Deklaration des Iterators ist zu beachten, daß er den richtigen Typ erhält. Sie können dies sicherstellen, indem Sie die Typdefinition aus der Container-Klasse übernehmen. Jede Container-Klasse definiert nämlich einen Typ für die zu ihr passenden Iteratoren. Um nicht bei jeder Deklaration eines Iterators die ganze Typdefinition wiederholen zu müssen, habe ich in Zeile 16 das Kürzel `iter` für diesen Iteratortyp definiert.

In den Schleifen-Headern wird also zuerst ein Iterator vom Typ `iter` deklariert und dann initialisiert. Allerdings kann man einen Iterator nicht wie eine Integer-Variable einfach auf Null setzen. Statt dessen

verwenden wir die Methode `begin()`, die einen Iterator auf das erste Element zurückliefert. Auch die Abbruchbedingung hat sich geändert. Der Iterator wird jetzt mit dem Iterator auf das Container-Ende verglichen.

Vorsicht! Die Methode `end()` liefert einen Iterator, der hinter das letzte Element im Container verweist. Dies ist wichtig, damit man Elementens Ende eines Containers einfügen kann. Da aber mit diesem Iterator kein Element mehr verknüpft ist, dürfen Sie ihn nicht dereferenzieren.



In der ersten Schleife werden also wie zuvor die fünf Elemente im Container mit zufälligen ungeraden Zahlen initialisiert.

Elemente in Container einfügen. Als nächstes werden dem Container sechs neue Zahlen hinzugefügt. Statt in einer Schleife wird dies durch einen Aufruf der Methode `insert()` erledigt. Diese Funktion liegt in drei überladenen Versionen vor:

➡ 1. Version

```
iterator insert(iterator position, const T& x = T());
```

Das erste Argument gibt die Position an, vor der eingefügt wird. Das zweite Argument bezeichnet das Objekt, das eingefügt werden soll.

➡ 2. Version

```
void insert (iterator position, size_type n, const T& x);
```

Das erste Argument ist hier wiederum die Einfügeposition, das letzte Argument bezeichnet das einzufügende Objekt. Das zweite Argument gibt an, wie viele Kopien des Objekts eingefügt werden sollen. Diese Version wird in Zeile 26 aufgerufen. Im Gegensatz zum letzten Beispielprogramm werden damit sechs identische statt sechs unterschiedliche Werte eingefügt.

➡ 3. Version

```
template <class InputIterator>
void insert (iterator position,
             InputIterator first,
             InputIterator last);
```

Die letzte Version dient dazu, eine Sequenz von Objekten einzufügen.



Allgemein gilt, daß das gleichzeitige Einfügen von Elementen effektiver ist, da für alle Objekte zusammen Speicher allokiert wird.

Elemente löschen. In ähnlicher Weise wie `insert()` ist die Methode `erase()` überladen, mit der Sie ein einzelnes Element oder eine Sequenz von Elementen löschen können. In Zeile 29 wird `erase()` aufgerufen, um die ersten drei Elemente im Container zu löschen. Wir nutzen dabei die Tatsache, daß die Container-Klasse `deque` Iteratoren mit wahlfreiem Zugriff unterstützt, weswegen wir den von `begin()` zurückgelieferten Iterator einfach um 3 erhöhen können.

Zum Schluß werden die Werte der Elemente im Container ausgegeben.

D.1.3 Die Algorithmen

<algorithm>

Nachdem wir uns in die Verwendung von Iteratoren eingearbeitet haben, ist es nur noch ein kleiner Schritt zur Verwendung der STL-Algorithmen.

Tabelle D.1:
Die STL-
Algorithmen

Algorithmus	Beschreibung
<code>adjacent_find</code>	benachbarte Vorkommen aufspüren
<code>binary_search</code>	Element in sortierter Sequenz suchen
<code>copy</code>	Sequenz kopieren (beginnt mit erstem Element)
<code>copy_backward</code>	Sequenz kopieren (beginnt mit letztem Element)
<code>count</code>	Elemente in Sequenz zählen
<code>count_if</code>	bestimmte Elemente in Sequenz zählen
<code>equal</code>	Elemente in Sequenz paarweise vergleichen
<code>equal_range</code>	einheitliche Sequenz suchen
<code>fill</code>	Elemente in Sequenz einen Wert zuweisen
<code>fill_n</code>	den ersten n Elemente in Sequenz einen Wert zuweisen
<code>find</code>	Wert in Sequenz suchen
<code>find_end</code>	letztes Vorkommen einer Teilsequenz suchen
<code>find_first_of</code>	Element in Sequenz suchen
<code>find_if</code>	bestimmtes Element suchen
<code>for_each</code>	Funktion auf Elemente anwenden
<code>generate</code>	Sequenz einen berechneten Wert zuweisen

Algorithmus	Beschreibung
generate_n	ersten Elementen einer Sequenz einen berechneten Wert zuweisen
includes	Sequenz enthalten in anderer Sequenz
inplace_merge	zwei Sequenzen vermengen
iter_swap	zwei Elemente tauschen
lexicographical_compare	zwei Sequenzen lexikographisch vergleichen
lower_bound	Element in sortierte Sequenz einfügen
make_heap	Sequenz in Heap umwandeln
max	Maximum zweier Werte bestimmen
max_element	Maximum einer Sequenz bestimmen
merge	zwei Sequenzen vermengen
min	Minimum zweier Werte bestimmen
min_element	Minimum einer Sequenz bestimmen
mismatch	die ersten Elemente zweier Sequenzen, die nicht gleich sind
next_permutation	nächste Permutation erzeugen
nth_element	n-tes Element einsortieren
partial_sort	Teilsequenz sortieren
partial_sort_copy	Teilsequenz sortiert kopieren
partition	Elemente, die eine Bedingung erfüllen, nach vorne kopieren
pop_heap	Element von Heap entfernen
prev_permutation	vorangehende Permutation erzeugen
push_heap	Element in Heap einfügen
random_shuffle	Elemente in Sequenz neu verteilen
remove	Elemente mit bestimmtem Wert entfernen
remove_copy	nur Elemente kopieren, die ungleich einem bestimmten Wert sind
remove_copy_if	nur Elemente kopieren, die eine bestimmte Bedingung nicht erfüllen
remove_if	bestimmte Elemente entfernen
replace	Elemente mit bestimmtem Wert ersetzen
replace_copy	Sequenz kopieren, bestimmte kopierte Elemente durch Wert ersetzen

Algorithmus	Beschreibung
<code>replace_copy_if</code>	Sequenz kopieren, bestimmte kopierte Elemente durch Wert ersetzen
<code>replace_if</code>	bestimmte Elemente ersetzen
<code>reverse</code>	Reihenfolge der Elemente umkehren
<code>reverse_copy</code>	Sequenz in umgekehrter Reihenfolge kopieren
<code>rotate</code>	Elemente rotieren
<code>rotate_copy</code>	Sequenz in rotierter Reihenfolge kopieren
<code>search</code>	Teilsequenz suchen
<code>search_n</code>	Teilsequenz mit identischen Werten suchen
<code>set_difference</code>	Differenz zweier sortierter Sequenzen (für Elemente aus A, aber nicht B)
<code>set_intersection</code>	Schnitt zweier sortierter Sequenzen
<code>set_symmetric_difference</code>	Differenz zweier sortierter Sequenzen (für Elemente entweder in A oder B)
<code>set_union</code>	Vereinigung zweier sortierter Sequenzen
<code>sort</code>	Sequenz sortieren
<code>sort_heap</code>	Heap sortieren
<code>stable_partition</code>	Elemente, die eine Bedingung erfüllen, unter Beibehaltung ihrer relativen Ordnung nach vorne kopieren
<code>stable_sort</code>	Sequenz sortieren (Reihenfolge gleicher Elemente bleibt erhalten)
<code>swap</code>	zwei Elemente tauschen.
<code>swap_ranges</code>	Elemente zweier Sequenzen tauschen
<code>transform</code>	Elementen einer Sequenz neue Werte zuweisen
<code>unique</code>	Duplikate aus Sequenz entfernen
<code>unique_copy</code>	Sequenz ohne Duplikate kopieren
<code>upper_bound</code>	Element in sortierte Sequenz einfügen

Damit kommen wir zu unserem letzten Beispiel zur STL.

Beispiel

In dem nachfolgenden Programm wird die Algorithmusfunktion `generate()` zur Initialisierung der Container-Elemente herangezogen. Sie erspart uns eine weitere `for`-Schleife.

Danach wird der Container – wie im letzten Beispiel auch – um sechs neue Elemente mit identischen Werten erweitert.

Zum Schluß wird der Container sortiert und die Werte der Elemente ausgegeben.

```

1:  /* STL_ALGO.CPP Verwendung von STL-Containern */
2:  #include <stdlib.h>
3:  #include <iostream>
4:  #include <deque>
5:  using namespace std;

6:  // Generator fuer ungerade Zufallszahlen
7:  int get_ungerade()
8:  {
9:      int zufallszahl;
10:     zufallszahl = random(10000);
11:     if (!(zufallszahl & 1L))
12:         return zufallszahl + 1;
13:     else
14:         return zufallszahl;
15: }

16: typedef deque<int>::iterator iter;

17: int main(int argc, char **argv)
18: {
19:     deque<int> container(5);

20:     // Werte initialisieren
21:     generate(container.begin(), container.end(), get_ungerade);

22:     // Werte hinzufuegen
23:     container.insert(container.end(), 6, get_ungerade());

24:     // Werte sortieren
25:     stable_sort(container.begin(),
26:                 container.end(), greater<int>());

27:     //Werte ausgeben
28:     int count = 1;
29:     for(iter i = container.begin(); i != container.end(); i++)
30:     {
31:         cout << count++ << ": " << *i << endl;
32:     }

33:     return 0;
34: }
```

Ausgabe

In Zeile 21 wird die Funktion `generate()` aufgerufen. Für diese Funktion gibt es zwei Versionen:

```

template<class ForwardIterator, class Generator>
void generate(ForwardIterator first,
              ForwardIterator last,
              Generator gen);

template<class OutputIterator, class Size, class Generator>
void generate_n(OutputIterator first,
                Size n,
                Generator gen);
```




```
1: 8635
2: 7633
3: 7617
4: 7617
5: 7617
6: 7617
7: 7617
8: 7617
9: 6119
10: 2023
11: 1121
```

Der ersten Version werden zwei Iteratoren übergeben, die den zu initialisierenden Bereich festlegen. Als drittes Argument erwartet Sie ein Funktionsobjekt (in unserem Beispiel wird einfach eine Funktion übergeben), das die Werte für die Initialisierung der Elemente zurückliefert. Im Gegensatz zur Methode `insert()` die im letzten Beispiel beschrieben wurde, wird das Funktionsobjekt allerdings zur Initialisierung jedes einzelnen Elementes aufgerufen, so daß man die Möglichkeit hat, den Bereich mit unterschiedlichen Werten zu initialisieren.

Für die zweite Version wird der zu initialisierende Bereich durch einen Iterator und die Anzahl der Elemente statt durch zwei Iteratoren spezifiziert.

Der zweite Algorithmus, den wir verwenden, ist `stable_sort()`.

```
template<class RandomAccessIterator>
void stable_sort(RandomAccessIterator first,
                 RandomAccessIterator last);

template<class RandomAccessIterator, class Compare>
void stable_sort(RandomAccessIterator first,
                 RandomAccessIterator last,
                 Compare comp);
```

Im Unterschied zu `sort()` ist bei `stable_sort()` garantiert, daß Elemente, die gemäß der Sortierfunktion identisch sind, zueinander in der gleichen relativen Reihenfolge bleiben, die sie vor der Sortierung hatten. Dies macht in unserem Beispiel zwar keinen Unterschied, kann aber in anderen Fällen bedeutsam sein.

Stellen Sie sich beispielsweise vor, die Elemente Ihres Containers wären Strukturen mit den Namen und Adressen Ihrer Bekannten. Jetzt sortieren Sie den Container zuerst nach den Nachnamen, dann nach den Wohnorten. Wenn Sie beim zweiten Mal `stable_sort()` verwendet haben, ist der Container jetzt nach Wohnorten sortiert und alle Einträge zu einem Wohnort sind wiederum nach Nachnamen sortiert. Haben Sie dagegen bei der zweiten Sortierung `sort()` verwendet, kann es sein, daß die Einträge zu einem Wohnort nicht mehr nach den Nachnamen sortiert, sondern durcheinandergelassen sind.



Standardmäßig wird `stable_sort()` der zu sortierende Bereich in Form zweier Iteratoren übergeben und die Sortierung erfolgt dann in aufsteigender Reihenfolge. Natürlich wollen wir dies gerade nicht, sondern ziehen es vor, unsere Werte in absteigender Folge auszugeben. Statt aber die Ausgaberoutine umzuschreiben, nutzen wir die zweite Version von `stable_sort()`, der ein Vergleichsobjekt übergeben werden kann. In der Header-Datei `<functional>` sind glücklicherweise schon eine ganze Reihe von Vergleichsobjekten, die zu der allgemeineren Kategorie der Funktionsobjekte gehören, vordefiniert.

D.1.4 Funktionsobjekte

`<functional>`

Die Implementierung generischer Funktionen beruht oftmals auf der Übergabe von Funktionen als Parameter (vergleiche `qsort()`). Die STL erweitert dieses Konzept, indem Sie Funktionen in Klassen kapselt. Allen diesen Klassen gemeinsam ist, daß der `()`-Operator für sie überladen wurde, damit die Instanzen der Klasse (die Funktionsobjekte) wie Funktionen aufgerufen werden können.

Funktionsobjekte sind mächtiger als Funktionen, da sie auf die internen Daten und Methoden ihrer Klasse zurückgreifen können. Zudem sind alle Funktionsobjekte der STL als Templates implementiert, um mit beliebigen Datentypen arbeiten zu können.

Die STL unterscheidet

- arithmetische Funktionsobjekte, die eine arithmetische Operation durchführen (`divides`, etc.) und
- Prädikate, die einen Vergleich durchführen (`equal_to`, etc.), `bool` als Ergebnistyp haben und in den Algorithmen der STL häufig verwendet werden, um eine Bedingung zu definieren, die die Elemente, auf denen die Algorithmenfunktion operiert, erfüllen müssen.

Die Funktionsobjekte

Algorithmus	Beschreibung
<code>divides</code>	$\text{arg1} / \text{arg2}$
<code>equal_to</code>	$\text{arg1} == \text{arg2}$
<code>greater</code>	$\text{arg1} > \text{arg2}$
<code>greater_equal</code>	$\text{arg1} \geq \text{arg2}$
<code>less</code>	$\text{arg1} < \text{arg2}$
<code>less_equal</code>	$\text{arg1} \leq \text{arg2}$
<code>logical_and</code>	$\text{arg1} \ \&\& \ \text{arg2}$
<code>logical_not</code>	$\neg \text{arg}$
<code>logical_or</code>	$\text{arg1} \ \ \text{arg2}$
<code>minus</code>	$\text{arg1} - \text{arg2}$
<code>modulus</code>	$\text{arg1} \% \text{arg2}$
<code>multiplies</code> (früher <code>times</code>)	$\text{arg1} * \text{arg2}$
<code>negate</code>	$-\text{arg}$
<code>not_equal_to</code>	$\text{arg1} \neq \text{arg2}$
<code>plus</code>	$\text{arg1} + \text{arg2}$

Funktionsadapter

Ergänzt werden die Möglichkeiten der Funktionsobjekte durch sogenannte Funktionsadapter, die die Aufrufe von Funktionsobjekten anpassen, indem sie die Funktionsobjekte selbst als Argument übernehmen und intern aufrufen.

D.1.5 Die Stream-Klassen

`<fstream>`

Algorithmus	Beschreibung
<code>basic_fstream</code>	Stellt HighLevel-Routinen zur Ein- und Ausgabe von und in Dateien zur Verfügung (mit Spezialisierungen für <code>char</code> und <code>wchar_t</code>).
<code>basic_ifstream</code>	Stellt HighLevel-Routinen zum Einlesen aus einer Datei zur Verfügung (mit Spezialisierungen für <code>char</code> und <code>wchar_t</code>).

Algorithmus	Beschreibung
basic_ofstream	Stellt HighLevel-Routinen zum Ausgeben in eine Datei zur Verfügung (mit Spezialisierungen für char und wchar_t).
basic_filebuf	Von basic_streambuf abgeleitete Pufferklasse, die die Ein- und Ausgabe mit einer Datei verknüpft (mit Spezialisierungen für char und wchar_t).

<iomanip>

Enthält verschiedene Manipulatoren zur formatierten Ein- und Ausgabe.

Algorithmus	Beschreibung
resetiosflags	Formatierungsflag löschen.
setbase	Basis des Zahlensystems festlegen.
setfill	Füllzeichen festlegen.
setiosflags	Formatierungsflag setzen.
setprecision	Genauigkeit festlegen.
setw	Feldbreite festlegen.

<ios>

Klassen

Algorithmus	Beschreibung
basic_ios	Basisklasse der istream-Klassen.
ios_base	Definiert Aufzählungstypen, Bitfelder und verwandte Funktionen für die Streamkonfiguration und -diagnose.

Enthält neben den I/O-Streamklassen auch verschiedene Manipulatoren zur formatierten Ein- und Ausgabe.

Algorithmus	Beschreibung
boolalpha	bool-Typ berücksichtigen.
dec	Dezimaldarstellung.
hex	Hexadezimaldarstellung.

Algorithmus	Beschreibung
fixed	Feste Stellenzahl.
internal	Bestimmte Stellen mit Füllzeichen.
left	Linksbündige Ausgabe.
nboolalpha	bool-Typ nicht berücksichtigen.
nshowbase	Basis des Zahlensystems nicht anzeigen.
nshowpoint	Gleitkommazahlen ohne Dezimalpunkt.
nshowpos	Positive Zahlen ohne Vorzeichen.
noskipws	White space miteinlesen.
nouppercase	Keine Konvertierung in Großbuchstaben.
oct	Oktaldarstellung.
right	Rechtsbündige Ausgabe.
scientific	Darstellung mit Mantisse und Exponent.
showbase	Basis des Zahlensystems anzeigen.
showpoint	Gleitkommazahlen mit Dezimalpunkt.
showpos	Positive Zahlen mit Vorzeichen.
skipws	White space nicht einlesen.
uppercase	Bestimmte Zeichen als Großbuchstaben.

<iosfwd>

Enthält Vorwärtsdeklarationen der Stream-Klassen und Stream-Objekte.

<iostream>

Enthält Vorwärtsdeklarationen der wichtigsten vordefinierten Ein- und Ausgabestreams (cin, cout, etc.).

<istream>

Klassen

Algorithmus	Beschreibung
basic_istream	Stellt HighLevel Routinen zum Einlesen aus einem Streampuffer zur Verfügung.
istream	Spezialisierung für Datentyp char.
wistream	Spezialisierung für Datentyp wchar.

Enthält neben den istream-Klassen auch den Manipulator ws.

Algorithmus	Beschreibung
ws	White-space-Zeichen lesen.

<ostream>

Klassen

Algorithmus	Beschreibung
basic_ostream	Stellt HighLevel-Routinen zum Schreiben in einem Streampuffer zur Verfügung.
ostream	Spezialisierung für Datentyp char.
wostream	Spezialisierung für Datentyp wchar.

Enthält neben den ostream-Klassen auch verschiedene Manipulatoren.

Algorithmus	Beschreibung
endl	Zeilenvorschub und Puffer leeren.
ends	Stringende-Zeichen schreiben.
flush	Puffer leeren.

<sstream>

Algorithmus	Beschreibung
basic_istream	Stellt HighLevel-Routinen zum Einlesen zur Verfügung.
istream	Spezialisierung für Datentyp char.
wistream	Spezialisierung für Datentyp wchar.
basic_ostream	Stellt HighLevel-Routinen zum Ausgeben zur Verfügung.
ostream	Spezialisierung für Datentyp char.
wostream	Spezialisierung für Datentyp wchar.
basic_stringstream	Stellt HighLevel-Routinen zum Einlesen und Ausgeben zur Verfügung.

Algorithmus	Beschreibung
stringstream	Spezialisierung für Datentyp char.
wstringstream	Spezialisierung für Datentyp wchar.
basic_stringbuf	Von basic_streambuf abgeleitete Pufferklasse für Objekte der Klasse basic_string.
stringbuf	Spezialisierung für Datentyp char.
wstringbuf	Spezialisierung für Datentyp wchar.

<streambuf>

Algorithmus	Beschreibung
basic_streambuf	Basisklasse für gepufferte LowLevel-Operationen.
streambuf	Spezialisierung für Datentyp char.
wstreambuf	Spezialisierung für Datentyp wchar.

D.1.6 Die String-Klassen

<string>

Klassen

Algorithmus	Beschreibung
basic_string	Template-Klasse zur String-Behandlung.
string	Spezialisierung von basic_string für Datentyp char.
char_traits	Definiert grundlegende Funktionen und Datentypen für die Zeichen eines Zeichensatzes.
char_traits<char>	Spezialisierung für Datentyp char.
char_traits<wchar_t>	Spezialisierung für Datentyp wchar_t.
wstring	Spezialisierung von basic_string für Datentyp wchar_t.

Funktionen

Algorithmus	Beschreibung
getline	Liest eine Zeile aus einem Eingabestream.
swap	Zuweisung eines neuen Textes

D.1.7 Sonstige Klassen

<complex>

Klassen

Algorithmus	Beschreibung
complex	Klasse für komplexe Zahlen.
complex<double>	Spezialisierung für Datentyp double.
complex<float>	Spezialisierung für Datentyp float.
complex<long double>	Spezialisierung für Datentyp long double.

Funktionen

Algorithmus	Beschreibung
abs	Betrag berechnen.
arg	Argument ermitteln.
conj	Konjugierte Komplexe bestimmen.
cos	Kosinus berechnen.
cosh	Kosinus hyperbolicus berechnen.
exp	Exponentialfunktion berechnen.
imag	Imaginärteil bestimmen.
log	Natürlichen Logarithmus berechnen.
log10	Logarithmus zur Basis 10 berechnen.
norm	Norm bestimmen.
polar	Komplexe Zahl aus Polarform bestimmen.
pow	Zahl potenzieren.
real	Realteil bestimmen.
sin	Sinus berechnen.

Algorithmus	Beschreibung
<code>sinh</code>	Sinus hyperbolicus berechnen.
<code>sqrt</code>	Quadratwurzel berechnen.
<code>tan</code>	Tangens berechnen.
<code>tanh</code>	Tangens hyperbolicus berechnen.

<exception>

Klassen

Algorithmus	Beschreibung
<code>exception</code>	Basisklasse, der in der Laufzeitbibliothek definierten Exception-Klassen.
<code>bad_exception</code>	Für Exception-Objekte, die von der Funktion <code>unexpected()</code> ausgelöst werden.

Enthält verschiedene Funktionen zur Fehlerbehandlung.

Algorithmus	Beschreibung
<code>set_terminate</code>	Richtet Funktion zum Programmabbruch ein.
<code>set_unexpected</code>	Richtet Funktion zur Behandlung unerwarteter Exceptions ein.
<code>terminate</code>	Ruft Funktion zum Programmabbruch auf.
<code>uncaught_exception</code>	Liefert true, wenn eine noch nicht abgefangene Exception ausgelöst wurde.
<code>unexpected</code>	Ruft Funktion zur Behandlung unerwarteter Exceptions auf.

<limits>

Enthält das Klassentemplate `numeric_limits` sowie Spezialisierungen für die verschiedenen elementaren Datentypen.

Algorithmus	Beschreibung
<code>numeric_limits</code>	Klasse mit Informationen zur implementierungsspezifischen Darstellung eines Datentyps.

<locale>

In bestimmten Anwendungen ist es notwendig, lokale Einstellungen zu berücksichtigen. Dies betrifft zum Beispiel landesspezifische Darstellungen von Zeichen oder Datumsangaben oder die Darstellung numerischer Werte. Die einzelnen Bereiche werden als Kategorien bezeichnet, die wiederum in Facetten (Spezialisierungen der entsprechenden Klassentemplates) untergliedert werden. Mit Hilfe der Klasse `locale` und ihrer Elementfunktionen können verschiedene lokale Einstellungen und ihre Kategorien oder Facetten ausgewählt werden.

Algorithmus	Beschreibung
<code>codecvt</code>	Kategorie <code>ctype</code> . Unterstützt die Konvertierung zwischen Codierungsformen.
<code>codecvt_base</code>	Definiert Aufzählungstyp für Ergebniswerte.
<code>codecvt_byname</code>	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
<code>collate</code>	Klasse zur gleichnamigen Kategorie. Unterstützt den Vergleich und die Aufspaltung von Zeichenketten.
<code>collate_byname</code>	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
<code>ctype</code>	Klasse zur gleichnamigen Kategorie. Unterstützt die Zeichenerkennung sowie das Einlesen und Formatieren von Zeichenketten.
<code>ctype<char></code>	Spezialisierung für den Datentyp <code>char</code> .
<code>ctype_base</code>	Definiert Bitfeld zur Zeichenklassifizierung.
<code>ctype_byname</code>	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
<code>ctype_byname<char></code>	Spezialisierung für den Datentyp <code>char</code> .
<code>locale</code>	Klasse zur Verwaltung lokaler Einstellungen.
<code>messages</code>	Klasse zur gleichnamigen Kategorie. Unterstützt das Herauslesen von Strings aus Botschaftskatalogen.
<code>messages_base</code>	Definiert den Datentyp <code>catalog</code> .
<code>messages_byname</code>	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
<code>money_base</code>	Definiert Aufzählungstyp <code>part</code> zur Formatierung von Währungsangaben und den Rückgabety <code>pattern</code> .

Algorithmus	Beschreibung
money_punct	Kategorie monetary. Unterstützt die Punktsetzung in Währungsangaben.
money_punct_byname	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
money_get	Kategorie monetary. Unterstützt das Einlesen von Währungsangaben.
money_put	Kategorie monetary. Unterstützt die formatierte Ausgabe von Währungsangaben.
num_punct	Kategorie numeric. Unterstützt die Punktsetzung in numerischen Werten.
num_punct_byname	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
num_get	Kategorie numeric. Unterstützt das Einlesen numerischer Werte.
num_put	Kategorie numeric. Unterstützt die Formatierung bei der Ausgabe numerischer Werte.
time_base	Definiert Aufzählungstyp zur Abfolge von Datumsangaben.
time_get	Kategorie time. Unterstützt das Einlesen von Zeitangaben in eine Variable des Typs tm.
time_get_byname	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.
time_put	Kategorie time. Unterstützt die Ausgabe von Zeitangaben.
time_put_byname	Auswahl spezifischer Implementierungen anhand des Namens der lokalen Einstellung.

Enthält überladene Formen zur Zeichenklassifizierung und -konvertierung, die als zweites Argument – neben dem Zeichen – ein `locale`-Objekt zur Festlegung landesspezifischer Besonderheiten erwarten.

Algorithmus	Beschreibung
isalnum	Testet, ob Zeichen eine Ziffer oder ein Buchstabe ist.
isalpha	Testet, ob Zeichen ein Buchstabe ist.
isctrl	Testet, ob Zeichen ein Kontrollzeichen ist.
isdigit	Testet, ob Zeichen eine Zahl ist.
isgraph	Testet, ob Zeichen darstellbar ist.

Algorithmus	Beschreibung
islower	Testet, ob Zeichen ein Kleinbuchstabe ist.
isprint	Testet, ob Zeichen druckbar ist.
ispunct	Testet, ob Zeichen ein Interpunktionszeichen ist.
isspace	Testet, ob Zeichen ein Zwischenraumzeichen ist.
isupper	Testet, ob Zeichen ein Großbuchstabe ist.
isxdigit	Testet, ob Zeichen eine Hexadezimalziffer ist.
tolower	Wandelt Groß- in Kleinbuchstaben um.
toupper	Wandelt Klein- in Großbuchstaben um.

<memory>

Klassen

Algorithmus	Beschreibung
allocator	Für spezielle Speicheranforderungen.
auto_ptr	Zur automatischen Freigabe dynamischen Speichers.
raw_storage_iterator	Zur Ablage in nicht initialisierten Speicherbereichen.

Funktionen

Algorithmus	Beschreibung
get_temporary_buffer	Pufferbereich suchen.
return_temporary_buffer	Pufferbereich liefern.
uninitialized_copy	Iteratoren: Bereich initialisieren.
uninitialized_fill	Iteratoren: Bereich initialisieren.
uninitialized_fill_n	Iteratoren: Bereich initialisieren.

<new>

Klassen

Algorithmus	Beschreibung
bad_alloc	Für Exception-Objekte, die bei gescheiterten Speicherreservierungen ausgelöst werden.
nothrow	Strukturtyp zur Übergabe an new-Operator.

Enthält zudem die überladenen Operatorfunktionen für `new` und `delete` sowie die Funktion:

Algorithmus	Beschreibung
<code>set_new_handler</code>	Richtet Funktion zur Fehlerbehandlung von <code>new</code> ein.

<numeric>

Algorithmus	Beschreibung
<code>accumulate</code>	Iteratoren: Summe berechnen.
<code>adjacent_difference</code>	Iteratoren: Differenz zuweisen.
<code>inner_product</code>	Iteratoren: »Skalarprodukt« berechnen.
<code>partial_sum</code>	Iteratoren: Teilsumme berechnen.

<stdexcept>

Algorithmus	Beschreibung
<code>domain_error</code>	Für Exception-Objekte, die bei Auftreten unzulässiger Werte ausgelöst werden.
<code>invalid_argument</code>	Für Exception-Objekte, die bei Übergabe ungültiger Argumente ausgelöst werden.
<code>length_error</code>	Für Exception-Objekte, die ausgelöst werden, wenn die zulässige Größe von Datenstrukturen oder -typen überschritten wird.
<code>logic_error</code>	Für Exception-Objekte, die logische Fehler betreffen.
<code>out_of_range</code>	Für Exception-Objekte, die ausgelöst werden, wenn ein Argument nicht im vorgesehenen Bereich liegt.
<code>overflow_error</code>	Für Exception-Objekte, die bei arithmetischen Überläufen ausgelöst werden.
<code>range_error</code>	Für Exception-Objekte, die bei Verletzung von Gültigkeitsbereichen ausgelöst werden.
<code>runtime_error</code>	Für Exception-Objekte, die Laufzeitfehler betreffen.

<utility>

Klassen

Algorithmus	Beschreibung
pair	Template-Klasse für Strukturen mit zwei Elementen unterschiedlicher Datentypen.

Funktionen

Algorithmus	Beschreibung
make_pair	Pair-Objekt erzeugen.

<typeinfo>

Algorithmus	Beschreibung
bad_cast	Für Exception-Objekte, die bei gescheiterten dynamic_cast-Operationen ausgelöst werden.
bad_typeid	Für Exception-Objekte, die ausgelöst werden, um einen Null-Zeiger in typeid-Ausdrücken anzuzeigen.
type_info	Verwaltet Informationen (beschreibender String, Prioritätswert) zu Datentypen.

<valarray>

Die Headerdatei <valarray> definiert die Datenstruktur `valarray`, die ihre Elemente wie in einem Array verwaltet (der Speicher wird allerdings dynamisch verwaltet). Die Klasse `valarray` erlaubt eine Vielzahl von Operationen, die jeweils auf sämtlichen Elementen des Arrays ausgeführt werden, sowie die Möglichkeit mit Unterstützung ihrer Hilfsklassen Teilmengen zu bilden.

Algorithmus	Beschreibung
gslice	Hilfsklasse zur Definition einer mehrdimensionalen Teilmenge aus einem eindimensionalen <code>valarray</code> -Objekt.
gslice_array	Hilfsklasse zur Manipulation einer mehrdimensionalen Teilmenge aus einem eindimensionalen <code>valarray</code> -Objekt.

Algorithmus	Beschreibung
indirect_array	Hilfsklasse zur Manipulation einer Teilmenge eines valarray-Objekts. Die Teilmenge wird mittels eines Index-Arrays erzeugt.
mask_array	Hilfsklasse zur Manipulation einer Teilmenge eines valarray-Objekts. Die Teilmenge wird mittels einer Maske erzeugt.
slice	Hilfsklasse zur Definition einer Teilmenge konstanter Schrittweite aus einem valarray-Objekt.
slice_array	Hilfsklasse zur Manipulation einer Teilmenge konstanter Schrittweite aus einem valarray-Objekt.
valarray	Eindimensionales dynamisches Array mit einer Vielzahl von Operationen, die gleichzeitig auf allen Elementen ausgeführt werden.

Funktionen

Algorithmus	Beschreibung
abs	Betrag berechnen.
acos	Arkuskosinus berechnen.
asin	Arkussinus berechnen.
atan	Arkustangens berechnen.
atan2	Arkustangens berechnen.
cos	Kosinus berechnen.
cosh	Kosinus hyperbolicus berechnen.
exp	Exponentialfunktion berechnen.
log	Natürlichen Logarithmus berechnen.
log10	Logarithmus zur Basis 10 berechnen.
min	Minimum
max	Maximum
pow	Zahl potenzieren.
sin	Sinus berechnen.
sinh	Sinus hyperbolicus berechnen.
sqrt	Quadratwurzel berechnen.
tan	Tangens berechnen.
tanh	Tangens hyperbolicus berechnen.

D.2 STL und MFC im Vergleich

Viele der Funktionen der STL-Container-Klassentemplates sind in den Klassen und Templates der MFC reproduziert. Wie stehen diese beiden Technologien zueinander, und vor allem, welche sollte man anwenden?

Zum Vergleich bringt Listing D.1 ein einfaches Beispiel aus einem früheren Kapitel (Kapitel 26, »Container-Klassen«), das die MFC-Klassen `CString` und `CMapStringToString` verwendet, um ein einfaches interaktives Wörterbuch zu implementieren. Kompilieren Sie von der Kommandozeile mit

```
CL -MT MFCVOCAB.CPP
```

```
#include <afx.h>
#include <iostream.h>
#include <stdio.h>

void main(void)
{
    CString wrd, def;
    CMapStringToString map;
    CStdioFile inf(stdin);

    cout << "Vokabeln einlesen\n";
    while (TRUE)
    {
        cout << "Vokabel eingeben: ";
        cout.flush();
        inf.ReadString(wrd);
        if (wrd == "Q") break;
        cout << "Definition: ";
        cout.flush();
        inf.ReadString(def);
        map[wrd] = def;
    }
    if (map.IsEmpty())
    {
        cout << "Wörterbuch ist leer!\n";
        exit(0);
    }
    cout << "Wörterbuch enthält ";
    cout << map.GetCount() << " Einträge.\n";
    cout << "Nach Vokabeln suchen\n";
    while (TRUE)
    {
        cout << "Vokabel eingeben: ";
        cout.flush();
        inf.ReadString(wrd);
        if (wrd == "Q") break;
        if (map.Lookup(wrd, def))
            cout << def << '\n';
        else
            cout << "nicht gefunden!\n";
    }
}
```

*Listing D.1:
Einsatz der MFC-
Collection-
Klassen*

Dieses Programm füllt das Objekt `CMapStringToString` mit einer Reihe von Schlüssel/Wert-Paaren, die interaktiv vom Anwender erhalten werden. Wenn der Anwender ein `Q` als Schlüsselstring eingibt, wechselt das Programm in den Abfragemodus; jedes eingegebene Zeichen wird verglichen und der entsprechende Wert angezeigt.

Listing D.2 stellt genau dasselbe Programm dar, mit der STL-Container-Klasse `map` und der String-Klasse `string` implementiert. Kompilieren Sie mit

CL /GX STLVOcab.CPP



Das Flag /GX ist nötig, da STL intern C++-Ausnahmen verwendet.

Listing D.2:
Einsatz der
STL-Container-
Klassen

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

typedef map<string, string> vocabulary;

void main(void)
{
    string wrd, def;
    vocabulary voc;
    vocabulary::iterator itr;

    cout << "Vokabeln einlesen" << endl;
    while (true)
    {
        cout << "Vokabel eingeben: ";
        cout.flush();
        cin >> wrd;
        if (wrd == "Q") break;
        cout << "Definition: ";
        cout.flush();
        cin >> def;
        voc[wrd] = def;
    }
    if (voc.empty())
    {
        cout << "Wörterbuch ist leer!" << endl;
        exit(0);
    }
    cout << "Wörterbuch enthält ";
    cout << voc.size() << " Einträge." << endl;
    cout << "Nach Vokabeln suchen" << endl;
    while (true)
    {
        cout << "Vokabel eingeben: ";
        cin >> wrd;
        if (wrd == "Q") break;
        itr = voc.find(wrd);
        if (itr != voc.end())
```

```
        cout << itr->second << endl;
    else
        cout << "nicht gefunden!" << endl;
    }
}
```

Wie Sie sehen, ist die Struktur der beiden Programme identisch, die Unterschiede sind im Detail. Die STL-Klasse `string` unterstützt die Operatoren `<<` und `>>` mit `iostream`-Klassen; somit brauchen wir keine Funktion wie `CStdioFile::ReadString`. Ein weiterer bemerkenswerter Unterschied ist der Einsatz eines Iterators in der STL-Version zum Finden und Referenzieren des zutreffenden Wertes für den vom Anwender eingegebenen Schlüssel.

Da nun die Fähigkeiten so ähnlich sind, welche Bibliothek sollte man einsetzen?

Das erste Kriterium sollte die Vertrautheit mit einer der beiden Technologien sein. Falls Sie zum Beispiel mit MFC gut vertraut sind, STL jedoch nie eingesetzt haben, sind Sie besser dran, wenn Sie bei der bekannten Lösung bleiben.

Wenn Ihre Anwendung bereits MFC verwendet, werden Sie durchgehend MFC einsetzen wollen. Das hilft, mögliche Konflikte und Kompatibilitätsprobleme zu vermeiden.

Andererseits bietet STL auf einigen Gebieten weit mehr Unterstützung. Die Anwendungen werden damit auch kleiner, was wichtig ist, wenn sie zum Beispiel durch das Internet verteilt werden. Außerdem ist STL zu mehreren Plattformen kompatibel. Anwendungen, die nur STL und Standard-C++-Bibliotheksfunktionen verwenden (oder Standard C), können unter zahlreichen verschiedenen Betriebssystemen rekompiliert werden.

Zur CD-ROM

Anhang E

Auf der CD-ROM, die dem Buch beiliegt, befindet sich neben den Quelltexten der Beispielpprogramme und einer Auswahl nützlicher Zusatztools auch die Autoren-Edition des Visual-C++-6.0-Compilers.

Mit dieser Autoren-Edition können Sie die in diesem Buch vorgestellten Beispielpprogramme nachvollziehen und sich mit der Entwicklungsumgebung Visual C++ 6.0 vertraut machen. Die bedeutendste Einschränkung der Autoren-Edition gegenüber der Standard-Version des Visual-C++-Compilers betrifft die Erstellung der ausführbaren Dateien. Die Lizenzbestimmungen der Autoren-Edition erlauben es nicht, die mit dieser Visual-C++-Version erstellten Programme kommerziell zu vermarkten oder zu vertreiben. Vorbeugend wird daher in die erzeugten EXE-Dateien ein Dialogfeld aufgenommen, das vor dem Start der eigentlichen Anwendung erscheint und den Anwender darauf aufmerksam macht, daß das vorliegende Programm unrechtmäßig vermarktet wurde. Als Privatanutzer soll Sie das aber nicht stören.

Installation der Visual-C++-Autoren-Edition

Bevor Sie das Setup-Programm der Visual-C++-Autoren-Edition aufrufen, sollten Sie alle anderen Programme schließen. Speichern Sie vor allem Ihre Daten und Dateien, da Windows im Laufe der Installation neu gestartet wird.

Danach folgen Sie einfach den Anweisungen des Setup-Programms.

Auf zwei Dinge möchte ich Sie noch hinweisen:

1. Das Setup-Programm legt für Sie im Laufe der Installation die Batch-Datei *vcvars32.bat* an und speichert diese im BIN-Verzeichnis des Visual-C++-Compilers. In dieser Datei sind unter anderem die Pfade zu den Kommandozeilenversionen des Compilers und des Linkers enthalten.

Viele der Beispielprogramme in diesem Buch werden am einfachsten mit Hilfe der Kommandozeilenversion des Compilers erstellt. (Die entsprechenden Kommandozeilenaufrufe sind jeweils bei den Listings der Programme angegeben.) Dazu müssen Sie ein MS-DOS-Fenster öffnen, die Datei *vcvars32.bat* ausführen, in das Verzeichnis der Quelltextdatei wechseln und dann den Compiler aufrufen.

Sollte es dabei Schwierigkeiten geben, kann es daran liegen, daß der Umgebungsspeicher für die Pfadangaben aus der *vcvars32.bat*-Batchdatei nicht groß genug ist. Rufen Sie in einem solchen Fall im Systemmenü der MS-DOS-Eingabeaufforderung den Befehl **EIGENSCHAFTEN** auf und setzen Sie auf der Seite **SPEICHER** den Umgebungsspeicher herauf.

2. Gegen Ende des Visual C++-Setups können Sie noch die MSDN, das neue Microsoft-Hilfesystem, installieren lassen. Auch wenn für die Installation der MSDN zusätzliche 150 Mbyte benötigt werden, sollten Sie nicht auf die MSDN verzichten. (Bei Bedarf kann die MSDN auch zu einem späteren Zeitpunkt über das Setup-Programm im MSDN-Unterverzeichnis der CD-ROM installiert werden.)

Literatur

Anhang

F

Black, Uyless: TCP/IP & Related Protocols, Zweite Ausgabe. McGraw-Hill, New York, 1994.

Bronstein, I.N., und Semendayev, K.A.: Spravotchnik po Matematike. Gostechizdat, Moskau, 1954; Hungarian translation, Matematikai Zsebkönyv, Muszaki Könyvkiadó, Budapest, 1963.

Date, C.J.: A Guide to the SQL Standard, Zweite Ausgabe, Addison-Wesley Publishing Company, Reading, MA, 1989.

Ellis, Margaret A., und Stroustrup, Bjarne: The Annotated C++ Reference Manual, Addison-Wesley Publishing Company, Reading, MA, 1991.

Kernighan, Brian W., und Ritchie, Dennis M.: The C Programming Language, Zweite Ausgabe, Prentice Hall, Englewood Cliffs, NJ, 1988.

Louis, Dirk: C/C++-Kompendium, Markt&Technik, München, 1998.

Microsoft Corporation, Microsoft Developer Network Developer Library, January, 1997.

Nelson, Mark: C++ Programmers Guide to the Standard Template Library, IDG Books Worldwide, Foster City, CA, 1995.

Petzold, Charles: Programming Windows 95, Microsoft Press, Redmond, 1996.

Press, William H., Teukolsky, Saul A., Vetterling, William T. und Flannery, Brian P.: Numerical Recipes in C, Second Edition, Cambridge University Press, Cambridge, United Kingdom, 1992.

Shirley, John, und Ward Rosenberry. Microsoft RPC Programming Guide, O'Reilly & Associates, Sebastopol, CA, 1995.

Stevens, W. Richard. UNIX Network Programming, Prentice Hall, Englewood Cliffs, NJ, 1990.

Stroustrup, Bjarne. The C++ Programming Language, Dritte Ausgabe, Addison-Wesley Publishing Company, Reading, MA, 1991.



Stichwortverzeichnis

A

Abbildungsmodi 213, 485

- MM_ANISOTROPIC 214
- MM_HIENGLISH 214
- MM_HIMETRIC 214
- MM_ISOTROPIC 214
- MM_LOENGLISH 214
- MM_LOMETRIC 214
- MM_TEXT 214
- MM_TWIPS 214

Abfragen 809

Access-Datenbank erstellen 763

ActiveX-Datenobjekte 797

ActiveX-Steuerelemente 577, 667, 717

- Anwendermodus 721
- Anwendungsgerüst 669
- ATL 697
- beständige Eigenschaften 682, 686
- Bitmap 683
- DDP_-Funktionen 693
- distribuieren 695
- Eigenschaften 682, 684, 704, 718
- Eigenschaftsseiten 674, 691
- Entwurfsmodus 721
- Ereignisse 682, 688, 708, 718
- erstellen 668
- Get-/Set-Methoden 683
- Klassen-Assistent 69, 684
- Lizenzierung 670
- mehrere Eigenschaftenseiten 694
- Methoden 682, 687, 706, 718

- MFC-Unterstützung 667

- mit ATL erstellen 699

- Nachrichten 726

- PX_-Funktionen 687

- Registrierung 673

- schreibgeschützte Eigenschaften 683

- Standardeigenschaften 686

- Test-Container 710

- testen 694

- verwenden 695, 717

- Vorteile der ATL 697

- Zeichnen 682, 689

ADO 797

- Objekte 797

- versus OLE DB 781

Adressen 895

- logische 272

- physikalische 272

Adreßformat, kanonisches 895

Adreßräume 271, 273

AfxEnableControlContainer 719

AfxMessageBox 88

AfxWinMain 382

Algorithmen (STL) 1072

Animation 460, 988

ANSI C++-Bibliothek 1057

- Algorithmen 1072

- Container 1058

- Funktionsobjekte 1077

- Iteratoren 1068

- Sonstige Klassen 1083

- Stream-Klassen 1078
- String-Klassen 1082
- ANSI-C-Bibliothek 1043
- Ansichten 403, 414
 - Ansichtsklasse deklarieren 414
 - CView-Elementfunktionen 416
 - GetDocument 416
 - OnDraw 416
 - überschreibbare Funktionen 416
 - und Nachrichten 418
 - Varianten 418
- Ansichtsklasse 394
- Anwendungsobjekt 381
 - InitInstance 384
 - Run 386
- API (Application Programming Interfaces) 123
- Arbeitsbereiche 42
 - Arbeitsbereichfenster 23
 - konfigurieren 43
- ARP 869
- ASSERT 89
- Assistenten 51, 56
 - Anwendungsassistent 52, 379
 - ATL-COM-Anwendungs-Assistent 700
 - eigene Assistenten schreiben 995
 - Klassen-Assistent 58, 401
 - MFC-ActiveX-Assistent 669
- Assistentenleiste 27
- ATL (ActiveX-Templatebibliothek) 697
 - ActiveX-Steuerelement erstellen 699
 - Anwendungsgerüst 700
 - Bitmap 715
 - Eigenschaften hinzufügen 704
 - Eigenschaftsseiten 711
 - Ereignisse hinzufügen 708
 - Methoden hinzufügen 706
 - Projekte testen 710
 - Steuerelement zeichnen 707
 - Verbindungspunkte unterstützen 703, 708
- ATL-COM-Anwendungs-Assistent 700
- Audio Compression Manager 952
- Ausgabefenster 26
- Auslagerungsdatei 274
- Ausnahmebehandlung 341
 - __except 343
 - __try 343
 - Filterausdruck 343
 - in C 342
 - in C++ 347
 - strukturiert 341
 - Überwachungsblock 343
 - universeller C++-Bearbeiter 349
- Ausnahmen
 - abfangen 342
 - C 342
 - C++-Klassen 349
 - CArchiveException 546
 - C-Ausnahmen übersetzen 352
 - CDaoException 548
 - CDBException 548
 - CException 543
 - CFileException 545
 - CInternetException 549
 - CMemoryException 544
 - CNotSupportedException 547
 - COleDispatchException 549
 - COleException 549
 - CResourceException 547
 - CUserException 550
 - in der MFC 542
 - Multithreading 551
 - verschachtelte 343
- Automatisierung 576, 645
 - Anwendungsgerüst 646
 - DISPATCH-Tabelle 654
 - Eigenschaften 651
 - IDispatch 659
 - Klassen-Assistent 67, 651
 - Methoden 654
 - Standardeigenschaften 659
 - Standardobjekte 659
 - Typenbibliothek 656
 - von Objekten 659
- AVI 937
- AVIFile 951
- AVIStream 951
- B**
- Befehlsverben 590
- Benutzerdefinierte Assistenten 995
- Bereiche 497
- Bibliotheken 263
- Bibliothekmanager 36
- Bilderliste 464

BitBlt 238, 489
Bitmaps 230, 237, 496
– BitBlt 238
– geräteabhängige 230
– geräteunabhängige 230, 238
– LoadBitmap 230
– Ressourcen 194

C

CArchive 501, 507
CAsyncSocket 883
CBitmap 487, 496
CBrush 487, 495
CClientDC 481
CCmdUI 560
CCommandLineInfo 559
CCommonDialog 452
CCreateContext 559
CCriticalSection 555
CDAODatabase 777
CDAOException 779
CDAOFieldExchange 779
CDAOQueryDef 778
CDAORecordset 776
CDAORecordView 777
CDAOTableDef 779
CDAOWorkspace 778
CDatabase 756
CDataExchange 560
CDC 479
CDialog 423
CDocItem 412
CDocument 403, 864
CDocument, siehe Dokumente
CEvent 554
CException 543
CFieldExchange 759
CFile 501
CFileStatus 559
CFont 487, 497
CFtpConnection 843
CFtpFindFile 846
CGopherConnection 843
CGopherFindFile 846
CGopherLocator 846
Child-Fenster 150
ChoosePixelFormat 963
CHttpConnection 843

CInternetException 846
CInternetSession 842
Client für Remote Procedures 925
Client-Server-Modell 956
Clipping 231, 492
Clipping-Pfade 231
CMC (Common Messaging Calls) 860
CMemFile 506
CMemoryState 90
CMemoryStatus 559
CMetaFileDC 482
CMultiLock 556
CMutex 555
CObject 359, 517
COleClientItem 630
COleServerItem 630
COleStreamFile 506
COM 565
Compiler 34
– Browser-Informationen 76
– Debug-Informationen 78
– MIDL-Compiler 656, 921, 923
– Vorkompilierte Header 47
Container 517, 607
– Ansichtsklasse 614
– CArray-Template 535
– CByteArray 525
– CDWordArray 525
– CList-Template 533
– CMapPtrToPtr 529
– CMapPtrToWord 530
– CMapStringToOb 529
– CMapStringToPtr 529
– CMapStringToString 526
– CMap-Template 536
– CMapWordToOb 530
– CMapWordToPtr 530
– CObArray 522
– CObject 517
– CObList 518
– Container-Objekte 612
– Container-Templates 531
– CPtrArray 525
– CPtrList 524
– CStringArray 525
– CStringList 524
– CTypedPtrArray-Template 538
– CTypedPtrList-Template 538

- CTypedPtrMap-Template 539
- CUIntArray 525
- CWordArray 525
- MFC-Anwendungsgerüst 608
- Objekte auswählen 624
- Objekte zeichnen 623
- Objektpositionen 621
- Ressourcen 619
- Zuordnungen 526
- Container (STL) 1058
- CPaintDC 481
- CPalette 497
- CPen 487, 494
- CPoint 557
- CPrintInfo 559
- CPropertyPage 439
- CPropertySheet 439
- CreateDC 209, 479
- CreateFile 297, 916
- CreateThread 256
- CreateWindow 155
- CreateWindowEx 157
- CRecordset 757
- CRecordView 759
- CRect 557
- CRectTracker 560, 625
- CRgn 487, 497
- CSemaphore 555
- CSingleDocTemplate 385
- CSingleLock 556
- CSize 557
- CSocket 885
- CSocketFile 507
- CStdioFile 505
- CString 557
- CTime 558
- CView 414
- CView, siehe Ansichten
- CWaitCursor 560
- CWindowDC 482
- D**
- DAO 761
 - Abfragedefinitionsobjekte 776
 - Anwendungsgerüst 766
 - CDaoDatabase 777
 - CDaoException 779
 - CDaoFieldExchange 779
 - CDaoQueryDef 778
 - CDaoRecordset 776
 - CDaoRecordView 777
 - CDaoTableDef 779
 - CDaoWorkspace 778
 - Datenaustausch 770
 - Datensätze filtern 773
 - DFX_-Funktionen 770
 - Elementvariablen für Datenbankfelder 772
 - Klassen 775
 - SQL-Anweisungen 761
 - Tabellendefinitionsobjekte 776
 - Übersicht 761
- Data Access Objects 761
- Dateien 295
 - CFile 501
 - Dateiobjekte 297
 - Dateiverwaltung 503
 - E/A-Funktionen 295
 - erstellen 297
 - Fehlerbearbeitung 504
 - in C 307
 - in C++ 307
 - Lesen und Schreiben 503
 - MFC-Unterstützung 501
 - öffnen 297, 502
 - Sicherheitsattribute 298
 - Speicherabbilder 506
 - sperren 504
- Dateisysteme 296
 - FAT 296
 - FAT32 296
 - NTFS 296
 - VFAT 296
- Datenbank-Assistent 816
- Datenbanken
 - Abfragen erstellen 809
 - Access 763
 - anlegen 816
 - Beziehungen einrichten 820
 - Daten hinzufügen 808
 - Datenbank-Projekt 804
 - erstellen 803
 - Excel-Tabellen 737
 - Gespeicherte Prozeduren 821
 - Primärschlüssel 819

- Tabellen anlegen 818
- Textdateien 747
- Datenbankformulare 754
- Datenbankprogrammierung
 - Abfragen 809
 - ADO 781, 797
 - Datenbanken erstellen 803
 - Datenbankformulare 754
 - Datenbank-Projekt 804
 - Datenbank-Tools 803
 - Datenquelle 732
 - Datenquellen einrichten 732
 - Datensätze filtern (DAO) 773
 - mit DAO 761
 - OLE DB 782
 - SQL 731
 - SQL Server 811
 - Treiber 731, 735
- Datenbank-Tools 803
 - Abfragen erstellen 809
 - Daten hinzufügen 808
- Datenquellen 732
- Datentransfer 573
- Datenzugriffsobjekte 761
 - Hierarchie 761
 - siehe auch DAO
- Datum/Zeitauswahl 473
- Debug-Fenster 26
 - Aufrufliste 83
 - Disassemblierung 85
 - Editorfenster 86
 - Register 85
 - Speicher 83
 - Überwachung 82
 - Variablen 84
- Debugger 77
 - Debug-Fenster 82
 - Haltepunkte 79
 - Just-In-Time-Testläufe 93
 - Remote-Testlauf 91
 - starten 78
- Debugging
 - Anwendungen debuggen 78
 - Bearbeiten und fortfahren 87
 - Debug-Sitzung anhalten 79, 81
 - DLLs debuggen 79
 - Informationen anzeigen 82
 - Schnellüberwachung 86
 - Vorbereitungen zum Debuggen 77
- Debug-Monitor 92
- Debug-Techniken 88
 - ASSERT 89
 - CMemoryState 90
 - Meldungsfenster 88
 - Objekt-Dump 89
 - TRACE 88
- DECLARE_DYNAMIC 363
- DECLARE_DYNCREATE 363
- DECLARE_SERIAL 363
- DefDlgProc 141
- DefWindowProc 141
- Dialoge 423
 - ActiveX-Steuerelemente hinzufügen 720
 - als Hauptfenster 53
 - aufrufen und anzeigen 430
 - benutzerdefinierte Datentypen 437
 - CDialog 423
 - CDialog-Objekt 423
 - DefDlgProc 141
 - Dialog-Datenaustausch 434
 - Dialog-Datenüberprüfung 435
 - Dialogklasse erstellen 425
 - DoDataExchange 435
 - DoModal 432
 - Einheiten 196
 - Elementvariablen zu Steuerelementen 427
 - erstellen 424
 - Klassen-Assistent 64, 425
 - modale 170, 432
 - Multimedia-Dialoge 940
 - Nachrichtebearbeitung 438
 - nicht-modale 171, 432
 - Registerdialoge 439
 - Ressourcen 196
 - Standarddialoge 174, 451
 - Steuerelement-Datentypen 437
- Dialogfelder 170
 - Dialogfeldprozedur 173
 - Dialogvorlagen 173
- Dialogvorlagen 424
- DIBs 230, 238
- DirectInput 974
- DirectX 974
 - Animation 988
 - Direct3D 974, 978
 - DirectDraw 974, 976

- DirectInput 982
- DirectPlay 974, 980
- DirectSetup 975, 982
- DirectSound 974, 979
- Sound 988
- und COM 975
- DispatchMessage 138
- DLLMain 264
- DLLs (Dynamische Linkbibliotheken) 263
 - debuggen 79
 - DllMain 264
 - Eintrittsfunktion 264
 - Funktionen exportieren 265
 - Funktionen importieren 266
 - laden 265
 - Ressourcen-DLLs 202
- Doc/View 53
- DoDataExchange 435
- Dokumente 403
 - CDocument-Elementfunktionen 405
 - Datenelemente 409
 - Dokumentklasse deklarieren 404
 - Dokumentvorlagenzeichenfolge 397
 - Serialisierung 411
 - überschreibbare Funktionen 408
 - und Nachrichten 407
 - Unterstützung für Datei-Dialoge 397
- Dokumentklasse 391
- Dokumentobjekte 595
- Dokumentvorlage 385
- Dokumentvorlagenzeichenfolge 397
- Domänensystem 872
- DoModal 432
- Drag&Drop 577, 629
 - Anwendungsgerüst 630
 - Drag-Quellen 629, 635
 - Drop-Ziele 629, 636
 - Nachrichtenbehandlungsfunktionen 637
 - Objekte verschieben 641
 - OnDragEnter 638
 - OnDragLeave 640
 - OnDragOver 639
 - OnDrop 640
- Drag-Quellen 629, 635
- DrawDib 952
- DrawText 239
- Drehregler 468

Drop-Ziele 629, 636

Drucken 240, 492

E

Eigenschaften 704

Eigenschaftsdialog 691

Eigenschaftsseiten 674

Ein- und Ausgabe 298

- asynchrone E/A 300
- Dateien 298, 501
- in C 307
- in C++ 307
- Konsole 309
- Low-Level 305
- Serialisierung 501
- serielle Schnittstelle 300
- überlappend 302

Ein- und Ausgabestrom 307

eingebettete Objekte 575

eingebettete SQL 735

EnumChildWindows 152

EnumThreadWindows 152

EnumWindows 152

Ereignis-Objekte 258, 554

Ereignisse 708

Exceptions, siehe Ausnahmen

F

FAT32-Dateisystem 296

FAT-Dateisystem 296

Fenster 149

- Ansichtsfenster (MFC) 366, 414
- CreateWindow 155
- DefWindowProc 141
- erzeugen 141
- Fensterfunktion 114, 139, 141, 162
- Fensterklasse registrieren 116
- Fensterklassen 116, 141, 162
- Fensterstile 156
- Hauptfenster (MFC) 385, 389
- Hierarchie 150
- Klassenstile 154
- mehrere Fensterfunktionen 143
- MFC-Klassen 364
- Rahmenfenster 365
- registrieren 153
- Standard-Fensterfunktion 141

- Subklassen 117, 164
- Superklassen 117, 168
- teilbare Fenster 366
- übergeordnet 151
- untergeordnet 151
- Verwaltungsnachrichten 160
- Vorort-Rahmenfenster 595
- WM_PAINT-Nachricht 142
- WNDCLASS-Struktur 153
- Z-Reihenfolge 150
- zugeordnet 151
- Fensterfunktion 114
- OpenGL 964
- Figuren 488
- FindWindow 152
- Fortschrittsleiste 465
- Freihandzeichnen 144
- FTP 828
- HTTP-Zugriff 837
- MFC-Zugriff 844, 847
- WinInet-Zugriff 834
- Funktionen, virtuelle 566
- Funktionsobjekte (STL) 1077

G

- GDI (Graphics Device Interface) 207, 477
- Abbildungsmodi 213, 485
- Bereiche 237, 497
- BitBlt 489
- Bitmap-Objekte 237
- Bitmaps 230, 496
- CBitmap 487, 496
- CBrush 487, 495
- CDC 479
- CDC-Attribute 483
- CFont 487, 497
- Clipping 231, 492
- CPalette 497
- CPen 487, 494
- CRgn 487, 497
- Drucken 492
- Figuren 237, 488
- gdi.dll 207
- GDI-Objekte 222, 487, 493
- Gerätekontext 208, 209, 478
- Gerätetreiber 207
- Koordinaten 211, 485

- Kurven 236
- Linien 234, 488
- Paletten 225, 497
- Pfade 239, 493
- Pinsel 223, 495
- Pixeloperationen 489
- Schriftarten 224, 497
- SelectObject 222
- Stifte 223, 494
- Textausgabe 239, 490
- Viewport 211
- Zeichenfunktionen 234, 486
- GDI-Objekte 493
- Gerätekontexte 126, 208, 478
- Arten 209
- Ausgabe in 209
- CDC-Attribute 483
- erstellen 479
- erzeugen 209
- Füllmodus 484
- Hintergrundfarbe 483
- Hintergrundmodus 483
- Informationsgerätekontexte 211
- löschen 480
- Metadateien 482
- Metadatei-Gerätekontexte 210
- MFC-Klassen 481
- SelectObject 222
- Speichergerätekontexte 210
- Zeichenmodus 483
- zurücksetzen 487
- Gerätetreiber 207
- Gespeicherte Prozeduren 821
- Get-/Set-Methoden 683
- GetClassInfo 117
- GetDC 209
- GetDesktopWindow 152
- GetLastError 130
- GetMessage 121, 138
- GetOpenFileNamePreview 940
- GetParent 152
- GetSaveFileNamePreview 940
- GetWindow 152
- GetWindowDC 209
- Global Unique Identifiers 569
- Gopher 830
- MFC-Zugriff 845, 848

- WinInet-Zugriff 836
- Grafik 207
- Animationen 988
 - DirectX 973
 - OpenGL 955
- Grafik, siehe GDI (Graphics Device Interface)
- GUID 569, 921

H

- Haltepunkte 79
- deaktivieren 80
 - löschen 80
 - setzen 79
 - spezielle Haltepunkttypen 80
- Hayes AT-Befehlssatz 896
- HeapCreate 286
- Heap-Funktionen 286
- HKEY_CLASSES_ROOT 328, 331
- HKEY_CURRENT_CONFIG 328
- HKEY_CURRENT_USER 328, 332
- HKEY_DYN_DATA 328
- HKEY_LOCAL_MACHINE 328, 330
- HKEY_USERS 328, 332
- Hostnamen 872
- HTTP 832
- MFC-Zugriff 845, 849

I

- ICMP 869
- IDispatch 659
- IMPLEMENT_DYNAMIC 363
- IMPLEMENT_DYNCREATE 363
- IMPLEMENT_SERIAL 363
- Informationsgerätekontext 211
- INI-Dateien 332
- InitInstance 384
- In-Place-Bearbeitung (Vorortbearbeitung) 590
- Internet-Programmierung
- blockierende Aufrufe 879
 - Byte-Anordnung 876
 - Domänensystem 872
 - FTP 828
 - Gopher 830
 - Hostnamen 872
 - HTTP 832
 - IP-Adressen 870
 - IP-Datagramme 869
 - IP-Header 870
 - MFC-Klassen 883

- Portnummern 873
 - Protokolle 828
 - Routing 871
 - Sockets 873
 - WinInet 827
 - WinInet-Bibliothek 833
 - WinSock 867, 874
- Internet-Protokolle 828, 868
- ARP 869
 - FTP 828
 - Gopher 830
 - HTTP 832
 - ICMP 869
 - IP 869
 - RARP 869
 - TCP 868
 - UDP 869
- IP 869
- IP-Adresse 473
- IP-Adressen 870
- IP-Datagramm 869
- IP-Header 870
- ISDN 894
- Iteratoren (STL) 1068
- IUnknown-Schnittstelle 569

K

- Klassen-Assistent 401
- ActiveX-Ereignisse 69
 - ActiveX-Steuerelemente 684
 - Automatisierung 67, 651
 - Bearbeitungsfunktionen einrichten 62
 - Dialoge 425
 - Klasseninformationsdatei (clw) 71
 - Member-Variablen für den Datenaustausch 64
 - neue Klassen anlegen 60
 - virtuelle Elementfunktionen 64
- Klasseninformationsdatei (clw) 60, 71
- Kommandozeilen-Tools 33
- Kommunikationsschnittstellen 310
- Komponenten, eigene erstellen 107
- Komponentenobjektmodell 565
- Komponentensammlung 105
- Komprimierungsmechanismen 297
- Konfiguration
- Arbeitsbereiche 43
 - Projekte 46
- Konsole 309

Koordinatentransformation 215
Kritische Abschnitte 259, 555

L

Laufzeitbibliotheken
– ANSI C 1043
– ANSI C++ 1057
 Algorithmen 1072
 Container 1058
 Funktionsobjekte 1077
 Iteratoren 1068
 Sonstige Klassen 1083
 Stream-Klassen 1078
 String-Klassen 1082
Laufzeittypinformationen 359, 362
LINE_CALLSTATE 910
lineMakeCall 905
Linker 35
– Debug-Informationen 78
– Einstellungen für Profiler 98
Listenansicht 463
Lokaler Thread-Speicher 291

M

m_pMainWnd 385
Mail 851
Makros
– DECLARE_DYNAMIC 363, 514
– DECLARE_DYNCREATE 363, 513
– DECLARE_SERIAL 363, 514
– der MFC 363
– für die Serialisierung 513
– IMPLEMENT_DYNAMIC 363, 514
– IMPLEMENT_DYNCREATE 363, 513
– IMPLEMENT_SERIAL 363, 514
malloc 279
MAPI 851
– Anwendungsassistent 865
– Architektur 852
– CMC 856, 860
– Dienstanbieter 854
– Erweitertes MAPI 857, 861
– Formen 862
– MAPI-Profile 855
– MAPI-Spooler 855
– MFC-Unterstützung 864
– OLE-Nachrichten 857, 862
– Simple Mail 856, 857

– Typen von MAPI-Anwendungen 853
Marshaling 570, 919
Mausbehandlung 139
Mauszeiger, Ressourcen 194
MCI 937, 945
– Befehlssatz 949
– Benachrichtigungen 950
– Funktionen 950
MCIWnd 938
– Benachrichtigungen 945
– Funktionen 939
– Makros 941
MCIWndCreate 933
MDI 53
Mediensteuerschnittstelle, siehe MCI
Mehrfachvererbung 363
Meldungsfenster 88, 172
Menüs, Ressourcen 198
Message Loop 113
MessageBox 172
Metadateien 482
Metadatei-Gerätekontext 210
Methoden 566, 706
MFC (Microsoft Foundation Classes)
– ActiveX-Unterstützung 667
– AfxWinMain 382
– Ansichten 403
– Ansichtsfenster 366
– Ansichtsklasse 394
– Anwendungsarchitekturklassen 370
– Anwendungsgerüst 379
– Anwendungsgerüst manuell bearbeiten 398
– Anwendungsgerüst mit Klassenassistent
 bearbeiten 401
– Anwendungsobjekte 371, 381
– Ausnahmenbehandlung 542
– CDocument 403
– CObject 359
– Container 517
– Container-Klassen 375
– CView 414
– DAO-Klassen 775
– Dialoge 367
– Dokumente 403
– Dokumentklassen 370, 391
– Dokumentobjekte 372
– Dokumentvorlagen 371, 385
– Fensterklassen 364

- FTP-Verbindungen 844, 847
 - Gopher-Verbindungen 845, 848
 - Grafikunterstützungsklassen 373
 - Grundlagen 359
 - Hauptfenster 385, 389
 - Hilfsklassen 557
 - HTTP-Verbindungen 845, 849
 - Internet-Dateien 846
 - Internet-Sitzungen 842
 - Internet-Unterstützungsklassen 841, 883
 - Internet-Verbindungen aufbauen 842
 - Laufzeittypinformationen 359, 362
 - m_pMainWnd 385
 - Makros 363
 - Mehrfachvererbung 363
 - Nachrichtenzuordnungstabelle 370, 386
 - ODBC-Klassen 756
 - ODBC-Unterstützung 746
 - OLE DB-Unterstützung 789
 - Rahmenfenster 365
 - Serialisierung 359, 392
 - Standarddialoge 451
 - Standardsteuerelemente 460
 - Steuerelemente 370
 - Systemunterstützungsklassen 374
 - Thread-Sicherheit 541, 551
 - Übersicht 357
 - und WinMain 382
 - Windows-Objekte 364
 - MFC-Anwendungsassistent 379
 - MFC-Bibliothek 357
 - Microsoft Access 763
 - Microsoft SQL Server 811
 - MIDI 936
 - MIDL-Compiler 656, 921, 923
 - MODL (Microsoft-Objektdefinitionssprache) 656
 - Monatskalender 473
 - Moniker 571
 - Multimedia 933
 - Audio Compression Manager 952
 - Audiodateien 936
 - AVICap 952
 - AVIFile 951
 - AVIStream 951
 - Befehlsnachrichten 947
 - Befehlsstrings 946
 - Dateidialoge 940
 - Dateiformate 936
 - DrawDib 952
 - kundenspezifische Handler 951
 - MCI 937, 945
 - MCI-Befehlssatz 949
 - MCI-Benachrichtigungen 950
 - MCI-Funktionen 950
 - MCIWnd 937, 938
 - MCIWnd-Benachrichtigungen 945
 - MCIWnd-Funktionen 939
 - MCIWnd-Makros 941
 - MIDI-Dateien 936
 - PlaySound 952
 - Schnittstellen 937
 - Video Compression Manager 952
 - Videoaufnahmen 952
 - Videos 934, 937
 - Multitasking 121, 244
 - Design-Ziele 244
 - Kontextschalter 244
 - kooperatives 245, 246
 - präemptives 245, 246
 - Win95/98 248
 - Multithreading 120, 551
 - MFC 551
 - Prozesse 121
 - Synchronisierung 553
 - Threads 121, 122
 - Mutex-Objekte 259, 555
- N**
- Nachrichten 112, 113
 - eigene Nachrichten definieren 118
 - Fensterfunktion 139
 - MAPI-Programmierung 851
 - Mausbehandlung 139
 - Message Loop 112
 - Nachrichtenschleife 137
 - senden 115
 - senden und hinterlegen 137
 - Standardverarbeitung 141
 - Threads 122
 - Typen 117
 - Umgehung der MessageLoop 114
 - Verfolgung mit Spy++ 118
 - von ActiveX-Steuerelementen 726
 - weiterleiten 138
 - WM_ACTIVATE 119, 161

- WM_CLOSE 160
- WM_COMMAND 118
- WM_CREATE 160
- WM_DESTROY 120, 160
- WM_ENABLE 161
- WM_ENDSESSION 161
- WM_GETTEXT 161
- WM_KILLFOCUS 119, 161
- WM_LBUTTONDOWN 119
- WM_LBUTTONUP 119
- WM_MOVE 161
- WM_NCACTIVATE 119, 162
- WM_NCCREATE 162
- WM_NCDESTROY 120, 162
- WM_NCPAINT 162
- WM_PAINT 119, 142, 158
- WM_QUERYENDSESSION 160
- WM_QUIT 160
- WM_SETFOCUS 161
- WM_SETTEXT 161
- WM_SHOWWINDOW 161
- WM_SIZE 161
- WM_WINDOWPOSCHANGED 119
- WM_WINDOWPOSCHANGING 119
- Nachrichtenwarteschlange 112
- Nachrichtenzuordnungstabellen 61, 386
- Netzwerklaufwerke 297
- Netzwerkprogrammierung 913
 - Pipes 913
 - Remote Procedure Calls 919
- Neuerungen 38
- new 279
- NMAKE 36
- NTFS-Dateisystem 296

O

- Objekt-Dump 89
- Objekte
 - eingebettete 575
 - verknüpfte 574
- ODBC 731
 - Anwenderdatenquellen 732
 - Anwendungsgerüst 749
 - API-Funktionen 736, 739, 740
 - API-Kern 735
 - API-Level-1 735
 - API-Level-2 735
 - Dateidatenquellen 732

- Datenquelle 732
- Datenquellen einrichten 732, 746
- Ebenen 735
- in MFC-Applikationen 746
- MFC-Klassen 756
- Setup 732
- SQL Server-Einrichtung 812
- Systemdatenquellen 732
- Treiber 731, 735
- und SQL 735
- OLE 565
 - Apartment-Modell 572
 - Automatisierung 576, 645
 - Container 607
 - Container, siehe Container
 - Datentransfer 573
 - Drag&Drop 577, 629
 - Einbettung 575
 - GUID 569
 - IUnknown 569
 - Komponentenobjektmodell 565
 - Marshaling 570
 - Methoden 566
 - Moniker 571
 - Registrierung 570, 586
 - Schnittstellen 566
 - Schnittstellenbezeichner 569
 - Server 575, 589
 - Server, siehe Server
 - Speichern von Verbunddokumenten 572
 - Speicherreservierung 567
 - Verbunddokumente 572, 574
 - Vererbung 568
 - Verknüpfung 574
- OLE DB 782
 - Anwendungsgerüst 789
 - Grundkonzepte 782
 - OLE DB-MFC-Beispiel 789
 - OLE DB-SDK 782
 - OLE DB-SDK-Beispiel 784
 - OLE DB-Templates 782
 - versus ADO 782
- OnDraw 969
- OpenGL 955
 - API-Anwendungen 961
 - Bibliotheken 960
 - Fenster 958
 - Fensterfunktion 964

- Initialisierung 958
- Initialisierung (API) 963
- Initialisierung (MFC) 966
- Kompilation 965
- MFC-Anwendungen 965
- Pixeldaten-Modi 958
- Primitive 956
- Projekteinstellungen 965
- Rendering-Kontext 958
- Vertex 956
- Zeichenfähigkeiten 956
- Zeichnen 959
- Optimierung 97
 - mit dem Compiler 97
 - mit dem Profiler 98
 - mit dem VS Analyzer 103

P

- Paletten 225, 497
 - logische 226
 - realisieren 227
 - Systempalette 226
- Palettenanimation 227
- PeekMessage 121
- Pipes 913
 - anonyme Pipes 914
 - Anschluß an Pipes 915
 - Datentransfer 916
 - erzeugen 914
 - Named Pipes 914
- Playback 935
- PlaySound 952
- Portnummern 873
- PostMessage 115
- POTS 893
- Primärschlüssel 819
- Primitive 956
- Profiler 98
 - Anwendung vorbereiten 98
 - Batch-Dateien 102
 - Daten absichern 101
 - Einstellungen 101
 - einzelne Module analysieren 102
 - einzelne Zeilen analysieren 102
 - PLIST 99
 - PREP 99
 - PROFILE 99
 - Profiltypen 99

- Tips zur Analyse 98
- Programmausführung unterbrechen 81
- Programme
 - API-Anwendungsgerüst 135
 - dialogbasierte Anwendungen 419
 - Fensterfunktion 114
 - MAPI-Unterstützung 853
 - MDI 386
 - MFC 359
 - MFC-Anwendungsgerüst 379
 - Nachrichtenschleife 137
 - Terminierung in C 346
 - Terminierung in C++ 348
 - Warteschleifen 113
- Programmerstellung 28
 - Dialog als Hauptfenster 53
 - MDI 53
 - OLE-Container 607
 - OLE-Server 590
 - SDI 53
- Projekte 41
 - Arbeitsbereiche 42
 - bearbeiten 45
 - Browser-Informationen 76
 - Debug-Informationen 77
 - Einstellungen für Profiler 98
 - erstellen 45
 - Klassen-Assistent 401
 - Konfigurationen 48
 - konfigurieren 46
 - MFC-Anwendungsassistent 379
 - Projekttypen 56
 - Unterprojekte 43
- Protokolle
 - Internet 828, 868
 - verbindungsloses 869
 - verbindungsorientiertes 869
- Proxy-Objekte 571
- Prozesse 121
 - Speicherverwaltung 272
- Prozess-Server 575

Q

- Quellcode-Browser 75
 - Browser-Informationen 76
 - Einsatz 76
- Quelltexteditor 25
- Quickinfo 471

R

- RARP 869
- ReadFile 298
- RegisterClass 116, 153
- Registerdialoge 439
 - CPropertyPage 439
 - CPropertyPage-Elementfunktionen 445
 - CPropertySheet 439
 - erstellen 439
 - nicht-modale 446
 - Seiten anlegen 440
 - Seiten erzeugen 442
- Register-Steuerelement 468
- Registrierung 325, 570
 - ActiveX-Steuerelemente 673
 - bearbeiten 329
 - HKEY_CLASSES_ROOT 328, 331
 - HKEY_CURRENT_CONFIG 328
 - HKEY_CURRENT_USER 328, 332
 - HKEY_DYN_DATA 328
 - HKEY_LOCAL_MACHINE 328, 330
 - HKEY_USERS 328, 332
 - INI-Dateien 332
 - Registrierungseditor 329
 - Schlüssel 325
 - Schlüssel erstellen 334
 - Schlüssel öffnen 333
 - vordefinierte Schlüssel 327
 - Wert abfragen 334
 - Wert setzen 334
 - Werte 326
 - Werttypen 327
- Remote Procedures 919
 - abgeleitete Schnittstellen 929
 - Aufruf 927
 - Ausnahmebehandlung 927
 - Client 925
 - Grundlagen 919
 - Marshaling 919
 - Namensdienst 929
 - Protokolle 924
 - Schnittstellenangaben 921
 - Server 923
 - Speicherverwaltung 925
 - Stub-Funktionen 919
 - Zeiger als Argumente 928
- Ressourcen 191
 - benutzerdefinierte 201
 - Bitmaps 194
 - Dialoge 196
 - lokalisieren 203
 - Mauszeiger 194
 - Menüs 198
 - mit gleichem Bezeichner 398
 - Ressourcen-DLLs 202
 - Ressourcevorlagen 204
 - Schriften 194
 - Symbole 194
 - Symbolleisten 200
 - Tastaturkürzel 195
 - Versionsinformationen 200
 - Zeichenfolgentabellen 199
- Ressourceneditor 26
- Ressourcenskripte 192
 - kompilieren 202
- RFCs 874, 886
- Routing 871
- RPC 913
- RPC, siehe Remote Procedures
- RTF-Steuerelement 466
- RTTI 359
- Run 386

S

- Schieberegler 467
- Schnittstellen 566
 - benutzerdefinierte 577
 - für Remote Procedures 921
 - IDispatch 659
- Schriftarten 224, 497
- Schriften
 - in Gerätekontexten 490
 - Ressourcen 194
- SDI 53
- SelectObject 222
- Semaphore 258, 555
- SendMessage 115, 941
- Serialisierung 359, 392, 501
 - CArchive 507
 - Lesen und Schreiben 508
 - Makros 513
 - Streamoperatoren 509
- Serielle Schnittstelle 300, 310
- Server 589
 - Automatisierungs-Server 645
 - Befehlsverben 590

- für Remote Procedures 923
- lokaler 575
- MFC-Anwendungsgerüst 592
- Mini-Server 589
- Prozess-Server 575
- registrieren, automatisch 604
- Registrierung 586
- Serverobjekte 593
- Serverobjekte zeichnen 600
- Server-Ressourcen 595
- Voll-Server 589
- Vorortaktivierung 575
- Vorortbearbeitung 590
- Server-Konzepte 589
- SetROP2 483
- Sockets 873
 - Berkeley Sockets 867
 - Daten austauschen 877
 - einrichten 875
 - erzeugen 875
 - Verbindung einrichten 877
 - Verbindung schließen 877
- Sound 973, 988
- Spaltenüberschrift 461
- Speicher
 - gemeinsamer 272, 287
 - strukturierter 572
 - virtueller 274, 281
- Speichergerätekontext 210
- Speichermodelle 278
- Speicherreservierung
 - Heap-Funktionen 286
 - malloc 279
 - new 279
 - virtueller Speicher 282
- Speicherverwaltung 271
 - 16- versus 32-Bit 276
 - Adreßräume 271, 273
 - Auslagerungsdatei 274
 - gemeinsamer Speicher 272, 287
 - Heap 286
 - logische Adressen 272
 - malloc 279
 - new 279
 - physikalische Adressen 272
 - Prozesse 272
 - Remote Procedures 925
 - Seitentabelle 274
 - Speichermodelle 278
 - und Threads 290
 - virtueller Speicher 274, 281
- Speicherzuordnungsdateien 288
- SQL 731, 742
 - ALTER TABLE 746
 - CREATE INDEX 746
 - CREATE TABLE 745
 - CREATE VIEW 745
 - Datendefinitionsanweisungen 742, 745
 - Datenmanipulationsanweisungen 742
 - DELETE 744
 - DROP 746
 - eingebettete 735
 - INSERT 744
 - SELECT 743
 - SQL Server 811
 - und DAO 761
 - UPDATE 744
- SQL Server
 - Beziehungen einrichten 820
 - Datenbank anlegen 816
 - Datenbank-Assistent 816
 - Gespeicherte Prozeduren anlegen 821
 - konfigurieren 812
 - Primärschlüssel festlegen 819
 - registrieren 815
 - Tabellen anlegen 818
- Standarddialoge 174, 451
 - CColorDialog 453
 - CCommonDialog 452
 - CFileDialog 453
 - CFinReplaceDialog 454
 - CFontDialog 456
 - COleDialog 459
 - CPageSetupDialog 457
 - CPrintDialog 457
- Standardsteuerelemente 460
 - Animation 460
 - Datum/Zeitauswahl 473
 - Drehregler 468
 - Fortschrittsleiste 465
 - IP-Adresse 473
 - Listenansicht 463
 - Monatskalender 473
 - Quickinfo 471
 - Register 468
 - RTF-Steuerelement 466

- Schieberegler 467
 - Spaltenüberschrift 461
 - Statusfenster 468
 - Strukturansicht 471
 - Symbolleiste 470
 - Werkzeugleiste 473
 - Zugriffstasten 462
 - Statusfenster 468
 - Steuerelemente 182
 - ActiveX-Steuerelemente 717
 - besitzergezeichnete 438
 - Bildlaufleisten 185
 - Eingabefelder 184
 - IE 4.0 185, 473
 - Kombinationsfelder 184
 - Listenfelder 184
 - Schaltflächen 184
 - Standardsteuerelemente 460
 - Textfelder 183
 - Windows_95 185, 473
 - Strukturansicht 471
 - Subklassen 164
 - Superklassen 168
 - Symbole, Ressourcen 194
 - Symbolleiste 470
 - Ressourcen 200
 - Synchronisierter Variablenzugriff 290
 - Synchronisierungsobjekte 257, 553
 - Ereignisse 258, 554
 - Kritische Abschnitte 555
 - Mutexe 259, 555
 - Semaphore 258, 555
 - Syp++ 118
 - Systempalette 226
- T**
- TAPI 891
 - Adressen 895
 - Anruf beenden 903
 - Anruf tätigen 903, 909
 - asynchrone Operationen 897
 - Besitzerrechte 904
 - Dienste 900
 - Geräte 894
 - Hayes AT-Befehlssatz 896
 - ISDN 894
 - Leitungsgeräte 894
 - Listing 905
 - Medien-Modus 904
 - POTS 893
 - Probing 905
 - Programmiermodell 902
 - Software-Architektur 896
 - synchrone Operationen 897
 - TAPI initialisieren 902, 909
 - Telefongeräte 894
 - Träger-Modus 904
 - Übersicht 891
 - variable Strukturen 899
 - Tastaturkürzel, Ressourcen 195
 - TCP 868
 - TCP/IP-Protokoll 867
 - Telefonie 891
 - Telefonie, siehe TAPI
 - TelNet 829
 - Test-Container 710
 - Text, Farbe bestimmen 490
 - Textausgabe 239
 - TextOut 239
 - Threads 121, 122, 243, 245
 - beenden 256
 - Benutzeroberflächen-Threads 552
 - erzeugen 256
 - in der MFC 551
 - lokaler Speicher 291
 - sekundäre 254
 - Speicherverwaltung 290
 - Synchronisierung 553
 - Synchronisierungsobjekte 257
 - Thread-Objekte 256
 - volatile Variablen 255
 - Wartezustand 257
 - Worker-Threads 122, 553
 - TRACE 88
 - Treiber 731, 735
 - Typenbibliothek 656
- U**
- UDP 869
 - UNC-Namen 297
 - Universal Naming Convention 297
- V**
- Variablenzugriff, synchronisierter 290
 - VBX-Zusatzsteuerelemente 717
 - VCVARS32.BAT 34

Verbindungspunkte 703, 708
Verbunddokumente 572, 574
Verknüpfte Objekte 574
Verknüpfung 574
Versionsinformationen, Ressourcen 200
Vertex 956
VFAT-Dateisystem 296
Video Compression Manager 952
View, siehe Ansichtsklasse
Viewport 211, 485
VirtualAlloc 282
Virtuelle Funktionen 566
Virtuelle Speicherfunktionen 282
Virtueller Speicher 274, 281
Visual Basic, Listing 863
Visual Studio 21
– Anwendungsassistent 52
– Arbeitsbereichfenster 23
– Assistenten 51, 56
– Assistentenleiste 27
– Ausgabefenster 26
– Debug-Fenster 26
– Debugger 77
– grundlegende Arbeitsschritte 28
– Klassen-Assistent 58
– Komponentensammlung 105
– Quellcode-Browser 75
– Quelltexteditor 25
– Ressourceneditor 26
Visual Studio Analyzer 103
volatile 256
Vorkompilierte Header 47
Vorortaktivierung 575
Vorortbearbeitung 590

W

WaitForMultipleObjects 305
Warteschlangen 112
Warteschleifen 113
Waveform 935, 936
Werkzeugleiste 473
wglMakeCurrent 959
WIN32S 111
Windows
– Betriebssystemübersicht 111
– C/C++-Bibliotheksfunktionen 131
– Funktionsaufrufe 123

– GDI-Dienste 126
– Kernel-Dienste 123
– Multitasking 244
– Nachrichten 112
– Speicherverwaltung 271
– User-Dienste 125
– Warteschlangen 112
Windows_95 Logo 851
Windows_95/98 111, 133
Windows_NT 111, 132
WinNet 827
– asynchroner Modus 838
– Cache 839
– FTP-Anfragen 834
– Gopher-Anfragen 836
– HTTP-Anfragen 837
– Verbindung herstellen 835
– WinNet-Bibliothek 833
WinSock 874
– blockierende Aufrufe 879
– Daten austauschen 877
– Initialisieren 875
– Sockets einrichten 875
– Verbindung einrichten 877
– Verbindung schließen 877
WM_EX_TOPMOST 151
WM_PAINT 142, 158
WndClass 141
WNDCLASS-Struktur 153
WndProc 141
WOW-Prozess 247
WriteFile 298, 310

Y

Yield 246

Z

Zeichenflächen 478
Zeichenfolgentabellen, Ressourcen 199
Zeichenfunktionen 234, 486
– DirectX 976
– OpenGL 959
Zeichenmodus 483
Zeichenobjekte 222
Zugriffstasten 462
Zuordnungen 526
Zwischenablage 313

- Daten kopieren in die 316
- Einfügen von Daten aus der 318
- Formate 313
- Nachrichten 319
- Operationen 316
- private Formate 316
- registrierte Formate 315
- Steuerelemente 318
- verzögerte Übertragung 317
- Viewer 319

