

**Start ohne
Vorwissen!**

Jetzt lerne ich

Visual C++ 6



**Komplettes Compiler-Kit für den schnellen und
einfachen Einstieg in die Windows-Programmierung**

Inhaltsverzeichnis.....

1. Teil: Anwendungsentwicklung mit Visual C++	15	
1 Zur Einstimmung	17	
1.1 »Hello, World!« als Konsolenanwendung	18	
1.1.1 Einsatz der Kommandozeilenversion des Compilers	18	
1.1.2 Einsatz der integrierten Entwicklungsumgebung (IDE)	23	
1.2 »Hello, World!« als GUI-Anwendung	26	
1.2.2 Einsatz der integrierten Entwicklungsumgebung (IDE)	29	
1.2.3 Einsatz der Kommandozeilenversion des Compilers	31	
1.3 Die Kommandozeilenversion des Compilers	34	
1.4 Zusammenfassung	37	
1.5 Fragen	38	
1.6 Aufgaben	38	
2 Von der Idee zum Programm	41	
2.1 Ein Programm zur Zinsberechnung	42	
2.2 Projekte und Arbeitsbereiche	45	
2.2.1 Was ist ein Projekt?	45	
2.2.2 Wozu braucht man Arbeitsbereiche?	48	
2.2.3 Wie legt man neue Projekte an?	49	
2.2.4 Das Arbeitsbereichsfenster	53	
2.2.5 Projekte öffnen und schließen	55	
2.2.6 Wie erweitert man ein Projekt um neue Quelltextdateien?	56	
2.3 Der Editor	59	
2.4 Ressourcen	61	
2.4.1 Vorteile des Ressourcenkonzepts	62	
2.4.2 Das Ressourcenkonzept	63	
2.4.3 Ressourcen anlegen	67	
2.4.4 Die verschiedenen Ressourcen-Arten	68	
2.5 Quellcode fertigstellen	81	

2.6	Compiler und Linker	83
2.6.1	Die Projekteinstellungen	83
2.6.2	Vorkompilierte Header-Dateien	85
2.6.3	Der inkrementelle Linker	88
2.6.4	Zwischendateien	88
2.6.5	Die Arbeit mit Projektkonfigurationen	89
2.6.6	Was geschieht bei der Projekterstellung?	90
2.7	Der Debugger	91
2.8	Zusammenfassung	94
2.9	Fragen	94
2.10	Aufgaben	95
3	Die Assistenten	97
3.1	Der MFC-Anwendungs-Assistent	99
3.2	Der Klassen-Assistent	103
3.2.1	Das Dialogfeld des Klassen-Assistenten	104
3.2.2	Erstellen einer neuen Klasse	105
3.2.3	Behandlungsmethoden einrichten	106
3.2.4	Member-Variablen und Datenaustausch	109
3.2.5	Automatisierung	110
3.2.6	ActiveX-Ereignisse	112
3.2.7	Die Klasseninformationsdatei (.clw)	113
3.3	Zusammenfassung	114
4	Fehlersuche mit dem Debugger	117
4.1	Der Debugger	119
4.2	Vorbereitung des Programms für das Debuggen	120
4.3	Befehle zur Programmausführung	121
4.3.1	Programm in Debugger laden und starten	121
4.3.2	Programm anhalten	121
4.3.3	Programm schrittweise ausführen	123
4.4	Die Debug-Fenster	124
4.5	Weitere Debug-Tools	127
4.6	Debug-Techniken	129
4.6.1	Debugfähigen Code erstellen	129
4.6.2	Kritischen Code überprüfen	130
4.6.3	Windows-Anwendungen debuggen	130

4.7	Beispiel	131
4.8	TRACE-Diagnosemakros	134
4.9	Das ASSERT-Makro	134
4.10	Zusammenfassung	135
4.11	Fragen	136
4.12	Aufgaben	136
2. Teil: Windows-Programmierung mit der MFC		137
5	Das MFC-Anwendungsgerüst	139
5.1	Das Projekt	140
5.2	Die Dateien und Klassen des Anwendungsgerüsts	144
5.2.1	CHello_W3App – das Anwendungsobjekt	145
5.2.2	CMainFrame – das Hauptfenster der Anwendung	145
5.2.3	CHello_W3View – das Ansichtsfenster der Anwendung	147
5.2.4	CHello_W3Doc – die Dokumentklasse	148
5.3	Was geschieht beim Starten der Anwendung?	149
5.4	Spezielle Makros und Konventionen der MFC	154
5.4.1	Die ungarische Notation	154
5.4.2	MFC-Makros	156
5.5	Erweitern des Anwendungsgerüsts	157
5.5.1	Hinzufügen einer Zeichenfolgen-Ressource	157
5.5.2	Modifizieren des Dokuments	158
5.5.3	Modifizieren der Ansicht	159
5.6	Zusammenfassung	160
5.7	Fragen	160
5.8	Aufgaben	161
5.9	Lösung zu Aufgabe 3	162
6	Anpassen des Hauptfensters	165
6.1	Fenster und Fensterobjekte	166
6.2	Anpassung über den Assistenten	168
6.3	Anpassung über die Methode PreCreateWindow()	170
6.3.1	Anpassung der Position und Größe des Hauptfensters	171
6.3.2	Anpassung des Fensterstils	172
6.4	Anpassung über die Methode OnCreate()	174
6.4.1	Symbolleiste und Statusleiste	174

6.5	Das Anwendungssymbol	175
6.6	Kommandozeilenargumente	178
6.7	Zusammenfassung	181
6.8	Fragen	182
6.9	Aufgaben	182
6.10	Lösungen zu den Aufgaben	184
7	I nteraktivität durch Nachrichten	187
7.1	Ereignisverarbeitung unter Windows und der MFC	188
7.1.1	Ereignisverarbeitung unter Windows	189
7.1.2	Jetzt übernimmt das Programm	191
7.1.3	Antworttabellen (MESSAGE_MAP)	192
7.2	Der Klassen-Assistent ist eine große Hilfe	192
7.3	Mausereignisse	194
7.4	Tastaturereignisse	198
7.5	Ganz wichtig: WM_PAINT	199
7.5.1	Außerhalb von OnDraw() zeichnen	200
7.5.2	Innerhalb von OnDraw() zeichnen	202
7.6	Zeitgeber	203
7.7	Zusammenfassung	207
7.8	Fragen	207
7.9	Aufgaben	208
7.10	Lösungen zu den Aufgaben	209
8	Menüs, Symbolleisten, Tastaturkürzel	213
8.1	Eine komplette Menüunterstützung	214
8.2	Bearbeitung der zugehörigen Ressourcen	220
8.2.1	Anpassung des Menüs	220
8.2.2	Anpassung der Tastaturkürzel	223
8.2.3	Anpassung der Symbolleiste	224
8.2.4	Anpassung der Stringtabelle	226
8.3	Methoden für Menübefehle einrichten	227
8.3.1	WM_COMMAND	227
8.3.2	Einsatz des Klassen-Assistenten	228
8.3.3	Die Klassen des Anwendungsgerüsts	229
8.4	Menübefehle deaktivieren	233
8.5	Kontextmenüs	235

8.6	Zusammenfassung	237
8.7	Fragen	238
8.8	Aufgaben	238
8.9	Lösung zur Aufgabe	239
9	Steuerelemente	243
9.1	Steuerelemente in Fenster integrieren	244
9.2	Statische Textfelder	245
9.3	Eingabefelder	248
9.4	Schaltflächen	250
9.5	Kontrollkästchen und Optionsfelder	252
9.6	Listenfelder und Kombinationsfelder	256
9.7	Ein einarmiger Bandit	259
9.8	Zusammenfassung	262
9.9	Fragen	262
9.10	Aufgaben	263
10	Dialogfelder	265
10.1	Dialog-Ressourcen erstellen	266
10.2	Erstellung von Dialogen auf der Grundlage von Ressourcen	272
10.2.1	Dialogklasse anlegen	272
10.2.2	Zugriff auf die Steuerelemente	275
10.3	Dialogfelder initialisieren	279
10.4	Dialogfelder aufrufen	280
10.5	Benutzereingaben auswerten	282
10.6	Modale und nicht-modale Dialoge	283
10.7	Zusammenfassung	284
10.8	Fragen	285
10.9	Aufgaben	285
11	Text und Dateien	287
11.1	Ein Problem – viele Lösungen	287
11.1.1	Meldungsfenster	287
11.1.2	Text zeichnen	290
11.1.3	Steuerelemente	292
11.1.4	Spezielle Ansichtsklassen	293
11.1.5	Weitere nützliche Klassen	294

11.2	Dateien	294
11.2.1	Die Klasse CFile	295
11.2.2	Dateien schreiben	298
11.2.3	Dateien lesen	302
11.3	Was bedeutet »Serialisierung«?	303
11.3.1	Die CArchive-Klasse	303
11.3.2	Lesen und Schreiben über CArchive	305
11.4	Ein einfacher Texteditor	307
11.5	Zusammenfassung	309
11.6	Fragen	310
11.7	Aufgaben	310
12	Zeichnen	311
12.1	Das Arbeitsmaterial des Künstlers	312
12.2	Gerätekontexte	314
12.2.1	Gerätekontextklassen	314
12.2.2	Gerätekontexte selbst erzeugen	315
12.2.3	OnDraw()	316
12.3	Die Zeichenmethoden	321
12.3.1	Übersicht	321
12.3.2	Der Mäuse-Editor	324
12.4	Die Zeichenwerkzeuge	330
12.4.1	Überblick	330
12.4.2	Vordefinierte GDI-Objekte	330
12.4.3	GDI-Objekte einrichten	331
12.5	Farbige Fraktale	335
12.6	Zusammenfassung	339
12.7	Fragen	340
12.8	Aufgaben	340
12.9	Lösungen zu den Aufgaben	341
13	Bitmaps	347
13.1	Bitmap-Ressourcen anlegen	348
13.2	Bitmaps laden und anzeigen	349
13.3	Bitmaps als Fensterhintergründe	352
13.4	Bitmaps manipulieren	354

13.5	Zusammenfassung	357
13.6	Fragen	358
13.7	Aufgaben	358
13.8	Lösungen zu den Aufgaben	358
3. Teil:	Verstehen	361
14	Der von den Assistenten erzeugte Code	363
14.1	Wie funktionieren Windows-Programme?	364
14.1.1	Die Eintrittsfunktion WinMain()	364
14.1.2	Erzeugung des Hauptfensters	367
14.1.3	Eintritt in die Nachrichtenverarbeitung	370
14.2	Von der API zur MFC	378
14.2.1	WinMain() und Anwendungsobjekt	378
14.2.2	Message Loop und Run()	379
14.2.3	Erzeugung der Fenster	379
14.2.4	Fensterfunktion und Antworttabellen	379
14.2.5	Gerätekontexte	382
14.3	Abschlußbemerkung	382
14.4	Zusammenfassung	383
14.5	Fragen	383
14.6	Aufgaben	384
14.7	Lösung zu Aufgabe 2	384
15	Das Doc/View-Modell	385
16	Das Doc/View-Gerüst anpassen	391
16.1	Ein Dokument – zwei Ansichten	391
16.2	Zusammenfassung	399
16.3	Fragen	399
16.4	Aufgaben	400
17	Programme ohne Doc/View	401
18	Rückbesinnung auf die API	405
18.1	Aufruf von API-Funktionen	405
18.2	Systeminformationen abfragen	408
18.3	Zusammenfassung	410

← Inhaltsverzeichnis		
18.4	Fragen	411
18.5	Aufgaben	411
4. Teil: MFC-Programmierung für Fortgeschrittene		413
19	Multimedia	415
19.1	Allgemeines	415
19.2	Sound	416
19.3	Video	417
19.4	Zusammenfassung	420
20	Dynamische Linkbibliotheken (DLLs)	421
20.1	Allgemeines	422
20.2	Erstellung der DLL	423
20.3	Erstellung der EXE-Anwendung	425
20.4	DLL debuggen und ausführen	427
20.5	Zusammenfassung	428
21	MDI -Anwendungen	429
21.1	Erstellung eines MDI-Editors	431
21.2	Die Klassen des MDI-Anwendungsgerüsts	432
21.3	MDI mit mehreren Dokumentvorlagen	434
21.4	Zusammenfassung	435
22	COM	437
22.1	OLE (Object Linking and Embedding)	439
22.2	Automatisierung	441
22.2.1	Der Server	441
22.2.2	Der Client	445
22.3	Zusammenfassung	447
23	Multithreading	449
23.1	Allgemeines	450
23.2	Threads erzeugen	451
23.3	Zusammenfassung	457
24	Internet-Programmierung	459
24.1	Die Internet-Protokolle	460
24.1.1	Das Dateitransferprotokoll FTP	460

24.1.2	Das Gopher-Protokoll	462
24.1.3	Das Hypertext Transfer Protokoll HTTP	462
24.2	Aufbau von Internet-Verbindungen	463
24.3	Erstellung eines Webbrowsers	464
24.4	Zusammenfassung	469
25	Datenbank-Programmierung	471
25.1	Grundlagen der Datenbanken	472
25.2	ODBC – die Verbindung zur Datenbank	474
25.3	Datenbanken anlegen	476
25.4	Datenbanktreiber und Treiberverbindungen	477
25.5	Datenbank-Programmierung	479
25.5.1	Die ODBC-Datenbankklassen	479
25.5.2	Datensätze einlesen	481
25.5.3	Datensätze anzeigen	485
25.5.4	In der Datenbank navigieren	487
25.5.5	Datensätze editieren	488
25.5.6	Nach Daten suchen	488
25.5.7	Daten grafisch darstellen	490
25.6	Zusammenfassung	495
Anhang A: Windows-Nachrichten		497
Anhang B: Objektorientierte Programmierung in C++		507
Anhang C: Antworten zu den Fragen		523
Anhang D: Buch-CD und Autoren-Edition		533
Stichwortverzeichnis		537

Vorwort zur 2. Auflage

Dieses Buch wendet sich an C++-Programmierer, die sich mit Hilfe von Visual C++ in die Windows-Programmierung einarbeiten wollen. Damit ist eigentlich schon alles und doch so gut wie nichts über das Buch gesagt, denn Bücher zur Windows-Programmierung mit Visual C++ gibt es viele.

Statt Ihnen das Buch an dieser Stelle weiter anzupreisen und Sie davon zu überzeugen, daß Sie ohne dieses Buch nicht mehr leben können, möchte ich den Raum lieber nutzen, um Ihnen darzustellen, für welchen Leser dieses Buch geschrieben wurde.

1. Sie sollten in C++ programmieren können und auch über Kenntnisse in objektorientierter Programmierung verfügen. Im Anhang des Buches finden Sie zwar eine kurze Übersicht und Darstellung der wichtigsten Konzepte der objektorientierten Programmierung, doch kann dies ein Lehrbuch und eigene Erfahrungen nicht ersetzen. Ein Profi muß man allerdings nicht sein, Grundkenntnisse sollten ausreichen, um den Ausführungen in diesem Buch folgen zu können.
2. Sie sollten Interesse, Engagement und Wißbegier mitbringen. Dies ist kein Buch für Leute, die programmieren lernen wollen, ohne selbst mitdenken zu müssen. Das Buch richtet sich zwar an Einsteiger in die Windows-Programmierung, doch heißt dies nicht, daß jedwede komplexere Sachverhalte deshalb ausgespart würden. Im Gegenteil, das Buch will Ihnen nicht nur zeigen, wie Sie mit Hilfe von Visual C++, der MFC und den Assistenten von Visual C++ schnell und bequem professionelle Anwendungen erstellen können, es möchte Ihnen auch auf verständliche Weise vermitteln, welche Konzepte hinter dem aufgesetzten Programmcode stehen, welche Anforderungen Windows an ein Windows-Programm stellt, was ein API-Programm ist, wie die MFC funktioniert.
3. Sie sollten Spaß an der Programmierung haben. Dies ist kein Buch für Leute, die die Programmierung bierernst nehmen. Wer nur an harten Fakten und trockenen Informationen interessiert ist, wird diese zwar finden, doch wird ihm die Verpackung nicht gefallen, denn das Buch ist vornehmlich als Lese- und Lehrbuch gedacht, das auch ein bißchen den Spaß an der Programmierung vermitteln will.

Wenn Sie Fragen oder Anregungen zum Inhalt des Buches haben, Kritik äußern oder ein Lob loswerden wollen, erreichen Sie mich per E-Mail unter

`rdlouis@compuserve.com`

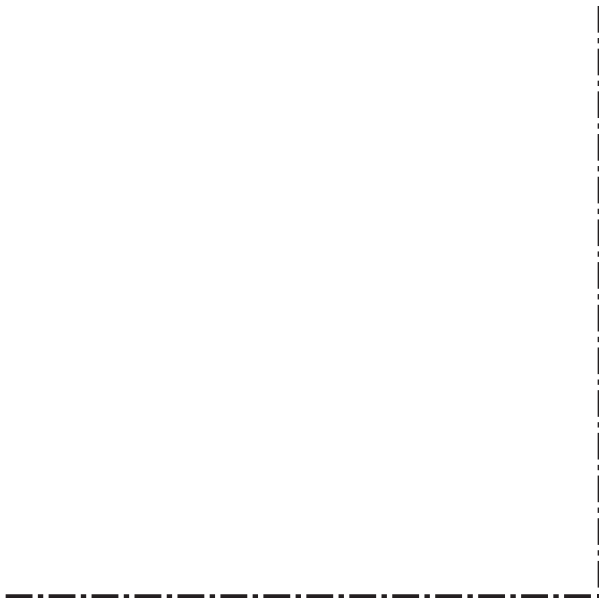
Ansonsten wünsche ich Ihnen viel Spaß und Erfolg mit dem Buch.

Dirk Louis



TEIL 1

Anwendungs- entwicklung mit Visual C++



Zur Einstimmung

In diesem Kapitel lernen Sie ... eigentlich gar nichts! Es soll Sie einfach nur in die Entwicklungsumgebung von Visual C++ einführen und Ihnen beweisen, daß die Anwendungsentwicklung mit Visual C++ ein Kinderspiel ist. So nebenbei erfahren Sie, wo man seinen Programmcode eintippt, wie man Programme kompiliert und was der Unterschied zwischen einer Konsolen- und einer GUI-Anwendung ist – aber von Lernen kann deswegen ja wohl noch keine Rede sein.

Sie erfahren in diesem Kapitel:

- ✗ Was eine Konsolenanwendung ist
- ✗ Wie man Konsolenanwendungen erstellt
- ✗ Was eine GUI-Anwendung ist
- ✗ Wie man GUI-Anwendungen erstellt
- ✗ Wie man mit der Kommandozeilenversion des Compilers arbeitet

Voraussetzung

Voraussetzung ist, daß Sie bereits über Grundkenntnisse in C/C++ verfügen. Im Anhang finden Sie zwar ein Referenzkapitel zur objektorientierten Programmierung in C++, doch ein Lehrbuch und eigene Erfahrungen kann dies nicht ersetzen.



1.1 »Hello, World!« als Konsolenanwendung

Welcher C-Programmierer kennt es nicht, das legendäre »Hello, World!«-Programm. Es kann fast nichts, es macht fast nichts, aber irgendwie haben wir es doch lieb gewonnen, denn es sagt uns, daß es auch in so komplizierte Bereiche wie die C-Programmierung einen einfachen Einstieg gibt. Schauen wir also, wie wir das Hello-World-Programm mit Visual C++ zum Laufen bringen.

Wie Sie aus Ihrer Praxis als C/C++-Programmierer wissen, braucht man dazu im Grunde nicht mehr als einen ganz simplen ASCII-Editor und einen Compiler. (Die Unterscheidung zwischen Compiler und Linker spare ich mir, da sie für das Thema dieses Buches praktisch ohne Belang ist.) Visual C++ ist aber viel mehr als nur ein Compiler. Es ist eine äußerst komplexe und leistungsfähige integrierte Entwicklungsumgebung, und Sie können zu recht stolz sein, mit einem solch professionellen Werkzeug zu arbeiten. Nur ... in dieser Entwicklungsumgebung ein Programm wie »Hello, World!« zu erstellen, ist wie mit Kanonen auf Spatzen zu schießen. Vergessen wir also zunächst die ganze integrierte Entwicklungsumgebung (und das Geld, das sie uns gekostet hat) und begnügen wir uns mit dem, was wir wirklich brauchen: dem Compiler.

1.1.1 Einsatz der Kommandozeilenversion des Compilers

Der Einsatz der Kommandozeilenversion des Compilers ist an sich ganz einfach, bedarf jedoch einiger vorbereitender Arbeiten. (Sollten Sie mit der Einrichtung der Kommandozeilenwerkzeuge Schwierigkeiten haben, braucht Sie dies nicht zu verdrießen. Zur Windows-Programmierung genügt es vollkommen, wenn Sie mit der IDE arbeiten – tatsächlich werden Sie in der Praxis wohl eher selten auf die Kommandozeilenwerkzeuge zurückgreifen. Lesen Sie die nachfolgenden Abschnitte aber auf jeden Fall durch, um ein wenig über die hinter der IDE stehenden Werkzeuge zu erfahren.)

- cl.exe* 1. Zuerst einmal muß man natürlich wissen, wie der Compiler heißt und in welchem Verzeichnis er zu finden ist.

Wie man erwarten würde, findet sich der Compiler im *BIN*-Verzeichnis von Visual C++. Seine EXE-Datei heißt: *cl.exe*, wobei »CL« vermutlich ein Akronym für Compiler/Linker ist.

2. Wie rufe ich den Compiler auf?

Eingabeaufforderung

Da wir den Compiler über eine Kommandozeile aufrufen wollen, brauchen wir ein Fenster, über das wir Befehle an das Betriebssystem schicken können. Unter Windows wäre dies die (MS-DOS-)Eingabeaufforderung, die über das Menü START/PROGRAMME aufgerufen wird.

3. Wie kann ich den Compiler aus einem beliebigen Verzeichnis aufrufen?

vcvars32.bat

Zur Erstellung eines Programms wechseln wir in der Eingabeaufforderung mit Hilfe des Befehls CD in das Verzeichnis, in dem der Quelltext des Programms steht und rufen von dort aus den Compiler auf. Dazu muß dem Betriebssystem aber bekannt sein, in welchem Verzeichnis die EXE-Datei des Compilers zu finden ist, und der Compiler wiederum muß wissen, wo die Include-Dateien und die LIB-Dateien der Laufzeitbibliothek zu finden sind. Zu diesem Zweck muß die Systemumgebung durch Erweiterung des Pfads (PATH) und Setzen einiger Umgebungsvariablen angepaßt werden. Wer aus den Tagen des alten DOS noch mit der Nutzung und Überarbeitung der *autoexec.bat* vertraut ist, den wird dies nicht schrecken. Aber auch wer mit diesen Techniken nicht so vertraut ist, muß sich nicht sorgen, denn Microsoft liefert mit dem Visual C++-Compiler eine Batch-Datei namens *vcvars32.bat* aus, die sämtliche Einstellungen für uns vornimmt.

Alles, was Sie tun müssen, ist *vcvars32.bat* auszuführen.

Die Ausführung der Batch-Datei *vcvars32.bat* unterscheidet sich ein wenig für die Betriebssysteme Windows 95/98 und Windows NT.

Unter Windows 95/98 gehen Sie folgendermaßen vor:

Windows 95/98

1. Rufen Sie das MS-DOS-Eingabefenster auf (über START/PROGRAMME).
2. Wechseln Sie im Fenster mit dem DOS-Befehl CD in das BIN-Verzeichnis von Visual C++. Wenn Sie jetzt den Befehl DIR eingeben (oder DIR /P), sollte in der Liste der Dateien auch *vcvars32.bat* aufgeführt werden.
3. Rufen Sie *vcvars32.bat* aus. Die entsprechende Kommandozeile könnte beispielsweise wie folgt aussehen:

```
C:\Microsoft Visual Studio\VC98\Bin> vcvars32.bat
```

Unter Umständen erhalten Sie jetzt eine Fehlermeldung, daß im Umgebungsbereich kein Speicher mehr frei ist. Sollte dies der Fall sein, rufen Sie das Systemmenü des Eingabefensters auf (mit der rechten Maustaste auf das Symbol links neben dem Fenstertitel klicken), wählen den Befehl EIGENSCHAFTEN aus, wechseln im erscheinenden Dialogfeld zur Seite

SPEICHER und setzen dort die Angaben zum Umgebungsspeicher herauf. (Versuchen Sie es zuerst einmal damit, den anfänglichen Umgebungsspeicher auf 1024 zu setzen und die restlichen Einstellungen auf AUTOMATISCH zu belassen.) Danach müssen Sie das Eingabefenster schließen und neu aufrufen, bevor Sie *vcvars32.bat* ausführen können.



Wenn Sie viel mit der Eingabeaufforderung arbeiten, lohnt es sich, die DOS-History zu aktivieren. In dieser History werden die im Eingabefenster abgeschickten Befehle aufgelistet. Mit Hilfe der Pfeilschalter (Pfeil nach oben und Pfeil nach unten) kann man einen Befehl aus der Liste auswählen, gegebenenfalls in der Kommandozeile bearbeiten und dann erneut abschicken. Zur Aktivierung der History geben Sie einfach von der Kommandozeile den Befehl *DOSKEY* ein, oder schreiben Sie den Aufruf von *DOSKEY* direkt in Ihre *autoexec.bat* (unter Windows NT braucht *DOSKEY* nicht extra aufgerufen zu werden).

Windows NT Unter Windows NT können Sie die Einstellungen aus *vcvars32.bat* direkt bei der Installation registrieren lassen. Haben Sie dies versäumt, verfahren Sie wie unter Windows 95.

Soweit zu den Vorbereitungen. Das Ganze mutet vielleicht ein wenig umständlich an, aber für den einen oder anderen Leser, der hier so nebenbei noch ein wenig über sein Betriebssystem erfahren konnte, dürften die Ausführungen auch ganz interessant gewesen sein. Warum sich die Auseinandersetzung mit der Kommandozeilenversion des Compilers auch für alle anderen Visual-C++-Programmierer lohnt, werde ich im letzten Teil dieses Kapitels ausführen. Jetzt wollen wir uns aber zunächst mit unserem ersten Programm belohnen.

Übung 1-1: Konsolenanwendung mit *cl.exe* erstellen

Endlich! Die Erstellung des Beispielprogramms.

1. Rufen Sie den Notepad-Editor auf oder irgendeinen beliebigen anderen Editor, mit dem man ASCII-Textdateien erstellen kann. Geben Sie den Quelltext des Programms ein.

```
#include <stdio.h>
int main()
{
    printf("Hello, World\n");
    return 0;
}
```

2. Speichern Sie den Quelltext unter dem Namen *Hello.cpp*.
3. Rufen Sie die Eingabeaufforderung auf, führen Sie *vcvars32.bat* aus (siehe oben), und wechseln Sie in das Verzeichnis, in dem Ihre Quelltextdatei steht.
4. Rufen Sie den Compiler zur Erstellung des Programms auf. In der Kommandozeile übergeben Sie dem Compiler die zu übersetzende Datei:
cl hello.cpp
5. Wenn Ihnen beim Eingeben des Quelltextes keine Fehler unterlaufen sind, können Sie das fertige Programm *hello.exe* ausführen.

```

Eingabeaufforderung
C:\TEMP>cd..
C:\>e:
E:\>cd temp
E:\TEMP>cd Hello_M1
E:\TEMP\HELLO_M1>cl Hello.cpp
Optimierender Microsoft (R) 32-Bit C/C++-Compiler, Version 12.00.8168, fuer x86
Copyright (C) Microsoft Corp 1984-1998. Alle Rechte vorbehalten.

Hello.cpp
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/out:Hello.exe
Hello.obj
E:\TEMP\HELLO_M1>Hello.exe
Hello, World
E:\TEMP\HELLO_M1>
E:\TEMP\HELLO_M1>

```

Bild 1.1:
Aufruf des
Compilers von
der Komman-
dozeile

Das Programm *hello.exe* ist übrigens eine Konsolenanwendung. Das Kennzeichen einer Konsolenanwendung ist, daß sie über keine echte grafische Benutzeroberfläche verfügt. Dies ist praktisch gleichbedeutend mit der Feststellung, daß Konsolenanwendungen keine Fenster (englisch windows) haben. Unter einem fensterorientierten Betriebssystem wie Windows stellt dies aber ein Problem dar, da hier die gesamte vom Betriebssystem vermittelte Kommunikation zwischen Anwender und Programm über Fenster geschieht. (Wie dies im einzelnen funktioniert, werden Sie im dritten Teil des Buches erfahren.) Statt dessen kommuniziert das Programm über die Standardeingabe (*stdin*, *cin*) und die Standardausgabe (*stdout*, *cout*) und überläßt es dem Betriebssystem, festzulegen, was die Standardeingabe und die Standardausgabe wirklich sind.

*Konsolen-
anwendungen
sind Anwen-
dungen ohne
grafische
Oberfläche*

Jetzt die Frage an Sie: Wie sehen unter Windows die Standardeingabe und die Standardausgabe aus?

Die Standardausgabe ist schnell gefunden. Als erfahrener C-Programmierer wissen Sie natürlich, daß die Funktion *printf()* in die Standardausgabe schreibt. Und wohin hat unser Programm seinen Gruß ausgegeben?

*Standard-
ausgabe*

Richtig! Folglich ist die Standardausgabe für Konsolenanwendungen das Fenster der Eingabeaufforderung.

Daß unser Programm nicht nur deshalb in die Eingabeaufforderung schreibt, weil das Programm von der Eingabeforderung aus gestartet wurde, können Sie leicht nachprüfen, indem Sie das Programm *Hello.exe* per Doppelklick aus dem Windows Explorer heraus starten. Auch in diesem Fall schreibt das Programm in die Standardausgabe. Das Betriebssystem erkennt dies und öffnet automatisch die Eingabeaufforderung, in der der ausgegebene Text erscheint. Der einzige Nachteil ist, daß das Betriebssystem die Eingabeaufforderung auch wieder automatisch schließt, wenn das Programm beendet wurde. Man sieht die Eingabeaufforderung daher nur kurz aufblinkern.

*Standard-
eingabe*

Die Standardeingabe ist die Tastatur. Doch erklärt dies noch nicht, wie eine Anwendung unter Windows Eingaben über die Tastatur empfangen kann. Schließlich können unter Windows mehrere Anwendungen gleichzeitig ausgeführt werden. Wenn der Anwender nun einen Text über die Tastatur eintippt, stellt sich die Frage, welche der gerade ausgeführten Anwendungen den eingegebenen Text empfängt. Aufgrund Ihrer Erfahrung als Windows-Nutzer können Sie diese Frage schnell beantworten. Die Anwendung, deren Fenster gerade den Fokus besitzt (d.h., das Fenster steht im Vordergrund, und seine Titelleiste wird nicht grau dargestellt), übernimmt die Eingaben.

Wie sieht dies nun für Konsolenanwendungen aus? Ganz einfach: Das Betriebssystem sieht die Eingabeaufforderung als Fenster der Konsolenanwendung an. Nur wenn die Eingabeaufforderung den Fokus hat, werden Tastatureingaben an die im Fenster der Eingabeaufforderung ausgeführte Konsolenanwendung geschickt.

Unter fensterbasierten Betriebssystemen sind Konsolenanwendungen also darauf angewiesen, daß ihnen das Betriebssystem ein Standardfenster zuweist, über das Ein- und Ausgabe erfolgen können. Dieses Fenster nennt man üblicherweise Konsole oder Konsolenfenster; unter Windows heißt es Eingabeaufforderung.



Daß Konsolenanwendungen in dem Fenster der Eingabeaufforderung ausgeführt werden, die unter Windows 95 noch MS-DOS-Eingabeaufforderung heißt, bedeutet nicht, daß Konsolenanwendungen DOS-Anwendungen seien. Die Konsolenanwendungen, die Sie mit dem Visual-C++-Compiler erstellen, sind alles echte 32-Bit-Anwendungen, die unter DOS gar nicht lauffähig wären.

1.1.2 Einsatz der integrierten Entwicklungsumgebung (IDE)

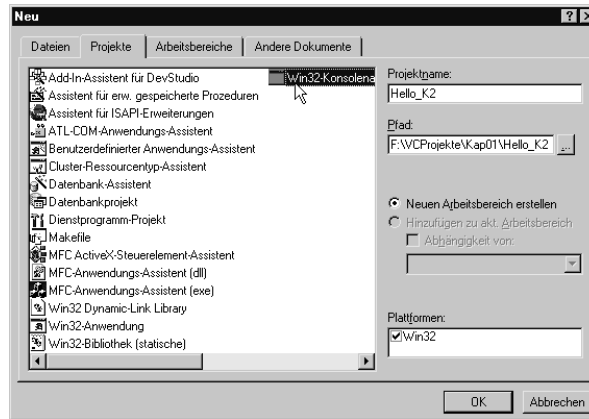
Als nächstes wollen wir das gleiche Konsolenprogramm aus der IDE heraus erstellen. Wenn Sie schon einmal mit einem Compiler mit integrierter Entwicklungsumgebung gearbeitet haben, wird dies überhaupt kein Problem für Sie darstellen. Wenn Sie bisher aber nur mit Kommandozeilen-Entwicklertools gearbeitet haben (etwa dem GNU-Compiler), steht zwischen Ihnen und der Erstellung Ihres Programms paradoxerweise eines der leistungsfähigsten und nützlichsten Features der IDE: die Projektverwaltung. Visual C++ erlaubt es uns nämlich nicht, einfach eine neue Datei anzulegen, unseren Quelltext einzugeben und daraus ein ausführbares Programm zu erstellen. Statt dessen verlangt Visual C++, daß alle Dateien, die zu einem Programm gehören, in einem Projekt zusammengefaßt werden. Projekte werden ihrerseits wieder in Arbeitsbereiche organisiert, was vor allem dann interessant ist, wenn mehrere ausführbare Dateien irgendwie zusammengehören (beispielsweise eine EXE-Datei und eine DLL, die von der EXE-Datei aufgerufen wird). Welche Vorteile uns die Projektverwaltung bringt und wie man sie nutzt, werden wir uns im nachfolgenden Kapitel anschauen. Im Moment finden wir uns einfach damit ab, daß wir selbst für ein so einfaches Programm wie *Hello World*, das nur aus einer einzigen Quelltextdatei besteht, einen Arbeitsbereich und ein Projekt anlegen müssen, in das wir unsere Quelltextdatei aufnehmen.

Übung 1-2: Konsolenanwendung in IDE erstellen

1. **Rufen Sie Visual C++ auf.** Falls nach dem Programmstart irgendwelche Quelldateien angezeigt werden oder bereits ein Arbeitsbereich geöffnet sein sollte (evtl. haben Sie an der Programmkonfiguration herumgespielt und im Dialogfeld OPTIONEN, Aufruf über das Menü EXTRAS, auf der Seite ARBEITSBEREICH die Option LETZTEN ARBEITSBEREICH BEIM START AUTOMAT. LADEN aktiviert), schließen Sie diese.
2. **Legen Sie einen Arbeitsbereich und ein Projekt an.** Rufen Sie dazu den Befehl DATEI/NEU auf, und lassen Sie die Seite PROJEKTE anzeigen.

Geben Sie einen NAMEN für das Projekt ein, beispielsweise *Hello_K2*, und wählen Sie ein im Feld PFAD übergeordnetes Verzeichnis für das Projekt aus. Visual C++ wird unter diesem Verzeichnis ein Unterverzeichnis für das Projekt anlegen, das den gleichen Namen wie das Projekt trägt.

Bild 1.2:
Anlegen eines
neuen Projekts



Anhand der per Voreinstellung ausgewählten Option NEUEN ARBEITSBEREICH ERSTELLEN können Sie erkennen, daß Visual C++ selbst dafür Sorge trägt, daß das neue Projekt einem Arbeitsbereich zugeordnet wird.

Zur Einrichtung eines Projekts gehört auch die Angabe eines Projekttyps. Über den Projekttyp teilen wir dem Compiler mit, welche Art von Programm aus dem Projekt erstellt werden soll (was beispielsweise Einfluß darauf hat, welche Laufzeitbibliotheken in das Programm eingebunden werden). Da wir eine Konsolenanwendung erstellen wollen, wählen wir aus der Liste im linken Bereich der Projektseite den Projekttyp WIN32-KONSOLENANWENDUNG aus.

Drücken Sie zuletzt auf OK.

- Es erscheint ein Dialogfeld, in dem Sie auswählen können, wie weit Ihr Projekt von Visual C++ schon vorab konfiguriert und mit Code ausgestattet werden soll. Erliegen Sie bitte nicht der Versuchung, die Option HALLO WELT-Anwendung auszuwählen, sondern verzichten Sie auf jegliche weitere Unterstützung durch die IDE, und wählen Sie die Option EIN LEERES PROJEKT aus. Drücken Sie auf FERTIGSTELLEN.

Es erscheint noch ein abschließendes Kontrollfenster, das Sie mit OK abschicken.

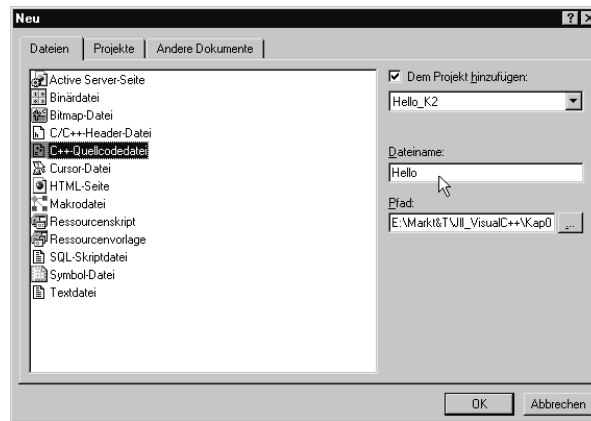


Bild 1.3:
Hinzufügen
einer Quell-
textdatei zu
einem Projekt

4. **Nehmen Sie eine Quelltextdatei in das Projekt auf.** Ein leeres Projekt nutzt uns nicht viel; wir brauchen eine Textdatei, in die wir unseren Quelltext eingeben können. Um eine solche Quelltextdatei in Ihr Projekt aufzunehmen, rufen Sie den Menübefehl PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU auf. Das Dialogfeld, das daraufhin aufspringt, gleicht dem Dialogfeld aus Schritt 2, mit dem Unterschied, daß weniger Registerseiten angeboten werden.

Bleiben Sie auf der Seite DATEIEN, geben Sie einen Namen für die hinzuzufügende Datei ein (nicht den Projektnamen verändern!) und wählen Sie als Dateityp C++-QUELLCODEDATEI aus der Liste aus. Klicken Sie auf OK, um die Datei in Ihr Projekt aufzunehmen.

5. **Geben Sie den Quelltext ein.** Die Datei wird auch gleich in den integrierten Quelltexteditor geladen, so daß Sie direkt den Quellcode eingeben können.

(Wenn Sie sich vergewissern wollen, daß die Datei korrekt in Ihr Projekt aufgenommen wurde, wechseln Sie im Arbeitsbereichfenster, das standardmäßig in den linken Rahmen integriert ist, zur Seite DATEIEN, und expandieren Sie die Knoten – das Pluszeichen – für das Projekt und das Verzeichnis der Quellcode-dateien.)

```
#include <stdio.h>
int main()
{
    printf("Hello, World\n");
    return 0;
}
```

Speichern Sie danach zur Sicherheit die Datei.

6. **Lassen Sie das Programm erstellen.** Rufen Sie dazu den Menübefehl ERSTELLEN/HELLO_K2.EXE ERSTELLEN auf.

Im Ausgabefenster, das in den unteren Rand des IDE-Rahmenfensters integriert ist, wird der Fortgang des Erstellungsprozesses angezeigt. Zum Schluß sollte *Hello_K2.exe* ohne Fehler erstellt worden sein. Die entsprechende Meldung sollte lauten:

Hello_K2.exe - 0 Fehler, 0 Warnung(en)

7. **Führen Sie das Programm aus.** Rufen Sie den Befehl ERSTELLEN/AUSFÜHREN VON HELLO_K2.EXE auf oder drücken und merken Sie sich gleich das Tastaturkürzel **[Strg] + [F5]**.

Bild 1.4:
Ausführung
einer Konso-
lenanwendung
aus der IDE
heraus



Die Meldung »Press any key to continue« wird von der Visual C++-Entwicklungsumgebung hinzugefügt. Sie ist nicht Teil unserer Anwendung, sondern ist ein Service der Visual C++-IDE, der verhindert, daß das Konsolenfenster gleich wieder verschwindet. Zum Schließen des Fensters drücken Sie eine beliebige Taste.

1.2 »Hello, World!« als GUI-Anwendung

Nachdem Sie im vorangehenden Abschnitt erfahren haben, daß Konsolenanwendungen Programme ohne echte grafische Benutzeroberfläche sind, werden Sie sich bereits denken können, daß GUI-Anwendungen folglich Programme mit echten grafischen Benutzeroberflächen sind. Tatsächlich steht GUI auch für nichts anderes als »Graphical User Interface«, zu Deutsch also »grafische Benutzerschnittstelle«.

Wir kommen damit zum eigentlichen Thema dieses Buches: zur Erstellung von Windows-Anwendungen mit Visual C++.

Das soll nicht etwa heißen, daß Konsolenanwendungen keine echten Windows-Anwendungen wären. Aber Sie verfügen eben nicht über die typische, auf Fenstern basierende Oberfläche und entsprechen daher nicht dem Bild, das wir von einer typischen Windows-Anwendung haben.



Unser nächstes Ziel wird nun sein, aus dem *Hello World*-Programm eine echte Windows-Anwendung mit eigenem Fenster zu machen. Das Problem ist nur, daß man dieses Ziel auf vielen Wegen erreichen kann, und für mich als Autor stellt sich die Frage, welcher Weg didaktisch am klügsten ist. Nach reiflicher Überlegung bin ich zu der Überzeugung gelangt, daß es ganz egal ist, welchen Weg ich wähle; viel interessanter und lehrreicher dürfte für Sie sein, wenn ich Ihnen kurz begründe, was für den einen oder anderen Weg spricht.

API oder MFC

Damit überhaupt irgend jemand Windows-Anwendungen schreiben kann, stellt Microsoft die Windows-API zur Verfügung (API steht für Application Programming Interface) – eine Sammlung von über 200 Funktionen und Makros, mit denen man beispielsweise Fenster erzeugen, in Fenster zeichnen oder auf Betriebssystemdienste zugreifen kann. Um mit diesen Funktionen Windows-Programmierung zu betreiben, muß man nicht nur wissen, welche Funktion wann und wofür einzusetzen ist, sondern darüber hinaus auch über intime Kenntnisse des Windows-Betriebssystems verfügen. Nur so kann man sicherstellen, daß die Programme mit dem Betriebssystem korrekt zusammenarbeiten. (Letzteres ist ein ganz wichtiger Punkt bei der Windows-Programmierung.)

Die API

Für den Einsteiger leichter zu überblicken ist die Programmierung mit der MFC. Die MFC ist eine Klassenbibliothek (MFC steht für Microsoft Foundation Classes), in deren Klassen die wichtigsten API-Funktionen gekapselt sind. Dies hat für den Programmierer zwei entscheidende Vorteile:

Die MFC

1. Statt einem Wust von scheinbar gleichberechtigten API-Funktionen gegenüberzustehen, hat man es mit einer relativ überschaubaren und logisch geordneten Klassenhierarchie zu tun.
2. Die MFC befreit den Programmierer von vielen Routineaufgaben und achtet auf die Einhaltung bestimmter Regeln und Formalismen.

Dem professionellen Programmierer geht die Arbeit daher dank der MFC schneller von der Hand, dem Anfänger erleichtert die MFC den Einstieg, da sie viele Details und komplizierte Konzepte der Windows-Programmierung vor dem Programmierer verbirgt. Genau darin liegt aber auch eine Gefahr, denn es verführt den Anfänger dazu, sich seine Programme schnell mit Hilfe der MFC zusammenzubasteln, ohne sich jemals Gedanken um die dahinter stehende Funktionsweise des Windows-Betriebssystems zu machen. Die Folge ist häufig, daß er in seinen Programmierbemühungen nie über ein gewisses Niveau hinauskommen und über kurz oder lang auf Probleme treffen wird, denen er ratlos gegenübersteht. Trotzdem wollen wir uns in diesem Buch die Vorteile der MFC nicht nur für die fortgeschrittene Programmierung, sondern auch für den Einstieg zunutze machen. In Teil 3 des Buches werden wir uns dann aber in einiger Ausführlichkeit mit den Hintergründen der MFC, mit der API-Programmierung und der Funktionsweise des Windows-Betriebssystems auseinandersetzen. Damit erschließen wir uns das Wissen, das bis dahin durch die rosa Brille der MFC vor uns verborgen war.

*Eigenes
Klassengerüst
oder Anwen-
dungs-
Assistent*

Nachdem wir uns nun für die MFC entschieden haben, stellt sich die Frage, ob wir uns mit Hilfe der MFC-Klassen ein eigenes Programm von Grund an aufbauen, oder ob wir auf den MFC-Anwendungs-Assistenten zurückgreifen sollen, der uns ein lauffähiges Anwendungsgerüst liefert, das man nur zu erweitern braucht. Im Grunde ist dies die gleiche Wahl wie zwischen API und MFC. Der Aufbau eines eigenen Anwendungsgerüsts ist ebenso lehrreich wie mühselig, der Einsatz des Assistenten hingegen erleichtert den Einstieg, lädt uns aber eine Schuld auf, die wir in den folgenden Kapiteln abtragen müssen.



Weil es mir wichtig ist, möchte ich es hier noch einmal betonen. Dieses Buch ist zwar für Anfänger und Einsteiger in die MFC-Programmierung geschrieben, aber das soll weder für mich noch für Sie eine Ausrede sein, allen auftauchenden Schwierigkeiten gezielt aus dem Weg zu gehen. Wenn Sie nur ein wenig in die Windows-Programmierung reinschnuppern wollen, steht es Ihnen natürlich frei, den 3. Teil einfach auszulassen. Sollten Sie aber ernsthaft an der Anwendungsentwicklung mit der MFC interessiert sein, studieren Sie den 3. Teil sorgfältig, und eignen Sie sich das nötige Hintergrundwissen an. Der 3. Teil ist zweifelsohne schwieriger zu verdauen als die anderen Teile des Buches, aber für ambitionierte Leser ist er auch der lohnendste.

»Hello, World!« als GUI-Anwendung

1.2.2 Einsatz der integrierten Entwicklungsumgebung (IDE)

Unsere Vorgehensweise zur Erstellung des *Hello World*-Programms als GUI-Anwendung wird so aussehen, daß wir uns vom MFC-Anwendungs-Assistenten ein vollständiges Anwendungsgerüst samt Fenster zaubern lassen, das wir um die Ausgabe des »Hello, World«-Grußes erweitern werden.

Übung 1-3: GUI-Anwendung in IDE erstellen

1. **Rufen Sie Visual C++ auf.** Falls nach dem Programmstart irgendwelche Quelldateien angezeigt werden oder bereits ein Arbeitsbereich geöffnet sein sollte, schließen Sie diese.

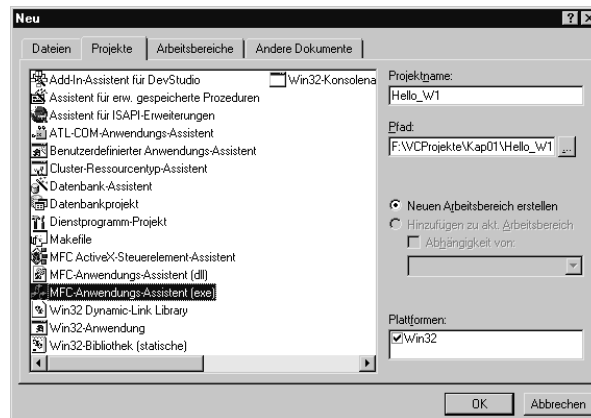


Bild 1.5:
Anlegen eines
neuen Projekts

2. **Legen Sie einen Arbeitsbereich und ein Projekt an.** Rufen Sie dazu den Befehl DATEI/NEU auf, und lassen Sie die Seite PROJEKTE anzeigen.

Geben Sie einen NAMEN für das Projekt ein, beispielsweise *Hello_W1*, und wählen Sie im Feld PFAD ein übergeordnetes Verzeichnis für das Projekt aus (Visual C++ wird unter diesem Verzeichnis ein Unterverzeichnis für das Projekt anlegen, das den gleichen Namen wie das Projekt trägt).

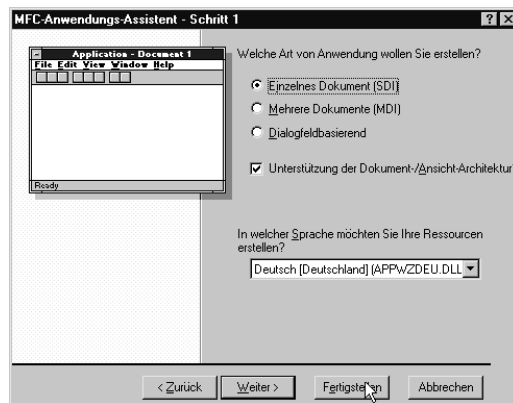
An der per Voreinstellung ausgewählten Option NEUEN ARBEITSBEREICH ERSTELLEN, können Sie erkennen, daß Visual C++ selbst dafür Sorge trägt, daß das neue Projekt einem Arbeitsbereich zugeordnet wird.

Aus der Liste im linken Teil des Dialogfeldes wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN (EXE) aus.

KAPITEL 1 Zur Einstimmung

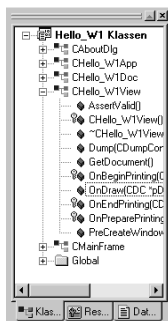
Drücken Sie zuletzt auf OK.

Bild 1.6:
Konfiguration
des Projekts
im Anwen-
dungs-Assi-
stenten



- Es erscheint das erste Dialogfeld des Anwendungs-Assistenten, in dem Sie die Option EINZELNES DOKUMENT (SDI) auswählen und danach auf FERTIGSTELLEN drücken. (Mit den einzelnen Optionen und Seiten des Assistenten werden wir uns im 3. Kapitel beschäftigen.)

Es erscheint noch ein abschließendes Kontrollfenster, das Sie mit OK abschicken.



- Lassen Sie einen Text im Fenster der Anwendung anzeigen.** Dazu müssen wir uns kurz darüber informieren, welche Klassen und Methoden der Anwendungs-Assistent bereits für uns aufgesetzt hat.

Auf der linken Seite der Visual-IDE sollten Sie das Arbeitsbereichfenster sehen, das standardmäßig an dieser Stelle in den Rahmen des IDE-Hauptfensters integriert wird. (Wenn nicht, können Sie das Fenster über den Menübefehl ANSICHT/ARBEITSBEREICH anzeigen lassen.) Expandieren Sie auf der Seite KLASSEN den Knoten HELLO_W1 KLASSEN, um die in dem Projekt deklarierten Klassen anzeigen zu lassen, und danach den Knoten CHELLO_W1VIEW, um zu sehen, welche Methoden für die Klasse CHello_W1View definiert wurden. Wie das Suffix »View« andeutet (View steht für Sicht, Ansicht), können wir über diese Klasse auf die Anzeige unserer Anwendung einwirken. Konkret ist es die Methode OnDraw(), der wir die Anweisung zur Anzeige des Hello World-Grußes hinzufügen.

Doppelklicken Sie im Arbeitsbereichfenster auf die Methode OnDraw(), um diese in den Editor zu laden.

Erweitern Sie die Methode um einen Aufruf der Methode `TextOut()`:

```
// CHello_W1View Zeichnen
void CHello_W1View::OnDraw(CDC* pDC)
{
    CHello_W1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    pDC->TextOut(30,30,"Hello, World!");
}
```

5. **Lassen Sie das Programm erstellen, und führen Sie es aus.** Rufen Sie dazu nacheinander die Menübefehle ERSTELLEN/HELLO_W1.EXE ERSTELLEN und ERSTELLEN/AUSFÜHREN VON HELLO_W1.EXE auf, oder rufen Sie gleich nur den Befehl ERSTELLEN/AUSFÜHREN VON HELLO_W1.EXE auf, und bestätigen Sie in dem erscheinenden Dialogfeld, daß die geänderten Dateien neu erstellt werden sollen.

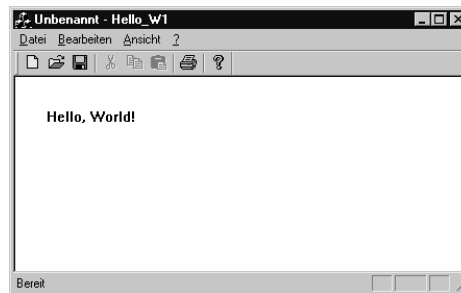


Bild 1.7:
Das Programm
Hello_W1

1.2.3 Einsatz der Kommandozeilenversion des Compilers

Der logische Abschluß unseres Ausflugs in die Anwendungserstellung wäre die Übersetzung des *Hello_W1*-Programms mit der Kommandozeilenversion des Visual C++-Compilers. Aber geht dies überhaupt? Blickt man in das Arbeitsbereichsfenster der IDE (siehe Schritt 4), können einem ob der Vielzahl an Dateien, die zu dem Programm gehören, Zweifel kommen. Zudem enthält das *Hello_W1*-Programm neben normalen C-Quelltextdateien (.cpp, .h) auch Ressourcendateien (.rc, .ico, etc.), die von einem C-Compiler gar nicht übersetzt werden können. Schließlich greift das Programm auf Klassen der MFC-Bibliothek zurück, was bedeutet, daß der MFC-Code irgendwie in das Programm eingebunden werden muß. Bedeutet dies aber

auch, daß wir das Programm nicht von der Kommandozeile aus erstellen können? Nein!

Ein Compiler – verschiedene Aufrufmöglichkeiten Tatsächlich gibt es nur *einen* Visual C++-Compiler. Der Compiler, den wir über die Kommandozeile aufrufen, ist der gleiche, der auch in der IDE verwendet wird. Anders ausgedrückt: Jedes Programm, das in der IDE erstellt werden kann, läßt sich auch von der Kommandozeile aus übersetzen. Der einzige Unterschied ist, daß man beim Aufruf von der Kommandozeile selbst dafür verantwortlich ist, dem Compiler alle für die Erstellung des Programms nötigen Informationen zu übergeben, während uns in der IDE diese Arbeit abgenommen wird. Für die Erstellung von Windows-Programmen ist dies eine große Erleichterung, so daß Sie Ihre Windows-Programme fast immer aus der IDE heraus erstellen werden. Daß wir an dieser Stelle einmal die andere Möglichkeit durchspielen, dient dem Zweck, Sie ein wenig besser mit der Funktionsweise der IDE und des Visual C++-Compilers vertraut zu machen und Sie mit einem Gefühl der Dankbarkeit für die Projektverwaltung der IDE zu erfüllen.



Bevor Sie mit dem Eintippen der nachfolgend beschriebenen Kommandozeilenaufrufe beginnen, sollten Sie den Befehl DOSKEY ausführen, um die Befehlsaufzeichnung zu aktivieren. Wenn Sie dann nach dem Abschicken eines Befehls bemerken, daß Sie sich irgendwo vertippt haben, können Sie den Befehl über die Pfeiltasten wieder aufrufen und editieren.

Übung 1-4: GUI-Anwendung mit cl.exe erstellen

1. **Legen Sie eine Kopie des Hello_W1-Projekts an.** Richten Sie dazu im Explorer ein Verzeichnis *Hello_W2* ein, und kopieren Sie sämtliche Quelltextdateien (Dateien mit den Endungen cpp, h und rc) aus dem Verzeichnis des *Hello_W1*-Projekts in das Verzeichnis *Hello_W2*. Auch das Unterverzeichnis *RES* muß samt Inhalt nach *Hello_W2* kopiert werden.
2. **Übersetzen Sie die Ressourcendatei.** Das Programm verwendet eine Ressourcendatei namens *Hello_W1.rc*, in der das Symbol, das Menü und das Info-Dialogfeld der Anwendung definiert sind. All diese Ressourcen sind in einer speziellen Ressourcenskriptsprache aufgesetzt, die vom C-Compiler nicht verstanden wird. Man braucht daher einen eigenen Ressourcen-Compiler, der aus der Ressourcenskriptdatei (.rc) eine binäre Ressourcendatei (.res) erzeugt. Diese kann später mit dem Linker in das Programm eingebunden werden. (In Kapitel 2 werden wir uns noch ein wenig eingehender mit den Ressourcen beschäftigen.)

Der Ressourcen-Compiler von Visual C++ heißt einfach `rc` und wird wie folgt aufgerufen:

```
rc Hello_W1.rc
```

3. **Übersetzen Sie die Quelltextdateien.** Wir können alle Quelltextdateien in einem Rutsch kompilieren lassen, indem wir hinter `cl` die Namen der zu kompilierenden Dateien auflisten. Zuvor müssen wir aber noch zwei Optionen übergeben:

`/c` sorgt dafür, daß die Dateien nur übersetzt und nicht gelinkt werden. Zum Binden der übersetzten Dateien werden wir den Linker im nächsten Schritt selbst aufrufen, denn nur so können wir ihn anweisen, auch die Ressourcendatei `Hello_W1.res` in das Programm mit einzubinden.

`/MT` sorgt dafür, daß eine passende Version der MFC-Bibliothek eingebunden wird. Für die Autorenedition von Visual C++ verwenden Sie: `/MD /D "_AFXDLL" /D "_MBCS"`.

Der vollständige Aufruf sieht damit so aus:

```
cl /c /MT Hello_W1.cpp Hello_W1Doc.cpp Hello_W1View.cpp
MainFrm.cpp stdafx.cpp
```

3. **Binden Sie die Objektdateien (.obj) zu einem ausführbaren Programm.** Hierzu bleibt uns wieder einmal nichts anderes übrig, als die einzelnen Objektdateien (.obj) plus der binären Ressourcendatei (.res) hinter dem Namen des Linkers aufzulisten. Zusätzlich müssen wir dem Linker anzeigen, daß es sich um eine Windows-Anwendung handelt (Option `/subsystem:windows`).

```
link /subsystem:windows Hello_W1.obj Hello_W1Doc.obj
Hello_W1View.obj
MainFrm.obj stdafx.obj Hello_W1.res
```

```

Eingabeaufforderung
F:\UCProjekte\KAP01\HELLO_W2>rc Hello_W1.rc
F:\UCProjekte\KAP01\HELLO_W2>cl /c /MT Hello_W1.cpp Hello_W1Doc.cpp Hello_W1View
.cpp MainFrm.cpp stdafx.cpp
Optimierender Microsoft (R) 32-Bit C/C++-Compiler, Version 12.00.8168, fuer x86
Copyright (C) Microsoft Corp 1984-1998. Alle Rechte vorbehalten.

Hello_W1.cpp
Hello_W1Doc.cpp
Hello_W1View.cpp
MainFrm.cpp
stdafx.cpp
Generieren von Code...

F:\UCProjekte\KAP01\HELLO_W2>link /subsystem:windows Hello_W1.obj Hello_W1Doc.ob
j Hello_W1View.obj MainFrm.obj stdafx.obj Hello_W1.res
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

F:\UCProjekte\KAP01\HELLO_W2>hello_w1.exe
  
```

Bild 1.8:
Aufruf des
Compilers von
der Komman-
dozeile

1.3 Die Kommandozeilenversion des Compilers

In den weiteren Kapiteln werden wir uns nicht mehr mit der Kommandozeilenversion des Compilers, sondern nur noch mit der integrierten Entwicklungsumgebung beschäftigen. Da ich das Thema Kommandozeilenversion aber nun einmal angefangen habe, möchte ich es auch ordentlich abschließen. Ich tue dies vornehmlich aus drei Gründen:

1. Um der IDE den Nimbus zu nehmen
3. Um Sie auf die Projektverwaltung von Visual C++ vorzubereiten
2. Um Ihnen den Austausch von Programmen zwischen verschiedenen Compilern zu erleichtern

IDE und Kommandozeilenversion

Die IDE greift zur Übersetzung und Erstellung der in ihr bearbeiteten Programme auf die gleichen Entwicklerwerkzeuge zurück, die wir auch direkt von der Kommandozeile aus aufrufen können. Die wichtigsten Entwicklerwerkzeuge sind:

- cl* Der eigentliche Compiler, der Ihren C/C++-Code in Maschinencode (Objektdateien) übersetzt und anschließend – falls Sie nicht die Option `/c` gesetzt haben – den Linker aufruft
- rc* Der Ressourcen-Compiler, der aus Ressourcenskriptdateien binäre Ressourcen (*res*-Datei) erzeugt.
- link* Der Linker, der Objektdateien (*.obj*), Bibliotheksdateien (*.lib*) und binäre Ressourcen (*.res*) zu ausführbaren Programmen zusammenbindet.

Der Aufruf dieser Entwicklerwerkzeuge sieht schematisch stets gleich aus:

Programm /Optionen Dateien

Welche Optionen Ihnen für die einzelnen Entwicklerwerkzeuge zur Verfügung stehen, können Sie der Online-Hilfe entnehmen (schlagen Sie unter den Stichworten »Compiler«, »Linker«, »Ressourcen-Compiler« oder »Optionen« nach). Alternativ können Sie auch das betreffende Werkzeug von der Kommandozeile aus mit dem Argument `/?` aufrufen.

Übersetzung modularer Programme

Größere Programme bestehen meist aus mehreren Modulen. Jedes Modul stellt eine eigene Übersetzungseinheit dar, die aus einer *.cpp*-Datei und den

zugehörigen Header-Dateien besteht. Im einfachsten Fall kann man diese Module übersetzen, indem man sie zusammen dem Compiler übergibt:

```
cl Modul1.cpp Modul2.cpp
```

Der Compiler erzeugt aus den Modulen die Objektdateien *Modul1.obj* und *Modul2.obj* und übergibt sie dem Linker, der daraus eine EXE-Datei erzeugt, die den Namen der ersten Objektdatei trägt (*Modul1.exe*).

Es ist leicht nachzuvollziehen, daß der Aufruf des Compilers um so aufwendiger wird, je mehr Module ein Projekt enthält. Hinzukommt, daß man dem Compiler meist eine Vielzahl von Optionen übergeben möchte und daß unter Umständen verschiedene Module unterschiedliche Compiler-Optionen erfordern. Die Erstellung eines Programms kann damit schnell in echte Arbeit ausarten – eine ebenso mühselige wie unnötige Arbeit.

Diese Arbeit möchte man sich natürlich ersparen. Hierzu gibt es zwei Ansätze:

- ✗ Make-Dateien zur Erstellung von der Kommandozeile aus
- ✗ Projekte zur Erstellung in integrierten Entwicklungsumgebungen

Make-Dateien enthalten, in einer eigenen einfachen Sprache verfaßt, alle Anweisungen zur Erstellung eines Programms. Ausgeführt wird die Make-Datei mit einem speziellen Dienstprogramm, das im Falle von Visual C++ NMAKE heißt.

Um das Projekt *Hello_W2* von oben mit Hilfe einer Make-Datei zu erstellen, könnte man beispielsweise eine Make-Datei *Hello_W2.mak* mit folgenden Anweisungen aufsetzen:

```
Hello_W1.exe : Hello_W1.obj Hello_W1Doc.obj \
                Hello_W1View.obj MainFrm.obj \
                StdAfx.obj Hello_W1.res
    link /subsystem:windows Hello_W1.obj \
        Hello_W1Doc.obj Hello_W1View.obj \
        MainFrm.obj StdAfx.obj Hello_W1.res
Hello_W1.res : Hello_W1.rc
    rc    Hello_W1.rc
Hello_W1.obj : Hello_W1.cpp
    cl    /c /MT Hello_W1.cpp
Hello_W1Doc.obj : Hello_W1Doc.cpp
    cl    /c /MT Hello_W1Doc.cpp
Hello_W1View.obj : Hello_W1View.cpp
    cl    /c /MT Hello_W1View.cpp
```



```
MainFrm.obj: MainFrm.cpp
           cl      /c /MT MainFrm.cpp
StdAfx.obj: StdAfx.cpp
           cl      /c /MT StdAfx.cpp
```

und das Programm dann mit Hilfe des folgenden Aufrufs von der Kommandozeile aus kompilieren:

```
nmake Hello_W2.mak
```

Eine andere Möglichkeit ist, die Make-Datei unter dem Namen *makefile* abzuspeichern und einfach *NMAKE* aufzurufen.

Der Vorteil der Make-Dateien liegt auf der Hand. Man braucht die Befehle zur Übersetzung und Bindung der Module nur einmal aufzusetzen und nicht für jede Kompilierung des Projekts neu in die Kommandozeile einzutippen.

Projektverwaltung In integrierten Entwicklungsumgebungen übernimmt die Projektverwaltung die Aufgabe der Make-Datei. Die Projektverwaltung verfügt über Befehle und Fenster, über die der Programmierer festlegen kann, welche Quelldateien zu dem Projekt gehören und wie diese zu kompilieren und zu linkern sind. Die IDE speichert diese Einstellungen in einer speziellen Datei. Im Grunde ist es also so, daß der Programmierer seine Programme über Menübefehle und Dialogfelder konfiguriert und die IDE aus diesen Informationen eine passende Make-Datei erzeugt. Wir werden uns dies im nächsten Kapitel für die Visual C++-Projektverwaltung noch näher ansehen.

Andere Compiler

Die Projektverwaltungen der integrierten Entwicklungsumgebungen definieren meist ein eigenes Format für die Abspeicherung der Projektkinformationen. Obwohl diese Projektdaten also alle Informationen enthalten, die man auch in Make-Dateien findet, sind es insofern keine echten Make-Dateien, als sie nicht mit *NMAKE* (oder anderen *MAKE*-Programmen) ausgeführt werden können. Es sollte aber für die IDE nicht schwer sein, aus einer ihrer Projektdaten eine zugehörige Make-Datei zu erstellen. Tatsächlich gibt es dazu in der Visual C++-IDE einen eigenen Befehl: *PROJEKT/MAKEFILE EXPORTIEREN*.

Wann werden Sie diesen Befehl benutzen?

- ✗ Wenn Sie Ihre Programme auf Systemen erstellen wollen, auf denen Sie mit den Kommandozeilenwerkzeugen (*cl*, *link*, *rc*), nicht aber der IDE arbeiten können (sei es, daß die IDE aus Platzgründen nicht installiert wurde, sei es, daß die IDE wegen Unverträglichkeit mit dem Betriebssystem nicht installiert werden kann).

- ✗ Wenn Sie Ihre Programme mit anderen C++-Compilern erstellen wollen (Borland-Compiler, GNU-Compiler). Die Portierung auf andere Compiler ist aber nur selten damit getan, daß man sich eine passende Make-Datei erzeugen läßt. Zum einen muß die Make-Datei üblicherweise nachbearbeitet werden (beispielsweise müssen die neuen Compiler-Werkzeuge eingetragen werden, Optionen müssen eventuell angepaßt werden, die Verzeichnispfade zu den Bibliotheken und Header-Dateien müssen verfügbar sein), zum anderen muß der Quelltext selbst portierbar sein (massive Probleme gibt es meist dann, wenn man compilerspezifische Makros oder Schlüsselwörter verwendet, aber auch die Verwendung spezieller Bibliotheken wie der MFC erschweren die Portierung erheblich).

Alles in allem ist die Portierung von Programmen auf andere Entwicklungssysteme und Compiler ein äußerst mühsames Unterfangen, mit dem Anfänger wie Professionelle glücklicherweise eher selten konfrontiert werden. Sollte die Portierung aber doch einmal zwingend erforderlich sein, ist es gut zu wissen, wo man ansetzen kann.



1.4 Zusammenfassung

In Visual C++ werden Programme in Form von Projekten verwaltet. Der erste Schritt bei der Anwendungsentwicklung ist daher stets die Einrichtung eines neuen Projekts (Befehl DATEI/NEU). Zur Einrichtung eines Projekts gehört neben Angaben zum Namen, Verzeichnis und übergeordnetem Arbeitsbereich auch die Auswahl eines Projekttyps (oder eines Assistenten), der die Art der zu erstellenden Anwendung festlegt.

- ✗ Für Konsolenanwendungen wählt man im Dialogfeld NEU den Projekttyp WIN32-KONSOLENANWENDUNG.
- ✗ Für Windows-Anwendungen wählt man im Dialogfeld NEU den Projekttyp WIN32-ANWENDUNG.
- ✗ Für Windows-MFC-Anwendungen kann man im Dialogfeld NEU auch den MFC-ANWENDUNGS-ASSISTENT (EXE) auswählen.

Die Befehle zur Kompilierung und Ausführung des Programms stehen im Menü ERSTELLEN.

Die gleichen Entwicklerwerkzeuge (Compiler, Linker), die intern aufgerufen werden, wenn man ein Programm in der integrierten Entwicklungsumgebung erstellen läßt, kann man auch selbst von der Kommandozeile aus auf-

rufen. Für kleinere Anwendungen, die nur aus einer Quelltextdatei bestehen und nur wenige zusätzliche Bibliotheken benötigen, kann es sogar bequemer sein, die Anwendungen von der Kommandozeile statt aus der IDE heraus zu erstellen.

1.5 Fragen

1. Was sind Konsolenanwendungen, und was sind GUI-Anwendungen?
2. Wofür steht das Akronym »IDE«?
3. Muß man MFC-Programme in der IDE von Visual C++ erstellen?
4. Wie heißt die Kommandozeilenversion des Compilers von Visual C++?
5. Wie beginnt man in Visual C++ mit der Erstellung eines neuen Programms?
6. Als wir in Visual C++ eine neue Konsolenanwendung anlegten, haben wir uns im Dialogfeld des Konsolen-Assistenten dafür entschieden, mit einem leeren Projekt zu beginnen (Schritt 3). Wir hätten aber auch den Assistenten eine fertige *Hallo Welt*-Anwendung erstellen lassen können. Tut man dies, stellt man fest, daß der Assistent neben dem Quellcode für den »Hallo Welt«-Gruß noch eine zweite Quelltextdatei namens *StdAfx.cpp* anlegt, die abgesehen von der Einbindung einer Header-Datei namens *StdAfx.h* keinen Code enthält. Wozu dient diese Datei?

(Tip: Wenn Sie nicht mehr weiter wissen, schauen Sie sich die Compiler-Einstellungen für die Datei an.)

1.6 Aufgaben

1. In Abschnitt 1.1 habe ich behauptet, daß für Konsolenanwendungen, die aus dem Windows Explorer heraus aufgerufen werden, automatisch ein Eingabeaufforderung-Fenster geöffnet wird. Leider ließ sich dies nur schwer überprüfen, da das Fenster nach Ablauf des Programms sofort wieder geschlossen wurde. Erweitern Sie daher die *Hello World*-Konsolenanwendung um einen Aufruf der C-Funktion `getchar()`, die auf die Eingabe eines beliebigen Zeichens wartet. Wenn Sie jetzt das Programm über den Windows Explorer starten (die EXE-Datei steht im Unterverzeichnis *Debug*), wird das Programm erst beendet, nachdem Sie die Eingabetaste gedrückt haben.

2. Über den Windows Explorer können Sie das Programm aus Übung 1 auch mehrfach starten. Rufen Sie es zwei- bis dreimal auf, und vergewissern Sie sich, daß jede Programminstanz nur auf Eingaben reagiert, die an ihr spezielles Konsolenfenster geschickt wurden.
3. Wenn Sie in der IDE ein Projekt geöffnet haben und den Befehl PROJEKT/EINSTELLUNGEN aufrufen, können Sie sich die Projekteinstellungen für das Gesamtprojekt und für die einzelnen Quelldateien anschauen. Konzentrieren Sie sich auf die Seiten C/C++, LINKER und RESSOURCEN, über die Sie die Compiler und den Linker konfigurieren können. Im ganz unten gelegenen OPTIONEN-Feld können Sie überprüfen, wie die Einstellungen aus den darüber gelegenen Steuerelementen in Optionen für den Compiler (Linker) umgewandelt werden. Spielen Sie ein bißchen mit den Einstellungen herum, und machen Sie sich mit den wichtigsten Optionen vertraut. Kompilieren Sie die *Hello World*-Konsolenanwendung noch einmal von der Kommandozeile aus, und lassen Sie das Programm hinsichtlich seiner Codegröße optimieren.
5. Die IDE speichert die Informationen zu den Projekten in der Datei mit der Endung *.dsp* ab. Schauen Sie sich den Inhalt dieser Datei für das *Hello_W1*-Projekt an. Sie müssen die Datei dazu in einen Texteditor (beispielsweise den Windows-Editor oder Word) laden. Laden Sie das Projekt noch einmal in die IDE. (Sie müssen dazu den Befehl DATEI/ARBEITSBEREICH ÖFFNEN aufrufen und die DSW-Datei des Arbeitsbereichs auswählen, in dem das Projekt enthalten ist.) Lassen Sie aus der Projektdatei (*.dsp*) eine Make-Datei (*.mak*) erstellen (Befehl PROJEKT/MAKEFILE EXPORTIEREN), und vergleichen Sie den Inhalt der beiden Dateien.

Von der Idee zum Programm

In diesem Kapitel wollen wir einmal den gesamten Prozeß der Anwendungsentwicklung anhand eines Beispiels und acht Übungen von Anfang bis Ende durchspielen. Unser Augenmerk gilt dabei der integrierten Entwicklungsumgebung (IDE) von Visual C++ und der Frage, welche Komponenten der IDE für uns wichtig sind und wann und wie uns diese Komponenten im Prozeß der Anwendungsentwicklung zum Vorteil gereichen. Wir werden uns also noch einmal etwas ausführlicher mit Projekten und Arbeitsbereichen beschäftigen, wir werden sehen, was uns der Quelltexteditor zu bieten hat, wir werfen einen Blick auf die Ressourceneditoren, den Compiler und den Debugger. Wie man ein Konzept für ein zu erstellendes Programm entwickelt oder wie der Compiler aus unserem Quelltext eine ausführbare EXE-Datei zaubert, soll uns hier nicht interessieren. Mit dem Compiler sind Sie sicherlich schon von der C/C++-Programmierung her vertraut, und Programm-Design und -Planung lernt man am besten aus Erfahrung und durch konsequente Modularisierung.

Sie erfahren in diesem Kapitel:

- ✕ Wie man Zinsen berechnet
- ✕ Wie man mit Projekten und Arbeitsbereichen arbeitet
- ✕ Wie man effizient mit dem Editor arbeitet
- ✕ Wie man eigene Quelltextdateien in ein Projekt integriert



- ✗ Wie man Ressourcen (beispielsweise Dialogfenster oder Symbole) aufsetzt und in ein Programm aufnimmt
- ✗ Wie man Programme erstellt und testet

2.1 Ein Programm zur Zinsberechnung

Das Beispielprogramm, anhand dessen wir die Anwendungsentwicklung mit Visual C++ exemplarisch durchspielen werden, ist ein einfaches Programm zur Zinsberechnung.

Am meisten Freude werden Sie an dem Programm haben, wenn Sie selbst über ein Sparbuch mit festem Zinssatz verfügen. Dann können Sie das Programm gleich dazu benutzen, sich auszurechnen, wie schnell sich ihr Kapital in den nächsten Jahren vermehren wird. Vermutlich wird das Ergebnis wegen der niedrigen Zinssätze, die wir derzeit haben, wohl eher enttäuschend sein. Man könnte das Beispiel natürlich abändern, so daß es ausrechnet, wie ein aufgenommenes Darlehen Jahr für Jahr abgetragen wird, aber erstens ist die Berechnung von Darlehenstilgungen etwas aufwendiger als eine einfache Zinsberechnung, und zweitens möchte ich Sie ja nicht dazu animieren, sich nur wegen der günstigen Darlehenszinsen zu verschulden.

Wir bleiben also beim Sparbuch und schreiben ein Programm, das ausrechnet, wieviel Geld sich auf dem Sparbuch ansammelt, wenn man ein Startkapital K_0 einzahlt und über n Jahre mit x Prozent verzinsen läßt.

Das Programm soll als Hauptfenster ein Dialogfenster erhalten, das Eingabefelder enthält, über die der Anwender

- ✗ das Startkapital,
- ✗ die Verzinsung (in Prozent) und
- ✗ die Laufzeit

eingeben kann.

Aus diesen Daten wird dann berechnet, wie sich das Kapital über die Jahre vermehrt. Dazu muß man wissen, nach welcher Formel die Kapitalentwicklung berechnet werden kann.

Zinsen und Folgen

Wenn wir einmal unterstellen, daß die Zinsen nicht auf das Spargbuch angerechnet, sondern auf ein getrenntes Konto überwiesen werden, kann die Kapitalentwicklung nach der folgenden Formel berechnet werden:

$$\text{Endkapital} = \text{Startkapital} * (1 + \text{Laufzeit} * \text{Verzinsung}/100)$$

wobei das Startkapital in DM, die Verzinsung in Prozent und die Laufzeit in Jahren angegeben werden.

Mehr brauchen wir im Grunde nicht für unser Programm zu wissen. Da die Mathematik aber für sich selbst auch eine spannende Sache ist und ich davon ausgehe, daß nicht jeder meiner Leser mit der Theorie der Folgen vertraut ist, möchte ich die Gelegenheit wahrnehmen und Sie auf einen kurzen Ausflug ins Reich der Mathematik mitnehmen. Wen dies nicht interessiert oder wer sich die obige Formel selbst herleiten kann, der möge einfach zum nächsten Abschnitt springen.

Wenn man ein Startkapital von K_0 mit 10% verzinst (ohne den Zinsertrag selbst weiter zu verzinsen), wächst das Kapital jedes Jahr um 10% des ursprünglichen Startkapitals – also um $0,1 * K_0$. *Ohne Zinseszins*

Jahr	Verhältnis K_n zu K_{n-1}	Verhältnis K_n zu K_0
0. Jahr	K_0	K_0
1. Jahr	$K_1 = K_0 + K_0 * 0.1$	$K_1 = K_0 + 1 * K_0 * 0.1 = K_0 * (1 + 1 * 0.1)$
2. Jahr	$K_2 = K_1 + K_0 * 0.1$	$K_2 = K_0 + 2 * K_0 * 0.1 = K_0 * (1 + 2 * 0.1)$
3. Jahr	$K_3 = K_2 + K_0 * 0.1$	$K_3 = K_0 + 3 * K_0 * 0.1 = K_0 * (1 + 3 * 0.1)$
...	...	...
n-tes Jahr	$K_n = K_{n-1} + K_0 * 0.1$	$K_n = K_0 + n * K_0 * 0.1 = K_0 * (1 + n * 0.1)$

Tabelle 2.1:
Entwicklung
einer arith-
metischen
Folge

Schreibt man – wie in der obigen Tabelle – auf, wie sich das Kapital Jahr für Jahr entwickelt, erhält man eine Folge von Kapitalerträgen $K_0, K_1, K_2 \dots K_n$, die sich dadurch auszeichnet, daß zwischen zwei aufeinanderfolgenden Elementen der Folge stets die gleiche Differenz besteht (siehe zweite Spalte der Tabelle). Eine solche Folge von Zahlen bezeichnet man in der Mathematik als eine arithmetische Folge.



Das Kennzeichen einer arithmetischen Folge ist, daß zwischen zwei beliebigen aufeinanderfolgenden Elementen der Folge immer die gleiche Differenz liegt.

Im Falle unserer Zinsformel ist die konstante Differenz zwischen den Elementen der Folge $K_0 \cdot 0.1$. Weit mehr als die Differenz zwischen zwei Elementen interessiert uns aber, wie wir am bequemsten ausrechnen können, wieviel Kapital sich nach n Jahren angesammelt hat.

Da wir mittlerweile wissen, daß die Kapitalentwicklung eine arithmetische Folge bildet, brauchen wir dazu nur in einer mathematischen Formelsammlung nachzuschlagen und erfahren dort, daß das n -te Element einer arithmetischen Folge zu

$$a_n = a_0 + n \cdot d$$

gegeben ist (für den Fall, daß a_0 das erste Element ist).



Beginnt die Indexierung mit 1, lautet die Formel $a_n = a_1 + (n-1) \cdot d$.

Überträgt man diese Formel auf unsere Zinsrechnung, ist n die Laufzeit und d die Differenz $K_0 \cdot 0.1$ und die Formel lautet $K_n = K_0 + n \cdot K_0 \cdot 0.1 = K_0 (1 + n \cdot 0.1)$ (siehe dritte Spalte der Tabelle).

Mit Zinseszins

Wie sähe die Formel aus, wenn die Zinserträge jedes Jahres auf dem Sparbuch gutgeschrieben und im nächsten Jahr mit 10% (Zinssatz von 0,1) verzinst würden?

Tabelle 2.2:
Entwicklung
einer geometri-
schen Folge

Jahr	Verhältnis K_n zu K_{n-1}	Verhältnis K_n zu K_0
0. Jahr	K_0	K_0
1. Jahr	$K_1 = K_0 + K_0 \cdot 0.1 = K_0 \cdot (1 + 0.1)$	$K_1 = K_0 \cdot (1 + 0.1)$
2. Jahr	$K_2 = K_1 + K_1 \cdot 0.1 = K_1 \cdot (1 + 0.1)$	$K_2 = K_0 (1 + 0.1) \cdot (1 + 0.1)$
3. Jahr	$K_3 = K_2 + K_2 \cdot 0.1 = K_2 \cdot (1 + 0.1)$	$K_3 = K_0 (1 + 0.1) \cdot (1 + 0.1) \cdot (1 + 0.1)$
...	...	...
n -tes Jahr	$K_n = K_{n-1} + K_{n-1} \cdot 0.1 = K_{n-1} \cdot (1 + 0.1)$	$K_n = K_0 \cdot (1 + 0.1)^n$

Wie man der zweiten Spalte der Tabelle entnehmen kann, ist in diesem Fall nicht die Differenz zweier aufeinanderfolgender Elemente konstant, sondern der Quotient. Eine solche Folge von Zahlen bezeichnet man in der Mathematik als eine geometrische Folge.

Das Kennzeichen einer geometrischen Folge ist, daß der Quotient zweier beliebiger aufeinanderfolgenden Elemente der Folge konstant ist.



Im Falle unserer Zinsformel ist der konstante Quotient zwischen den Elementen der Folge $q = (1 + 0.1)$. Weit mehr als die mathematische Theorie interessiert uns aber auch hier, wie wir am bequemsten ausrechnen können, wieviel Kapital sich nach n Jahren angesammelt hat.

Da wir mittlerweile wissen, daß die Kapitalentwicklung eine geometrische Folge bildet, brauchen wir dazu wieder nur in unserer mathematischen Formelsammlung nachzuschlagen und erfahren dort, daß das n -te Element einer geometrischen Folge zu

$$a_n = a_0 * q^n$$

gegeben ist (für den Fall, daß a_0 das erste Element ist und q^n die n -te Potenz von q bezeichnet).

Überträgt man diese Formel auf unsere Zinsrechnung, ist n die Laufzeit und der Quotient $1 + 0.1$, und die Formel lautet $K_n = K_0 * (1 + 0.1)^n$ (siehe dritte Spalte der Tabelle).

Soviel zur zugrundeliegenden Mathematik, zurück zu unserem Programm.

2.2 Projekte und Arbeitsbereiche

Daß in Visual C++ Programme in Form von Projekten verwaltet werden, haben Sie bereits im ersten Kapitel dieses Buches erfahren. Wenn Sie auch den letzten Abschnitt im ersten Kapitel gelesen haben, wissen Sie auch bereits, daß die Kompilation von Programmen durch die Projektverwaltung wesentlich vereinfacht wird. Doch was konkret ist jetzt ein Projekt?

2.2.1 Was ist ein Projekt?

Ein Projekt ist im Grunde ein Bausatz für die Erstellung eines Programms. Es besteht aus

- ✕ den Quelltextdateien des Programms (dem Baumaterial),
- ✕ der Information, wie diese Quelltextdateien übersetzt und zu einer ausführbaren Datei (dem Programm) gelinkt werden können (der Bauanleitung),

KAPITEL 2 Von der Idee zum Programm

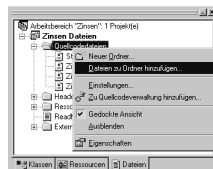
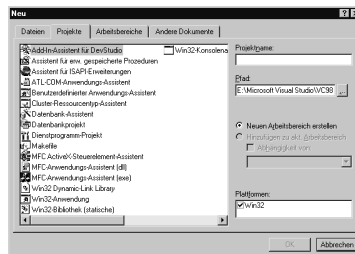
- ✗ einem Verzeichnis auf Ihrer Festplatte, in dem die Dateien des Projekts abgelegt sind (der Baukasten).

Wie unterstützt die IDE die Arbeit mit Projekten?

Die IDE stellt uns eine Reihe von Menübefehlen und Dialogfeldern zur Verfügung, mit deren Hilfe Projekte erzeugt, erweitert, überwacht und konfiguriert werden können.

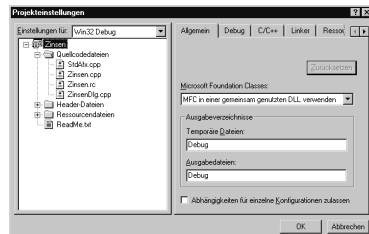
Tabelle 2.3:
Die wichtigsten Elemente
der Projekt-
verwaltung

Funktion	Menübefehl / Dialogfeld	Beschreibung
Neue Projekte anlegen	DATEI/NEU; Dialogfeld NEU	Im Dialogfeld NEU auf der Seite PROJEKTE sind Sie aufgefordert, alle erforderlichen Angaben zu dem neuen Projekt zu machen (Name, Verzeichnis, Arbeitsbereich, Projekttyp). Wenn Sie das Dialogfeld bestätigen, wird das neue Projekt angelegt.
Quelldateien hinzufügen	Dialogfeld NEU, Seite DATEIEN Befehle im Menü PROJEKT/DEM PROJEKT HINZUFÜGEN Kontextmenü des Arbeitsbereichsfensters	Wie Sie sehen, gibt es mehrere Wege, Quelldateien in ein Projekt aufzunehmen. Teilweise führen die Befehle aber zu dem gleichen Ergebnis (sprich zu dem gleichen Dialogfeld). Unterscheiden muß man allerdings, ob man eine bereits bestehende Quelltextdatei in das Projekt aufnehmen will oder ob man eine ganz neue Datei anlegen und im Projekt abspeichern möchte. Im ersten Fall kann man über das Kontextmenü des Arbeitsbereichsfensters gehen oder den Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN aufrufen. Im zweiten Fall wählt man einen der Menübefehle DATEI/NEU oder PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU.



Funktion	Menübefehl / Dialogfeld	Beschreibung
Quelldateien löschen	Arbeitsbereichsfenster, Seite DATEIEN	Markieren Sie im Arbeitsbereichsfenster die zu entfernende Datei, und drücken Sie die [Entf] -Taste, oder rufen Sie den Menübefehl BEARBEITEN/LÖSCHEN auf. Beachten Sie, daß die Datei lediglich aus dem Projekt entfernt wird; sie wird nicht von der Festplatte gelöscht.
Projekte konfigurieren	PROJEKTE/EINSTELLUNGEN; Dialogfeld PROJEKTEINSTELLUNGEN	Über das Dialogfeld PROJEKTEINSTELLUNGEN können Sie festlegen, mit welchen Optionen das Programm kompiliert und gelinkt werden soll.
Projekte überwachen	Arbeitsbereichsfenster; Browser; Befehl BEARBEITEN/SUCHEN IN DATEIEN	Daß die IDE Ihre Quelldateien in Form von Projekten überwacht, hat auch über die bequeme Programmerstellung hinaus noch seine Vorteile. So eignet sich das Arbeitsbereichsfenster bestens als zentrale Schaltstelle, von der aus Sie schnell zu bestimmten Stellen Ihres Programms springen können (siehe unten). Der Browser informiert Sie über die in Ihrem Programm verwendeten globalen und lokalen Variablen, Klassenhierarchien etc. Mit dem Befehl BEARBEITEN/SUCHEN IN DATEIEN können Sie bequem in allen oder ausgewählten Dateien Ihres Projekts nach bestimmten Textstellen suchen.

Tabelle 2.3:
Die wichtigsten Elemente der Projektverwaltung (Fortsetzung)





Visual C++ speichert die Informationen über den Aufbau und die Konfiguration des Projekts in einer Datei mit der Extension `.dsp`. Wenn Sie möchten, können Sie sich die Datei in einem ASCII-Texteditor (beispielsweise Notepad) anschauen, Sie sollten aber nicht versuchen, die Datei selbst zu editieren und Visual C++ ins Handwerk zu pfuschen.

Welche Vorteile bringt uns die Projektverwaltung?

Die zwei wesentlichen Vorteile der Projektverwaltung sind

- ✗ die problemlose Kompilierung der Programme (im Gegensatz zu NMAKE)
- ✗ die übersichtliche Quellcodeverwaltung im Arbeitsbereichsfenster

2.2.2 Wozu braucht man Arbeitsbereiche?

In Visual C++ werden Projekte immer in Arbeitsbereichen verwaltet. Ein Arbeitsbereich ist nichts anderes als eine höhere Organisationsebene, die es erlaubt, mehrere Projekte gemeinsam zu verwalten. Auf diese Weise können Sie beispielsweise verschiedene Versionen eines Programms in einem Arbeitsbereich verwalten, oder Sie legen ein Programm und eine zugehörige DLL in einem gemeinsamen Arbeitsbereich ab.

Arbeitsbereiche mit einem Projekt

Wenn es Ihnen aber lediglich darum geht, ein Projekt für ein einzelnes Programm zu erstellen, ist die Verwaltung in Arbeitsbereichen im Grunde nur unnötiger Ballast für Sie. Die IDE tut daher ihr Möglichstes, um Sie von diesem Ballast zu befreien.

- ✗ Der erste Punkt ist, daß es zur Einrichtung eines neues Projekts nicht nötig ist, zuerst einen Arbeitsbereich zu erstellen und im nächsten Schritt ein Projekt in den Arbeitsbereich aufzunehmen. Statt dessen legt man einfach auf der Seite PROJEKTE im Dialogfeld NEU (Aufruf über DATEI/NEU) das neue Projekt an und kontrolliert nur kurz, ob die Option NEUEN ARBEITSBEREICH ERSTELLEN gesetzt ist. Der Arbeitsbereich für das Projekt wird von der IDE dann automatisch erstellt.
- ✗ Der zweite Punkt ist, daß es für die weitere Arbeit mit dem Projekt ganz unerheblich ist, daß es in einem Arbeitsbereich liegt.

Arbeitsbereiche mit mehreren Projekten

Anders liegt der Fall, wie gesagt, wenn man mehrere Projekte in einem Arbeitsbereich verwalten möchte (im Kapitel zu den Dynamischen Linkbibliotheken – den DLLs – werden wir hierzu noch ein Beispiel sehen). Dazu müssen Sie wissen, wie Sie Projekte in einen bestehenden Arbeitsbereich aufnehmen und wie Sie in einem Arbeitsbereich ein Projekt zur Bearbeitung auswählen:

- ✗ Um einem bestehenden Arbeitsbereich ein Projekt hinzuzufügen, öffnen Sie den Arbeitsbereich (DATEI/ARBEITSBEREICH ÖFFNEN) und gehen dann im Dialogfeld NEU (Aufruf über DATEI/NEU) zur Seite PROJEKTE, um das neue Projekt anzulegen. Achten Sie diesmal darauf, daß die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH gesetzt ist.
- ✗ Von den verschiedenen Projekten eines Arbeitsbereichs ist immer nur eines aktiv (durch Fettschrift im Arbeitsbereichsfenster hervorgehoben). Auf dieses Projekt beziehen sich die Menübefehle (beispielsweise zur Kompilation und Erstellung). Um festzulegen, welches Projekt Sie bearbeiten wollen, verwenden Sie den Menübefehl PROJEKT/AKTIVES PROJEKT FESTLEGEN.

Arbeitsbereiche konfigurieren

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

2.2.3 Wie legt man neue Projekte an?

Möchte man mit der Implementierung eines neuen Programms beginnen, richtet man zuerst ein passendes Projekt ein. Neue Projekte legt man, wie Sie bereits wissen, über den Befehl DATEI/NEU an.

Übung 2-1: Neue Projekte anlegen

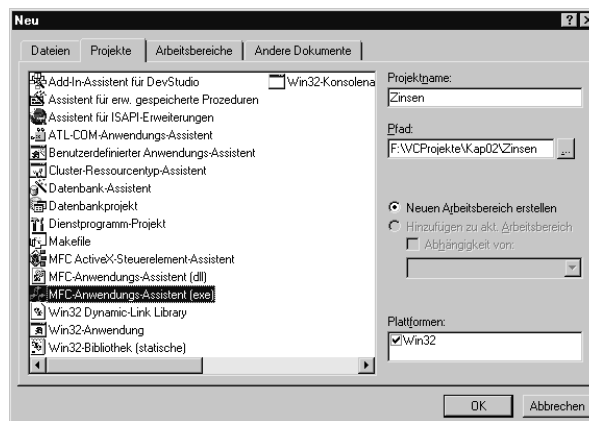
Beginnen wir damit, ein Projekt für unser Zinsprogramm anzulegen.

1. Rufen Sie den Befehl DATEI/NEU auf, und lassen Sie im Dialogfeld NEU die Seite PROJEKTE anzeigen.
2. Geben Sie im Feld PROJEKTNAME einen Namen (beispielsweise *Zinsen*) und im Feld PFAD ein Verzeichnis für das Projekt an.

Im Feld PFAD steht das Projektverzeichnis. In diesem Verzeichnis werden die Dateien des Projekts abgespeichert.

Standardmäßig verwendet Visual C++ den Namen des Projekts auch als Namen für das Projektverzeichnis, d.h., wenn Sie über den Schalter  neben dem Feld PFAD das Dialogfeld VERZEICHNIS WÄHLEN aufrufen und mit dessen Hilfe ein übergeordnetes Verzeichnis auswählen, hängt Visual C++ den Projektnamen automatisch an das ausgewählte Verzeichnis an.

Bild 2.1:
Anlegen eines
neuen Projekts



Theoretisch können Sie den Verzeichnispfad im Feld PFAD manuell bearbeiten, indem Sie die Eingabemarke in das Feld setzen und es direkt editieren. Auf diese Weise können Sie für das Projektverzeichnis einen anderen Namen vorsehen als für das Projekt selbst, doch ist dies nicht empfehlenswert, da es eher Verwirrung stiftet.

3. Wählen Sie einen Arbeitsbereich für das Projekt aus. Per Voreinstellung ist die Option NEUEN ARBEITSBEREICH ERSTELLEN ausgewählt. Achten Sie darauf, daß diese Option gesetzt ist, damit Visual C++ für das neue Projekt einen eigenen Arbeitsbereich anlegt.

Die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH ist nur verfügbar, wenn bei Aufruf des Befehls DATEI/NEU bereits ein Arbeitsbereich in der IDE geöffnet war. Sollte dies der Fall sein, können Sie durch Aktivieren der Option Ihr Projekt dem aktuell geöffneten Arbeitsbereich hinzufügen. Auf diese Weise können Sie beispielsweise verschiedene Versionen eines Programms in einem Arbeitsbereich verwalten, oder Sie speichern ein Programm und eine DLL, die von dem Programm benötigt wird, in einem gemeinsamen Arbeitsbereich ab. Von letzterer Möglichkeit werden wir im Kapitel 20 zu den Dynamischen Linkbibliotheken Gebrauch ma-

chen. Dort können Sie dann auch praktische Erfahrungen zum Umgang mit Arbeitsbereichen sammeln.

4. Wählen Sie im linken Teil des Dialogfeldes einen Projekttyp aus der Liste aus. Für unser Beispiel verwenden wir den MFC-ANWENDUNGS-ASSISTENTEN (EXE). Ich möchte Ihnen aber zuvor noch die anderen Projekttypen kurz vorstellen.

Projekttyp	Beschreibung
Add-in-Assistent für DevStudio	Assistent zur Erstellung von Add-In-DLLs, die der Erweiterung und Automatisierung der Visual-Studio-IDE dienen.
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)).
ATL-COM-Anwendungs-Assistent	Mit diesem Assistenten können Sie ActiveX-Steuer-elemente auf der Grundlage der ATL (Active Template Library) erstellen.
Benutzerdefinierter Anwendungs-Assistent	Dieser Assistent ermöglicht Ihnen die Erstellung eigener Anwendungs-Assistenten.
Cluster-Ressourcentyp-Assistent	Erzeugt Projekte für Microsoft Cluster Server (MSCS)-Ressourcen. Die Cluster-Architektur verbindet mehrere verbundene Systeme zu einem nach außen einheitlich erscheinenden Netzwerk.
Datenbank-Assistent	Zur Erstellung von und Anbindung an SQL-Server-Datenbanken.
Datenbankprojekt	Zur Erstellung datenbankgestützter Anwendungen.
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.
MFC-ActiveX-Steuerelement-Assistent	Mit diesem Assistenten können Sie ActiveX-Steuer-elemente auf der Grundlage der MFC erstellen.
MFC-Anwendungs-Assistent (dll)	Assistent zur Erstellung von MFC-Programmgerüsten für DLLs (Dynamische Linkbibliotheken).
MFC-Anwendungs-Assistent (exe)	Assistent zur Erstellung von MFC-Programmgerüsten für Windows-Anwendungen.

*Tabelle 2.4:
Die verschiedenen Projekt-
typen*

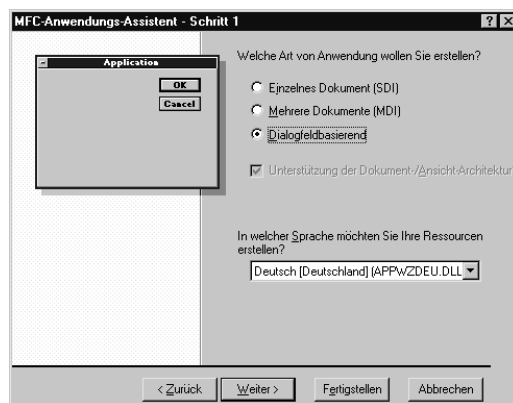
KAPITEL 2 Von der Idee zum Programm

Tabelle 2.4:
Die verschiedenen Projekt-
typen
(Fortsetzung)

Projekttyp	Beschreibung
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.

5. Bestätigen Sie mit OK.

Bild 2.2:
Ein Dialogfeld
als Haupt-
fenster



6. Es erscheint das erste Dialogfeld des Anwendungs-Assistenten, in dem Sie die Option DIALOGFELDBASIEREND auswählen. »Dialogfeldbasierend« bedeutet nichts anderes, als daß das Hauptfenster der Anwendung ein Dialogfeld sein wird. Für unser Zinsprogramm ist dies genau die richtige Wahl.

Im Grunde könnten Sie jetzt schon auf FERTIGSTELLEN drücken, aber wir wollen noch einige untergeordnete Änderungen vornehmen. Drücken Sie daher den Schalter WEITER.

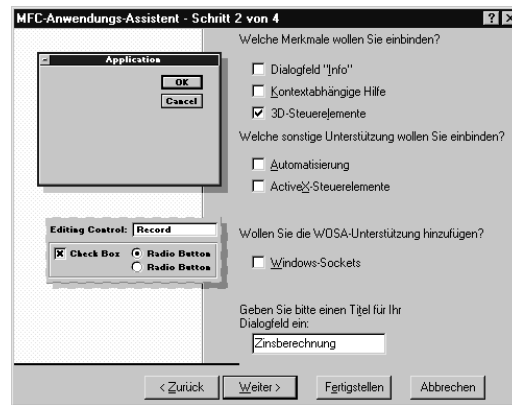


Bild 2.3:
Konfiguration
des Projekts
im Anwen-
dungs-Assi-
stenten

7. Es erscheint das zweite Dialogfeld des Anwendungs-Assistenten, in dem Sie die Optionen DIALOGFELD "INFO" und ACTIVEX-STEUERELEMENTE löschen, da wir an diesen Elementen in unserem Projekt nicht interessiert sind.

Wenn Sie möchten, können Sie dafür im letzten Eingabefeld des Dialogs einen netten Titel für den Dialog der Anwendung eingeben.

Drücken Sie jetzt auf FERTIGSTELLEN.

8. Es erscheint noch ein abschließendes Kontrollfenster, das Sie mit OK bestätigen.

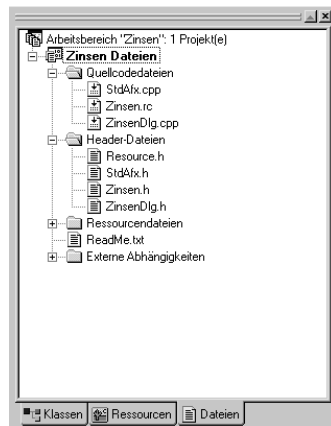
Mit den einzelnen Optionen und Seiten des Assistenten werden wir uns im dritten Kapitel näher beschäftigen.



2.2.4 Das Arbeitsbereichsfenster

Das Arbeitsbereichsfenster ist die Schaltzentrale der Projektverwaltung. Hier verschaffen Sie sich einen Überblick über den Aufbau Ihrer Programme (Dateien, Klassen, globale Variablen, Ressourcen), und von hier aus laden Sie Quelltextdateien und Ressourcen zur Bearbeitung in den Editor. Wenn Sie sich erst einmal ein wenig an die Arbeit mit dem Arbeitsbereichsfenster gewöhnt haben, werden Sie es gar nicht mehr missen wollen. Sollten Sie es aber einmal aus Versehen geschlossen haben, können Sie es jederzeit über den Befehl ANSICHT/ARBEITSBEREICH wieder anzeigen lassen.

*Bild 2.4:
Das Arbeitsbereichsfenster,
Seite Dateien,
für das Zinsen-
Projekt*



Das Arbeitsbereichsfenster enthält mehrere Seiten, die über die »Reiter« (ähnlich den Registerkarten) am unteren Rand des Fensters ausgewählt werden können. Für MFC-Anwendungen finden Sie hier die Seiten KLASSEN, RESSOURCEN und DATEIEN. (Welche Seiten im einzelnen verfügbar sind, hängt vom Projekttyp ab.)

*Tabelle 2.5:
Die Standard-
seiten des
Arbeits-
bereichs-
fensters*

Seite	Beschreibung
Klassen	Diese Seite verschafft Ihnen einen Überblick über die in Ihrem Programm deklarierten Klassen (einschließlich der Klassenelemente) und globalen Symbole. Mit den Befehlen aus den Kontextmenüs der Klassen-Namen können Sie sich weitere Informationen anzeigen lassen, in Deklarationen und Definitionen springen, Haltepunkte setzen etc. Wenn Sie auf den Namen einer Klasse doppelklicken, springen Sie in den Quelltext der Klassendeklaration. Wenn Sie auf den Namen einer Methode doppelklicken, springen Sie in den Quelltext der Methodendefinition.
Ressourcen	Diese Seite zeigt die in der Ressourcenskriptdatei des Projekts abgelegten Ressourcen – nach Ressourcentypen geordnet – an. Mit den Befehlen aus den Kontextmenüs der Ressourcen-Knoten können Sie neue Ressourcen anlegen, die Ressourcen-IDs bearbeiten, oder die Ressourcen-Includedatei festlegen. Wenn Sie auf den Namen einer Ressource doppelklicken, wird die Ressource zur Bearbeitung in den zugehörigen Ressourceneditor geladen.

Seite	Beschreibung
Dateien	Diese Seite zeigt die Arbeitsbereich-Hierarchie mit den zugehörigen Projekten und Quelldateien an. Mit den Befehlen aus den Kontextmenüs der Dateien-Knoten (Klick mit der rechten Maustaste auf den jeweiligen Dateinamen) können Sie die Quelldateien verwalten, Unterverzeichnisse für die Anzeige einrichten, die Projekte erstellen, Knoten konfigurieren. Wenn Sie auf den Namen einer Datei doppelklicken, wird die Datei in den Texteditor geladen.

Tabelle 2.5:
Die Standard-
seiten des
Arbeits-
bereichs-
fensters
(Fortsetzung)

2.2.5 Projekte öffnen und schließen

Wie man ein neues Projekt anlegt, wissen Sie nun. Wenn Sie aber Visual C++ beenden und später neu starten, um an Ihrem Projekt weiterzuarbeiten, stehen Sie vor einem Problem: Es scheint gar keine Befehle für das Öffnen von Projekten zu geben.

Erschrecken Sie nicht, sondern erinnern Sie sich daran, daß Projekte ihrerseits in Arbeitsbereichen verwaltet werden. Was Sie tun müssen, ist, den *Arbeitsbereich* zu öffnen, in dem Ihr Projekt abgelegt ist.

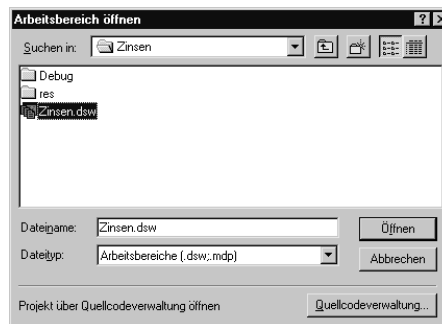


Bild 2.5:
Arbeitsbereich
und Projekt
öffnen

1. Rufen Sie hierzu den Befehl DATEI/ARBEITSBEREICH ÖFFNEN auf, und wählen Sie in dem gleichnamigen Dialogfenster die entsprechende .dsw-Datei. Die .dsw-Datei finden Sie im entsprechenden Projektverzeichnis.

Analog schließt man Projekt und Arbeitsbereich über den Befehl DATEI/ARBEITSBEREICH SCHLIESSEN.

Erlauben Sie mir noch ein paar Hinweise und Tips zum Öffnen von Arbeitsbereichen:

- ✗ Arbeitsbereiche, an denen Sie aktuell arbeiten, öffnen Sie am schnellsten über den Befehl DATEI/ZULETZT GEÖFFNETE ARBEITSBEREICHE. Per

Voreinstellung werden hier die vier zuletzt bearbeiteten Arbeitsbereiche aufgelistet. Sie können aber auch mehr als vier Arbeitsbereiche auflisten lassen. Rufen Sie dazu den Befehl EXTRAS/OPTIONEN auf, und gehen Sie zur Registerseite ARBEITSBEREICH. Geben Sie hier für die LISTE DER ZULETZT VERWENDETEN ARBEITSBEREICHE einen höheren Wert ein.

- ✗ Der Befehl ÖFFNEN eignet sich weniger zum Öffnen von Projekten oder Arbeitsbereichen (obwohl dies durchaus möglich ist), sondern zum Öffnen von Quellcodedateien. Wenn Sie Quellcodedateien über diesen Befehl öffnen, werden diese in den Editor geladen, ohne jedoch dem aktuellen Projekt hinzugefügt zu werden.
- ✗ Wenn Sie möchten, können Sie die IDE so einstellen, daß der zuletzt bearbeitete Arbeitsbereich automatisch beim Starten von Visual C++ geöffnet wird. Rufen Sie dazu den Befehl EXTRAS/OPTIONEN auf, und gehen Sie zur Registerseite ARBEITSBEREICH. Aktivieren Sie hier die Option LETZTEN ARBEITSBEREICH BEIM START AUTOM. LADEN.

2.2.6 Wie erweitert man ein Projekt um neue Quelltextdateien?

Solange man mit den Assistenten arbeitet, werden alle Quelltextdateien, die man benötigt, automatisch angelegt und in das Projekt aufgenommen. Es gibt aber auch Situationen, in denen man Quelltextdateien selbst in das Projekt aufnehmen möchte:

- ✗ sei es, daß man ein Projekt ohne Assistentenunterstützung bearbeitet (beispielsweise eine Windows-API-Anwendung),
- ✗ sei es, daß man auf eine Unterstützung durch Assistenten verzichten muß (beispielsweise beim Anlegen einer Datei für eine Funktionensammlung),
- ✗ sei es, daß man eine bereits existierende Datei in das Projekt aufnehmen möchte (beispielsweise eine Datei mit hilfreichen mathematischen Funktionen).

Wie man dabei vorgeht, hängt davon ab, ob man eine bereits bestehende Quelltextdatei in das Projekt aufnehmen will oder ob man eine ganz neue Datei anlegen und im Projekt abspeichern möchte.

Bestehende Datei aufnehmen

Um eine bereits existierende Datei in Ihr Projekt aufzunehmen, gehen Sie am besten so vor, daß Sie

1. im Arbeitsbereichfenster die Seite DATEIEN öffnen,
2. mit der rechten Maustaste auf den Ordner (erste Ebene) des Projekts klicken, unter dem die Datei eingeordnet werden soll, dann
3. im erscheinenden Kontextmenü den Befehl DATEIEN ZU ORDNER HINZUFÜGEN aufrufen und schließlich
4. im erscheinenden Dialogfeld die gewünschte Datei auswählen.

Die Alternative wäre der Befehl aus dem Menü PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN. Die Datei wird dann anhand ihrer Extension in die Unterordner des Projekts eingeordnet.



Bild 2.6:
Bestehende
Datei in ein
Projekt auf-
nehmen

Neue Datei aufnehmen

Das Anlegen einer ganz neuen Datei erfolgt immer über die Seite DATEIEN des Dialogfelds NEU, das Sie über einen der Befehle DATEI/NEU oder PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU aufrufen können. Die neue Datei wird anhand ihres Typs (Quelldatei, Header-Datei etc.) in die jeweiligen Unterordner des Projekts eingeordnet.

Die Unterordner des Projektordners dienen im übrigen nur der übersichtlicheren Dateiverwaltung. Sie können eigene Ordner anlegen (über den Befehl NEUER ORDNER des Kontextmenüs) sowie Ordner und Dateien in der Knotenhierarchie mit der Maus verschieben.

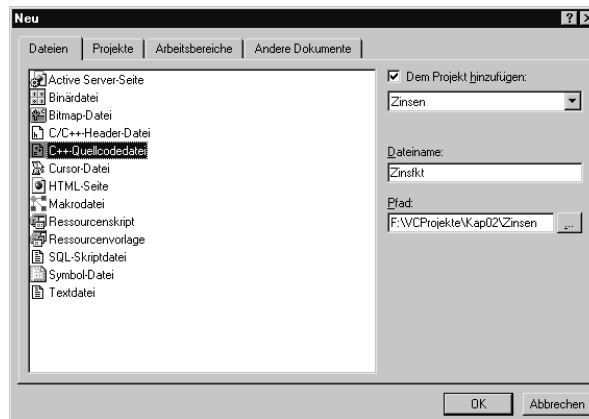


Übung 2-2: Neue Quelltextdatei in ein Projekt aufnehmen

Getreu den Prinzipien der modularen Programmierung wollen wir die Funktion zur Berechnung der Kapitalentwicklung bei Verzinsung ohne Zinseszins

in einer eigenen Datei implementieren. Diese kann später bei Bedarf ohne Probleme in andere Projekte aufgenommen werden.

Bild 2.7:
Neue Datei in
ein Projekt
aufnehmen



1. Rufen Sie den Befehl DATEI/NEU auf, und lassen Sie die Seite DATEIEN anzeigen.
2. Geben Sie einen Dateinamen ein (beispielsweise *Zinsfkt*)
3. Kontrollieren Sie, ob die Option DEM PROJEKT HINZUFÜGEN aktiviert und das Projekt *Zinsen* in dem darunterliegenden Listenfeld angezeigt wird. (Sollte dies nicht der Fall sein, war vermutlich bei Aufruf des Befehls DATEI/NEU das Projekt *Zinsen* nicht geöffnet.)
4. Wählen Sie links den Typ der neuen Datei aus – für unser Beispiel also C++-QUELLCODEDATEI.
5. Wiederholen Sie die ganze Prozedur noch einmal, um eine gleichnamige C/C++-Header-Datei anzulegen.

Jetzt ist die leere Datei *Zinsfkt.cpp* Teil unseres Projekts und wird zusammen mit diesem kompiliert. Sie ist aber noch nicht Teil unseres Programms. Dazu müssen wir erst einmal Code in die Datei schreiben und dafür sorgen, daß dieser Code aus den anderen Dateien des Projekts aufgerufen werden kann – und auch wird. Wir müssen die neuen Dateien also editieren.

2.3 Der Editor

»Den« Editor gibt es in Visual C++ eigentlich gar nicht. Vielmehr gibt es verschiedene Editoren, mit denen die verschiedenen Arten von Quelldateien und Programmelementen bearbeitet werden können. Die wichtigsten Editoren sind für uns der Quelltexteditor und die Ressourceneditoren (mit letzteren beschäftigen wir uns im nächsten Abschnitt).

Quelltextdatei laden

Um eine Quelltextdatei in ein Editorfenster zu laden, bedient man sich üblicherweise des Arbeitsbereichsfensters und doppelklickt in der DATEIEN-Ansicht auf den Namen der zu öffnenden Datei.

Zum Quellcode einer Klasse springen

Möchte man gezielt zur Deklaration einer Klasse oder der Definition einer Elementfunktion einer Klasse springen, kann man diese in der KLASSEN-Ansicht des Arbeitsbereichsfensters doppelklicken.

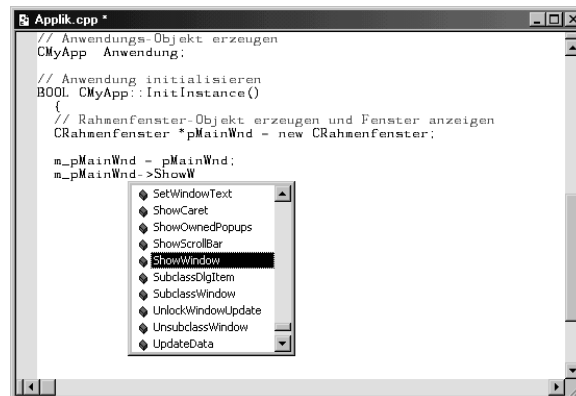


Bild 2.8:
Anweisungs-
vervollständi-
gung imEditor-
fenster

Der Quelltexteditor verfügt über eine übersichtliche Syntaxhervorhebung sowie neuerdings über eine Anweisungsvervollständigung, d.h., der Editor kann Ihnen während des Eintippens Ihres Quellcodes Vorschläge für Klassen/ Strukturelemente oder Aufrufparameter machen (siehe Abbildung 2.8).

- ✗ Wenn Sie nach dem Namen einer Klasseninstanz einen Zugriffsoperator (., ->) eintippen, springt ein Listefeld auf, in dem die verschiedenen Elemente der Klasse aufgeführt werden. Wenn Sie weitertippen, wird das Listefeld zu dem Eintrag gescrollt, der Ihrer bisherigen Buchstabenfolge am meisten entspricht. Durch Drücken der Eingabetaste können

Sie das aktuell ausgewählte Listenelement in den Quelltext einfügen lassen, wobei etwaige Tippfehler in Ihrer Buchstabenfolge sogar 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.

Übung 2-3: Quelltextdatei bearbeiten

Zur Übung laden wir jetzt die Quelltextdatei *Zinsfkt.cpp* (und später auch noch die zugehörige Header-Datei *Zinsfkt.h*) und implementieren darin die Funktion zur Berechnung der Zinserträge.

1. Laden Sie die Datei *Zinsfkt.cpp* zur Bearbeitung in den Editor. Gehen Sie dazu im Arbeitsbereichsfenster zur Seite DATEIEN, und doppelklicken Sie auf den Namen der Datei.
2. Setzen Sie den Code für die Zinsberechnungsfunktionen ein, und binden Sie der Ordnung halber auch die zugehörige Header-Datei ein:

```
#include "stdafx.h"
#include "Zinsfkt.h"
double ErtragOhneZinseszins( double startKapital,
                           double prozZins,
                           double laufzeit )
{
    return startKapital * ( 1 + prozZins/100.0 * laufzeit);
}
```

3. Laden Sie die Header-Datei *Zinsfkt.h* in den Editor. Deklarieren Sie hier die Zinsberechnungsfunktion. Andere Dateien, in denen die Zinsberechnungsfunktion aufgerufen werden soll, brauchen dann nur die Header-Datei einzubinden.

```
#include <math.h>
double ErtragOhneZinseszins( double startKapital,
                           double prozZins,
                           double laufzeit );
```

Jetzt ist es an der Zeit, daß wir uns um die Benutzeroberfläche unseres Programms kümmern. In unserem Fall besteht die Benutzeroberfläche aus einem einzigen Dialogfeld, das beim Einrichten des Projekts vom Assistenten bereits für uns angelegt wurde. Was der Assistent nicht für uns erledigen konnte, war die Ausstattung des Dialogfelds mit Steuerelementen (Eingabefeldern, Beschriftungen etc.). Dies müssen wir nun nachholen.

2.4 Ressourcen

Unter den Ressourcen eines Programms verstehen wir bestimmte Elemente der Benutzeroberfläche. Im einzelnen sind dies

- ✕ Dialoge
- ✕ Menüs
- ✕ Tastaturkürzel
- ✕ Symbolleisten
- ✕ Bitmaps
- ✕ Anwendungssymbole
- ✕ Zeigersymbole
- ✕ Stringtabellen
- ✕ HTML-Code
- ✕ Versionsinformationen
- ✕ Benutzerdefinierte Ressourcen

Was all diesen Ressourcen gemeinsam ist, und was sie als »Ressourcen« kennzeichnet, ist, daß wir sie nicht durch gewöhnliche C++-Anweisungen aufbauen (obwohl dies möglich wäre). Statt dessen bedient man sich zur Erstellung spezieller Editoren, speichert die Ressourcen in speziellen Dateien (den *rc*-Ressourcenskriptdateien) und verwendet im C++-Quelltext spezielle Methodenaufrufe, um die fertigen Ressourcen zu laden.

2.4.1 Vorteile des Ressourcenkonzepts

Dem Programmierer bietet dieses Verfahren etliche handfeste Vorteile:

1. Das Erstellen und Bearbeiten der Ressourcen wird durch die Bereitstellung spezieller Editoren stark vereinfacht.

So steht Ihnen für Bitmaps und Symbole (für Mauszeiger, Schaltflächensymbole etc.) ein eigener Grafikeditor zur Verfügung, und Sie brauchen die Bitmaps nicht als zweidimensionales Feld (Array) von Farbwerten niederzuschreiben.

Haben Sie eine Ahnung, wie aufwendig es ist, ein Bitmap in einem Programm zu erzeugen? Sie müßten erst einmal ein Array mit den Farbwerten für die einzelnen Pixel definieren. Nehmen wir an, wir beschränken uns auf 256 Farben, dann bräuchten wir für jedes Pixel ein Byte:

```
BYTE bits[16] = { 255, 255, 255, 255,
                  255,  0,  0, 255,
                  255,  0,  0, 255,
                  255, 255, 255, 255
                };
```

Das obige Bitmap erzeugt ein schwarzes Rechteck in weißem Rahmen – was man noch beim Blick auf die Zahlen nachempfinden kann. Aber haben Sie Lust, auf diese Weise auch nur einen Smiley von 40 mal 40 Pixeln zu definieren? Als nächstes müßten Sie mit Hilfe der Klasse `CBitmap` ein leeres Bitmap-Objekt erzeugen und in dieses dann die Farbwerte für die einzelnen Pixel kopieren. (Wie man dies im einzelnen macht, wird Thema einer Aufgabe von Kapitel 13 sein.) Allerdings wird das auf diese Weise erzeugte Bitmap nur auf Systemen mit 256 Farben korrekt angezeigt. Auf Systemen mit 65.536 oder mehr Farben stimmt die Zuordnung der Farbwerte aus dem Array zu den Pixeln nicht mehr.

Wieviel einfacher ist es da, das Bitmap im Bitmap-Editor von Visual C++ zu entwerfen.

Beim Speichern wird das Bitmap in eine eigene BMP-Datei geschrieben, und in die Ressourcendatei wird ein Verweis auf diese Datei aufgenommen. Im Programm kann das Bitmap mit Hilfe der Methode `LoadBitmap()` jederzeit ohne Schwierigkeiten geladen werden, wobei das Problem mit den verschiedenen Farbauflösungen so ganz nebenbei durch den Umweg über die Abspeicherung im *bmp*-Format auch gelöst wurde.

2. Ressourcen sind besser wiederverwendbar.

Dadurch, daß Ressourcen in externen Dateien abgelegt werden, kann man sie leicht kopieren und in anderen Programmen verwenden.

Möchte man beispielsweise eine Menüstruktur aus Programm A in Programm B übernehmen, braucht man nur die Ressourcenskriptdatei zu übernehmen (in der das Menü definiert ist) oder das Menü aus der Ressourcenskriptdatei von A in die Ressourcenskriptdatei von B zu kopieren (Ressourcen-Header nicht vergessen). Noch einfacher ist der Austausch für Bitmaps und Symbole, die zusätzlich in eigenen Bilddateien (*.bmp*, *.ico*) abgespeichert werden.

3. Ressourcen sind besser lokalisierbar.

Wenn Sie eines Ihrer Programme nicht nur in Deutschland, sondern auch in den USA vertreiben wollen, müssen Sie alle Texte in Englisch übersetzen. Neben dem Hilfesystem betrifft dies vor allem die Oberflächenelemente der Benutzerschnittstelle: Menüs, Dialoge, Meldungen. Wenn Sie diese Elemente als Ressourcen implementiert haben (Meldungen und alle Strings, die Sie in Ihrer Anwendung anzeigen und die nicht schon Teil einer anderen Ressource sind, sollten dabei in einer Stringtabelle abgespeichert sein), werden Sie damit keine Schwierigkeiten haben. Sie brauchen lediglich eine Kopie der Ressourcendatei anzulegen, in der Sie die deutschen Texte in Englisch übersetzen. Den Quelltext des Programms brauchen Sie nicht zu ändern, da die Ressourcen vom Programm aus über Ressourcenbezeichner angesprochen werden, die für die deutschen und englischen Ressourcen identisch sind. Zum Schluß lassen Sie Ihr Programm neu erstellen, wobei Sie darauf achten, daß die englische Ressourcenskriptdatei verwendet wird.

Aber das sind schon recht fortgeschrittene Techniken, mit denen wir uns nicht gleich zu Anfang belasten sollten. Wichtig ist zu erkennen, wie wertvoll das Ressourcenkonzept für den Programmierer ist, und zu verstehen, wie es funktioniert.

Ach ja, wie funktioniert es eigentlich?

2.4.2 Das Ressourcenkonzept

Die Grundidee des Ressourcenkonzepts ist die Auslagerung bestimmter Elemente der grafischen Benutzeroberfläche (Dialoge, Menüs, Bitmaps etc.). Auslagerung bedeutet dabei, daß diese Elemente nicht im Quelltext, sondern in separaten Dateien definiert werden. Um nun aber aus dem C++-Quelltext heraus eine in einer externen Ressourcendatei definierte Ressource aufrufen zu können, muß wieder eine Verbindung zu der Ressource hergestellt werden. Diese Verbindung ist die Ressourcen-ID (auch Ressourcenbezeichner genannt).



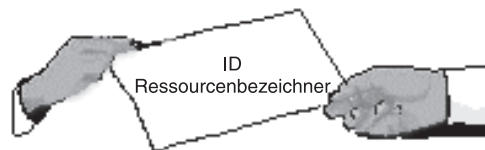
Jede Ressource verfügt über eine Ressourcen-ID, über die das Programm auf sie zugreifen kann.

Bild 2.9:
Ressourcen-
bezeichner als
Bindeglied
zwischen Res-
sourcendatei
und Programm

Ressourcenskriptdatei

Definition der
Ressourcen

Bindeglied



Programm

Methoden zum
Laden der
Ressourcen

Ressourcenskriptdatei

Ressourcen werden in sogenannten Ressourcenskriptdateien definiert (Extension `.rc`). Bei diesen Dateien handelt es sich um einfache Textdateien, in denen die verschiedenen Ressourcen nach den Regeln einer eigenen Syntax definiert werden.

Die Definition eines Menüs mit drei Popup-Menüs Datei, Bearbeiten und Hilfe könnte beispielsweise wie folgt in der Ressourcenskriptdatei definiert sein:

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&Datei"
    BEGIN
        MENUITEM "&Neu\tStrg+N",          ID_FILE_NEW
        MENUITEM "Ö&ffnen...\tStrg+O",    ID_FILE_OPEN
        MENUITEM "S&peichern\tStrg+S",      ID_FILE_SAVE
        MENUITEM SEPARATOR
        MENUITEM "&Beenden",                ID_APP_EXIT
    END
    POPUP "&Bearbeiten"
    BEGIN
        MENUITEM "&Rückgängig\tStrg+Z",    ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "&Ausschneiden\tStrg+X",    ID_EDIT_CUT
        MENUITEM "&Kopieren\tStrg+C",        ID_EDIT_COPY
        MENUITEM "E&infügen\tStrg+V",        ID_EDIT_PASTE
    END
    POPUP "&Hilfe"
    BEGIN
        MENUITEM "Inf&o...",                ID_APP_ABOUT
    END
END
```

```
END
END
```

Bitmaps dagegen werden in eigenen Bitmap-Dateien gespeichert, auf die von der Ressourcenskriptdatei aus verwiesen wird:

```
IDB_BITMAP1      BITMAP  DISCARDABLE  "res\\bitmap1.bmp"
```

RES-Datei

Wie Sie sehen, folgen diese Definitionen bestimmten Syntaxregeln, die der menschlichen Sprache (wenn auch nur im Telegrammstil) viel näher sind als C/C++-Anweisungen. Ressourcenskriptdateien können daher nicht vom C-Compiler übersetzt werden, sie bedürfen eines eigenen Compilers, des Ressourcen-Compilers. Dieser erzeugt aus der Ressourcenskriptdatei eine binäre .res-Datei.

Wenn Sie mit Visual C++ arbeiten, brauchen Sie sich dank der integrierten Projektverwaltung nicht um den Aufruf des Ressourcen-Compilers zu kümmern. Wenn Ihr Projekt Ressourcen verwendet und Sie es über die Befehle im Menü ERSTELLEN kompilieren lassen, ruft die IDE automatisch den Ressourcen-Compiler für die Übersetzung der Ressourcenskriptdatei(en) auf.



Ressourcen-IDs

Zu jeder Ressourcendefinition gehört die Angabe einer Ressourcen-ID. Grundsätzlich sollten diese IDs eindeutig sein, da mit ihrer Hilfe die einzelnen Ressourcen vom Programm aus angesprochen werden. (Eine Ausnahme bilden zusammengehörende Ressourcen – beispielsweise ein Menü und seine Tastaturkürzel oder die Standardressourcen von Programmen: Menü, Anwendungssymbol, String für den Titel des Hauptfensters.)

Diese Ressourcen-IDs müssen nun dem C++-Compiler bekanntgemacht werden. Dazu müssen die einzelnen Ressourcen-IDs mittels #define-Direktiven mit Integer-Konstanten verbunden werden. Am sinnvollsten geschieht dies in einer eigenen Header-Datei (der MFC-Assistent nennt diese Header-Datei standardmäßig *resource.h*), denn diese Header-Datei muß sowohl in die Ressourcenskriptdatei als auch in die Quelltextdateien, die Ressourcen verwenden, per #include-Direktive aufgenommen werden. Für unsere oben definierten Ressourcen könnte die Ressourcen-Headerdatei wie folgt aussehen:

```
#define IDR_MAINFRAME      128
#define IDB_BITMAP1        129
```

Die Ressourcenmethoden

Der letzte Schritt besteht darin, die Ressourcen zu laden, um sie anzuzeigen und mit ihnen zu arbeiten. Dazu stehen sowohl verschiedene API-Funktionen als auch MFC-Methoden zur Verfügung. Die folgende Tabelle zeigt Ihnen eine kleine Auswahl.

Tabelle 2.6:
MFC-Methoden zum
Laden von
Ressourcen *

Ressource	Methoden
Dialog	CDialog (LPCTSTR lpszTemplateName, CWnd* pParentWnd = NULL); CDialog (UINT nIDTemplate, CWnd* pParentWnd = NULL);
Menü	virtual BOOL CFrameWnd::LoadFrame (UINT nIDResource, DWORD dwDefaultStyle = WS_OVERLAPPEDWINDOW FWS_ADDTOTITLE, CWnd* pParentWnd = NULL, CCreateContext* pContext = NULL);
Tastaturkürzel	virtual BOOL CFrameWnd::LoadFrame (UINT nIDResource, DWORD dwDefaultStyle = WS_OVERLAPPEDWINDOW FWS_ADDTOTITLE, CWnd* pParentWnd = NULL, CCreateContext* pContext = NULL);
Anwendungs- symbol	virtual BOOL CFrameWnd::LoadFrame (UINT nIDResource, DWORD dwDefaultStyle = WS_OVERLAPPEDWINDOW FWS_ADDTOTITLE, CWnd* pParentWnd = NULL, CCreateContext* pContext = NULL);
Zeigersymbol	HCURSOR CWinApp::LoadCursor (LPCTSTR lpszResourceName); HCURSOR CWinApp::LoadCursor (UINT nIDResource);
Symbolleiste	BOOL CToolBar::LoadToolBar (LPCTSTR lpszResourceName); BOOL CToolBar::LoadToolBar (UINT nIDResource);
String	virtual BOOL CFrameWnd::LoadFrame (UINT nIDResource, DWORD dwDefaultStyle = WS_OVERLAPPEDWINDOW FWS_ADDTOTITLE, CWnd* pParentWnd = NULL, CCreateContext* pContext = NULL); BOOL CString::LoadString (UINT nID);
Bitmap	BOOL CBitmap::LoadBitmap (LPCTSTR lpszResourceName); BOOL CBitmap::LoadBitmap (UINT nIDResource);

* Wie man diese Methoden aufruft und die Ressourcen im Programm verwendet, erfahren Sie im zweiten Teil des Buches (in den Kapiteln zu den verschiedenen Programmelementen).

Die RES-Datei

Wie bereits oben erwähnt, werden Ihre Ressourcenskriptdateien von einem speziellen Ressourcen-Compiler (der für Visual C++ einfach `rc.exe` heißt), in eine binäre Ressourcendatei mit der Extension `res` übersetzt. In dieser einen Ressourcendatei sind alle Ressourcen enthalten! Bei der Projekterstellung wird diese binäre Ressourcendatei im letzten Schritt des Linkers mit den Objektdateien des C++-Quelltextes zur ausführbaren EXE-Datei zusammengebunden (siehe auch Abbildung 2.20).

Die Ressourcen sind also in der ausführbaren Datei enthalten!

2.4.3 Ressourcen anlegen

Der grundlegende Ablauf zum Anlegen oder Bearbeiten von Ressourcen ist immer gleich:

1. **Eine neue Ressource anlegen.** Um in eine Ressourcenskriptdatei eine neue Ressource aufzunehmen, klicken Sie in der RESSOURCEN-Ansicht des Arbeitsbereichsfensters auf den übergeordneten Ressourcen-namen und rufen im Kontextmenü den Befehl EINFÜGEN auf (oder rufen Sie den Menübefehl EINFÜGEN/RESSOURCE auf). Daraufhin erscheint das Dialogfeld RESSOURCE EINFÜGEN, in dem Sie den Typ der neu anzulegenden Ressource auswählen.

Wenn Sie Ihre Programme mit dem MFC-Anwendungs-Assistenten starten, legt dieser automatisch für Sie eine Ressourcenskriptdatei und eine zugehörige Header-Datei (`resource.h`) an. Wenn Sie mit leeren Projekten starten (beispielsweise für API-Programme), müssen Sie diese Dateien selbst anlegen (Befehl DATEI/NEU) und auch dafür Sorge tragen, daß die Header-Datei mit den Ressourcen-IDs per `#include`-Anweisung in die Quelltextdateien aufgenommen wird.

2. **Ressource bearbeiten.** Um eine bestehende Ressource zur Bearbeitung zu öffnen, müssen Sie nur in der RESSOURCEN-Ansicht des Arbeitsbereichsfensters auf den Namen der zu öffnenden Ressource doppelklicken. (Die Ressourcen sind in der RESSOURCEN-Ansicht nach Kategorien geordnet. Eventuell müssen Sie zuerst die übergeordneten Knoten expandieren.)

Die IDE ruft daraufhin einen zum Typ der Ressource passenden Ressourceneditor auf, in dem Sie diese bequem bearbeiten können.

3. **Ressource im Programm verwenden.** Damit eine Ressource von Ihrem Programm verwendet wird, müssen Sie die Ressource in das Programm laden. Dazu stellt Ihnen die MFC eine Reihe von Methoden zur

Verfügung, siehe Tabelle 2.6. Um der Methode mitzuteilen, welche Ressource geladen werden soll, übergeben Sie der Methode die ID der Ressource. Die meisten der Methoden sind überladen, so daß Sie die ID der zu ladenden Ressource als String (beispielsweise "IDR_MAINFRAME") oder als Wert (beispielsweise 128) übergeben können.

4. **Ressourcen kompilieren.** Die Übersetzung der Ressourcen geschieht üblicherweise automatisch im Zuge der Programmerstellung.

Sie können die Ressourcendatei aber – wie im übrigen jede Quelltextdatei – auch einzeln kompilieren, indem Sie in der DATEIEN-Ansicht des Arbeitsbereichsfensters mit der rechten Maustaste auf den Namen der Ressourcenskriptdatei klicken und den Befehl KOMPILIEREN VON... aufrufen.



Der schnellste Weg, neue Ressourcen anzulegen, führt über die Symbolleiste RESSOURCE, die Sie anzeigen lassen können, indem Sie mit der rechten Maustaste in den Hintergrund einer beliebigen Symbolleiste klicken und in der aufspringenden Liste das entsprechende Markierungskästchen aktivieren.



Um eine angelegte Ressource zu löschen, markieren Sie die Ressource in der RESSOURCEN-Ansicht und drücken die Entf-Taste.

2.4.4 Die verschiedenen Ressourcen-Arten

Dialogfelder

Im Dialog-Editor werden Dialogfelder durch Auswahl, Platzierung und Konfiguration einzelner Steuerelemente erstellt und bearbeitet. Das Fenster, in dem der Dialog bearbeitet wird, ist eine grafische Arbeitsfläche, in der Sie genau sehen, wie der Dialog später angezeigt wird (WYSIWYG = What You See Is What You Get). Nachdem Sie den Dialog-Editor zur Bearbeitung einer Dialog-Ressource geöffnet haben, wird die Menüleiste der IDE um das Popup-Menü LAYOUT erweitert. Hier finden Sie alle wichtigen Befehle zur Einrichtung und Bearbeitung des Dialogs, während Sie über den Befehl EIGENSCHAFTEN aus dem Kontextmenü des Dialog-Editors den Dialog und seine einzelnen Steuerelemente konfigurieren können.

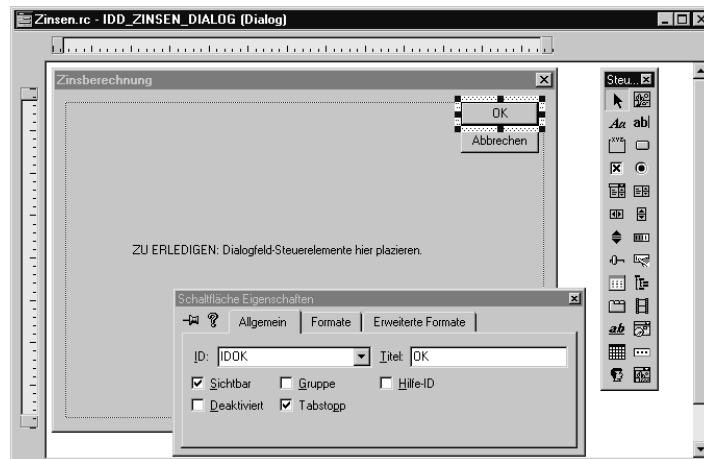


Bild 2.10:
Bearbeitung
von Dialog-
feldern

Grundsätzlich geht man zur Einrichtung eines Dialogs nach folgendem Schema vor:

1. **Legen Sie eine neue Dialog-Ressource an.** Klicken Sie in der RESSOURCEN-Ansicht des Arbeitsbereichfensters auf den übergeordneten Ressourcenknoten, und rufen Sie im Kontextmenü den Befehl EINFÜGEN auf. Daraufhin erscheint das Dialogfeld RESSOURCE EINFÜGEN, in dem Sie als Typ der neu anzulegenden Ressource DIALOG auswählen. Danach klicken Sie auf NEU.
2. **Steuerelemente in Dialog einfügen.** Die benötigten Steuerelemente werden in der Symbolleiste STEUERELEMENTE ausgewählt und per Klick mit der Maus in den Dialog aufgenommen.
3. **Steuerelemente platzieren.** Die einzelnen Steuerelemente werden im Dialog platziert und übersichtlich angeordnet. Die Grobarbeit kann dabei direkt mit der Maus vorgenommen werden (Verschieben oder Aufziehen des Markierungsrahmens), für Feinarbeiten empfiehlt sich der Einsatz der entsprechenden Befehle aus dem Menü LAYOUT.
4. **Steuerelemente konfigurieren.** Dies geschieht über den Befehl EIGENSCHAFTEN aus dem Kontextmenü des Steuerelements. In dem zu diesem Befehl erscheinenden Dialogfeld stehen Ihnen alle Optionen zur Konfiguration des Steuerelements zur Verfügung: Vergabe einer eigenen Ressourcen-ID, Angabe eines Titels, Festlegung des Anfangszustands, Tastenkombination für schnellen Zugriff etc. – die einzelnen Optionen variieren, je nach Art des Steuerelements.
5. **Zum Schluß wird der Dialog getestet** (Befehl LAYOUT/TESTEN).



Zum Löschen eines Steuerelements markieren Sie es im Editor und drücken die **Entf**-Taste.

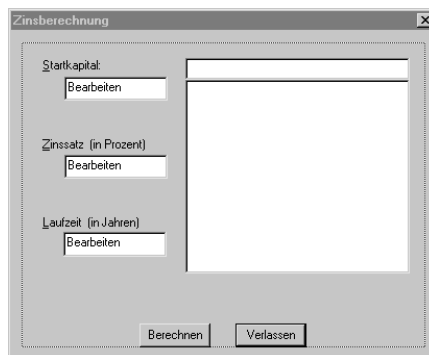


Mehr zum Einsatz von Dialogfeldern in Anwendungen (und besonders zur Festlegung der Tabulatorreihenfolge) sowie der Einrichtung von gruppierten Optionsfeldern erfahren Sie im Kapitel 10 zu den Dialogen.

Übung 2-4: Dialogfeld des Zinsenprogramms bearbeiten

In dieser Übung wollen wir das Erscheinungsbild des Hauptfensters unseres Zinsprogramms festlegen. (Sie erinnern sich: Unser Hauptfenster ist ein Dialogfenster.) Danach sollte Ihr Dialog ungefähr wie in Abbildung 2.11 aussehen.

Bild 2.11:
Das Haupt-
fenster des
Zinspro-
gramms



1. Laden Sie den Dialog zur Bearbeitung in den Editor. Gehen Sie dazu im Arbeitsbereichsfenster zur Seite RESSOURCEN, expandieren Sie den übergeordneten Knoten und den Knoten DIALOG, und doppelklicken Sie auf den Knoten IDD_ZINSEN_DIALOG.
2. Löschen Sie das ZU ERLEDIGEN-Feld.
3. Verschieben Sie die Schalter nach unten.

Halten Sie dazu die **⇧**-Taste gedrückt, und markieren Sie per Mausklick zuerst den Abbrechen-Schalter und dann den OK-Schalter. Rufen Sie den Menübefehl LAYOUT/SCHALTFLÄCHEN ANORDNEN/UNTEN auf. Der Abbrechen-Schalter sollte jetzt vor dem OK-Schalter liegen.

4. Konfigurieren Sie die Schalter.

Klicken Sie mit der rechten Maustaste auf den Schalter Abbrechen und wählen Sie im Kontextmenü den Befehl EIGENSCHAFTEN aus. Ändern Sie die ID des Schalters in ID_BN_RECHNEN und den TITEL in Berechnen.

Klicken Sie mit der rechten Maustaste auf den Schalter OK, und wählen Sie im Kontextmenü den Befehl EIGENSCHAFTEN aus. Ändern Sie den TITEL in »Verlassen«.

5. Nehmen Sie drei Textfelder auf.



Halten Sie dazu die **[Strg]**-Taste gedrückt, wenn Sie in der Steuerelementleiste auf das Symbol des Textfeldes klicken. Die **[Strg]**-Taste aktiviert die Mehrfachablage, das heißt, mit jedem nachfolgenden Klick in Ihr Formular fügen Sie ein neues Textfeld ein. Legen Sie auf diese Weise drei Textfelder in Ihrem Formular ab.

Benennen Sie die Textfelder, die als Beschriftungen für die Eingabefelder dienen sollen (siehe Abbildung 2.11).

6. Nehmen Sie drei Eingabefelder in Ihr Formular auf.



Über die Eingabefelder kann der Anwender bei Ausführung des Programms Startkapital, Zinssatz und Laufzeit eingeben.

7. Nehmen Sie ein Listenfeld in Ihr Formular auf.



In dem Listenfeld werden wir das Ergebnis unserer Berechnungen anzeigen.

Klicken Sie mit der rechten Maustaste auf das Listenfeld, und wählen Sie im Kontextmenü den Befehl EIGENSCHAFTEN aus. Wählen Sie auf der Seite FORMATE im Feld AUSWAHL den Eintrag EINZEL, und deaktivieren Sie die Option SORTIEREN.

8. Arrangieren Sie die Steuerelemente übersichtlich auf Ihrem Formular (siehe Abbildung 2.11).

Mit der Maus können Sie einzelne Steuerelemente im Dialog verschieben oder ihre Größe verändern (durch Ziehen an den Markierungspunkten).

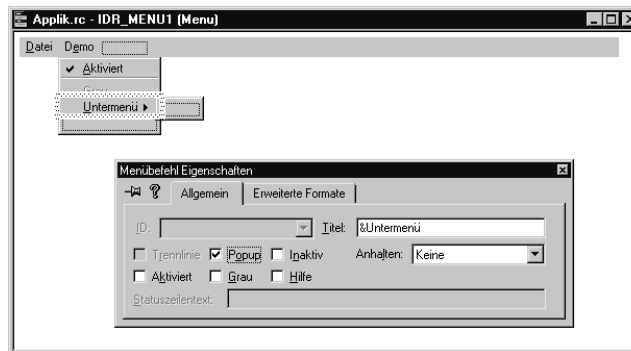
Indem Sie mehrere Steuerelemente gleichzeitig markieren (Aufziehen eines Markierungsrahmens um die Steuerelemente oder Anklicken mit der Maus bei gleichzeitigem Gedrückthalten der **[⇧]**-Taste) und den entsprechenden Befehl im Menü LAYOUT aufrufen (AUSRICHTEN oder

GLEICHMÄSSIG VERTEILEN), können Sie Steuerelemente zueinander ausrichten – beispielsweise in gleichem vertikalem Abstand und linksbündig.

9. Testen Sie den Dialog (Menübefehl LAYOUT/TESTEN).

Menüs

Bild 2.12:
Bearbeitung
von Menüs



Im Menü-Editor werden Menüs und Menüleisten schrittweise aufgebaut – Menü für Menü und Befehl für Befehl. Das Editorfenster, in dem das Menü bearbeitet wird, zeigt das Menü schon ganz so, wie es später in der Anwendung aussehen wird (WYSIWYG). Einzelne Menüelemente legt man an, indem man den Titel des Menüelements direkt in den dafür vorgesehenen Platzhalter eingibt und dann das Menüelement im Dialogfeld EIGENSCHAFTEN anpaßt.

Grundsätzlich geht man zur Einrichtung eines Menüs nach folgendem Schema vor:

1. **Legen Sie eine neue Menü-Ressource an.** Klicken Sie in der RESSOURCEN-Ansicht des Arbeitsbereichfensters auf den übergeordneten Ressourcenknoten, und rufen Sie im Kontextmenü den Befehl EINFÜGEN auf. Daraufhin erscheint das Dialogfeld RESSOURCE EINFÜGEN, in dem Sie als Typ der neu anzulegenden Ressource MENU auswählen. Danach klicken Sie auf NEU.
2. **Titel des Menüelements eingeben.** Der Menü-Editor zeigt Ihnen hierzu stets eine leere Schablone an, die als Platzhalter für ein neues Menüelement dient. Tippen Sie in diese Schablone den Titel des Menüelements ein (unabhängig davon, ob es sich um ein Popup-Menü, einen Menübefehl oder eine Trennlinie handelt).

3. **Menüelement anpassen.** Schon während Sie den Titel für ein Menüelement eingeben, wird das Dialogfeld EIGENSCHAFTEN zur Konfiguration des Menüelements angezeigt. Hier können Sie festlegen, ob es sich um ein Popup-Menü, einen Menübefehl oder eine Trennlinie handeln soll. Sie können einen Buchstaben für den Schnellaufruf mit der **[Alt]**-Taste auswählen (Kaufmännisches Und »&« in Beschriftung voranstellen) und den Anfangszustand des Menüelements festlegen. Für Menübefehle können Sie eine ID eingeben, über die später eine Botschaftsverarbeitung für den Befehl eingerichtet werden kann (wird ansonsten automatisch zugeteilt).

Das Dialogfeld EIGENSCHAFTEN kann auch nachträglich über den gleichnamigen Befehl im Kontextmenü des Menü-Editors aufgerufen werden.

4. **Weitere Elemente einfügen.** Danach bauen Sie das Menü oder die Menüleiste Schritt für Schritt auf, wofür Ihnen der Menü-Editor immer entsprechende Platzhalter am jeweiligen Ende der Menüleiste zur Verfügung stellt.

Zum Löschen eines Menüelements markieren Sie es im Editor und drücken die **[Entf]**-Taste.

Mehr zum Einsatz von Menüs in Anwendungen wie auch zur Deaktivierung von Menübefehlen oder der Implementierung von Kontextmenüs erfahren Sie im Kapitel 8 zu den Menüs.

Tastaturkürzel

Mit dem Zugriffstasten-Editor können Sie Tastaturkürzel für Ihre Anwendung einrichten. Die Zuordnung der IDs dieser Kürzel zu einer Antwortfunktion erfolgt natürlich im Code der Anwendung.

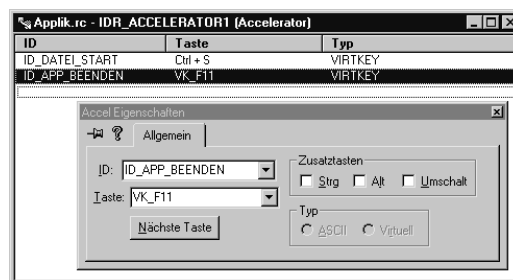


Bild 2.13:
Bearbeitung
von Tastatur-
kürzeln

Um ein Tastaturkürzel einzurichten, gehen Sie wie folgt vor:

1. **Legen Sie eine neue Tastaturkürzel-Ressource an.** Klicken Sie in der RESSOURCEN-Ansicht auf den übergeordneten Ressourcenknoten, und rufen Sie im Kontextmenü den Befehl EINFÜGEN auf. Daraufhin erscheint das Dialogfeld RESSOURCE EINFÜGEN, in dem Sie als Typ der neu anzulegenden Ressource ACCELERATOR auswählen. Danach klicken Sie auf NEU.
2. **Richten Sie ein neues Tastaturkürzel ein.** Doppelklicken Sie in den leeren Eintrag am Ende der Liste.
3. **Geben Sie eine ID für das Tastaturkürzel ein.** Wählen Sie im Feld ID des Dialogfelds EIGENSCHAFTEN eine vordefinierte ID aus, oder geben Sie eine eigene ID ein.

Für Tastaturkürzel, die Menübefehle aufrufen sollen, geben Sie beispielsweise die ID des betreffenden Menübefehls ein. Haben Sie das Menü bereits eingerichtet und in der gleichen Ressourcenskriptdatei gespeichert, wird die ID in der Liste des Feldes ID mit aufgeführt.

4. **Geben Sie die gewünschte Tastenkombination ein.** Drücken Sie im Dialogfeld EIGENSCHAFTEN den Schalter NÄCHSTE TASTE. Danach können Sie die Tastenkombination direkt über die Tastatur eingeben (so wie sie später ausgelöst wird), und der Zugriffstasten-Editor setzt die Optionen im Dialogfeld EIGENSCHAFTEN.

5. **Schließen Sie das Dialogfeld EIGENSCHAFTEN.**



Zum Löschen eines Tastaturkürzels markieren Sie die betreffende Zeile im Editor und drücken die **[Entf]**-Taste.



Mehr zum Einsatz von Tastaturkürzeln erfahren Sie im Kapitel 8 zu den Menüs.

Bitmaps

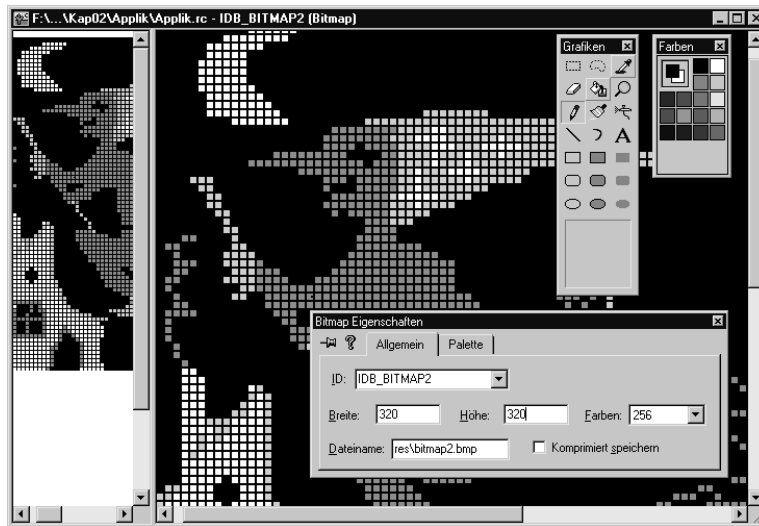


Bild 2.14:
Bitmaps
bearbeiten

Mit dem Grafik-Editor können Sie Bitmaps (hierzu gehören letztendlich auch Symbole und Mauszeiger) erstellen. Das Fenster des Grafik-Editors ist per Voreinstellung zweigeteilt, so daß man in einem Ausschnitt das Bild in Originalgröße betrachten kann, während man im anderen Abschnitt das Bild zur besseren Bearbeitung vergrößern läßt.

Nachdem Sie den Grafik-Editor zur Bearbeitung einer Bitmap-Ressource geöffnet haben, wird zudem die Menüleiste um das Popup-Menü **BILD** erweitert. Hier finden Sie eine Reihe interessanter Befehle zur Bildbearbeitung.

Die eigentliche Arbeit (Zeichnen von Linien oder Kreisen, Einfärben etc.) erfolgt aber über die (für Grafikeditoren typischen) Werkzeugleisten **GRAFIKEN** und **FARBEN**. Die Anzeige dieser Symbolleisten wird über die Liste im Kontextmenü der IDE gesteuert (Aufruf durch Klick in Symbolleistenhintergrund).

Ansonsten ist festzustellen, daß der Umgang mit dem Grafik-Editor keine großen Geheimnisse birgt und für jemanden, der schon einmal mit einem Grafikprogramm gearbeitet hat, keine Schwierigkeit darstellen dürfte. Einige Besonderheiten, über die Sie vielleicht stutzen könnten, möchte ich dennoch kurz erwähnen:

- ✗ **Rastergrafiken.** Bitmaps sind punktweise definierte Grafiken. Werden sie auf dem Bildschirm dargestellt, entspricht jedem Punkt in dem Bit-

map genau ein Pixel in der Bildschirmdarstellung. In der Bitmap-Datei wird für jeden Punkt des Bildes festgelegt, in welcher Farbe er gezeichnet werden soll. Das Gegenstück zu den Rastergrafiken sind Vektorgrafiken, in denen Zeichnungen nicht als Farbinformationen einzelner Punkte, sondern als mathematische Beschreibungen ganzer Objekte (Linien, Rechtecke etc.) gespeichert werden. Wenn Sie in einer Vektorgrafik ein Rechteck zeichnen, können Sie dieses Rechteck jederzeit wieder durch Anklicken markieren und verändern oder löschen. Wenn Sie dagegen in einer Rastergrafik ein Rechteck zeichnen, wird das Rechteck sofort in Farbinformationen für einzelne Punkte umgewandelt. Nach Abschluß der Operation ist es nicht mehr als Objekt vorhanden und kann auch nicht mehr markiert und weiterbearbeitet werden. Wenn Sie es löschen wollen, müssen Sie die Zeichenoperation rückgängig machen oder das Rechteck Pixel für Pixel übermalen.

- ✕ **Größe.** Wenn Sie ein neues Bitmap anlegen, öffnet Visual C++ automatisch den Dialog BITMAP EIGENSCHAFTEN. Hier können Sie die Maße des Bildes in Pixeln angeben. Wenn Sie die Größe des Bildes nachträglich ändern wollen, rufen Sie den Dialog durch Doppelklick in den Grafik-Editor oder über den Menübefehl ANSICHT/EIGENSCHAFTEN auf. (Achtung! Der gleichnamige Befehl aus dem Kontextmenü der RES-SOURCEN-Ansicht öffnet ein anders Dialogfeld.) Ansonsten können Sie die Größe des Bitmaps auch durch Ziehen der Markierungspunkte um das Bitmap verändern, wobei Sie die aktuellen Maße in der Statusleiste der IDE kontrollieren können.
- ✕ **Farbauflösung.** Die Farbauflösung bestimmen Sie ebenfalls über den EIGENSCHAFTEN-Dialog (Aufruf über ANSICHT/EIGENSCHAFTEN). Im Feld FARBEN haben Sie die Auswahl zwischen MONOCHROM, 16 oder 256 Farben. Mehr Farben unterstützt der Grafik-Editor nicht, mehr Farben benötigen wir aber auch gar nicht. Schließlich verwenden wir den Grafik-Editor vor allem für Symbole, Schaltflächen, kleine Grafiken zur Ausschmückung von Dialogen oder einfache Bildschirmhintergründe. Für alle diese Grafiken ist es ratsam, sich auf 16 oder 256 Farben zu beschränken (erstens schon dies die Systemressourcen, zweitens bleiben die Anwendungen besser kompatibel, drittens bedingt eine höhere Farbauflösung nicht zwangsweise eine bessere Bildqualität). Wenn Sie wirklich hochwertige Grafiken und Bilder benötigen, werden Sie diese sowieso nicht mit dem Grafik-Editor bearbeiten, sondern einscannen oder mit einem speziellen Grafikprogramm erstellen. Wenn Sie sich für eine Farbauflösung entschieden haben, können Sie über die Dialogseite PALETTE einzelne Farben zum Zeichnen auswählen (einfach anklicken) oder neu definieren (doppelt anklicken).

- ✗ **Geräteabbilder.** Geräteabbilder sind verschiedene Varianten (andere Maße und Farbauflösung) eines Bildes, die allerdings nur für Mauszeiger und Symbole (Anwendungssymbol, Schaltflächen der Symbolleiste) benötigt werden. Für die Bearbeitung von Bitmaps sind die entsprechenden Befehle im Menü LAYOUT daher deaktiviert.

Mauszeiger (Cursor)

Mauszeiger sind im Grunde Bitmaps, die wie andere Bitmaps im Grafik-Editor bearbeitet werden können, doch gibt es einige Besonderheiten zu beachten.

- ✗ **Abmessung.** Mauszeiger haben eine feste Größe (standardmäßig beträgt diese 32x32 Pixel).
- ✗ **Farben.** Mauszeiger sind üblicherweise monochrom. Zu den Farben Schwarz und Weiß kommen allerdings noch die Sonderfarben »Transparent« und »Invertiert«.
- ✗ **Transparent und Invertiert.** Da Cursor Bitmaps sind, haben sie immer rechteckige Abmessungen. Damit auf dem Bildschirm aber nur die eigentliche Cursor-Form sichtbar wird, gibt es die Farbe *Transparent*, die man üblicherweise für den Hintergrund des Cursors benutzt. Damit ein schwarzer Cursor nicht auf einem schwarzen Bildschirmhintergrund unsichtbar wird, kann man den Cursor in invertierter Farbe erscheinen lassen.
- ✗ **Geräteabbilder.** Wenn man möchte, kann man einen Cursor in unterschiedlichen Farbauflösungen und Ausmaßen definieren (die zusammen in einer Datei abgespeichert werden). Rufen Sie dazu den Menübefehl BILD/NEUES GERÄTEABBILD auf), und drücken Sie den Schalter BENUTZERDEFINIERT.
- ✗ **HotSpot.** Cursor sind Zeiger. Daher muß festgelegt sein, auf welchen Punkt ein Cursor zeigt. Dies ist der sogenannte Hotspot, dessen Position Sie definieren, indem Sie auf das Symbol HOTSPOT SETZEN klicken und dann auf das entsprechende Pixel der Cursor-Bitmap klicken.

Symbole (Icons)

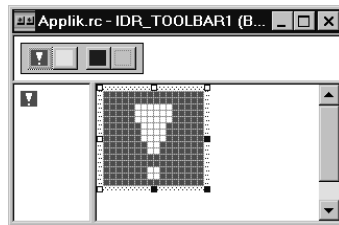
Symbole sind ebenfalls Bitmaps, die im Grafik-Editor bearbeitet werden, doch auch hier gibt es einige Besonderheiten zu beachten.

- ✗ **Abmessung.** Symbole haben eine feste Größe (standardmäßig beträgt diese 32x32 Pixel).
- ✗ **Farben.** Symbole bescheiden sich üblicherweise mit 16 (oder weniger) Farben. Dies sollten unbedingt die Windows-Systemfarben sein, damit das Symbol auf jedem Windows-Rechner gleich gut zu erkennen ist.

- ✗ **Geräteabbilder.** Wenn man möchte, kann man ein Symbol in verschiedenen Maßen und Auflösungen definieren (die zusammen in einer Bitmapdatei abgespeichert werden). Klicken Sie dazu auf den Schalter NEUES GERÄTEABBILD (oder rufen Sie den gleichnamigen Befehl aus dem Menü BILD auf). Im Feld ZIELGERÄT sehen Sie dann die verschiedenen Auflösungen, deren Bearbeitung empfehlenswert ist. Sie brauchen diese Auflösungen aber nicht unbedingt zu bearbeiten. Wenn Sie beispielsweise nur ein großes Symbol (32x32) erstellen, erzeugt Windows ein kleineres Symbol (z.B. für die Anzeige im Explorer) durch automatische Stauchung des großen Symbols. Zudem können Sie Darstellungen in benutzerdefinierten Maßen und Farben anlegen.

Symbolleisten

Bild 2.15:
Symbolleisten
bearbeiten



Mit dem Symbolleisten-Editor können Sie schnell ein Bitmap für eine Symbolleiste erstellen. In dem Bitmap werden die Darstellungen für die Schaltflächen der Symbolleiste gespeichert. Zum Zeichnen der einzelnen Schaltflächen stehen Ihnen die üblichen Hilfsmittel des Grafik-Editors zur Verfügung.

Ansonsten gibt es bei der Erstellung von Symbolleisten-Ressourcen folgende Besonderheiten zu beachten:

- ✗ **Neue Schaltfläche anlegen.** Klicken Sie in die leere Schablone, die im oberen Teilfenster der Symbolleiste angezeigt wird. Sowie Sie dann in einem der unteren Fenster mit der Bearbeitung beginnen, wird im oberen Fenster eine neue leere Schablone eingefügt.
- ✗ **Schaltflächen bearbeiten.** Klicken Sie einfach im oberen Fenster auf die zu bearbeitende Schaltfläche.
- ✗ **Schaltflächen umordnen.** Verschieben Sie die Schaltflächen in der Symbolleiste im oberen Fenster.
- ✗ **Schaltflächen löschen.** Nehmen Sie die Schaltfläche im oberen Fenster mit der Maus auf, und ziehen Sie die Schaltfläche aus der Symbolleiste heraus.

- ✗ **Leerräume einfügen.** Um vor einer Schaltfläche innerhalb einer Symbolleiste einen Leerraum einzufügen, schieben Sie die Schaltfläche einfach um etwa eine halbe Breite über die nachfolgende Schaltfläche.
- ✗ **Abmessung der Schaltflächen.** Symbole haben eine feste Größe (standardmäßig beträgt diese 16x15 Pixel). Sie können die Maße selbst festlegen, indem Sie für die Symbolleiste den Befehl ANSICHT/EIGENSCHAFTEN aufrufen und auf der Seite ALLGEMEIN Breite und Höhe angeben. (Beachten Sie, daß die Darstellungen aller Schalter in einem gemeinsamen Bitmap abgelegt werden. Die Breitenangabe für die Schaltflächen gibt daher auch an, wie das Bitmap intern in einzelne Schaltflächendarstellungen zu unterteilen ist. Daraus folgt, daß alle Schaltflächen in der Symbolleiste die gleichen Ausmaße haben müssen).
- ✗ **Farben.** Zur Darstellung der Schaltflächen sollte man sich mit den 16 Systemfarben begnügen.
- ✗ **Statuszeilentext.** Wenn Sie möchten, daß ein beschreibender Hilfetext in der Statuszeile eingeblendet wird, wenn die betreffende Schaltfläche ausgewählt wird, rufen Sie das Dialogfeld EIGENSCHAFTEN auf (über den Befehl ANSICHT/EIGENSCHAFTEN), und geben Sie den Text in das Feld STATUSZEILENTEXT ein.
- ✗ **Quickinfo.** Um ein Quickinfo zu einem Schalter einzurichten, hängen Sie an den STATUSZEILENTEXT das Zeilenumbruchzeichen (\n) und den Quickinfo-Text ein. Als Quickinfo bezeichnet man einen Hinweistext, der erscheint, sobald der Mauszeiger einen Moment über einem Schalter oder Symbol verweilt.
- ✗ **Schalterzustände.** Windows kennt vier verschiedene Schalterzustände. Wenn Sie für einzelne Schalter eigene Bitmaps anlegen, können Sie in diesen für jeden Schalterzustand eine eigene Darstellung des Schalterbitmaps vorsehen. Mit dem Symbolleisten-Editor geht dies allerdings nicht. Dies ist in den meisten Fällen auch nicht notwendig, da Windows automatisch für die Kennzeichnung der verschiedenen Schalterzustände sorgt (durch Manipulation des Originalbitmap).

Schalterzustand	Beschreibung
Nicht gedrückt	Standardzustand der Schaltfläche
Deaktiviert	Schaltfläche ist inaktiviert, d.h. sie löst keine Aktionen aus
Gedrückt	Schaltfläche wird gerade angeklickt
Dauerhaft gedrückt	Wird verwendet, um die Schaltfläche auch nach dem Anklicken im gedrückten Zustand anzuzeigen

Tabelle 2.7:
Schalter-
zustände

Stringtabellen

Bild 2.16:
Stringtabellen
bearbeiten



ID	Wert	Beschriftung
AFX_IDS_APP_TITLE	57344	MFC_Applik
AFX_IDS_IDLEMESSAGE	57345	Bereit
ID_FILE_NEW	57600	Erstellt ein neues Dokument\
ID_FILE_OPEN	57601	Öffnet ein bestehendes Dokun
ID_FILE_CLOSE	57602	Schließt das aktive Dokument
ID_FILE_SAVE	57603	Speichert das aktive Dokumei
ID_FILE_SAVE_AS	57604	Speichert das aktive Dokumei
ID_FILE_MRU_FILE1	57616	Öffnet dieses Dokument.
ID_FILE_MRU_FILE2	57617	Öffnet dieses Dokument.
ID_FILE_MRU_FILE3	57618	Öffnet dieses Dokument.
ID_FILE_MRU_FILE4	57619	Öffnet dieses Dokument.

Stringtabellen sind recht einfache Ressourcen, in denen nicht mehr geschieht, als daß einzelne Strings mit IDs in Verbindung gebracht werden.

Der Vorteil dieses Verfahrens liegt darin, daß die Texte (beispielsweise für die Statusleiste oder ein Meldungsfenster) im Quellcode über die passenden IDs geladen werden. Dies vereinfacht die Lokalisierung (Übersetzung in mehrere Sprachen), da lediglich für jede Sprache eine eigene Ressourcen-datei erstellt werden muß, die dann mit dem ausführbaren Programm zusammenge-linkt wird. Eine Überarbeitung des Quelltextes der Anwendung ist dazu nicht erforderlich.

1. **Legen Sie eine neue Stringtabelle an.** Klicken Sie in der RESSOURCEN-Ansicht des Arbeitsbereichsfensters auf den übergeordneten Ressourcennamen, und rufen Sie im Kontextmenü den Befehl EINFÜGEN auf. Daraufhin erscheint das Dialogfeld RESSOURCE EINFÜGEN, in dem Sie als Typ der neu anzulegenden Ressource STRING TABLE auswählen. Danach klicken Sie auf NEU.
2. **Neuen String eingeben.** Doppelklicken Sie auf die leere Schablone am Ende der Tabelle. Geben Sie in dem erscheinenden Dialogfeld den Text und eine ID für den String ein.

Versionsinformationen

Versionsinformations-Ressourcen sind eine zusätzliche, optionale Zugabe für Ihre Anwendungen. Wenn Sie einer Anwendung eine Versionsinformations-Ressource mitgeben, können Sie in dieser Informationen wie Versionsnummer, Copyright, Autor etc. abspeichern.

Diese Informationen können dann zum Beispiel zur Laufzeit mit Hilfe der API-Funktionen `GetFileVersionInfo()` und `VerQueryValue()` abgefragt werden.

2.5 Quellcode fertigstellen

Jetzt müssen wir noch dafür sorgen, daß beim Anklicken des Schalters BE-RECHNEN die Werte in den Eingabefeldern (Startkapital, Zinssatz und Laufzeit) in das Programm eingelesen werden und daraus die Kapitalentwicklung berechnet wird. Das Ergebnis der Berechnungen soll im Listenfeld angezeigt werden (mit dem Endkapital im oberen Feld des Listenfelds).

Dies zu implementieren, ist gar nicht so einfach, aber auch nicht unlösbar. Folgen Sie zunächst einfach den Anweisungen. Wenn Sie später den zweiten Teil des Buches durchgearbeitet haben, werden Sie die hier beschriebenen Arbeitsschritte ohne Probleme verstehen.

Übung 2-5: Fertigstellung des Zinsprogramms

Zuerst müssen wir Elementvariablen einrichten, die eine Verbindung zwischen der Zinsen-Dialogklasse unseres Programms und den Steuerelementen im Dialog herstellen.

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und wechseln Sie zur Seite MEMBER-VARIABLEN.
2. Richten Sie eine Member-Variable für das Startkapital-Eingabefeld ein.

Wählen Sie im Feld KLASSENNAME die Klasse des Dialogs CZINSENDLG aus.

Markieren Sie im Feld STEUERELEMENT-IDS die Ressourcen-ID des Eingabefelds. (Wenn Sie nicht wissen, welche ID zu dem Steuerelement gehört, wechseln Sie zum Dialog-Editor, klicken Sie mit der rechten Maustaste auf das Eingabefeld und schauen im EIGENSCHAFTEN-Dialog nach).

Drücken Sie den Schalter VARIABLE HINZUFÜGEN.

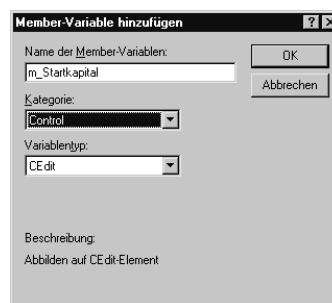


Bild 2.17:
Element-
variable für
Dialogsteuer-
element ein-
richten

Im Dialogfeld MEMBER-VARIABLE HINZUFÜGEN geben Sie einen Namen ein, wählen als Kategorie CONTROL und drücken auf OK.

3. In gleicher Weise richten Sie Elementvariablen für die Eingabefelder zum Zinssatz und zur Laufzeit sowie für das Listenfeld ein. Letztere sollte vom Typ CListBox sein.
4. Wechseln Sie im Klassen-Assistenten zur Seite NACHRICHTENZUORDNUNGSTABELLEN, und richten Sie eine Behandlungsmethode für den Berechnen-Schalter ein.

Wählen Sie im Feld KLASSENNAME den Wert CZINSENDLG aus.

Wählen Sie im Feld OBJEKT-IDS die Ressourcen-ID des Schalters ID_BN_RECHNEN aus.

Wählen Sie im Feld NACHRICHTEN den Eintrag BN_CLICKED aus.

Drücken Sie den Schalter FUNKTION HINZUFÜGEN, und übernehmen Sie den vorgegebenen Wert im Feld NAME DER MEMBER-FUNKTION durch Bestätigen (Klick auf OK).

Drücken Sie den Schalter CODE BEARBEITEN.

5. Setzen Sie den Code für die Methode auf.

```
void CZinsenDlg::OnBnRechnen()
{
    // TODO: Code für die Behandlungsroutine der
    // Steuerelement-Benachrichtigung hier einfügen
    double startKapital, zinsSatz;
    int laufzeit;
    double Ertrag;
    char str[100];

    // alte Werte aus Liste löschen
    m_Ausgabe.ResetContent();

    // Benutzereingaben einlesen
    m_startKapital.GetWindowText(str, 100);
    startKapital = atof(str);
    m_zinsSatz.GetWindowText(str, 100);
    zinsSatz = atof(str);
    m_laufzeit.GetWindowText(str, 100);
    laufzeit = (int) atof(str);
    // Kapitalentwicklung über die Jahre
    for (int i = 0; i <= laufzeit; i++)
```

```

    {
        Ertrag = ErtragOhneZinseszins(startKapital,
                                     zinsSatz, i );
        sprintf(str,"%lf", Ertrag);
        m_Ausgabe.AddString(str);
    }
    m_Ausgabe.SetCurSel(laufzeit);
}

```

6. Da wir die Funktion `ErtragOhneZinseszins()` aufrufen, müssen wir am Anfang der Datei noch die Header-Datei *Zinsfkt.h* einbinden:

```

// ZinsenDlg.cpp : Implementierungsdatei
//
#include "stdafx.h"
#include "Zinsen.h"
#include "ZinsenDlg.h"
#include "Zinsfkt.h"

```

2.6 Compiler und Linker

So langsam naht die Stunde der Wahrheit. Der Quellcode steht, es ist an der Zeit, den Compiler aufzurufen und aus dem Projekt ein ausführbares Programm zu erstellen.

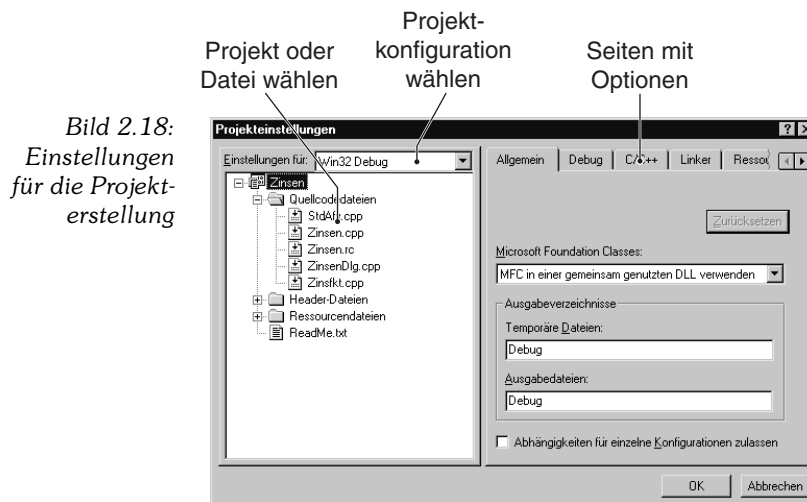
Übung 2-6: Fertigstellung des Zinsprogramms

1. Alles was Sie dazu tun müssen, ist, im Menü ERSTELLEN den Befehl ZINSEN.EXE ERSTELLEN aufzurufen (oder die Taste **F7** zu drücken).

Daß es so einfach ist, verdanken wir der Projektverwaltung von Visual C++. Die Projektverwaltung sorgt aber nicht nur dafür, daß aus einem losen Haufen verschiedenster Quelldateien ein ausführbares Programm wird, sie erlaubt es uns auch, die Kompilation der einzelnen Dateien bis ins Detail zu steuern.

2.6.1 Die Projekteinstellungen

Um auf die Arbeit des Compilers sowie des Linkers Einfluß zu nehmen, ruft man über den Befehl PROJEKT/EINSTELLUNGEN das Dialogfeld PROJEKT-EINSTELLUNGEN auf.



In diesem Dialogfeld können Sie auf mehreren Seiten festlegen, mit welchen Optionen Compiler und Linker Ihr Programm erstellen sollen. Dabei können Sie Optionen für das gesamte Projekt wie auch für einzelne Dateien definieren. Alle Optionen zusammengekommen werden in sogenannten Projektkonfigurationen verwaltet. Letzteres ist äußerst praktisch, erlaubt es uns doch, mehrere Projektkonfigurationen mit unterschiedlichen Compiler- und Linkereinstellungen anzulegen und zwischen diesen durch Aufruf eines einzigen Menübefehls hin- und herzuschalten (statt jedesmal die Projektoptionen selbst bearbeiten zu müssen). So legt Visual C++ beispielsweise von vorneherein stets zwei Konfigurationen an: DEBUG und RELEASE. In der Debug-Konfiguration sind die Compiler-Optionen so eingestellt, daß alle wichtigen Debug-Informationen für den Debugger in die EXE-Datei mit aufgenommen werden. In der Release-Konfiguration, die für die letzte Erstellung des fertigen Programms gedacht ist, wird dagegen auf diese Optionen, die lediglich die Größe der EXE-Datei unnötig aufblähen würden, verzichtet. Statt dessen werden in der Release-Konfiguration Schalter für die Optimierung des Codes gesetzt.

Wie geht man nun also vor, wenn man die Compiler-Einstellungen für ein Projekt anpassen will?

1. **Zuerst wählen Sie im Listenfeld über dem linken Teil des Dialogs die Konfiguration aus**, die Sie bearbeiten wollen.
2. **Dann klicken Sie im linken Teil auf den Projektknoten, um die globalen Einstellungen vorzunehmen.** Rechts sollte nun eine ganze Reihe verschiedener Registerseiten mit einer Vielzahl von Einstellungs-

möglichkeiten angezeigt werden. Beachten Sie, daß einige dieser Seiten (beispielsweise C/C++) über ein Listenfeld KATEGORIE verfügen, über das man auf der gleichen Seite unterschiedliche Optionensätze zu verschiedenen Themen anzeigen lassen kann. Wie Ihre Projekteinstellungen in Compiler-Optionen umgewandelt werden, sehen Sie im Feld PROJEKT OPTIONEN.

3. **Danach können Sie in der linken Projektansicht des Dialogs eine einzelne Datei auswählen**, um für diese Einstellungen vorzunehmen, die von den globalen Einstellungen abweichen.

Welches sind nun die Einstellungen, die man in diesem Dialogfeld vornehmen kann? Dies zu erkunden, überlasse ich Ihnen. Ich erspare es mir damit, ca. 100 Optionen zu erklären, die auch in der Online-Hilfe des Dialogs beschrieben sind. Sie mögen selbst entscheiden, welche Optionen Sie interessieren. Im übrigen ist es nur selten erforderlich, an den voreingestellten Einstellungen Änderungen vorzunehmen. Meist kommt man mit den Konfigurationen DEBUG und RELEASE, so wie sie sind, aus. Und wenn Sie doch einmal auf die Arbeit des Compilers oder des Linkers Einfluß nehmen wollen, werden Sie feststellen, daß die meisten Optionen selbsterklärend oder zumindest mit Unterstützung der Online-Hilfe (Fragezeichen oder **F1**) leicht zu verstehen sind.

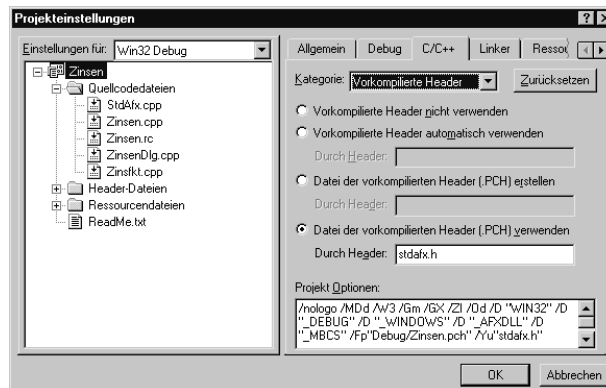
2.6.2 Vorkompilierte Header-Dateien

Windows- und insbesondere MFC-Programme beinhalten meist umfangreiche Header-Dateien, deren Kompilierung meist mehr Zeit in Anspruch nimmt als die Kompilierung des restlichen Programms. Da diese Header-Dateien andererseits kaum geändert werden, bietet es sich an, die erforderliche Kompilierung nur einmal vorzunehmen. So wird die Symboltabelle zu den Header-Dateien nicht durch Parsen derselben neu erstellt, sondern von Festplatte geladen. Wichtig dabei ist, daß die im Quelltext aufgerufenen Header-Dateien und die Header-Dateien im vorkompilierten Header das gleiche Datum haben, in der gleichen Reihenfolge vorliegen und kompatible Compiler-Einstellungen aufweisen. Und als echte Header-Dateien dürfen sie natürlich nur Deklarationen und keine Anweisungen enthalten.

Es gibt mehrere Mechanismen, um vorkompilierte Header zu erstellen und zu verwenden. Eine Möglichkeit ist beispielsweise, vorkompilierte Header automatisch erstellen und verwenden zu lassen – ein Verfahren, daß allerdings nur dann wirklich sinnvoll ist, wenn der Programmierer bei der Erstellung seiner `#include`-Anweisungen diszipliniert auf eine immer gleiche Abfolge achtet. Es gibt aber darüber hinaus eine narrensichere Methode, die

von den Anwendungs-Assistenten verwendet wird (und auf diese bezieht sich auch Abbildung 2.19).

Bild 2.19:
Einstellungen
für vorkompilierte Header



Vorkompilierten Header erstellen

Vielleicht haben Sie sich schon gefragt, wofür die Quelltextdatei *Stdafx.cpp* unseres Zinsen-Projekts ist. Ihre Aufgabe ist es lediglich, als Vorlage für den vorkompilierten Header zu dienen. Sie enthält daher keinen Quellcode, sondern nur eine `#include`-Direktive zur Einbindung der Header-Datei *Stdafx.h*. *Stdafx.h* wiederum enthält eine Reihe von `#include`-Direktiven zur Einbindung der wichtigsten Header-Datei der MFC-Bibliothek.

Ziel dieser Übung ist, daß der Compiler bei der Übersetzung der Quelldatei *Stdafx.cpp* einen vorkompilierten Header anlegt, in dem alle Informationen aus den in *Stdafx.h* aufgeführten Header-Dateien gespeichert sind. Dazu bedarf es spezieller Compiler-Einstellungen. Wenn Sie noch Ihr Zinsen-Projekt geöffnet haben (oder irgendein anderes mit dem MFC-Anwendungs-Assistenten erstelltes Programm), rufen Sie doch einmal den Befehl PROJEKT/EINSTELLUNGEN auf, und klicken Sie im linken Fenster auf den Namen der Datei *Stdafx.cpp*. Wechseln Sie dann zur Seite C/C++, und wählen Sie im Listenfeld KATEGORIE den Eintrag VORKOMPILIERTER HEADER aus. Wie Sie jetzt sehen können, ist für diese Datei die Option DATEI DER VORKOMPILIERTEN HEADER (.PCH) ERSTELLEN aktiviert, und *Stdafx.h* ist als Header-Datei angegeben. Dies soll nichts anderes besagen, als daß der Compiler die aktuelle Datei (*Stdafx.cpp*) von oben nach unten durchgeht, bis er auf die `#include`-Anweisung zur Einbindung der Header-Datei *Stdafx.h* trifft. Aus den Informationen (Deklarationen von Klassen, Funktionen, Konstanten etc.) dieser Header-Dateien (einschließlich *Stdafx.h*) erzeugt der Compiler den vorkompilierten Header.

Vorkompilierten Header verwenden

Der vorkompilierte Header allein nutzt uns nichts, wenn er nicht auch von anderen Dateien verwendet wird. Anders ausgedrückt: Der Compiler soll feststellen, ob in anderen Quelltextdateien die gleichen Header-Dateien verwendet werden. Und wenn ja, so soll er sich die Mühe sparen, diese zu parsen und auszuwerten, sondern statt dessen die benötigten Informationen aus dem vorkompilierten Header beziehen.

Damit dies funktioniert, müssen zwei Dinge erfüllt sein:

- ✗ Die Compiler-Optionen für die anderen Quelltextdateien müssen so eingestellt sein, daß der Compiler für diese Dateien nach einem passenden vorkompilierten Header suchen soll. Wenn Sie im Fenster PROJEKT-EINSTELLUNGEN auf den Namen einer anderen *cpp*-Datei klicken, können Sie sehen, daß für diese Dateien die Option DATEI DER VORKOMPILIERTEN HEADER (.PCH) VERWENDEN aktiviert und *Stdafx.h* als Header-Datei angegeben ist. Das heißt, der Compiler schaut in der entsprechenden Quelltextdatei nach, welche Header-Dateien bis einschließlich *Stdafx.h* eingebunden werden, und sucht dann nach einem vorkompilierten Header für genau diese Header-Dateien.
- ✗ Die Quelltextdatei, für die der vorkompilierte Header verwendet werden soll, muß so aufgebaut sein, daß sie zuallererst die gleichen Header-Dateien einbindet, auf deren Grundlage auch der vorkompilierte Header erstellt wurde. Danach dürfen beliebige weitere Header-Dateien, Deklarationen und schließlich der Quellcode folgen. Da in unserem Fall der vorkompilierte Header allein auf der Grundlage der Header-Datei *Stdafx.h* angelegt wurde, kann man diese Bedingung leicht erfüllen. Man muß nur darauf achten, daß die Datei mit der `#include`-Anweisung für *Stdafx.h* beginnt.

Daß der vorkompilierte Header allein auf der Grundlage der Header-Datei *Stdafx.h* erstellt wird und erst in dieser die eigentlich interessierenden Header-Dateien aufgeführt sind, hat mehrere Vorteile:

- ✗ Man muß sich nicht merken, welche Header-Dateien in welcher Reihenfolge in den vorkompilierten Header eingeflossen sind. Man muß sich nur merken, daß der vorkompilierte Header auf der Datei *Stdafx.h* gründet.
- ✗ Will man den vorkompilierten Header um eine Header-Datei erweitern oder eine Header-Datei entfernen, muß man die entsprechenden Änderungen nicht in allen Quelltextdateien, sondern nur in *Stdafx.h* vornehmen.



Die erste Anweisung in Quelltextdateien von Projekten, die mit dem MFC-Anwendungs-Assistenten erstellt wurden, sollte immer die `#include`-Anweisung für `Stdafx.h` sein. Ansonsten wird der voreingestellte Mechanismus zur Erstellung und Verwendung vorkompilierter Header gestört.



Die angelegten Header-Dateien (Extension `pch`) sind meist sehr umfangreich. Bearbeitet man gleichzeitig eine Reihe von Projekten mit eigenen vorkompilierten Headern, kann dies unter Umständen schnell zu einer Verknappung des Festplattenspeichers führen. In diesem Fall sollte man sich überlegen, ob verschiedene Projekte mit gemeinsamen vorkompilierten Headern auskommen könnten oder ob auf einige vorkompilierte Header verzichtet werden kann.

2.6.3 Der inkrementelle Linker

Das Pendant zu den vorkompilierten Headern des Compilers ist das inkrementelle Binden des Linkers. Wenn Sie auf der Seite **LINKER** der Projekteinstellungen die Option **INKREMENTELLES BINDEN** aktivieren, beschleunigen Sie nachfolgende Linkprozesse. Glücklicherweise sind zur Nutzung des inkrementellen Bindens keine weiteren Einstellungen nötig, weshalb es zum inkrementellen Binden sonst nicht mehr viel zu sagen gibt.

2.6.4 Zwischendateien

Compiler und Linker erzeugen in Abhängigkeit von den Projekteinstellungen bei der Projekterstellung eine Reihe von Zwischendateien, die im Ausgabeverzeichnis des Projekts (*Debug* oder *Release*) abgelegt werden. Diese Zwischendateien sind Hilfsdateien, die keine essentiellen Informationen enthalten und jederzeit neu erzeugt werden können. Um Ihre Festplattenspeicher zu schonen, sollten Sie diese Dateien für Projekte, an denen Sie augenblicklich nicht mehr arbeiten, löschen.

Gehen Sie dazu im Windows Explorer zum Ausgabeverzeichnis des Projekts. Lassen Sie sich die Dateien mit allen Detailinformationen (**ANSICHT/DETAILS**) und sortiert nach Dateityp (Klick auf Spaltenüberschrift **TYP**) anzeigen, und löschen Sie alle Dateien mit dem Typ **ZWISCHENDATEI**.

2.6.5 Die Arbeit mit Projektkonfigurationen

Die Anpassungsmöglichkeiten, die uns das Dialogfeld PROJEKTEINSTELLUNGEN bietet, sind großartig, doch wenn man laufend in den Projekteinstellungen von Seite zu Seite springen müßte, um die Projekterstellung an die augenblicklichen Bedürfnisse anzupassen, wäre dies äußerst lästig. Visual C++ erlaubt es daher, komplette Sätze von Projekteinstellungen als »Konfigurationen« abzuspeichern und stellt gleich zwei Konfigurationen standardmäßig zur Verfügung:

- ✕ **Debug.** In dieser Konfiguration sind die Schalter zur Aufnahme von Debug-Informationen für den Debugger gesetzt, Optimierungen sind deaktiviert. Verwenden Sie diese Konfiguration während der Arbeit an Ihrem Projekt. Wenn Sie wollen, können Sie die Konfiguration anpassen, so daß auch Browser-Informationen erzeugt werden (siehe Hinweis weiter unten), oder Sie erzeugen zu diesem Zweck eine eigene Konfiguration, die auf der Debug-Konfiguration aufbaut (siehe Übung 2-7).
- ✕ **Release.** In dieser Konfiguration sind die Schalter zur Aufnahme von Debug-Informationen für den Debugger deaktiviert, Optimierungen sind eingeschaltet. Diese Konfiguration ist für die abschließende Erstellung des fertigen Programms gedacht. Debug-Informationen, die für das fertige Programm ja ohne Bedeutung sind, die Größe der ausführbaren Datei aber beträchtlich aufplustern können, werden daher bei der Kompilierung mit diesen Projekteinstellungen nicht erzeugt. Dafür wird das Programm optimiert. Sie können auf der Seite C/C++ der PROJEKTEINSTELLUNGEN selbst auswählen, in welcher Hinsicht das Programm optimiert werden soll (*Codegröße, Laufzeit*).

Für die Arbeit mit Konfigurationen müssen Sie wissen,

- ✕ **wie Sie Konfigurationen anpassen.** Rufen Sie das Dialogfeld PROJEKTEINSTELLUNGEN auf (Befehl PROJEKT/EINSTELLUNGEN), und wählen Sie im Listenfeld neben EINSTELLUNGEN FÜR den Eintrag WIN32 DEBUG aus. Bearbeiten Sie dann die Konfiguration durch Anpassung der Einstellungen auf den verschiedenen Registerseiten.
- ✕ **wie Sie Konfigurationen neu anlegen.** Rufen Sie dazu den Befehl ERSTELLEN/KONFIGURATIONEN auf, und klicken Sie in dem erscheinenden Dialogfeld auf den Schalter HINZUFÜGEN (siehe nachfolgende Übung).
- ✕ **wie Sie Konfigurationen auswählen.** Rufen Sie den Befehl ERSTELLEN/AKTIVE KONFIGURATION FESTLEGEN auf.

Übung 2-7: Einrichtung einer Projektkonfiguration mit Browser-Informationen

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 im obersten Eingabefeld einen Namen für die neue Konfiguration an (zum Beispiel *Debug_Browser*). Darunter können Sie eine bereits bestehende Konfiguration auswählen, auf deren Einstellungen Sie aufbauen wollen. Wählen Sie hier die WIN32 DEBUG-Konfiguration aus. Schließen Sie dann die Dialogfelder.
3. 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. Um Browser-Informationen zu erzeugen, aktivieren Sie im Dialogfeld PROJEKTEINSTELLUNGEN für den Projektknoten die Optionen BROWSE-INFO GENERIEREN auf der Seite C/C++, Kategorie ALLGEMEIN, und BROWSER-INFORMATIONSDATEI ERSTELLEN auf der Seite BROWSE-INFORMATION.
4. Lassen Sie das Programm neu erstellen (F7).



Der Quellcode-Browser dient rein der Analyse des Quelltextes. Wenn Sie ihn über den Befehl EXTRAS/QUELLCODE-BROWSER aufrufen, können Sie sich über die verschiedenen in Ihrem Programm verwendeten Bezeichner (Namen von Variablen, Klasse, Funktionen etc.) informieren, zu Deklarationen und Definitionen springen, sich Klassenhierarchien oder Aufrufdiagramme für Ihr Programm anzeigen lassen. Zudem steht der Quellcode-Browser hinter verschiedenen Befehlen, die in den Kontextmenüs der KLASSEN-Ansicht verfügbar sind, beispielsweise der Darstellung von Klassenhierarchien.

2.6.6 Was geschieht bei der Projekterstellung?

Als erstes werden die einzelnen Quellmodule (.cpp, .h etc.) des Projekts in binäre Objektdateien (.obj) übersetzt.



Als Quellmodul, oder Übersetzungseinheit, bezeichnet man dabei eine einzelne Quelltextdatei (.cpp) mit allen per *#include*-Direktiven aufgenommenen Dateien.

Gleichzeitig erzeugt der Ressourcen-Compiler aus den Ressourcendateien (.rc, .ico etc.) eine binäre .res-Datei (.res).

Dann bindet der Linker alle Objektdateien, plus der verwendeten Bibliotheksdateien (.lib) zur ausführbaren Datei des Projekts (.exe, .dll etc.) zusammen.

Im letzten Schritt werden die Ressourcen in die ausführbare Datei aufgenommen.

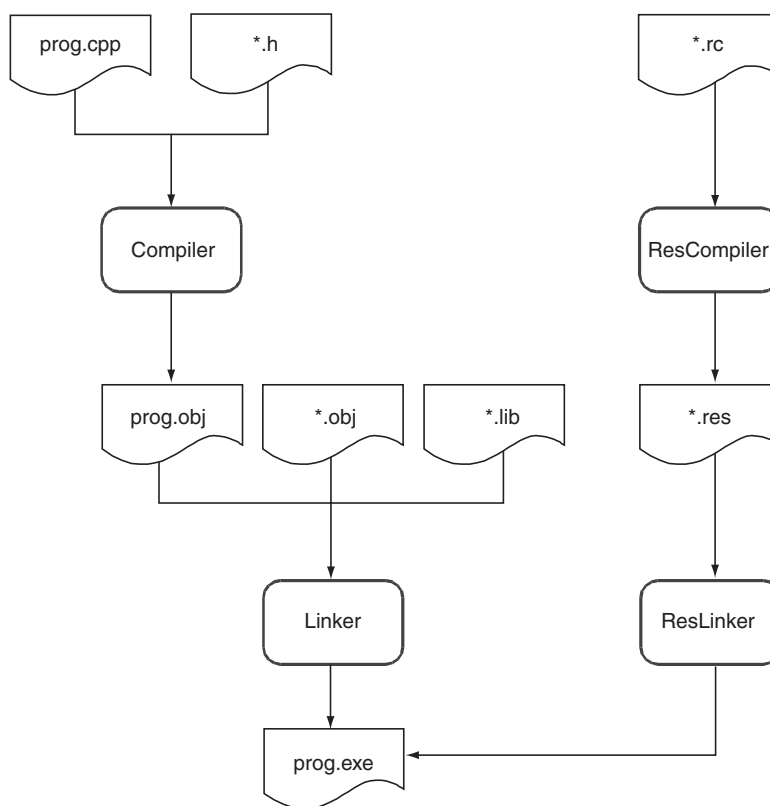


Bild 2.20:
Ablauf der
Projekter-
stellung

2.7 Der Debugger

Nur in den allerseltensten Fällen ist die Arbeit des Programmierers mit dem erfolgreichen Kompilieren und Linken der Anwendung abgeschlossen. Danach beginnt das Austesten des Programms, verbunden mit dem Ausmerzen auftretender Fehler (Bugs).



Der Begriff »Bug« (englisch für »Wanze«, »Insekt«) wurde übrigens an der Harvard University geprägt, wo eine in die Schaltungen eingedrungene Motte den Computer für Tage lahmlegte.

Bugs sind Fehler, die zur Laufzeit des Programms auftreten. Bugs bewirken, daß Berechnungen zu falschen Ergebnissen führen, daß ein Programm nicht so reagiert, wie wir es erwartet haben, oder daß es gar abstürzt oder sich in einer Endlosschleife aufhängt. Kurz gesagt, wir haben zwar ein Programm, das auch ausgeführt werden kann, aber nicht das macht, wofür wir es eigentlich vorgesehen hatten, weil wir irgendwo bei der Konzeption oder beim Implementieren der Routinen einen Fehler gemacht haben. Das Versteckteste an diesen Laufzeitfehlern ist, daß sie meist nur schwer zu lokalisieren sind. Zwar kann man häufig aus dem Zeitpunkt, zu dem das Programm unerwartet reagiert, auf die Quelltextstelle rückschließen, die den Fehler ausgelöst haben muß, doch erstens ist dies nicht immer so, und zweitens ist diese Eingrenzung oft nicht ausreichend. Was man braucht, ist also ein leistungsfähiges Tool, das uns bei der Lokalisierung der Laufzeitfehler unterstützt.

Der Debugger

Ein Debugger ist ein Programm, das ein anderes Programm schrittweise ausführen kann. Sofern in dem »debuggten« Programm spezielle Debug-Informationen vorhanden sind (im Programm verwendete Bezeichner, Zeilennummern etc.), kann der Debugger während der Ausführung anzeigen, welche aktuellen Werte in den Variablen des Programms gespeichert sind, welche Quelltextzeile gerade ausgeführt wird und wie der Aufrufstack aussieht.

In der Praxis sieht eine Debug-Sitzung so aus, daß man ständig die folgenden drei Schritte wiederholt:

1. **Programm im Debugger ausführen.** Um das Programm, das Sie gerade in der IDE bearbeiten, im Debugger ausführen zu lassen, rufen Sie einen der Befehle im Menü ERSTELLEN/DEBUG STARTEN auf. Um ein im Debugger angehaltenes Programm weiter ausführen zu lassen, rufen Sie einen der entsprechenden Befehle im Menü DEBUG auf (das Menü ERSTELLEN wird nach dem Starten des Debuggers durch das Menü DEBUG ersetzt).

2. **Programmausführung anhalten.** An den Stellen des Programms, die einem verdächtig vorkommen und in denen man einen Fehler vermutet, hält man die Ausführung des Programms an. Dazu verwendet man den Befehl DEBUG/ANHALTEN, setzt Haltepunkte oder lässt das Programm nur schrittweise ausführen.
3. **Programmzustand prüfen.** Bevor man das Programm weiter ablaufen lässt, lässt man sich in den Anzeigefenstern des Debuggers (Aufruf über ANSICHT/DEBUG-FENSTER) Informationen über den aktuellen Zustand des Programms anzeigen, beispielsweise den Inhalt von Variablen, den Zustand des Aufrufstacks oder der Register. Mit Hilfe dieser Informationen versucht man Rückschlüsse auf Ort und Art des Fehler zu ziehen.

Jetzt wäre es angebracht, Ihnen die verschiedenen Anzeigefenster und Befehle des Debuggers vorzustellen, doch ich möchte dieses Kapitel nicht mit Tabellen und Referenzen überladen. Alle weiteren Ausführungen zum Debuggen, einschließlich eines kleinen Beispiels, finden Sie daher im Kapitel 4.



Übung 2-8: Testen des Zinsprogramms

1. Rufen Sie jetzt im Menü ERSTELLEN den Befehl AUSFÜHREN VON ZINSEN.EXE auf (oder drücken Sie **Strg** + **F5**).

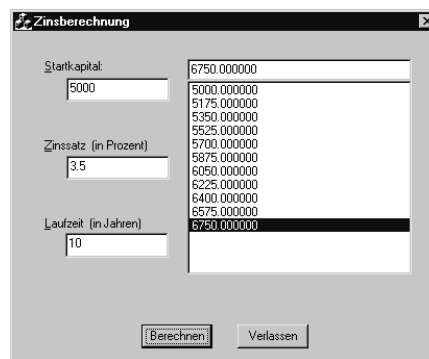


Bild 2.21:
Das fertige
Programm

2.8 Zusammenfassung

In diesem Kapitel sind wir den typischen Ablauf bei der Programmerstellung durchgegangen und haben dies zu einem Streifzug durch die Konzepte und Elemente der Visual C++-IDE genutzt. Eine Vielzahl von Begriffen, Menübefehlen und Einstellmöglichkeiten wurden Ihnen vorgestellt, und ich will gar nicht erst versuchen, diese Informationen hier noch einmal zusammenzufassen. Wenn Sie sich schnell über ein bestimmtes Element der IDE informieren wollen, schlagen Sie direkt im betreffenden Abschnitt dieses Kapitels nach. Die einzelnen Abschnitte sind zu diesem Zweck extra ein wenig referenzartig aufgebaut worden. Vielleicht ist Ihnen aber bei all den Informationen und Exkursen der Überblick über die Programmerstellung abhanden gekommen. Ich möchte diese Zusammenfassung daher nutzen, um noch einmal den typischen Ablauf bei der Programmerstellung zu referieren.

1. Projekt anlegen
2. Quelltext bearbeiten
3. Ressourcen erstellen und in Programm einbinden
4. Quelltext fertigstellen
5. Programm kompilieren und linken
6. Programm testen und debuggen

2.9 Fragen

1. Welches sind die drei wichtigsten Elemente der Projektverwaltung?
2. Welches sind die für uns am interessantesten Projekttypen und Assistenten?
3. Wie öffnet man ein bestehendes Projekt?
4. Wie lädt man am schnellsten eine bestimmte Datei eines Projekts in den Editor?
5. Wie springt man am schnellsten zum Quelltext einer Methode einer Klasse?
6. Nennen Sie fünf Arten von Ressourcen!
7. Lassen Sie das Zinsen-Projekt neu erstellen (Befehl ERSTELLEN/ALLES NEU ERSTELLEN), und beobachten Sie die Anzeige im Ausgabefenster. Warum wird die Datei *Stdafx.cpp* als erstes kompiliert?

2.10 Aufgaben

1. Schauen Sie sich die Ressourcenskriptdatei des *Zinsen*-Projekts einmal als Textdatei an. Rufen Sie dazu den Befehl DATEI/ÖFFNEN auf, und wählen Sie im Öffnen-Dialog unter ÖFFNEN ALS die Option TEXT aus, bevor Sie die rc-Datei laden lassen. Schauen Sie sich an, wie die einzelnen Ressourcen (Tastaturkürzel, Dialog, Menü, Stringtabelle, Anwendungssymbol) definiert werden.

Ganz Mutige dürfen sich auch daran versuchen, die Ressourcen direkt in der Ressourcenskriptdatei zu bearbeiten. Verwenden Sie dazu aber nicht unser Zinsen-Projekt, sondern legen Sie sich mit Hilfe des MFC-Anwendungs-Assistenten ein neues Test-Projekt an.

2. Erstellen Sie das Programm sowohl in der Debug-Konfiguration als auch in der Release-Konfiguration. Vergleichen Sie die Größen der jeweils erzeugten EXE-Datei. (Letztere sind in den Unterverzeichnissen Debug und Release zu finden.)
3. Implementieren Sie eine Funktion `ErtragMitZinseszins()`, die genau wie `ErtragOhneZinseszins()` aufgerufen wird, aber den Ertrag unter Berücksichtigung von Zinseszinsen zurückliefert. Verwenden Sie zur Abwechslung einmal diese Funktion in dem *Zinsen*-Programm.
4. Wenn Sie in objektorientierter Programmierung bewandert sind und Lust haben, sollten Sie die beiden Zinsberechnungsfunktionen in einer Klasse zusammenfassen. Vielleicht in einer Klasse *Sparbuch*?

Die Assistenten

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 Abschließen Ihrer Angaben erstellt der Assistent für Sie ein passendes Projekt, legt erste Dateien an und setzt ein Codegerüst 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-Steuerelemente etc.) befreit, so daß Sie gleich mit der kreativen Arbeit beginnen können. Projekte, die mit einem der MFC-Anwendungs-Assistenten erstellt wurden, können darüber hinaus mit dem Klassen-Assistenten in vielfacher Weise weiterbearbeitet werden.

In diesem Kapitel werde ich kurz über den MFC-Anwendungs-Assistenten und den Klassen-Assistenten referieren. Ich werde Ihnen die einzelnen Seiten des Anwendungs-Assistenten vorstellen und Sie mit den Möglichkeiten des Klassen-Assistenten vertraut machen. Richtig grün werden Sie den Assistenten dadurch sicherlich nicht – insbesondere dem Klassen-Assistenten werden Sie wohl noch einige Zeit skeptisch gegenüberstehen. Doch dies wird sich legen, wenn wir in den nachfolgenden Kapiteln des zweiten Teils immer wieder auf die beiden Assistenten zurückgreifen und sie für unsere tägliche Arbeit nutzen.



Damit taucht aber gleich ein weiteres Problem auf: Die beiden genannten Assistenten sind formidable Hilfsmittel, die den Programmierer bei der Erledigung bestimmter Standardaufgaben tatkräftig unterstützen. Teilweise ist diese Unterstützung aber so weit fortgeschritten, daß der Programmierer gar nicht mehr merkt, was mit seinem Quelltext passiert. Gerade Einsteiger in die MFC-Programmierung erliegen leicht der Versuchung, voll und ganz auf die Assistenten zu vertrauen und mit ihrer Hilfe in kürzester Zeit die schönsten Programme zu erstellen, ohne jedoch zu verstehen, was in dem Programm tatsächlich vor sich geht, ohne je den eigenen Quelltext durchgelesen zu haben. Auf diese Weise kann man zwar nette Windows-Anwendungen generieren, doch man bleibt im Grunde immer ein Anfänger der Windows-Programmierung, weil man nichts von Windows und den Erfordernissen der Windows-Programmierung verstanden hat. Genau dies soll uns nicht passieren. Wir wollen nicht die Sklaven, sondern die Herren der Assistenten sein. Die Assistenten sollen uns die Arbeit erleichtern und uns lästige Routineaufgaben abnehmen, aber sie sollen nicht unser eigenes Unwissen bemänteln.

Aus diesem Grunde werden wir uns mit den Assistenten in drei Stufen beschäftigen.

- ✗ In diesem Kapitel finden Sie eine kurze Referenz der wichtigsten Optionen des MFC-Anwendungs- und des Klassen-Assistenten. Hier können Sie sich einen Überblick über die Möglichkeiten der Assistenten verschaffen oder gezielt bestimmte Optionen nachschlagen.
- ✗ Im zweiten Teil des Buches werden wir die Assistenten für unsere Programmierarbeit nutzen. Sie werden lernen, für welche Arbeiten die Assistenten zu gebrauchen sind und wie man sie einsetzt.
- ✗ Im dritten Teil des Buches schließlich werden wir die Arbeitsweise der Assistenten etwas kritischer beleuchten. Hier nehmen wir den von den Assistenten erzeugten Code etwas genauer unter die Lupe und arbeiten ein wenig Hintergrundwissen zu Windows und der Windows-Programmierung auf.

Sie verschaffen sich in diesem Kapitel einen Überblick:

- ✗ über die Dialogseiten und Optionen des MFC-Anwendungs-Assistenten
- ✗ über die Dialogseiten und Einsatzmöglichkeiten des Klassen-Assistenten

3.1 Der MFC-Anwendungs-Assistent

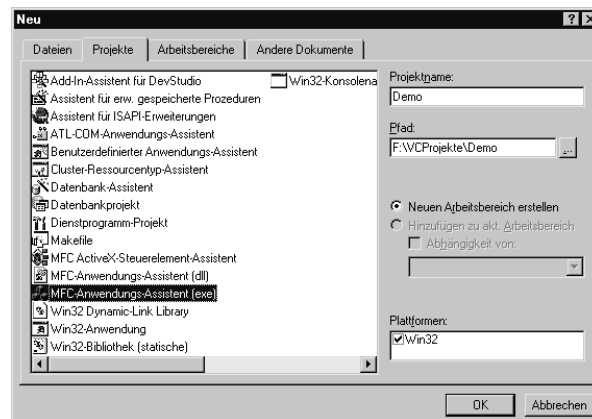


Bild 3.1:
Aufruf des
MFC-Anwen-
dungs-
Assistenten

Der MFC-Anwendungs-Assistent 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
- ✗ Datenbankanbindung
- ✗ OLE- und ActiveX-Features
- ✗ verschiedenen Fensterdekorationen
- ✗ und anderem

auszustatten.

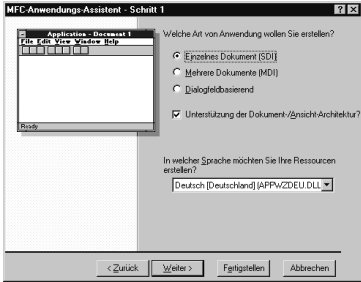
Aufgerufen wird der Anwendungs-Assistent über den Befehl DATEI/NEU. Auf der Seite PROJEKTE kann er dann aus der Liste der Projekttypen und Assistenten ausgewählt werden.

Klickt man auf OK, erscheint das Dialogfeld des Assistenten. Das Dialogfeld verfügt über mehrere Seiten, die allerdings nicht über Reiter, sondern über die Schalter ZURÜCK und WEITER aufgerufen werden. So arbeitet man sich Schritt für Schritt durch die Optionen des Anwendungs-Assistenten und teilt ihm mit, wie die Anwendung, für die er ein Projekt und ein Grundgerüst anlegen wird, beschaffen sein soll.

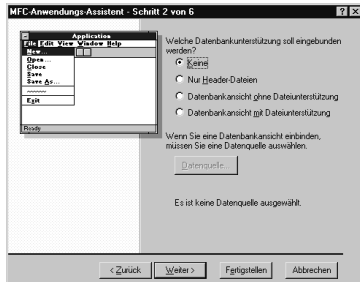
KAPITEL 3 Die Assistenten

Schauen wir uns die einzelnen Seiten des Assistenten einmal etwas genauer an. Die Einstellungen der Abbildungen in Tabelle 3.1 zeigen übrigens nicht die Voreinstellungen des Assistenten, sondern die Einstellungen, die wir im zweiten Teil für die meisten der Beispielanwendungen verwenden werden.

Tabelle 3.1:
Dialogseiten
des MFC-
Anwendungs-
Assistenten

Seite	Optionen
	Auf der ersten Seite legen Sie den grundlegenden Typ Ihrer Anwendung fest: EINZELNES DOKUMENT (SDI), eine Anwendung mit einem Rahmenfenster, in dem immer nur ein Dokumentfenster zur Zeit angezeigt werden kann (vgl. Notepad-Editor). Fast alle Beispiele in diesem Buch sind SDI-Anwendungen MEHRERE DOKUMENTE (MDI), eine Anwendung mit einem Rahmenfenster, in dem mehrere Dokumentfenster verwaltet werden können (vgl. WinWord und Kapitel 21) DIALOGFELDBASIEREND, eine Anwendung mit einem Dialogfeld als Rahmenfenster (vgl. MFC-Anwendungs-Assistent und Übungen aus Kapitel 2) Als Sprache für die anzulegenden Ressourcen sollten Sie natürlich DEUTSCH auswählen. Wenn Sie aber lieber gleich für den englischsprachigen Markt programmieren wollen, können Sie auch ENGLISCH als Sprache auswählen – das vom Assistenten angelegte Menü erzeugt dann englische Beschriftungen für die Menübefehle. Seit VC 6.0 gibt es die Möglichkeit, auf DOC/VIEW-UNTERSTÜTZUNG zu verzichten. Doc/View ist ein Programmiermodell, das sich rein auf die Konzeption 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). Für unsere kleinen Beispielanwendungen bringt das Doc/View-Modell im Grunde wenig, doch sollten Sie sich ruhig schon langsam daran gewöhnen.

Seite



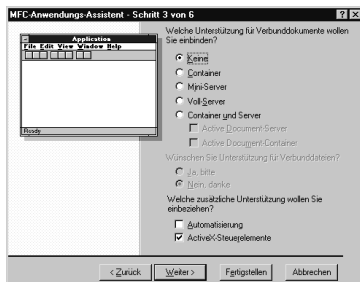
Optionen

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.

Wenn Sie sich für eine Datenbankverbindung 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.

Für uns werden diese Optionen aber erst im Kapitel 25 interessant.



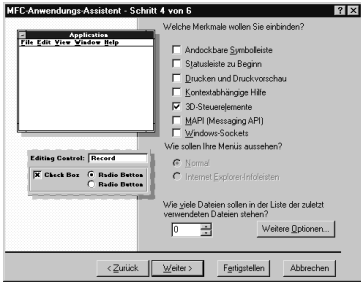
Die dritte Seite führt die OLE-Möglichkeiten und ActiveX-Features auf. Geben Sie hier an, ob Ihre Anwendung OLE-Verbindungs-funktionalität als SERVER, CONTAINER, MINI-SERVER oder CONTAINER/SERVER unterstützen soll (siehe Kapitel 22).

Wenn Sie sich für eine Form der Unterstützung von Verbindungsdateien entschieden haben, können Sie danach auch die Unterstützung für VERBUNDDATEIEN aktivieren. Nach Auswahl dieser Option werden die Objekte Ihrer Anwendung im Verbindungsdateiformat gespeichert, welches das Laden nach Aufforderung und sukzessives Speichern ermöglicht (aber größere Dateien bedingt).

Außerdem können Sie Ihrem Projekt Unterstützung für AUTOMATIONS-SERVER sowie für ACTIVE-X-STEUELEMENT-CONTAINER hinzufügen. Aktivieren Sie letztere Option, wenn Sie ActiveX-Steuerelemente in den Dialogen Ihrer Anwendung verwenden möchten.

*Tabelle 3.1:
Dialogseiten
des MFC-
Anwendungs-
Assistenten
(Fortsetzung)*

Tabelle 3.1:
Dialogseiten
des MFC-
Anwendungs-
Assistenten
(Fortsetzung)

Seite	Optionen
	Die vierte Dialogseite des Anwendungs-Assistenten enthält unterschiedliche Optionen. Die meisten Optionen sind selbsterklärend oder verfügen über eine ausreichende Online-Hilfe. Wenn Sie die Option KONTEXTABHÄNGIGE HILFE auswählen, werden Ihrer Anwendung 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. Wenn Sie das Kontrollkästchen MAPI (MESSAGING API) auswählen, werden die MAPI-Bibliotheken an Ihre Anwendung gebunden, und dem Menü DATEI wird der Eintrag SENDEN hinzugefügt. Das Aktivieren der Option WINDOWS-SOCKETS führt dazu, daß Ihrem Projekt WinSock-Bibliotheken und entsprechende Header-Dateien hinzugefügt werden. Die WinSock-Funktionalität, die Ihr Programm netzwerkfähig macht, müssen Sie jedoch selbst dem Projekt hinzufügen. Ihr besonderes Interesse sollte dem Dialog WEITERE OPTIONEN gelten, der nach einem Klick auf die gleichnamige Schaltfläche aufgerufen wird. In diesem Dialog bestimmen Sie einige zusätzliche Optionen, die hauptsächlich das äußere Erscheinungsbild Ihrer Anwendung betreffen. Der Dialog WEITERE OPTIONEN besteht aus zwei Registern. Das erste Register mit der Bezeichnung ZEICHENFOLGEN FÜR DOKUMENTVORLAGE ermöglicht Ihnen die Angabe verschiedener Zeichenfolgen, die für den Datei Öffnen-Dialog, die Dokumentvorlagen und den Fenstertitel (MDI-Anwendungen) relevant sind. Die eingegebenen Zeichenfolgen werden in der Stringtabelle der Ressourcenskriptdatei der Anwendung unter IDR_MAINFRAME gespeichert (siehe auch Kapitel 11 zur Erstellung des Texteditors)

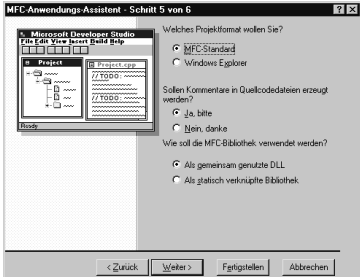
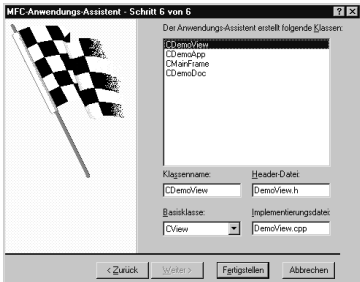
Seite	Optionen
	Das zweite Register des Dialogs WEITERE OPTIONEN ist mit FENSTERSTILE bezeichnet und dient der Konfiguration des Rahmenfensters Ihrer Anwendung. Auf der fünften Seite legen Sie fest, ✗ ob Sie ein normales oder ein Explorer-ähnliches Fenster haben möchten, ✗ ob ausführliche Kommentare angelegt werden sollen, ✗ ob die MFC statisch oder als DLL eingebunden werden soll.
	Auf der letzten Seite werden die zu generierenden Klassen angezeigt, und Sie haben die Möglichkeit, zu den einzelnen Klassen andere Basisklassen auszuwählen und/oder die Dateinamen zu ändern. Klicken Sie nun auf die Schaltfläche FERTIGSTELLEN, um das Projekt anlegen zu lassen.
	

Tabelle 3.1:
Dialogseiten
des MFC-
Anwendungs-
Assistenten
(Fortsetzung)

Bevor der Anwendungs-Assistent an die Arbeit geht, zeigt er noch ein Kontrollfenster mit einer kurzen Beschreibung des zu erstellenden Projekts an. Wenn Sie dieses durch Drücken von OK bestätigen, beginnt der Assistent damit, das Projekt einzurichten und ein lauffähiges Programmgerüst zu implementieren.

Wenn Sie wollen, können Sie das Programm direkt erstellen und ausführen lassen (Befehl ERSTELLEN/AUSFÜHREN VON ...).

3.2 Der Klassen-Assistent

Der Klassen-Assistent dient der halbautomatischen Bearbeitung von MFC-Projekten, die über eine CLW-Datei verfügen (beispielsweise die vom MFC-Anwendungs-Assistenten generierten Projekte).

Mit seiner Hilfe können Sie

- ✗ neue Klassen erstellen
- ✗ virtuelle Methoden überschreiben

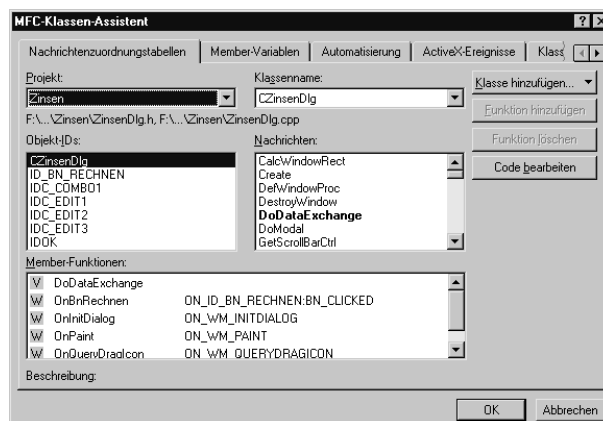
- ✗ Antwortmethoden zur Nachrichtenverarbeitung einrichten
- ✗ Dialogklassen erstellen
- ✗ Elementvariablen für Steuerelemente aus Dialogen anlegen
- ✗ Ereignisse für ActiveX-Steuerelemente definieren
- ✗ Klassen automatisieren
- ✗ Klassen aus Typbibliotheken erstellen

Vermutlich werden Sie im Moment nur mit den wenigsten dieser Optionen etwas anfangen können. Das ist aber nicht weiter schlimm, und es sollte Sie auch nicht verdrießen, daß Sie nach dem Durchlesen dieses Kapitels nicht wesentlich klarer sehen. So richtig anfreunden werden wir uns mit dem Klassen-Assistenten erst in den Kapiteln zur Nachrichtenbearbeitung und zu den Dialogfenstern. Im letzten Teil werden wir noch einmal im Zusammenhang mit der Automatisierung auf den Klassen-Assistenten zurückkommen, doch werden wir die Themenbereiche ActiveX, OLE und Automatisierung nur flüchtig streifen (für eine gründliche Einarbeitung in diese Programmierbereiche bedarf es schon spezieller Literatur). Betrachten Sie die nachfolgenden Abschnitte also als eine Referenz, auf die Sie nach dem Studium des zweiten Teils des Buches zurückkommen können.

3.2.1 Das Dialogfeld des Klassen-Assistenten

Wenn Sie den Klassen-Assistenten aufrufen (beispielsweise über den Befehl ANSICHT/KLASSEN-ASSISTENT), erscheint ein fünfseitiges Dialogfeld.

Bild 3.2:
Das Dialogfeld
des Klassen-
Assistenten



- ✗ **Nachrichtenzuordnungstabellen.** Auf dieser Seite können Sie sich darüber informieren, welche Antwortmethoden in welchen Klassen zu welchen Windows-Nachrichten deklariert sind (soweit diese vom Klassen-Assistenten verwaltet werden). Sie können Methoden zur Nachrichtenverarbeitung 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 (um sie auf diese Weise anderen Anwendungen zugänglich zu machen).
- ✗ **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 die Nachrichtenfilter für die verschiedenen Klassen auswählen. Dies beeinflusst die Anzeige der Nachrichten im Listefeld NACHRICHTEN der Seite NACHRICHTENZUORDNUNGSTABELLEN. Sie können auch ein Fremdobjekt anzeigen oder setzen, das mit der Formularansichts- oder Datensatzansichtsklasse Ihres Dialogs verbunden ist (für Datenbank-Anwendungen relevant).

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



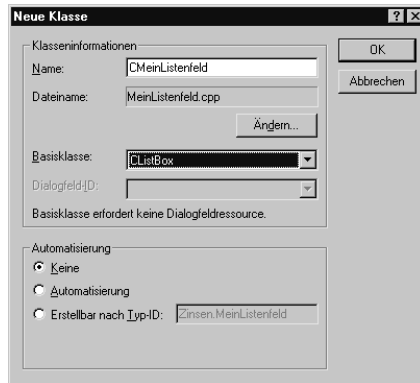
3.2.2 Erstellen einer neuen Klasse

Mit einem Klick auf die Schaltfläche KLASSE HINZUFÜGEN erstellen Sie eine neue Klasse für Ihre Anwendung.

Der Klassen-Assistent ermöglicht es Ihnen,

- ✗ eine neue Klasse auf der Grundlage einer MFC-Basisklasse zu erstellen,
- ✗ eine Klasse auf der Grundlage einer Typenbibliothek zu erstellen.

Bild 3.3:
Neue Klasse
hinzufügen



Neue Klassen

Wenn Sie eine Klasse 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 Anwendung vermerkt sind.

Unterstützt die Basisklasse die Automatisierung, können Sie die entsprechenden Optionen im unteren Bereich des Dialogs selektieren.

Der Assistent legt Header- und Quelltextdatei für die Klasse an und erstellt den Code für die Klassendeklaration und den Konstruktor.



Ein Beispiel für die Einrichtung einer neuen Dialogklasse mit Hilfe des Klassen-Assistenten finden Sie im Kapitel 10.

3.2.3 Behandlungsmethoden einrichten

Um sich über die Behandlungsmethoden einer Klasse, eines Objekts oder einer Nachricht zu informieren oder eine solche Methode einzurichten oder zu bearbeiten, gehen Sie wie folgt vor:

1. Wechseln Sie zur Seite NACHRICHTENZUORDNUNGSTABELLEN.

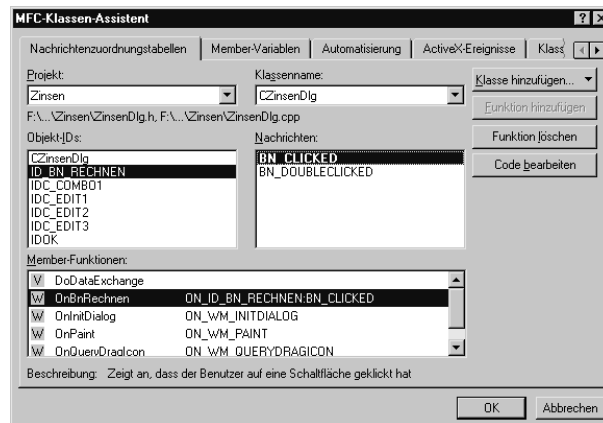


Bild 3.4:
Bearbeitungs-
methoden
einrichten

2. Wählen Sie im Feld KLASSENNAME die Klasse aus, in der die Behandlungsmethode deklariert werden soll.
3. Wählen Sie im Feld OBJEKT-IDS das Objekt aus, dessen Behandlungsmethoden Sie einsehen möchten oder für das Sie eine Behandlungsmethode einrichten wollen.

Die bereits eingerichteten Behandlungsmethoden werden jetzt im Feld MEMBER-FUNKTIONEN angezeigt.

Wenn Sie eine bereits eingerichtete Behandlungsmethode bearbeiten wollen:

- 4a. Wählen Sie die Behandlungsmethode im Feld MEMBER-FUNKTIONEN aus, und klicken Sie auf den Schalter CODE BEARBEITEN.

Wenn Sie eine neue Behandlungsmethode für eine Nachricht einrichten wollen:

- 4b. Wählen Sie im Feld NACHRICHTEN eine Nachricht 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 dessen Deaktivierung) zur Verfügung. Für Klassen stehen Ihnen die virtuellen Methoden, die vordefinierten Bearbeitungsmethoden (ON...) und die Windows-Nachrichten (WM...) zur Verfügung.

Nachrichten (oder Methoden), für die bereits Implementierungen vorliegen, werden im Feld NACHRICHTEN durch Fettschrift hervorgehoben.

5. Klicken Sie auf den Schalter FUNKTION HINZUFÜGEN.



Beispiele für die Einrichtung von Nachrichtenbehandlungsmethoden mit Hilfe des Klassen-Assistenten finden Sie in den Kapitel 7 bis 13.

Virtuelle Methoden überschreiben

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

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

Member-Funktionen löschen

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

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

Member-Funktionen bearbeiten

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



Ein Wort zur Terminologie. In der objektorientierten Terminologie hat es sich im deutschen Sprachgebrauch eingebürgert, Funktionen, die als Elemente von Klassen deklariert sind, als Elementfunktionen oder Methoden, und Variablen, die als Elemente von Klassen deklariert sind, als Elementvariablen zu bezeichnen. Microsoft verwendet entgegen dieses Usus im Dialogfeld des Klassen-Assistenten die Begriffe Member-Funktion und Member-Variable. Lassen Sie sich davon nicht beirren, mit Member-Funktionen sind einfach Methoden und mit Member-Variablen sind Elementvariablen gemeint.

3.2.4 Member-Variablen und Datenaustausch

Auf der zweiten Registerseite des Klassen-Assistenten-Dialogs können Sie Member-Variablen definieren und modifizieren.

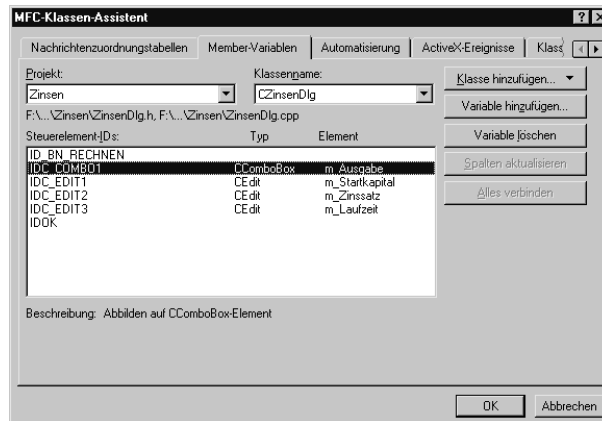


Bild 3.5:
Bearbeiten von
Member-Variablen

Wenn Sie einen Dialog implementieren, reicht es nicht, die Ressource zu erzeugen, 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 aus 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).

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 Bestätigen (und Beenden) des Dialogs die Elementvariablen umgekehrt mit den Eingaben aus den Steuerelementen aktualisiert werden.

Elementvariablen für Dialogelemente einrichten:

1. 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 (siehe Übung aus Kapitel 10).
 - ✗ Im Feld VARIABLENTYP wählen Sie den genauen Datentyp aus (üblicherweise ist dieser bereits durch den Typ des Steuerelements festgelegt, siehe Übung 2.5 aus Kapitel 2).

Wenn Sie auf diese Weise eine neue Member-Variable definieren, aktualisiert der Klassen-Assistent den Quellcode Ihrer Anwendung an verschiedenen Positionen. Insbesondere überarbeitet er die Methode `DoDataExchange()`. In dieser Funktion werden Informationen zwischen dem Steuerelementobjekt selbst und der Variable ausgetauscht, die den Wert des Objekts repräsentiert.



Beispiele für die Einrichtung von Member-Variablen bei Dialogsteuerelementen mit Hilfe des Klassen-Assistenten finden Sie in den Kapitel 2 und 10.

3.2.5 Automatisierung

Das Register AUTOMATISIERUNG im Dialog des Klassen-Assistenten ermöglicht Ihnen das Hinzufügen und Modifizieren von Automatisierungs-Eigenschaften und -Methoden (häufig bezeichnet als OLE-Automatisierung).

Voraussetzung ist, daß die betreffende Klasse die Automatisierung unterstützt. Unterstützung für Automatisierung können Sie beispielsweise vorsehen, indem Sie bei der Erstellung der Klasse im MFC-Anwendungs-Assistenten die Option AUTOMATISIERUNG aktivieren.



Automatisierung ist für Sie nur dann interessant, wenn Sie bestimmte Methoden und Eigenschaften Ihrer Klasse anderen Programmen zur Verfügung stellen wollen (siehe Kapitel 22).

Automatisierungs-Methode hinzufügen

1. Wechseln Sie zur Seite AUTOMATISIERUNG.
2. Wählen Sie im Feld KLASSEN-NAME 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 RÜCKGABETYP.
 - ✗ 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.

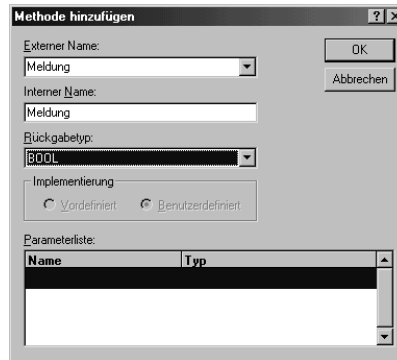


Bild 3.6:
Hinzufügen
einer Auto-
matisierungs-
Methode

Eigenschaften automatisieren

1. 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.

3.2.6 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.

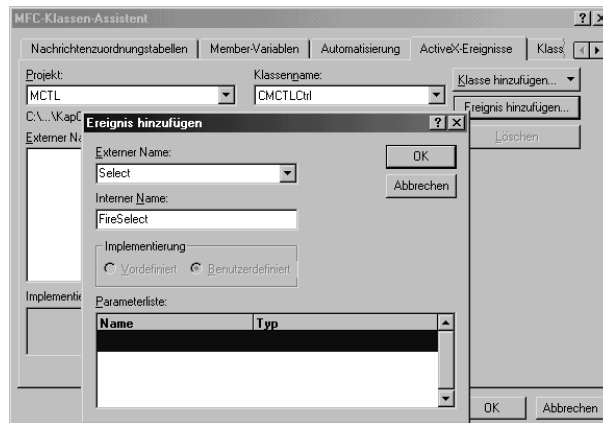


Bild 3.7:
Erstellen eines
ActiveX-Ereignisses

Erstellen eines ActiveX-Ereignisses

1. Wechseln Sie zur Seite ACTIVEX-EREIGNISSE.
2. Wählen Sie im Feld KLASSENNAME die Klasse des ActiveX-Steuerelements aus.
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.

3.2.7 Die Klasseninformationsdatei (.clw)

Der Klassen-Assistent kann nicht den gesamten Quellcode Ihrer Anwendungen 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 Dateierweiterung 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 Behandlungsmethoden zu Nachrichten 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.

3.3 Zusammenfassung

Der MFC-Anwendungs-Assistent richtet nicht nur neue Projekte für Sie ein, er erzeugt auch gleich ein passendes Anwendungsgerüst, auf dem Sie aufbauen können. Wie dieses Anwendungsgerüst aussieht und ausgestattet ist, können Sie selbst in den Dialogseiten des Anwendungs-Assistenten festlegen. So können Sie SDI-, MDI- oder dialogbasierte Anwendungen erstellen lassen, Sie können den Code für Datenverwaltung und Datenanzeige trennen lassen (Doc/View-Modell), Sie können Ihre Anwendungen mit Datenbank-, ActiveX- und OLE-Unterstützung versehen und vieles mehr.

Um ein neues Projekt von dem MFC-Anwendungs-Assistenten anlegen zu lassen, rufen Sie den Befehl DATEI/NEU auf und wählen auf der Seite PROJEKTE den MFC-ANWENDUNGS-ASSISTENTEN (EXE) aus.

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

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

- ✕ Das erste dieser Register, das mit NACHRICHTENZUORDNUNGSTABELLEN bezeichnet ist, ermöglicht Ihnen die Definition von Bearbeitungsmethoden für verschiedene Windows-Nachrichten.
- ✕ In dem Register MEMBER-VARIABLEN können Sie Elementvariablen für den Datenaustausch mit Dialog-Steuerelementen oder Datenbankformularen einrichten.
- ✕ Das Register AUTOMATISIERUNG dient der Einrichtung verschiedener Automatisierungs-Methoden und -Eigenschaften für Klassen, die Automatisierung unterstützen. Für ActiveX-Steuerelemente können Sie vordefinierte sowie benutzerdefinierte Methoden und Eigenschaften bestimmen.
- ✕ Das Register ACTIVEX-EREIGNISSE dient der Definition von ActiveX-Ereignissen, die von einem ActiveX-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.

Fehlersuche mit dem Debugger

Mehr und mehr beherrschen Computer und Microchips unser Leben. In immer neue Bereiche dringen Hard- und Software vor. Wurden Computer anfangs nur in großen Forschungsinstituten eingesetzt, begegnet man ihnen heute praktisch schon in jedem Büro und jedem Haushalt. Die nächste Stufe wird sein, daß die Software den Computer verläßt und mit Hilfe hochintegrierter Chips in praktisch jedes beliebige Objekt eindringen kann. Technische Geräte, die bis dato noch rein elektronisch gesteuert wurden, werden damit programmierbar und ... intelligent. Haben Sie vielleicht auch einen Infrarot-Melder vor Ihrer Haustür, der das Außenlicht anschaltet, wenn Sie abends nach Hause kommen? Nun, wie wäre es, wenn Sie diesen Infrarot-Melder so programmieren könnten, daß er nicht nur das Licht anknipst, sondern Sie gleichzeitig mit einem freundlichen »Guten Abend« willkommen heißt. Oder vielleicht kaufen Sie sich gleich die Luxusversion mit eingebauter Videotechnik, die Ihr Gesicht erkennt und Sie mit Namen anredet, während zwielichtige Gestalten, die sich abends Ihrer Haustür nähern, mit einem grimmigen »Hau ab« verscheucht werden. Ist das nicht der absolute Wahnsinn?

O tempora, o mores! Nein, ganz so schlimm wird es wohl doch nicht werden. Fest steht aber, daß immer weitere Bereiche der Technik wie des alltäglichen Lebens von Computern und Programmen abhängen. Dies erschließt uns ganz neue Möglichkeiten, birgt aber auch Gefahren, denn Programme können fehlerhaft sein:

- ✗ Vielleicht wurde bei einer Berechnung fälschlicherweise ein Pluszeichen statt eines Minuszeichen verwendet.



- ✗ Vielleicht werden Benutzereingaben nicht auf Korrektheit geprüft. (So muß ein Programm, das zur Berechnung eines Bruchs die Werte für Zähler und Nenner vom Anwender abfragt, auch damit rechnen, daß der Anwender eine 0 für den Nenner eingibt.)
- ✗ Vielleicht geht der Programmierer beim Schreiben des Programms einfach von falschen Voraussetzungen aus. (Beispielsweise, daß ein 1986 implementiertes DOS-Programm im Jahre 2000 mit Sicherheit nicht mehr im Gebrauch ist und man daher die Jahreszahl auf die letzten zwei Ziffern begrenzen kann.)

Wie auch immer, wir müssen uns damit abfinden, daß ein Programm, das vom Compiler ohne Probleme übersetzt werden konnte und ausführbar ist, immer noch voller Fehler sein kann. Es bleibt uns daher nichts anderes übrig, als unser Programm gründlich zu testen. Dabei erweisen sich diese Fehler, die wir zur Unterscheidung von den syntaktischen Fehlern, die bereits vom Compiler abgefangen werden, als Laufzeitfehler bezeichnen, in zweierlei Hinsicht als besonders heimtückisch:

Erstens treten diese Fehler nicht unbedingt bei jeder Ausführung des Programms auf. So könnte es sein, daß bei einem Programm mit ungefähr zwanzig Menübefehlen der Fehler nur dann auftritt, wenn ein bestimmter Menübefehl aufgerufen wird. Es könnte sogar so sein, daß der Fehler nur dann auftritt, wenn vor dem Aufruf von Menübefehl A der Menübefehl E aufgerufen wurde. Vielleicht tritt der Fehler nur dann auf, wenn der Anwender als Dateinamen den Namen einer nicht vorhandenen Datei angibt und dann zuerst den Menübefehl E und direkt danach den Menübefehl A aufruft... Worauf ich hinaus will, ist, daß man Programme gründlich testen sollte, dabei aber trotzdem nie vollkommen sicherstellen kann, daß ein Programm wirklich ohne Fehler ist (außer bei ganz einfachen Anwendungen).

Zweitens ist es nicht damit getan, Laufzeitfehler zu provozieren – man muß sie auch lokalisieren und korrigieren. Gerade die Lokalisierung ist aber oftmals ein Riesenproblem. Häufig sind es ganz einfach zu behebende Fehler (wie zum Beispiel eine Schleifenvariable, die um Eins zu weit hochgezählt wurde), die verheerende Folgen zeitigen und die man erst nach Stunden ermüdenden Suchens aufspürt.

Damit Sie bei der Aufspürung möglicher Laufzeitfehler nicht ganz auf sich allein gestellt sind, stellt Ihnen Visual C++ einen leistungsfähigen integrierten Debugger zur Seite, dessen einziger Sinn und Zweck darin liegt, die Aufindung von Laufzeitfehlern zu erleichtern.

Sie lernen in diesem Kapitel:

- ✗ wie eine Debug-Sitzung abläuft
- ✗ wie man im Debugger ein Programm ausführt
- ✗ wie man Haltepunkte setzt
- ✗ wofür man die verschiedenen Debug-Fenster gebrauchen kann
- ✗ was beim Debuggen allgemein zu beachten ist

4.1 Der Debugger

Ein Debugger ist ein Programm, das ein anderes Programm schrittweise ausführen kann. Sofern in dem »debuggten« Programm spezielle Debug-Informationen vorhanden sind (im Programm verwendete Bezeichner, Zeilennummern etc.), kann der Debugger während der Ausführung anzeigen, welche aktuellen Werte in den Variablen des Programms gespeichert sind, welche Quelltextzeile gerade ausgeführt wird und wie der Aufrufstack aussieht.

In der Praxis sieht eine Debug-Sitzung so aus, daß man ständig die folgenden drei Schritte wiederholt:

1. **Programm im Debugger ausführen.** Um das Programm, das Sie gerade in der IDE bearbeiten, im Debugger ausführen zu lassen, rufen Sie einen der Befehle im Menü ERSTELLEN/DEBUG STARTEN auf. Um ein im Debugger angehaltenes Programm weiter ausführen zu lassen, rufen Sie einen der entsprechenden Befehle im Menü DEBUG auf (das Menü ERSTELLEN wird nach dem Starten des Debuggers durch das Menü DEBUG ersetzt).
2. **Programmausführung anhalten.** An den Stellen des Programms, die einem verdächtig vorkommen und in denen man einen Fehler vermutet, hält man die Ausführung des Programms an. Dazu verwendet man den Befehl DEBUG/ANHALTEN, setzt Haltepunkte oder läßt das Programm nur schrittweise ausführen.
3. **Programmzustand prüfen.** Bevor man das Programm weiter ablaufen läßt, läßt man sich in den Anzeigefenstern des Debuggers (Aufruf über ANSICHT/DEBUG-FENSTER) Informationen über den aktuellen Zustand des Programms anzeigen, beispielsweise den Inhalt von Variablen,

den Zustand des Aufrufstacks oder der Register. Mit Hilfe dieser Informationen versucht man Rückschlüsse auf Ort und Art des Fehlers zu ziehen.

4.2 Vorbereitung des Programms für das Debuggen

Um diese Debug-Informationen in ein Programm aufnehmen zu lassen, kompilieren Sie das Programm in der Debug-Konfiguration oder aktivieren Sie selbst die folgenden Optionen.

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).
- ✗ OPTIMIERUNGEN: DEAKTIVIEREN (DEBUG), um eine möglichst vollständige Übereinstimmung zwischen Ihrem Quelltext und dem »debuggten« Objektcode zu erreichen.

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

- ✗ Wählen Sie unter Laufzeitbibliothek eine passende Debug-Version der C-Laufzeitbibliothek aus.

Auf der Seite LINKER, Kategorie ALLGEMEIN der Projekteinstellungen:

- ✗ DEBUG-INFO GENERIEREN setzen.

Auf der Seite LINKER, Kategorie ANPASSEN der Projekteinstellungen:

- ✗ PROGRAMMDATENBANK VERWENDEN setzen.

Auf der Seite LINKER, Kategorie DEBUG der Projekteinstellungen:

- ✗ Die Option DEBUG-INFO setzen.



Für DLLs ist das Feld AUSFÜHRBARES PROGRAMM FÜR DEBUG-SITZUNG auf der Seite DEBUG 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.

4.3 Befehle zur Programmausführung

Um herauszufinden, in welcher Zeile ein Programm eine Windows-Ausnahme auslöst, oder um während der Programmausführung die aktuellen Werte von Variablen einer bestimmten Quelltextstelle zu kontrollieren, muß man wissen,

- ✗ wie man ein Programm in den Debugger lädt,
- ✗ an einer bestimmten Stelle im Programm anhält und
- ✗ es gegebenenfalls schrittweise weiter ausführt.

4.3.1 Programm in Debugger laden und starten

Nachdem Sie Ihre Anwendung mit den Debug-Einstellungen kompiliert haben, können Sie die Anwendung im Debugger ausführen lassen, indem Sie einen der Einträge aus dem Untermenü ERSTELLEN/DEBUG STARTEN auswählen:

- ✗ 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.

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.

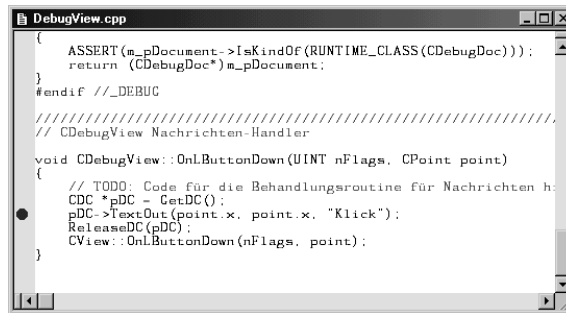
4.3.2 Programm anhalten

Einlaufendes Programm kann

- ✗ entweder mit Hilfe des Befehls DEBUG/ANHALTEN
- ✗ oder mit Hilfe von Haltepunkten angehalten werden.

Haltepunkte setzen

Bild 4.1:
Haltepunkt
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 roter Punkt links der Programmzeile zeigt an, daß in der Quelltextzeile ein Haltepunkt gesetzt wurde (siehe Abbildung 4.1). Kommt der Debugger bei der Ausführung des Programms zu dem Code, der dieser Quelltextzeile entspricht, hält er das Programm an und ermöglicht Ihnen die Begutachtung der Variablen des Programms (siehe Debug-Fenster).

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 (beispielsweise, daß ein Haltepunkt erst berücksichtigt werden soll, wenn die Schleifenvariable `loop = 50` ist). 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 stark 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 Nachrichtenbehandlungsmethoden der MFC angeben.

4.3.3 Programm schrittweise ausführen

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	↶ + F10	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.

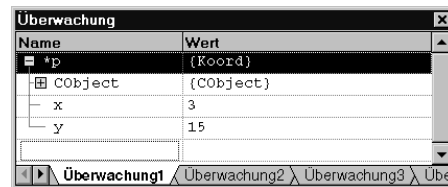
*Tabelle 4.1:
Befehle zur
schrittweisen
Ausführung*

4.4 Die Debug-Fenster

Die Debug-Fenster sind die Ausgabefenster des Debuggers. Hier kann man sich Informationen über den aktuellen Zustand des Programms (Werte von Variablen, Aufrufstack) anzeigen lassen. Alle Debug-Fenster können über das Menü ANSICHT aufgerufen werden. Einige Debug-Fenster werden standardmäßig beim Starten des Debuggers angezeigt.

Das Fenster Überwachung

Bild 4.2:
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 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

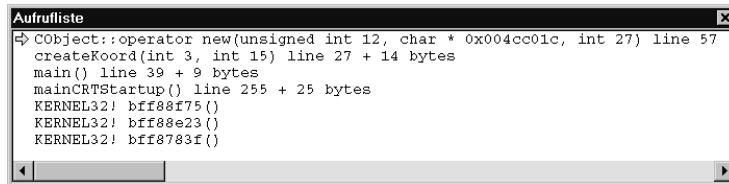


Bild 4.3:
Das Fenster
Aufrufliste

Dieses Fenster zeigt Ihnen an, welche Funktionen (bzw. Methoden) 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

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.

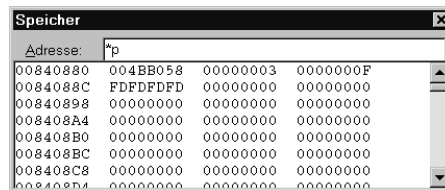
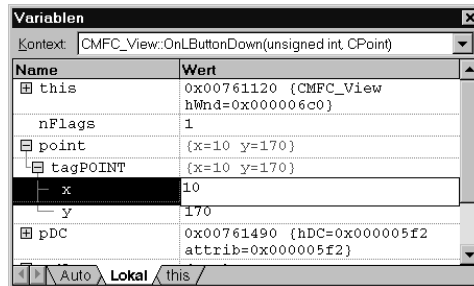


Bild 4.4:
Das Fenster
Speicher

- ✖ 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

Bild 4.5:
Das Fenster
Variablen



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

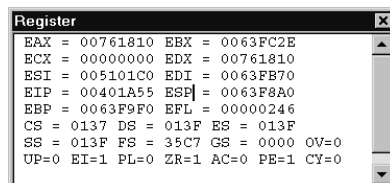
Das Fenster verfügt über drei verschiedene Seiten:

- ✖ 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

Bild 4.6:
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

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

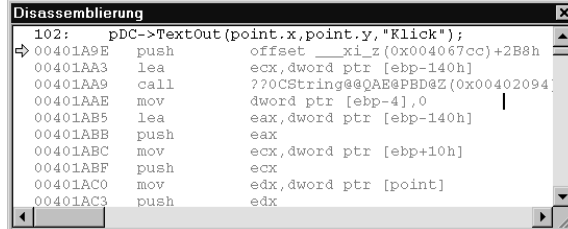


Bild 4.7:
Das Fenster
Disassemblierung

- ✗ Wenn Sie im Kontextmenü des Fensters den Befehl QUELLCODEANMERKUNG gesetzt haben, wird über den Assembleranweisungen der zugehörige Quelltext angezeigt.

4.5 Weitere Debug-Tools

Neben den oben vorgestellten klassischen Debug-Optionen stellen Ihnen IDE und Debugger noch drei weitere sehr interessante Debug-Tools zur Verfügung:

- ✗ Dateninfo
- ✗ Schnellüberwachung und
- ✗ Bearbeiten und Fortfahren

DatenInfo

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 4.8).

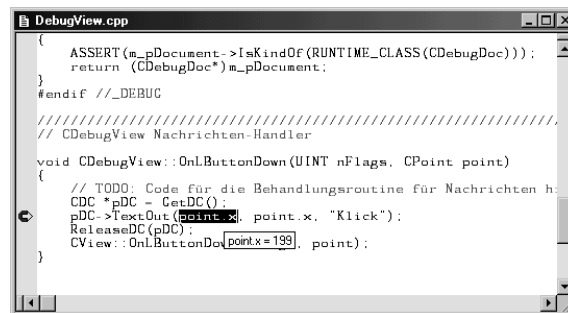
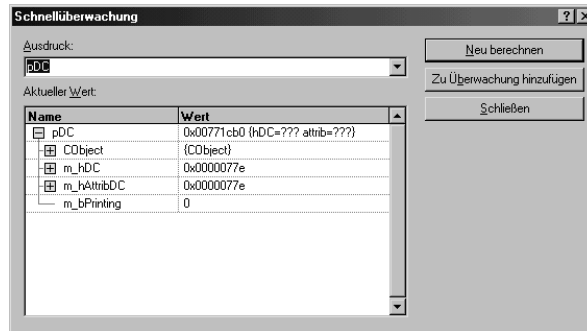


Bild 4.8:
DatenInfo im
Debug-Modus

Schnellüberwachung

Bild 4.9:
Schnellüberwachung



Genügen Ihnen die Informationen des DatenInfos nicht, können Sie den Dialog SCHNELLÜBERWACHUNG aufrufen (siehe Abbildung 4.9). 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.

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.

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.

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

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.**

4.6 Debug-Techniken

Je komplexer und umfangreicher ein Programm, desto aufwendiger ist das Debuggen desselbigen. Während man beispielsweise für einfache Programme noch theoretisch beweisen kann, daß sie stets korrekt arbeiten (allerdings sind diese Programme meist so simpel, daß sich der Nachweis nicht lohnt), sind umfangreichere Programme meist so komplex, daß der Beweis der Korrektheit nicht einmal mehr mit Hilfe von Supercomputern erbracht werden kann (weswegen Microsoft auch lieber auf Millionen von Beta-Testern vertraut). Daraus ergibt sich natürlich die Forderung, schon bei der Programmerstellung die Fehleranalyse zu berücksichtigen.

4.6.1 Debugfähigen Code erstellen

Sie können das Debuggen Ihrer Anwendungen wesentlich vereinfachen, wenn Sie den Code modular aufbauen.

Die Modularisierung eines Programms geschieht auf verschiedenen Ebenen:

- ✕ Code in Schleifen zusammenfassen
- ✕ Teilaufgaben als Funktionen implementieren und auslagern
- ✕ Objekte als Klassen implementieren (C++)
- ✕ Bibliotheken verwenden
- ✕ Programme in mehrere Quelltextdateien aufteilen

Der Vorteil der Modularisierung für das Debuggen liegt darin, daß die einzelnen Module (z.B. eine implementierte Funktion) für sich untersucht werden können.

Statt also ein komplexes Programm als Ganzes zu debuggen, checkt man zuerst die einzelnen, übersichtlicheren Module und danach das Zusammenspiel dieser Module im Programm – im Vertrauen darauf, daß die Module für sich genommen ja korrekt arbeiten (wovon man ja beispielsweise auch bei der Verwendung der Funktionen und Klassen der Laufzeitbibliotheken ausgeht).

4.6.2 Kritischen Code überprüfen

Fehler sind selten gleichmäßig über den Code eines Programms verteilt, vielmehr sind bestimmte Stellen eines Programms besonders anfällig.

- ✗ Schleifen sind stets darauf zu überprüfen, ob ihre Abbruchbedingungen irgendwann erfüllt werden. Nicht abbrechende Programmläufe weisen meist auf Endlosschleifen hin.
- ✗ Ebenso muß für Rekursionen sichergestellt sein, daß irgendwann die gewünschte Rekursionstiefe erreicht und die Rekursion nicht endlos fortgesetzt wird. Endlosrekursionen enden meist damit, daß der Stack vollgeschrieben ist.
- ✗ Der indizierte Zugriff auf Felder, sei es auf Elemente eines Arrays oder eines mit `malloc()` bzw. `new` eingerichteten Speicherblocks, ist besonders anfällig dafür, mit ungültigen Indizes auf Speicher außerhalb des vorgesehenen Speicherbereichs zuzugreifen.
- ✗ Zeiger sind eine häufige Fehlerquelle. Zeiger müssen mit einem entsprechenden Speicherraum verbunden oder auf `NULL` gesetzt werden. Zeiger, die auf keinen Speicherraum weisen, dürfen nicht dereferenziert (ausgewertet) werden.
- ✗ Eingaben müssen stets auf ihre Korrektheit überprüft werden. Dies gilt für die Eingabe seitens des Benutzers, der vielleicht »eins« statt »1« eintippt, ebenso wie für die Funktionsaufrufe innerhalb des Programms, wo eine Funktion beispielsweise »-1« oder »0« als Argument erhält, aber nur natürliche Zahlen verarbeiten kann.



Die MFC stellt zur Überprüfung boolescher Ausdrücke das `ASSERT`-Makro zur Verfügung, das nur in der Debug-Version des Programms (nicht in der Release-Version) ausgeführt wird und das eine Fehlermeldung ausgibt und die Programmausführung abbricht, wenn der Ausdruck innerhalb der Klammern zu Null ausgewertet wird.

4.6.3 Windows-Anwendungen debuggen

Beim Debuggen von Windows-Anwendungen sind zwei Besonderheiten zu beachten.

- ✗ Wegen der Nachrichtenverarbeitung und der damit verbundenen besonderen Ablaufstruktur von Windows-Anwendungen können diese nur in begrenztem Maße schrittweise ausgeführt werden. Insbesondere für

MFC-Anwendungen empfiehlt es sich zu Beginn des Debuggens, Haltepunkte in die interessierenden Nachrichtenbehandlungsmethoden zu setzen, um so bei Ausführung des Programms aus der Nachrichtenverarbeitung wieder in den Debugging-Modus zu kommen.

- ✗ **DEBUG/AUSNAHMEN.** In dem gleichnamigen Dialogfeld können Sie festlegen, im Falle welcher Systemausnahmen der Debugger anhalten soll.

4.7 Beispiel

Das folgende Beispiel, eine einfache Konsolenanwendung, soll eine Datei einlesen und verschlüsseln, indem die einzelnen Buchstaben durch Zeichen ersetzt werden, die ihnen im ASCII-Code um *shift* Positionen vorausgehen. Das Ergebnis wird in die Datei *codiert.txt* geschrieben.

```

1:  // Codierung.cpp
2:  //
3:  #include "stdafx.h"
4:  #include <stdio.h>
5:  #include <stdlib.h>
6:  #include <ctype.h>
7:
8:  int main(int argc, char* argv[])
9:  {
10:     FILE *fp_in, *fp_out;
11:     char zeichen, dateiname[100];
12:     int shift;
13:
14:     printf("Welche Datei soll codiert werden: \n");
15:     scanf("%s", dateiname);
16:     printf("Um wie viele Einheiten die Buchstaben \
           verruecken: \n");
17:     scanf("%d", shift);
18:
19:     if( (fp_in = fopen(dateiname, "rt")) == NULL)
20:     {
21:         printf("Fehler beim Öffnen der Datei\n");
22:         exit(0);
23:     }
24:
25:     if( (fp_out = fopen("Codiert.txt", "wt")) == NULL)
26:     {
27:         printf("Fehler beim Öffnen der Datei\n");

```



```

28:         exit(0);
29:     }
30:
31:     while((zeichen = fgetc(fp_in)) != EOF)
32:     {
33:         fputc(zeichen - shift, fp_out);
34:     }
35:
36:     fclose(fp_in);
37:     fclose(fp_out);
38:     return 0;
39: }

```

Übung 4-1: Beispiel debuggen

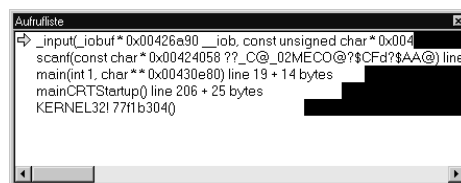
1. Führen Sie das Programm aus (**F5**). Kurz nach dem Einlesen des zweiten Parameters bricht das Programm mit einer allgemeinen Schutzverletzung ab.

Bild 4.10:
Es tritt eine
Ausnahme auf



2. Bestätigen Sie das Meldungsfenster. Wenn Ihre Visual C++-Version den Quelltext zur Laufzeitbibliothek parat hat, landen Sie danach im Quelltext der Datei *Input.c*. Andernfalls kann der Debugger nicht den verursachenden Quelltext anzeigen, was meist darauf hindeutet, daß der Fehler im Code einer der verwendeten Funktionen der Laufzeitbibliothek auftritt. Prinzipiell könnte dies bereits der zweite `scanf()`-Aufruf sein. Lassen Sie uns dies überprüfen.

Bild 4.11:
Die Aufrufliste



3. Rufen Sie die Aufrufliste auf (**Alt**+**7**). Dieser können Sie entnehmen, daß zuletzt die RTL-Funktion `scanf()` aufgerufen wurde. Unser Verdacht erhärtet sich.

5. Setzen Sie den Cursor in Zeile 17, und führen Sie das Programm bis dorthin im Debugger aus (**Strg** + **F10**).
6. Führen Sie die 17. Zeile aus (**F10**), und tippen Sie noch eine Zahl ein (beispielsweise 3). Die Schutzverletzung tritt wieder auf, unser Verdacht hat sich also bestätigt.
7. Betrachten wir den Funktionsaufruf. Der Fehler liegt darin, daß `scanf()` die Variable `shift`, statt der Adresse von `shift` übergeben wurde.
8. Beenden Sie das Programm, und korrigieren Sie die Zeile zu
`scanf("%d", &shift);`
9. Führen Sie das Programm erneut aus, und betrachten Sie die Ausgabe-datei, die vermutlich ziemlich chaotisch aussieht. Ist die Codierung etwa nicht korrekt?
10. Setzen Sie einen Haltepunkt in die Zeile 33. Setzen Sie den Cursor auf die Zeile, und führen Sie das Programm bis dorthin aus.
11. Wählen Sie die zu überwachenden Variablen. Rufen Sie den Befehl ANSICHT/DEBUG-FENSTER/ÜBERWACHUNG auf, und geben Sie im Feld NAME *zeichen* ein. Doppelklicken Sie in die neue Zeile, und geben Sie als neuen Ausdruck *zeichen - shift* ein.
11. Führen Sie das Programm schrittweise aus (**F10**). Die Codierung erfolgt korrekt.
12. Betrachten Sie noch einmal die Ausgabedatei. Wenn Sie diese in Visual C++ laden, wird sie als Binärdatei geöffnet, obwohl die Ausgabedatei im Textmodus geöffnet wurde. Dies könnte darauf zurückzuführen sein, daß der Text keine korrekten Zeilenumbrüche mehr enthält, da diese ja auch umcodiert wurden. Es liegt also ein logischer Fehler vor, denn wir codieren mehr als eigentlich erwünscht.
13. Ersetzen Sie daher abschließend die Zeile

```
fputc(zeichen - shift, fp_out);
```

durch

```
if(isprint(zeichen) && isprint(zeichen - shift))
    fputc(zeichen - shift, fp_out);
else
    fputc(zeichen, fp_out);
```

4.8 TRACE-Diagnosemakros

Wenn eine MFC-Anwendung 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("Rufe 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 Anwendung für den Debug-Vorgang kompiliert haben. Ihre Anwendung muß außerdem auch dann über den Debugger gestartet worden sein, wenn Sie keine anderen Debug-Features nutzen möchten.



Die `TRACE`-Makros werden nur in der Debug-Konfiguration berücksichtigt.



Für die Makros `TRACE0`, `TRACE1`, `TRACE2` und `TRACE3` werden weniger Debug-Informationen erzeugt.

4.9 Das ASSERT-Makro

Das Makro `ASSERT` unterbricht die Programmausführung, wenn eine bestimmte Bedingung falsch (`false`) ist. Dieses Makro kann in Debug-Versionen Ihrer Anwendung 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 `CMeinDoc`-Objekt zurückgibt, prüfen Sie den Zeiger wie folgt:

```
CMeinDoc *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.

Das `ASSERT`-Makro wird nur in der Debug-Konfiguration berücksichtigt.



4.10 Zusammenfassung

Der in Visual C++ integrierte Debugger wird ausgeführt, wenn Sie eine Anwendung über eine der Debug-Anweisungen im Menü `ERSTELLEN/DEBUG STARTEN` starten. Der Debugger verfügt über verschiedene Ansichtsfenster, über die Sie den Zustand Ihrer Anwendung überwachen können. Dazu zählen Editorfenster 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-Anwendungen, die mit dem Anwendungs-Assistenten erzeugt wurden, automatisch. Für solche Anwendungen erstellt der Anwendungs-Assistent 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.

Nützliche Helfer beim Debuggen von MFC-Anwendungen sind auch die `TRACE`- und `ASSERT`-Makros, die zum Beispiel vom Anwendungs-Assistenten verwendet werden.

4.11 Fragen

1. Was ist ein Bug?
2. Was ist ein Debugger?
3. Wie funktioniert ein Debugger?
4. Welches sind die wichtigsten Techniken beim Debuggen?
5. Wozu dienen die Makros ASSERT und TRACE?
6. Wie kann man die Ausgaben der Debug-Makros ASSERT und TRACE ausschalten?

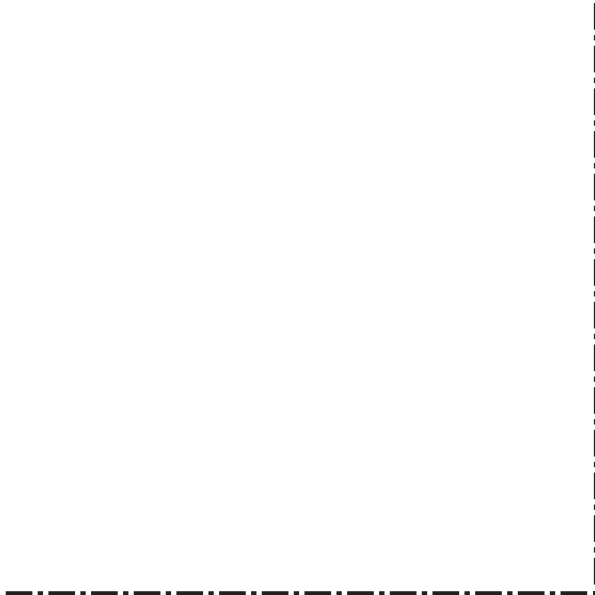
4.12 Aufgaben

1. Wenn Sie einen Kollegen oder Freund haben, der auch mit Visual C++ arbeitet, bitten Sie ihn doch, ein kleines Programm zu implementieren, das zwei bis drei Laufzeitfehler (auch logische Fehler) enthält. Setzen Sie ebenfalls ein solches Programm auf. Tauschen Sie Ihre Programme aus, und debuggen Sie das Programm Ihres Bekannten.



TEIL 2

**Windows-
Programmierung
mit der MFC**



Das MFC-Anwendungs- gerüst

Mit diesem Kapitel steigen wir jetzt endlich richtig in die Windows-Programmierung ein. Eine aufregende Zeit liegt vor Ihnen. Sie werden nicht nur Ihre Fähigkeiten als Programmierer erweitern, sondern Sie werden auch die Gelegenheit erhalten, Windows von einer ganz anderen Seite kennenzulernen. Bisher dürften Sie Windows nämlich nur aus der Perspektive des Anwenders kennen. Das »nur« ist dabei nicht abschätzig gemeint – im Gegenteil! Nur wer mit der typischen Benutzeroberfläche und der Arbeitsweise von Windows-Programmen vertraut ist, ist in der Lage, bedienungsfreundliche Programme für Windows-Anwender zu konzipieren und zu implementieren. Doch nur mit diesem Wissen allein kann man keine Windows-Programme erstellen. Man muß sich auch mit den Interna von Windows auskennen, mit den Möglichkeiten und Eigenheiten des Betriebssystems und mit den Anforderungen, die Windows an Programme und Programmierer stellt. Wir werden uns also nicht nur mit der MFC und verschiedenen Programmiertechniken beschäftigen, sondern auch einen Blick hinter die Kulissen von Windows werfen.

Doch wieder einmal, ein allerletztes Mal, muß ich Sie vertrösten. Bevor wir ausziehen, die Windows-Programmierung zu lernen, müssen wir uns noch einmal hinsetzen und geduldig unsere Hausaufgaben machen: Das Anwendungsgerüst, das der MFC-Anwendungs-Assistent für uns erstellt und das wir schon in den vorangehenden Kapitel kennengelernt haben, muß noch besprochen werden. Keine einfache Aufgabe, denn das vom Assistenten generierte Anwendungsgerüst ist selbst für einfache Programme bereits



äußerst komplex. Um es bis ins Detail zu erklären, müßte man weit ausholen und auf einmal alles lehren, was in diesem Buch steht. Ein ganz unmögliches Unterfangen (wäre es möglich, hätte ich mir das Schreiben dieses Buches sparen können). Wenn wir uns also auf den folgenden Seiten etwas eingehender mit dem Anwendungsgerüst beschäftigen, es sezieren und analysieren, dann geschieht dies in der Hoffnung, etwas über den Aufbau des Programmgerüsts, seine Funktionsweise und etwaige Ansatzpunkte zur Erweiterung des Codes zu erfahren. Detailkenntnisse interessieren uns nicht – diese kommen von selbst, wenn wir im Laufe unserer täglichen Programmierarbeit mit dem Programmgerüst immer vertrauter werden.

Sie lernen in diesem Kapitel:

- ✗ Noch einmal, wie man Programme mit dem MFC-Anwendungs-Assistenten beginnt
- ✗ Aus welchen Dateien das Anwendungsgerüst besteht
- ✗ Wie das Anwendungsgerüst aufgebaut ist
- ✗ Wie man das Anwendungsgerüst um eine einfache Textausgabe erweitern kann

5.1 Das Projekt

Wir beginnen damit, ein Projekt für eine Hello World-Anwendung anzulegen. Da es unsere dritte Version dieses Programms ist, nennen wir das Projekt *Hello_W3*.

1. **Rufen Sie Visual C++ auf.** Falls nach dem Programmstart irgendwelche Quelldateien angezeigt werden oder bereits ein Arbeitsbereich geöffnet sein sollte, schließen Sie diese.

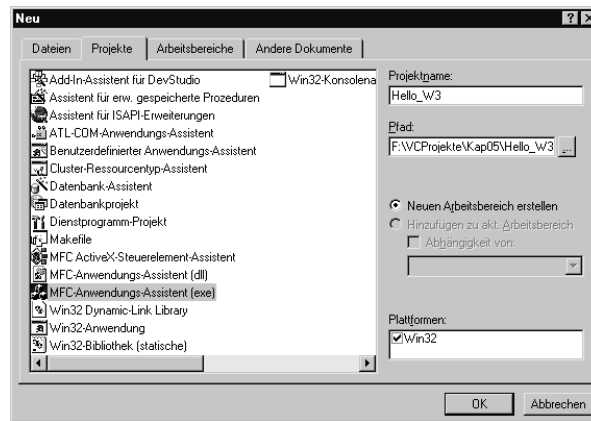


Bild 5.1:
Aufruf des
Anwendungs-
Assistenten

2. **Legen Sie einen Arbeitsbereich und ein Projekt an.** Rufen Sie dazu den Befehl DATEI/NEU auf, und lassen Sie die Seite PROJEKTE anzeigen.

Geben Sie einen NAMEN für das Projekt ein, hier *Hello_W3*, und wählen Sie im Feld PFAD ein übergeordnetes Verzeichnis für das Projekt aus.

Im linken Teil des Dialogfeldes wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN (EXE) aus.

Drücken Sie zuletzt auf OK.

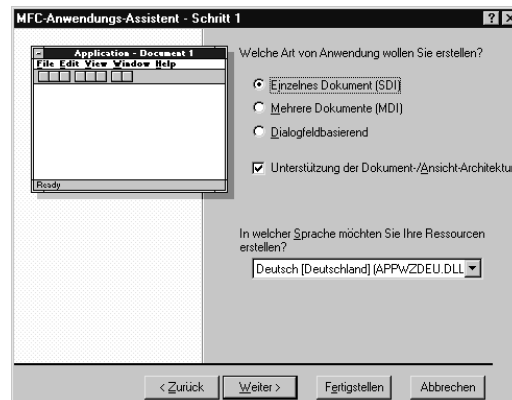


Bild 5.2:
Konfiguration
des Projekts
im Anwen-
dungs-Assi-
stenten

3. Es erscheint das Dialogfeld des Anwendungs-Assistenten, in dem Sie die Option EINZELNES DOKUMENT (SDI) auswählen.

Auf der vierten Seite des Anwendungs-Assistenten deaktivieren Sie die Optionen für SYMBOLLEISTE, STATUSLEISTE und DRUCKEN und geben im Eingabefeld für die LISTE DER ZULETZT VERWENDETEN DATEIEN 0 ein. Danach drücken Sie auf FERTIGSTELLEN.

Es erscheint noch ein abschließendes Kontrollfenster, das Sie mit OK bestätigen.

Warum SDI, warum Dokument/Ansicht-Architektur?

SDI, MDI SDI-Grundgerüste sind universeller einsetzbar als MDI-Gerüste. Beide Grundgerüste, SDI wie MDI, sind vorrangig für Dateibearbeitungsprogramme konzipiert – Programme, die den Inhalt von Dateien laden (Grafiken, Texte, Sounddateien), sie bearbeiten und wieder abspeichern. SDI-Programme erlauben dabei jeweils nur das Laden und Bearbeiten einer einzigen Datei zur gleichen Zeit. Wenn der Anwender eine zweite Datei lädt, wird die aktuelle Datei gespeichert und geschlossen. MDI-Programme erlauben dagegen die gleichzeitige Bearbeitung mehrerer Dateien. Die einzelnen Dateien werden in eigenen Bearbeitungsfenstern geöffnet, die innerhalb des Rahmens des Hauptfensters des Programms verschoben werden können. Zur Auswahl und Anordnung der Bearbeitungsfenster verfügen MDI-Programme üblicherweise über ein FENSTER-Menü mit entsprechenden Befehlen.

Ob man die Windows-Programmierung, beispielsweise die Behandlung der Maus oder die Ausgabe von Text und Grafiken, anhand von SDI- oder MDI-Programmen lehrt, ist im Grunde gleich. SDI-Programme haben jedoch den Vorteil, daß man sich nicht zusätzlich mit der Unterstützung mehrerer Bearbeitungsfenster abmühen muß und daß sie universeller einsetzbar sind. Schließlich dienen nicht alle Programme der Bearbeitung von Dateien. Für solche Programme läßt sich das SDI-Grundgerüst leichter anpassen als das MDI-Grundgerüst.

Doc/View Mit der Unterstützung des Doc/View-Modells (Dokument/Ansicht-Architektur) verhält es sich genau umgekehrt: Das Grundgerüst wird durch das Doc/View-Modell eher komplizierter als einfacher, und eindeutige Vorteile bringt das Modell auch nicht immer. Daß ich Sie dennoch von Anfang an mit dem Doc/View-Modell vertraut machen möchte, hat folgende Gründe:

- ✗ Je komplexer und professioneller Ihre Anwendungen mit der Zeit werden, um so mehr zahlt sich die Verwendung von Doc/View für Sie aus.

- ✗ Es fällt wesentlich schwerer, sich nachträglich an das Doc/View-Modell zu gewöhnen, als sich mit der Programmierung ohne Doc/View vertraut zu machen.
- ✗ Doc/View besitzt in Visual C++ eine langjährige Tradition, da bis zur aktuellen 6.0-Version alle SDI- und MDI-Anwendungen vom Assistenten automatisch unter Verwendung von Doc/View eingerichtet wurden. Wenn Sie Programme anderer Visual C++-Programmierer studieren oder Bücher zu Visual C++ lesen, werden Sie also fast immer mit Doc/View konfrontiert werden. Zudem zieht sich die Doc/View-Unterstützung durch die gesamte MFC.
- ✗ Doc/View schadet nicht. Auch wenn Ihr Programm genauso gut ohne Doc/View implementiert werden könnte, schadet es nichts, auf einem Doc/View-Gerüst aufzubauen. Die erzeugte EXE-Datei ist vielleicht etwas größer, und in Einzelfällen ist der Quelltext eher ein wenig schlechter als besser zu warten, doch sind diese Unterschiede keineswegs so gravierend, daß man unbedingt auf Doc/View verzichten müßte.

Zur Erinnerung: Doc/View ist kein substantielles Element einer Windows-Anwendung und auch keine spezielle Programmiertechnik, die uns eine neue Funktionalität erschließt (wie zum Beispiel OLE/ActiveX oder die Programmierung von DLLs). Doc/View bezeichnet lediglich ein Modell zur Code-Organisation, das die Trennung von Datenverwaltung und Datenanzeige propagiert. Die MFC stellt zur Unterstützung dieses Modells eine Reihe von speziellen Klassen und Methoden zur Verfügung.



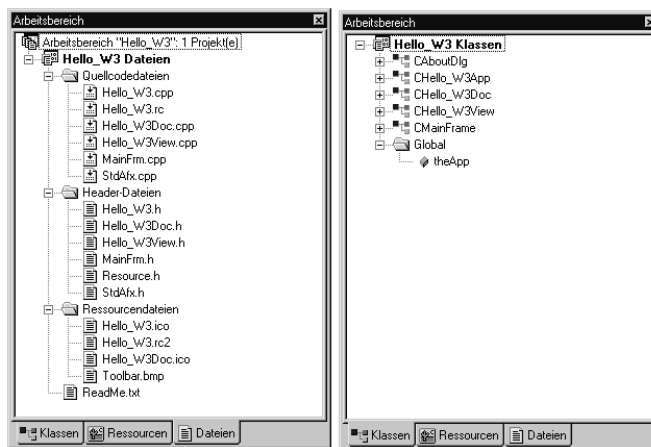
Wer lieber ohne Doc/View und Anwendungs-Assistent auskommen möchte, der findet in den Aufgaben dieses und der nachfolgenden Kapitel bis einschließlich Kapitel 8 Hinweise und Beispiele für die Erstellung und Bearbeitung eigener Anwendungsgerüste. Die Techniken zur Anzeige von Dialogfenstern, Text und Grafiken ist von der Verwendung von Doc/View weitgehend unabhängig.



5.2 Die Dateien und Klassen des Anwendungsgerüsts

Wechseln Sie jetzt im Arbeitsbereichfenster zur Seite DATEIEN, expandieren Sie die übergeordneten Knoten, und betrachten Sie sich die einzelnen vom Assistenten erzeugten Dateien.¹ Vergleichen Sie die Quelltextdateien mit den in den Dateien deklarierten Klassen (Seite KLASSEN).

Bild 5.3:
Dateien und
Klassen des
Anwendungs-
gerüsts



Wie man unschwer erkennen kann, gibt es eine Beziehung zwischen Dateien und Klassen. Dies liegt einfach daran, daß der Anwendungs-Assistent zum Aufbau des Anwendungsgerüsts vier Klassen erzeugen muß (die Klasse `CAboutDlg`, die den INFO-Dialog enthält, ist eine nette Zugabe, die mit dem Funktionieren des Anwendungsgerüsts nichts zu tun hat und die ich daher einfach unterschlage) und für jede dieser Klassen eine Header-Datei und eine Quelltextdatei erzeugt. In der Header-Datei wird die Klasse deklariert, in der Quelltextdatei werden die Methoden der Klasse definiert. Der Assistent hat den von ihm erzeugten Code also auf vorbildliche Weise modularisiert.

Schauen wir uns einmal an, welche grundlegenden Bestandteile ein MFC-Anwendungsgerüst enthalten muß:

¹ Welche Dateien im Einzelfall angelegt werden, hängt natürlich von den Einstellungen im Dialogfeld des Anwendungs-Assistenten ab – ob Sie beispielsweise eine SDI- oder eine MDI-Anwendung erstellen ließen oder ob Sie das Doc/View-Modell verwenden.

5.2.1 CHello_W3App – das Anwendungsobjekt

Zuerst benötigt man ein Objekt, das geeignet ist, die Anwendung zu repräsentieren. Die MFC definiert hierfür die Klasse `CWinApp`. Da es zur Anpassung der Anwendung an unsere Vorstellungen nötig ist, einige Methoden der Klasse `CWinApp` zu überschreiben, hat der Anwendungs-Assistent eine eigene Klasse namens `CHello_W3App` von `CWinApp` abgeleitet.

In der Datei `Hello_W3.cpp` sind nicht nur die überschriebenen Methoden implementiert, hier wird auch ein globales Objekt der Anwendungsklasse erzeugt: `theApp`. Im Quelltext unseres Projekts wird dieses Objekt nicht weiter verwendet, doch fügt Visual C++ unserem MFC-Programm noch zusätzlichen Startcode hinzu, aus dem auf dieses Objekt zugegriffen wird.

Die wichtigsten Aufgaben des Anwendungsobjekts sind die Erzeugung des Hauptfensters der Anwendung und die Kommunikation mit Windows.

5.2.2 CMainFrame – das Hauptfenster der Anwendung

Bis auf wenige Ausnahmen verfügt jede Windows-Anwendung über ein sichtbares Hauptfenster – sonst würde der Anwender ja nichts von der Anwendung zu sehen bekommen. Ein Hauptfenster kann ein Dialogfenster (siehe Übung aus Kapitel 2) oder ein sogenanntes Rahmenfenster sein.

Rahmenfenster sind Fenster mit einem speziellen Rahmen, in den bestimmte Dekorationselemente (Menü, Symbolleisten und Statusleiste) integriert werden können. Der innere Bereich des Fensters, der nach Abzug der Dekorationselemente frei bleibt, ist der sogenannte **Client-Bereich**.



Die Klasse `CMainFrame` ist von der MFC-Klasse `CFrameWnd` abgeleitet. Um nun ein Rahmenfenster zu erzeugen, es mit der Anwendung zu verbinden und anzuzeigen, sind im Prinzip folgende Schritte notwendig:

1. Um das Rahmenfenster zu erzeugen, wird ein Objekt unserer abgeleiteten Klasse `CMainFrame` instantiiert:

```
CMainFrame *pMainWnd = new CMainFrame;
```

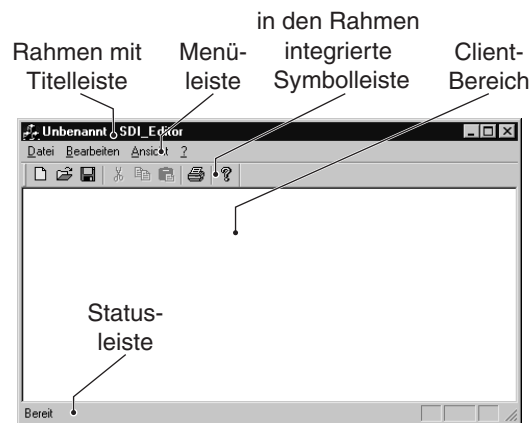
2. Um das Rahmenfenster mit der Anwendung zu verbinden, wird der Zeiger auf das Fensterobjekt an die globale MFC-Variable `m_pMainWnd` übergeben:

```
m_pMainWnd = pMainWnd;
```

3. Um das Rahmenfenster anzeigen zu lassen, werden die Methoden `ShowWindow()` und `UpdateWindow()` aufgerufen:

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

Bild 5.4:
Rahmenfenster mit Deko-
rationen und
Client-Bereich



Doch in dem Quelltext unseres Programms werden Sie diese Anweisungen so nicht wiederfinden. Dies liegt an der Doc/View-Unterstützung der MFC. Diese arbeitet mit sogenannten Dokumentvorlagen. Was Dokumentvorlagen sind, braucht uns im Moment nicht zu interessieren, wichtig ist lediglich, daß bei der Einrichtung der Dokumentvorlage auch das Rahmenfenster eingerichtet wird.

Expandieren Sie einmal in der KLASSEN-Ansicht des Arbeitsbereichsfensters den Knoten der Klasse `CHello_W3App`, und doppelklicken Sie auf die Methode `InitInstance()`. Sofort erscheint im Editor die Definition der Methode. Scrollen Sie nun nach unten bis zu den Zeilen:

```
// Dokumentvorlagen der Anwendung registrieren.
// Dokumentvorlagen dienen als Verbindung zwischen
// Dokumenten, Rahmenfenstern und Ansichten.
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CHello_W3Doc),
    RUNTIME_CLASS(CMainFrame), // Haupt-SDI-Rahmenfenster
    RUNTIME_CLASS(CHello_W3View));
AddDocTemplate(pDocTemplate);
```

Hier wird mit Hilfe des `new`-Operators die Dokumentvorlage erzeugt; dahinter verbirgt sich auch die Erzeugung und Einrichtung des Hauptfensters. Naja, man sieht dies dem Code nicht unbedingt an, aber Sie können es mir glauben.

Wenn Sie weiter nach unten zum Ende der Methode scrollen, sehen Sie dann auch die Aufrufe der Methoden `ShowWindow()` und `UpdateWindow()`, die das Rahmenfenster auf den Bildschirm zaubern und sichtbar machen.

5.2.3 CHello_W3View – das Ansichtsfenster der Anwendung

Mittlerweile verfügt unsere Anwendung also über ein Rahmenfenster. Wozu brauchen wir dann noch ein Ansichtsfenster, und wo und wann soll dieses Fenster angezeigt werden?

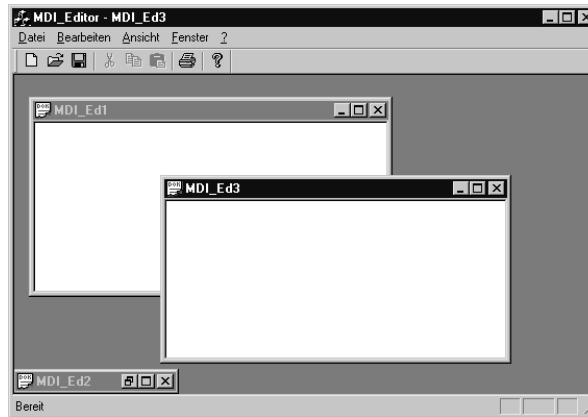
Das Ansichtsfenster benötigen wir, um dem Anwender etwas anzuzeigen: eine Meldung, den Text aus einer Datei, eine Grafik. Gegen Ende dieses Kapitels werden wir beispielsweise unser Programm dahingehend erweitern, daß dem Anwender der Text »Hello World!« angezeigt wird.

Zu Recht werden Sie jetzt sagen, daß dies immer noch nicht erklärt, wozu wir ein Ansichtsfenster benötigen. Schließlich haben wir ja das Rahmenfenster. »Warum zeigen wir den Text nicht einfach im Client-Bereich des Rahmenfensters an?«, werden Sie fragen.

Nun ja, das ginge schon – aber man macht es nicht so. Es ist einfach nicht üblich, direkt ins Rahmenfenster zu zeichnen. Statt dessen erzeugt man ein weiteres, untergeordnetes Fensterobjekt und fügt dies in den Client-Bereich des Rahmenfensters ein. Der Anwender merkt davon gar nichts: Er kann nicht unterscheiden, ob der Client-Bereich des Hauptfensters von einem Ansichtsfenster ausgefüllt wird oder nicht. Bei MDI-Editoren wie Word 95 liegt der Fall etwas anders. Da bei diesen Programmen in den Rahmenfenstern mehrere Dokumentfenster verwaltet werden müssen, werden diese Dokumentfenster mit eigenem Rahmen und Titelleiste ausgestattet, und hinter diesen Dokumentfenstern wird der graue Client-Bereich des Rahmenfensters sichtbar.

Nur wenn der Anwender auf die Fensterschaltfläche zum Maximieren eines Dokumentfensters klickt, wird das Dokumentfenster in den Rahmen des Hauptfensters eingepaßt und füllt den Client-Bereich aus – dann sieht der MDI-Editor wie ein SDI-Editor aus.

Bild 5.5:
Rahmen-
fenster und
Dokument-
fenster in
einem MDI-
Editor



Zwei Punkte sprechen für die Einrichtung eines eigenen Ansichtsfensters:

- ✘ Erstens erzielt man eine klare Aufgabenteilung: Das Rahmenfenster ist die Hauptschnittstelle der Anwendung. Es enthält die Menüleiste mit den Befehlen zur Steuerung der Anwendung. Und wenn man das Hauptfenster schließt, wird auch die Anwendung automatisch beendet. Dem Ansichtsfenster obliegt demgegenüber die Anzeige von Programmmuständen und Informationen.
- ✘ Zweitens gehört das Ansichtsfensterobjekt einer eigenen Klasse an, die uns weitere interessante Methoden zur Verfügung stellt. Tatsächlich kann man in der MFC sogar unter mehreren Ansichtsfensterklassen auswählen und sich die Fensterklasse für das Ansichtsfenster aussuchen, die in Hinblick auf die Funktion der Anwendung die meiste Unterstützung bietet (beispielsweise `CEditView` oder `CRichEditView` für Texteditoren, siehe Kapitel 11).

Zum Schluß bleibt noch die Frage, wie man es schafft, daß das Ansichtsfenster sich genau in den Client-Bereich des Rahmenfensters einpaßt?

Man übergibt ihm bei der Erzeugung einen Zeiger auf das übergeordnete Rahmenfenster. Doch in unserem Doc/View-Beispiel sieht man davon wieder einmal nichts, denn die Einrichtung des Ansichtsfensters und seine Verknüpfung mit dem Rahmenfenster ist ebenfalls in der Erzeugung der `CSingleDocTemplate`-Dokumentvorlage enthalten.

5.2.4 CHello_W3Doc – die Dokumentklasse

Unsere Klasse `CHello_W3Doc` ist von der MFC-Klasse `CDocument` abgeleitet. Sie unterstützt den Befehl `DATEI/NEU` sowie das Laden und Speichern von Daten aus und in Dateien.

Wie nicht anders zu erwarten, wird auch das Dokumentobjekt zusammen mit der Dokumentvorlage eingerichtet.

Erkundung des Quelltextes

Bevor wir noch einmal zusammenfassen, wie Anwendung, Rahmenfenster und Ansichtsfenster beim Start der Anwendung erzeugt und zusammengesetzt werden, sollten Sie sich selbst ein wenig mit dem Quelltext vertraut machen. Nutzen Sie dazu die Klassen-Ansicht, und gehen Sie Klasse für Klasse vor.

- ✗ Wenn Sie auf den Knoten einer Klasse doppelklicken, springt der Editor in die zugehörige Klassendeklaration.
- ✗ Wenn Sie den Knoten einer Klasse expandieren, werden darunter die in der Klasse deklarierten Methoden aufgeführt. Dies verschafft Ihnen eine gute Übersicht.
- ✗ Durch Doppelklick auf den Namen einer Methode springen Sie direkt in die Definition der Methode.

Wenn Sie die Klassen-Ansicht das erste Mal öffnen, kann es ein wenig dauern, bis die einzelnen Klassen angezeigt werden. Dies liegt daran, daß die IDE die Klassen-Ansicht direkt aus dem Quelltext aufbaut (und nicht aus Informationen, die beim Anlegen des Projekts gesammelt wurden). Der Vorteil dieses Verfahrens ist, daß die Klassen-Ansicht automatisch aktualisiert wird, wenn Sie Ihren Quelltext ändern. Wenn Sie also in einer Klasse neue Methoden deklarieren oder Methoden löschen, spiegelt sich dies wenige Augenblicke später in der KLASSEN-Ansicht wider.



5.3 Was geschieht beim Starten der Anwendung?

Von der C-Programmierung sind Sie gewohnt, daß jede Anwendung mit der Haupteintrittsfunktion `main()` beginnt. Windows-Programme verwenden dagegen eine Eintrittsfunktion namens `WinMain()`, die beim Starten der Anwendung vom Windows-Betriebssystem aufgerufen wird. In dieser Funktion wird dann üblicherweise das Hauptfenster der Anwendung erzeugt. In MFC-Programmen ruft diese Funktion dagegen die globale MFC-Funktion `AfxWinMain()` auf, und erst in dieser Funktion wird die Verbindung zum Quellcode unseres Programms hergestellt. Von all dem merken Sie nichts, denn Sie bekommen diesen Startcode üblicherweise gar nicht zu Gesicht.

Trotzdem ist es ganz interessant zu wissen, wo unsere Anwendung eigentlich beginnt.

In `AfxWinMain()` geschehen drei wichtige Dinge:

1. Die Funktion besorgt sich einen Zeiger auf unser Anwendungsobjekt. Sie erinnern sich? Das Anwendungsobjekt wird als globales Objekt in der Datei *Hello_W3.cpp* deklariert.

```
// Hello_W3.cpp
//
#include "stdafx.h"
#include "Hello_W3.h"
...
////////////////////////////////////
//
// Das einzige CHello_W3App-Objekt
CHello_W3App theApp;
...
```

2. Die Funktion ruft die Methode `InitInstance()` des Anwendungsobjekts auf. Diese Methode werden wir uns gleich etwas genauer anschauen.
3. Die Funktion ruft die Methode `run()` des Anwendungsobjekts auf. Die `run()`-Methode ist im wesentlichen eine Endlosschleife, in der die Anwendung abfragt, ob irgendwelche Anwenderaktionen vorliegen (Aufruf eines Menübefehls, Klick mit der Maus etc.), und wenn ja, eine passende Methode aufruft zur Beantwortung des Ereignisses.

Okay, die Eintrittsfunktion greift auf unser Anwendungsobjekt zu und ruft dessen `InitInstance()`-Methode auf. Die wichtigste Aufgabe dieser Methode ist es, das Hauptfenster der Anwendung zu erzeugen. Dazu wurde die Methode in der Klasse `CHello_W3App` überschrieben:

```
1: // Hello_W3.cpp
2: //
3: #include "stdafx.h"
4: #include "Hello_W3.h"
5: ...
6:
7: //////////////////////////////////////
8: // CHello_W3App Initialisierung
9:
10: BOOL CHello_W3App::InitInstance()
11: {
12:     AfxEnableControlContainer();
```

```

13:
14:     // Standardinitialisierung
15:     // Wenn Sie diese Funktionen nicht nutzen und die
16:     // Größe Ihrer fertigen ausführbaren Datei reduzieren
17:     // wollen, sollten Sie die nachfolgenden spezifischen
18:     // Initialisierungsroutinen entfernen.
19: #ifdef _AFXDLL
20:     Enable3dControls(); // Diese Funktion bei Verwendung
21:                        // von MFC in gemeinsam genutzten
22:                        // DLLs aufrufen
23: #else
24:     Enable3dControlsStatic(); // Diese Funktion bei
25:                             // statischen MFC-Anbindungen
26:                             // aufrufen
27: #endif
28: // Ändern des Registrierungsschlüssels, unter dem
29: // unsere Einstellungen gespeichert sind.
30: // ZU ERLEDIGEN: Sie sollten dieser Zeichenfolge einen
31: // geeigneten Inhalt geben wie z.B. den Namen Ihrer
32: // Firma oder Organisation.
33:     SetRegistryKey(_T("Local AppWizard-Generated \
        Applications"));
34:
35:     LoadStdProfileSettings(0); // Standard INI-
36:                             // Dateioptionen laden
37:                             // (einschließlich MRU)
38:
39:     // Dokumentvorlagen der Anwendung registrieren.
40:     // Dokumentvorlagen dienen als Verbindung zwischen
41:     // Dokumenten, Rahmenfenstern und Ansichten.
42:
43:     CSingleDocTemplate* pDocTemplate;
44:     pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CHello_W3Doc),
        RUNTIME_CLASS(CMainFrame),
        RUNTIME_CLASS(CHello_W3View));
45:     AddDocTemplate(pDocTemplate);
46:     // Befehlszeile parsen, um zu prüfen auf Standard-
47:     // Umgebungsbefehle DDE, Datei öffnen
48:     CCommandLineInfo cmdInfo;
49:     ParseCommandLine(cmdInfo);

```

```

50:    // Verteilung der in der Befehlszeile angegebenen
51:    // Befehle
52:    if (!ProcessShellCommand(cmdInfo))
53:        return FALSE;
54:    // Das einzige Fenster ist initialisiert und kann
55:    // jetzt angezeigt und aktualisiert werden.
56:    m_pMainWnd->ShowWindow(SW_SHOW);
57:    m_pMainWnd->UpdateWindow();
58:
59:    return TRUE;
60: }

```

In den Zeilen 12 bis 35 werden zuerst einige interne Initialisierungen ausgeführt, die mit dem Aufbau der Anwendung und der Einrichtung des Hauptfensters wenig zu tun haben.

Danach folgt in Zeile 43 die Erzeugung der Dokumentvorlage. Wie Sie bereits aus den vorangehenden Ausführungen wissen, verbirgt sich hinter diesem Aufruf auch die Einrichtung des Haupt- und des Ansichtsfensters. Doch wir sehen nur, daß dem Konstruktor von `CSingleDocTemplate` die Ressourcen-ID für die Anwendungsressourcen (Menü, Tastaturkürzel, Symbolleiste, Anwendungssymbol, Titel-String) sowie die Klassen für das Dokument, das Rahmenfenster und das Client-Fenster übergeben werden, und wir können lediglich ahnen, daß in dem Konstruktor von `CSingleDocTemplate` Objekte dieser Klassen erzeugt und die Fenster eingerichtet werden. (Die entsprechenden Anweisungen stehen allerdings nicht direkt im Konstruktorcode von `CSingleDocTemplate`, sondern sind unter etlichen Methodenaufrufen verborgen.)

Die Erzeugung von Klassenobjekten ist immer mit einem Aufruf des Konstruktors der Klasse verbunden. Schauen wir uns also einmal den Code für den Konstruktor unserer `CMainFrame`-Klasse an:

```

1:  // MainFrm.cpp : Implementierung der Klasse CMainFrame
2:  //
3:  #include "stdafx.h"
4:  #include "Hello_W3.h"
5:  ...
6:
7:
8:  //////////////////////////////////////
9:  // CMainFrame Konstruktion/Zerstörung
10: CMainFrame::CMainFrame()

```

```

11: {
12:     // ZU ERLEDIGEN: Hier Code zur Member-Initialisierung
13:
14: }
15:
16: CMainFrame::~CMainFrame()
17: {
18: }
19:
20: BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
21: {
22:     if( !CFrameWnd::PreCreateWindow(cs) )
23:         return FALSE;
24:     // ZU ERLEDIGEN: Ändern Sie hier die Fensterklasse
25:     // oder das Erscheinungsbild, indem Sie
26:     // CREATESTRUCT cs modifizieren.
27:
28:     return TRUE;
29: }

```

Erstaunlicherweise geschieht in diesem Konstruktor nichts. Dabei hätte man doch erwarten können, daß hier wenigstens irgendwelche Anweisungen stehen, die festlegen, wie groß das Rahmenfenster sein soll, oder die das Menü der Anwendung mit dem Rahmenfenster verbinden.

Wenn Sie sich den Konstruktor der Ansichtsklasse `CHello_W3View` ansehen, haben Sie das gleiche Bild: ein leerer Konstruktor und das, obwohl wir doch bereits wissen, daß dem Fenster irgendwo mitgeteilt werden muß, daß ein übergeordnetes Fenster existiert, in dessen Client-Bereich es sich einfügen soll.

Die Antwort darauf ist, daß die entsprechenden Anweisungen nicht im Konstruktor, sondern im Zuge der Einrichtung der Dokumentvorlage vorgenommen werden. Da dieser Code aber tief in der MFC verborgen ist, stellt sich die Frage, wie man auf die Darstellung des Rahmenfensters Einfluß nehmen kann. Wie kann man zum Beispiel Breite und Höhe des Fensters festlegen? Einen ersten Hinweis darauf gibt die Methode `PreCreateWindow()`, die der Anwendungs-Assistent bereits für uns überschrieben hat (Zeile 20–29). Im nächsten Kapitel werden wir dieser Frage nachgehen.

Kehren wir zurück zur Methode `InitInstance()`.

```

46:     // Befehlszeile parsen, um zu prüfen auf Standard-
47:     // Umgebungsbefehle DDE, Datei offen
48:     CCommandLineInfo cmdInfo;

```

```

49:     ParseCommandLine(cmdInfo);
50:     // Verteilung der in der Befehlszeile angegebenen
51:     // Befehle
52:     if (!ProcessShellCommand(cmdInfo))
53:         return FALSE;
54:     // Das einzige Fenster ist initialisiert und kann
55:     // jetzt angezeigt und aktualisiert werden.
56:     m_pMainWnd->ShowWindow(SW_SHOW);
57:     m_pMainWnd->UpdateWindow();
58:
59:     return TRUE;
60: }

```

Nach der Erzeugung der Dokumentvorlage besitzt die Anwendung also bereits ein Hauptfenster mit integriertem Ansichtsfenster.

Als nächstes prüft die Methode, ob der Anwendung beim Aufruf Kommandozeilenargumente übergeben wurden (Zeilen 48 bis 53). Sie kennen dies vermutlich noch von DOS her. Wie man diese Argumente auswertet, erfahren Sie ebenfalls im nächsten Kapitel.

Zu guter Letzt wird das Hauptfenster angezeigt.

5.4 Spezielle Makros und Konventionen der MFC

Beim Durchsehen des vom Anwendungs-Assistenten generierten Quelltextes fallen einige Besonderheiten auf, die für Programmierer, die noch nie mit Visual C++ gearbeitet haben, recht ungewohnt sein dürften. Dies sind

- ✗ die Benennung der Variablen
- ✗ die vielen MFC-Makros

5.4.1 Die ungarische Notation

Die Namensgebung für Variablen folgt in Visual C++ der Ungarischen Notation. Diese Ungarische Notation, die auf den Microsoft-Programmierer Charles Simonyi zurückgeht sieht vor, daß jeder Variablen ein Präfix aus einem oder mehreren Kleinbuchstaben vorangestellt wird, welches den Typ der Variablen kennzeichnet. Nach diesem Präfix folgt dann der eigentliche Name der Variablen, der zur Absetzung vom Präfix mit einem Großbuchstaben beginnt.

Präfix	Datentyp	Beispiel
a	Array	aFeld
b	Boolean	bVorhanden
by	Byte, unsigned char	byFlag
ch	char	chZeichen
s	String	sText
d	double	dWert
h	Handle	hWnd
i	int (Index)	iZaehler
lp	far-Zeiger	lpPtr (veraltet, für 32 Bit sind alle Zeiger near)
n	int (Wert)	nSumme
p	Zeiger	pszString (Zeiger auf NULL-terminierten String)
s	String	sName
sz	ASCII-Z-String	sName (NULL-terminierter String)
u	unsigned int	uMax
v	void	vPointer

Tabelle 5.1:
Konventionen
der Ungari-
schen Notation

Weitere Präfixe sind:

Präfix	Datentyp	Beispiel
afx, AFX	Variable/Funktion des Anwendungsgestüts	afxDump, AfxWinMain()
_afx, _AFY	Variable/Funktion des Anwendungsgestüts	_afxExLink, _AfxGetPtrFromPrt()
C	Klasse	CString
m_	Elementvariable einer Klasse	CWinThread::m_pMainFrame

Beachten Sie, daß die Verwendung dieser Präfixe keineswegs verbindlich sind, es sind lediglich allgemeine Konventionen, die vom Anwendungs-Assistenten und vielen Visual C++-Programmierern befolgt werden.

5.4.2 MFC-Makros

Die MFC verwendet eine Vielzahl von Makros und Makrogruppen.

- ✗ Da wären zum Beispiel die Makros zur Einrichtung von Nachrichtentabellen (`DECLARE_MESSAGE_MAP`, `BEGIN_MESSAGE_MAP`, `ON_COMMAND` etc.), die wir uns im Kapitel 7 »Interaktivität durch Nachrichten«, speziell in den Aufgaben, noch genauer ansehen werden.
- ✗ Des weiteren gibt es etliche Debug-Makros, wie `ASSERT`, `ASSERT_VALID` und `TRACE`, die ich Ihnen bereits im Kapitel 4 zum Debugger vorgestellt habe.
- ✗ Und es gibt eine Gruppe von Makros, die der Serialisierung und Bereitstellung von Laufzeitinformationen dienen.

Zur letzten Gruppe gehören drei Paare von Makros:

Tabelle 5.2:
Makros für
Laufzeit-
informationen

Makros	Beschreibung
<code>DECLARE_DYNAMIC</code> , <code>IMPLEMENT_DYNAMIC</code>	Es werden Laufzeitinformationen für die Klasse bereitgestellt. Mit Hilfe der Methode <code>ISKINDOF()</code> kann dann ermittelt werden, ob ein Objekt eine Instanz einer bestimmten Klasse ist oder eine Instanz einer Klasse, die von dieser Klasse abgeleitet ist.
<code>DECLARE_DYNCREATE</code> , <code>IMPLEMENT_DYNCREATE</code>	Sorgt für die dynamische Erstellung von Objekten der Klasse. Dies ist insbesondere für Doc/View-Klassen wichtig.
<code>DECLARE_SERIAL</code> , <code>IMPLEMENT_SERIAL</code>	Unterstützt die Serialisierung, d.h. das Lesen und Schreiben von Klassenobjekten. Ein- und Ausgabe dieser Klassenobjekte können dann mit Hilfe der Operatoren <code><<</code> und <code>>></code> der Klasse <code>CArchive</code> erfolgen.

Klassen, die von `CObject` (der obersten Basisklasse der MFC) abgeleitet sind, können mit Hilfe dieser Makros mit Laufzeitinformationen ausgestattet werden. Diese Informationen werden immer dann benötigt, wenn eine Methode ein Objekt übergeben bekommt, dessen genauer Klassentyp nicht feststeht (denken Sie beispielsweise daran, daß man Parametern vom Typ einer Basisklasse auch Objekte übergeben kann, die vom Typ einer abgeleiteten Klasse sind), die Methode aber darauf angewiesen ist, den genauen Klassentyp des Objekts zu kennen (beispielsweise um eine Typumwandlung vorzunehmen). Verfügt das Objekt über Laufzeitinformationen, ist dies kein Problem.



Auch wenn Sie selbst keine Methoden implementieren, die auf Laufzeitinformationen angewiesen sind, sollten Sie beachten, daß etliche Methoden des Anwendungsgerüsts oder allgemeiner der MFC auf diese Informationen angewiesen sind.

Um Laufzeitinformationen für die Objekte einer Klasse bereitzustellen,

1. nimmt man in die Deklaration der Klasse eines der `DECLARE`-Makros auf (siehe beispielsweise `MainFrm.h`)
2. nimmt man in die CPP-Quelltextdatei, in der die Methoden der Klasse definiert sind, das korrespondierende `IMPLEMENT`-Makro auf (siehe `MainFrm.cpp`).

Auf diese Weise wird die Klasse um ein eingebettetes Objekt der Klasse `CRuntimeClass` erweitert. Mit Hilfe des Makros `RUNTIME_CLASS` kann man nun für Objekte der Klasse das eingebettete `CRuntimeClass`-Objekt zurückliefern lassen und sich über den Typ des Objekts informieren.

5.5 Erweitern des Anwendungsgerüsts

So schwer das Anwendungsgerüst für den Anfänger zu verstehen ist, so einfach läßt sich damit programmieren. Um dies zu beweisen, werden wir das Programm jetzt dahingehend erweitern, daß im Fenster der Anwendung ein Text angezeigt wird. Gemäß den Konventionen des Doc/View-Modells, das die Trennung der Datenverwaltung und Datenanzeige fordert, werden wir dabei so vorgehen, daß wir den Text in unserem Dokumentobjekt speichern und das Ansichtsfenster dazu bringen, den Text aus dem Dokumentobjekt abzufragen und anzuzeigen.

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.

5.5.1 Hinzufügen einer Zeichenfolgen-Ressource

Natürlich könnten wir den auszugebenden Text einfach im Quellcode angeben (beispielsweise im Konstruktor der Dokumentklasse), doch ich möchte Ihnen auch einmal zeigen, wie man mit Stringtabellen arbeitet. Wir speichern den Text also in einer Stringtabellen-Ressource und laden den Text

mit Hilfe seiner Ressourcen-ID in unser Programm. Der Vorteil dieses Verfahrens liegt vor allem darin, daß der auszugebende Text vom Programmcode getrennt und eine spätere Lokalisierung der Anwendung dadurch wesentlich erleichtert wird.

Bild 5.6:
Zeichenfolgen-
Ressource
anlegen



Übung 5-1: Stringtabelle des Anwendungsgerüsts bearbeiten

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 Menübefehl EINFÜGEN/NEUE ZEICHENFOLGE auf (Visual Studio- oder Kontextmenü), 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).

5.5.2 Modifizieren des Dokuments

Als nächstes nehmen wir den Text in unser Dokumentobjekt auf. Dazu deklarieren wir in der Dokumentklasse zunächst eine Elementvariable für den Text.

Übung 5-2: Daten in Dokumentklasse aufnehmen

1. Klicken Sie in der DATEIEN-Ansicht auf den Knoten der Datei *Hello_W3Doc.h*.
2. Fügen Sie dem Attribute-Abschnitt der Klasse `CHello_W3Doc` die Deklaration einer Elementvariablen vom Typ `CString` hinzu:

```
// Attribute
public:
    CString m_sDaten;
```

Die Elementvariable `m_sDaten` muß natürlich noch initialisiert werden. Dies geschieht in der Methode `OnNewDocument()`, die beim Start der Anwendung sowie bei Aufruf des Befehls DATEI/NEU aufgerufen wird.

3. Doppelklicken Sie in der KLASSEN-Ansicht auf den Knoten der `CHello_W3Doc`-Methode `OnNewDocument()`.
4. Rufen Sie hier die `CString`-Methode `LoadString()` auf. Zur Spezifikation des zu ladenden Strings übergeben Sie dessen Ressourcen-ID.

```

BOOL CDemoDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
    // (SDI-Dokumente verwenden dieses Dokument)
    m_sDaten.LoadString(IDS_HELLO);
    return TRUE;
}

```

Die Zeichenfolge muß jetzt nur noch angezeigt werden.

5.5.3 Modifizieren der Ansicht

Damit unsere Zeichenfolge angezeigt wird, müssen wir die `OnDraw()`-Methode der Ansichtsklasse modifizieren.

Übung 5-3: Daten in Ansichtsfenster ausgeben

1. Doppelklicken Sie in der KLASSEN-Ansicht auf den Knoten der `CHello_W3View`-Methode `OnDraw()`.
2. Zur Ausgabe des Textes verwenden wir die `CDC`-Methode `TextOut()`, der wir die Ausgabekoordinaten, den auszugebenden String und die Größe des Strings in Byte übergeben:

```

void CHello_W3View::OnDraw(CDC* pDC)
{
    CHello_W3Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(20, 20, pDoc->m_sDaten,
                strlen(pDoc->m_sDaten));
}

```

Programm testen

1. Compilieren und starten Sie das Programm (Befehl ERSTELLEN/AUSFÜHREN VON ...).

Haben Sie die Anleitung korrekt befolgt, sollte sich Ihnen das Anwendungsfenster, wie in Abbildung 5.7 dargestellt, präsentieren.

Bild 5.7:
Die Hello
World-
Anwendung



5.6 Zusammenfassung

MFC-Anwendungen werden üblicherweise mit dem MFC-Anwendungs-Assistenten erstellt. Den Kern jeder MFC-Anwendung bildet ein von `CWinApp` abgeleitetes Objekt, das die Initialisierung sowie die Hauptnachrichtenschleife des Programms implementiert.

Die visuelle Präsentation der Anwendung sowie die Verwaltung der Anwendungsdaten sind ein Ergebnis der Kooperation zwischen Rahmenfenstern, Ansichtsfenstern und Dokumentobjekten. Das Dokumentobjekt verwaltet die Anwendungsdaten, das Ansichtsfenster zeigt die Daten des Dokumentobjekts an 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.

Anwendungen, die mit dem Anwendungs-Assistenten erstellt wurden, eignen sich auch für die Weiterbearbeitung mit dem Klassen-Assistenten.

5.7 Fragen

1. Welches sind die vier wichtigsten Klassen des MFC-Anwendungsgerüsts?
2. Auf welcher Idee beruht die Dokument/Ansicht-Architektur?
3. Wie gibt man in einem Doc/View-Anwendungsgerüst Daten aus?

5.8 Aufgaben

1. Schauen Sie sich mit Hilfe des Debuggers den Startcode der *Hello_W3*-Anwendung an. Laden Sie dazu das Programm in die IDE, und rufen Sie den Befehl ERSTELLEN/DEBUG STARTEN/IN AUFRUF SPRINGEN auf (oder drücken Sie einfach **F11**). Drücken Sie weiter **F11**, um in die Methode `AfxWinMain()` zu verzweigen.
2. Erstellen Sie mit Hilfe des MFC-Anwendungs-Assistenten noch einmal das gleiche Programm, mit gleichen Einstellungen, aber ohne Unterstützung der Dokument/Ansicht-Architektur. Schauen Sie sich den erzeugten Quelltext an, und vergleichen Sie ihn mit dem Quelltext des *Hello_W3*-Programms. Achten Sie insbesondere auf die Einrichtung des Hauptfensters in der Methode `InitInstance()`.
3. Erstellen Sie ein eigenes Anwendungsgerüst. Beginnen Sie mit einem leeren Win32-Anwendungsprojekt, oder lassen Sie den MFC-Anwendungs-Assistenten ein neues Projekt anlegen, und löschen Sie dann alle Dateien. Wenn Sie mit einem Win32-Anwendungsprojekt beginnen, müssen Sie dem Linker mitteilen, daß die MFC verwendet werden soll. Rufen Sie dazu den Menübefehl PROJEKT/EINSTELLUNGEN auf, und wählen Sie auf der Seite ALLGEMEIN im Feld MICROSOFT FOUNDATION CLASSES eine der Optionen zur Verwendung der MFC aus.

Nehmen Sie eine Quelltextdatei und eine Header-Datei in das Projekt auf. In der Header-Datei deklarieren Sie die Klassen für die Anwendung, das Rahmen- und das Ansichtsfenster. In der Quelldatei definieren Sie die Konstruktoren der Klassen. Für die Anwendungsklasse überschreiben Sie die Methode `InitInstance()`, in der ein Rahmenfenster eingerichtet wird, für die Ansichtsklasse überschreiben Sie `OnDraw()`, um einen Text auszugeben.

Neu ist, daß Sie in den Konstruktoren der beiden Fensterklassen jeweils die Methode `Create()` aufrufen müssen. Schauen Sie sich dazu die Online-Erklärung zu der Methode an, oder lesen Sie zuerst das nachfolgende Kapitel.

Diese Übung ist sicherlich nicht einfach. Geben Sie nicht gleich bei den ersten Schwierigkeiten auf, Sie werden im Laufe Ihres Visual C++-Programmiererdaseins noch häufiger in scheinbar ausweglose Lagen geraten...

5.9 Lösung zu Aufgabe 3

Die beiden folgenden Listings zeigen eine mögliche Implementierung eines selbst aufgebauten Anwendungsgerüsts.

Die Header-Datei:

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

class CAnsicht : public CView {
public:
    CAnsicht(CFrameWnd *parent);
protected:
    afx_msg void OnDraw(class CDC *);
};

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

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

Die Quelltextdatei:

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

// Anwendungs-Objekt erzeugen
CMyApp Anwendung;
// Ansichtsfenster
CAnsicht::CAnsicht(CFrameWnd *parent) {
    Create(0, 0, WS_CHILD | WS_VISIBLE,
          CRect(), parent, AFX_IDW_PANE_FIRST);
}
// muss überschrieben werden, da sonst abstrakte Klasse
void CAnsicht::OnDraw(class CDC *dc) {
    RECT rect;
    TEXTMETRIC tm;
    GetClientRect(&rect);
    dc->GetTextMetrics(&tm);
```

```

        dc->TextOut(10,(rect.bottom-tm.tmHeight)/2,
                    "Dies ist die Ansichtsklasse!");
    }
// Rahmenfenster
CRahmenfenster::CRahmenfenster() {
    // Fenster erzeugen
    Create(0, "Rahmen mit View",
           WS_OVERLAPPEDWINDOW, rectDefault);
    // Ansicht erzeugen
    new CAnsicht(this);
}
// 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;
}

```


Anpassen des Haupt- fensters

Das vom MFC-Anwendungs-Assistenten generierte Programmgerüst ist schon ziemlich perfekt. Aber bevor wir dazu übergehen, auf der Grundlage dieses Programmgerüsts funktionelle Anwendungen aufzusetzen, die mehr können, als einen einfachen Textstring anzuzeigen, wollen wir uns anschauen, wie wir das Programmgerüst an unsere Vorstellungen und Bedürfnisse anpassen können.

Im einzelnen lernen Sie in diesem Kapitel:

- ✗ Was Windows unter einem Fenster versteht
- ✗ Worin der Unterschied zwischen Windows-Fenstern und Fensterobjekten besteht
- ✗ Wie man den Titel des Anwendungsfensters anpaßt
- ✗ Wie man die Größe des Anwendungsfensters anpaßt
- ✗ Wie man den Fensterstil anpaßt
- ✗ Wie man Symbolleiste und Statusleiste hinzufügt
- ✗ Wie man das Anwendungssymbol anpaßt
- ✗ Wie man Kommandozeilenargumente abfragt



6.1 Fenster und Fensterobjekte

Im vorangehenden Kapitel haben wir uns bereits ein wenig mit Fenstern beschäftigt. Wir haben Rahmenfenster und Ansichtsfenster kennengelernt. Aber immer, wenn wir genauer nachforschen wollten, wie diese Fenster erzeugt und eingerichtet werden, sind wir an der Dokumentvorlage gescheitert, die uns all diese Arbeiten abnimmt und sie dadurch vor uns verbirgt.

Doch damit wollen wir uns nun nicht mehr zufrieden geben. Um dem Geheimnis der Fenster auf die Spur zu kommen, fragen wir uns zuerst, was Windows unter einem Fenster versteht.

Fenster unter Windows

Der Anwender denkt in Anwendungen, Windows denkt in Fenstern. Der Anwender sieht, daß er auf seinem Desktop den Windows Explorer, Word und ein Grafikprogramm laufen hat, Windows sieht Fenster. Natürlich ist dies ein wenig übertrieben, denn natürlich sieht auch der Anwender Fenster und natürlich erkennt Windows, welche Anwendungen laufen und zu verwalten sind. Aber die Perspektive ist anders. Will man Windows verstehen, muß man sich angewöhnen, mehr in Fenstern zu denken. Kaum hat man dies getan, sieht man plötzlich Fenster, wo früher gar keine waren ...

- ✘ Der Anwender erkennt Fenster an ihrem Rahmen und der Titelleiste. Dies wären also die Hauptfenster der Anwendungen und die Dialoge (für MDI-Editoren auch noch die Dokumentfenster).
- ✘ Für Windows ist ein Fenster schlicht ein rechteckiger Teil der Benutzeroberfläche, der eine bestimmte Funktionalität aufweist und prinzipiell mit dem Anwender interagieren kann.

Die einzelnen Fenster sind hierarchisch angeordnet (von hinten nach vorne):

- ✘ Zuoberst steht das Fenster des Desktops.
- ✘ Darunter folgen die TopLevel-Fenster. Dies sind die Hauptfenster der Anwendungen und die Dialogfenster.
- ✘ Darunter folgen untergeordnete Fenster, die im Client-Bereich ihrer übergeordneten Fenster erscheinen. Als Beispiel haben wir bereits das Ansichtsfenster aus *Hello_W3* kennengelernt, das dem Hauptfenster der Anwendung untergeordnet war. Weitere Beispiele wären die verschiedenen Steuerelemente, die man in Dialogen findet. Ja, jedes Steuerelement – jede Schaltfläche, jedes Eingabefeld, jedes Listenfeld etc. – ist ein Fenster!

Allen diesen Fenstern ist gemeinsam, daß sie über einen Handle verfügen, über den Windows (wie auch der Programmierer) auf die Fenster zugreifen kann. Alle diese Fenster basieren auf sogenannten Fensterklassen. Diese haben nichts mit den Klassen der MFC zu tun. Es sind Strukturen, die Kategorien von Fenstern beschreiben. So gibt es beispielsweise für jedes Windows-Steuerelement eine eigene Fensterklasse, die integraler Teil des Windows-Betriebssystems ist. Für Hauptfenster, die in Anwendungen definiert werden, muß dagegen eine passende Fensterstruktur von der Anwendung erzeugt und unter Windows registriert werden.

Die Fensterklasse ist aber nur eine Schablone, die einen bestimmten Typus von Fenster charakterisiert. Der nächste Schritt ist, auf der Basis einer solchen Fensterklasse ein Fenster zu erzeugen. Die Windows-API sieht dafür die Funktion `CreateWindow()` vor.

Die Fensterklassen der MFC

Machen wir jetzt einen Sprung zurück zur MFC. Wenn Sie ein Objekt einer MFC-Fensterklasse erzeugen:

```
CMainFrame *pMainWnd = new CMainFrame;
```

haben Sie noch lange kein Fenster! Wir haben lediglich eine Instanz einer Klasse, die geeignet ist, ein wirkliches Windows-Fenster zu repräsentieren. Doch dieses Windows-Fenster ist noch nicht vorhanden. Dazu muß die Methode `Create()` aufgerufen werden. Die Methode `Create()` ist in `CWnd`, der Basisklasse aller MFC-Fensterklassen, definiert und dient dazu, ein Windows-Fenster zu erzeugen (analog zu der API-Funktion `CreateWindow()`) und dieses mit dem Fensterobjekt, für das die Methode aufgerufen wurde, zu verbinden.

In dem Beispiel *Hello_W3* aus dem vorangehenden Kapitel haben wir von der `Create()`-Methode nichts gesehen, weil die gesamte Einrichtung der Fenster an die Erzeugung der Dokumentvorlage gekoppelt war. Eigentlich schade, denn in einem Programm ohne Dokument/Ansicht-Architektur, in dem man gezwungen ist, die `Create()`-Methode selbst aufzurufen, kann man diese gleich zur Konfiguration des Fensters nutzen.

Der folgende Quelltextauszug zeigt, wie die `Create()`-Methode direkt im Konstruktor der Rahmenfensterklasse aufgerufen wird. Dabei werden unter anderem der Titel, die Position und die Größe des Fensters festgelegt:

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

```
// Fenster erzeugen
Create(classname,           // 0 für MFC-Vorgabe
      "Erstes Programm (ohne Doc/View)", // Titel
      WS_OVERLAPPEDWINDOW | WS_VSCROLL, // Stil
      CRect(200, 200, 600, 400),         // Pos., Größe
      0,                               // kein übergeordn. Fenster
      0,                               // kein Menü
      WS_EX_TOPMOST,           // immer im Vordergrund
      0);                       // kein Doc/View
}
```

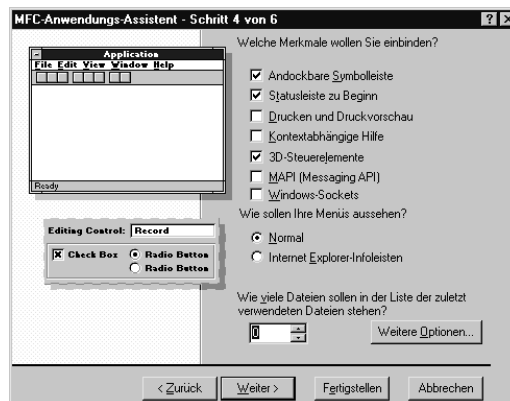
Wie sieht es aber nun mit der Anpassung des Hauptfensters einer Doc/View-Anwendung aus?

6.2 Anpassung über den Assistenten

Die einfachste Möglichkeit ein vom MFC-Anwendungs-Assistenten generiertes Anwendungsgerüst an die eigenen Vorstellungen anzupassen, besteht natürlich darin, die entsprechenden Optionen auf den Dialogseiten des Anwendungs-Assistenten zu setzen. Diese konfigurieren und beeinflussen nämlich nicht nur die Art und die Funktionalität der Anwendung, sondern auch deren Erscheinungsbild.

Alle Einstellungen zum Erscheinungsbild der Anwendung werden in Schritt 4 vorgenommen.

Bild 6.1:
Anpassung des
äußeren Er-
scheinungs-
bilds der
Anwendung



Auf dieser Seite können Sie entscheiden,

- ✗ ob Sie eine SYMBOLLEISTE, passend zum Menü, haben wollen,
- ✗ ob am unteren Rand des Fensters eine STATUSLEISTE angezeigt werden soll (in der automatisch kurze Hilfetexte zu den Befehlen der Menü- und Symbolleiste angezeigt werden)
- ✗ welches Aussehen Ihre Menüs haben sollen.

Des weiteren können Sie die Titelleiste des Rahmenfensters weiter konfigurieren. Sie müssen nur auf den Schalter WEITERE OPTIONEN klicken.



Bild 6.2:
Anpassung der
Titelleiste

Die meisten der Einstellungen auf der Seite ZEICHENFOLGEN FÜR DOKUMENTVORLAGE sind Optionen für die Unterstützung von Dateitypen, die vor allem für Anwendungen zur Bearbeitung von Dateien interessant sind. Zwischen- drin befindet sich aber auch das Eingabefeld BESCHRIFTUNG DES HAUPTFEN- STERS, in das man den gewünschten Titel für das Hauptfenster eingeben kann.

Auf der Seite FENSTERSTILE können Sie sich für einen breiten oder schmalen Rahmen entscheiden und festlegen, ob die Titelleiste des Hauptfensters mit einem Systemmenü und Schaltflächen zum Minimieren und Maximieren ausgestattet sein soll. Des weiteren können Sie festlegen, ob das Hauptfen- ster beim Start der Anwendung in normaler Größe oder minimiert bezie- hungsweise maximiert angezeigt werden soll.

6.3 Anpassung über die Methode PreCreateWindow()

Die Konfigurationsmöglichkeiten im vierten Schritt des MFC-Anwendungs-Assistenten sind zwar schon recht brauchbar, aber doch nicht so weitreichend, wie die Möglichkeiten, die sich dem Programmierer bieten, wenn er seine Fenster durch Aufruf der `Create()`-Methode selbst erzeugt.

Um diesem Manko zu begegnen, stellt uns der Anwendungs-Assistent die Methode `PreCreateWindow()` zur Verfügung. Diese Methode wird automatisch vor der Erzeugung des eigentlichen Windows-Fenster aufgerufen – also kurz bevor das Anwendungsgerüst intern die Methode `Create()` aufruft. Dabei wird der Methode die Adresse auf eine Strukturvariable vom Typ `CREATESTRUCT` übergeben. Diese Struktur ist es wert, daß wir sie genauer unter die Lupe nehmen:

```
typedef struct tagCREATESTRUCT {
    LPVOID    lpCreateParams;    // Fensterdaten
    HANDLE    hInstance;        // Modul des Fensters
    HMENU      hMenu;            // Menü
    HWND       hwndParent;       // übergeordn. Fenster
    int        cy;                // Höhe
    int        cx;                // Breite
    int        y;                // y-Koord des oberen Rands
    int        x;                // x-Koord des linken Rands
    LONG       style;            // Fensterstil
    LPCSTR     lpszName;         // Fenstername
    LPCSTR     lpszClass;        // Fensterstruktur
    DWORD      dwExStyle;        // erw. Fensterstil
} CREATESTRUCT;
```

Was wir in `PreCreateWindow()` machen können, ist, auf einzelne Elemente der Struktur zuzugreifen und diesen neue Werte zuzuweisen. Da die Strukturvariable nicht als Kopie, sondern als Referenz übergeben wurde, werden alle unsere Änderungen an den vor uns verborgenen Code des Anwendungsgerüsts zurückgemeldet und bei der Einrichtung des Fensters (Aufruf von `Create()`) berücksichtigt.

Lassen Sie uns anschauen, wie dies funktioniert:

6.3.1 Anpassung der Position und Größe des Hauptfensters

Übung 6-1: Position und Größe des Hauptfensters anpassen

1. Legen Sie wie in Kapitel 5 mit Hilfe des MFC-Anwendungs-Assistenten ein neue SDI-Anwendung mit Unterstützung für Doc/View an.
2. Führen Sie das Programm zum Test aus (**Strg** + **F5**), und achten Sie auf Position und Größe des Fensters. Beenden Sie das Programm.
3. Expandieren Sie in der KLASSEN-Ansicht den Knoten der Klasse CMainFrame, und doppelklicken Sie auf die Methode PreCreateWindow().
4. Weisen Sie den Koordinaten x und y für die obere linke Ecke des Fensters neue Werte zu. Weisen Sie den Elementen cx und cy, die Breite und Höhe des Fensters bestimmen, neue Werte zu.

```

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.
    cs.x = 100;
    cs.y = 100;
    cs.cx = 400;
    cs.cy = 100;
    return TRUE;
}

```

5. Führen Sie die Anwendung erneut aus (**Strg** + **F5**).

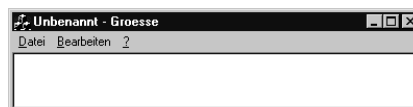


Bild 6.3:
Fenster mit
angepaßter
Größe

Die CREATESTRUCT-Struktur bietet uns aber noch mehr Möglichkeiten. Man könnte eine andere Menüressource oder ein übergeordnetes Fenster angeben (eher interessant für untergeordnete Ansichtsfenster) oder den Fensterstil beeinflussen.

6.3.2 Anpassung des Fensterstils

Der Fensterstil wird durch eine Kombination bestimmter Konstanten festgelegt.

Tabelle 6.1:
Die Fenster-
stile (für
cs.style)¹

Stil	Bedeutung
WS_BORDER	Dünnere Rand.
WS_CAPTION	Titel und dünnere Rand.
WS_CHILD	Kindfenster (nicht zu verwenden mit WS_POPUP).
WS_CHILDWINDOW	Entspricht WS_CHILD.
WS_CLIPCHILDREN	Beim Zeichnen ins Elternfenster werden Bereiche der Kindfenster ausgenommen.
WS_CLIPSIBLINGS	Beim Zeichnen in ein Kindfenster werden alle Bereiche anderer überlappender Kindfenster ausgenommen.
WS_DISABLED	Fenster anfänglich deaktiviert.
WS_DLGFRAAME	Für Dialogfenster.
WS_GROUP	Zur Gruppierung von Steuerelementen.
WS_HSCROLL	Horizontale Bildlaufleiste.
WS_ICONIC	Entspricht WS_MINIMIZED.
WS_MAXIMIZE	Fenster ist anfänglich maximiert.
WS_MAXIMIZEBOX	Schalter zum Maximieren (erfordert WS_SYSMENU, nicht in Kombination mit WS_EX_CONTEXTHELP).
WS_MINIMIZE	Fenster ist anfänglich maximiert.
WS_MINIMIZEBOX	Schalter zum Minimieren (erfordert WS_SYSMENU, nicht in Kombination mit WS_EX_CONTEXTHELP).
WS_OVERLAPPED	Titelleiste und Rahmen.
WS_OVERLAPPEDWINDOW	Kombination der Stile WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX und WS_MAXIMIZEBOX.

¹ Die Konstanten für die erweiterten Fensterstile (cs.dwExStyle) können Sie der Online-Hilfe entnehmen (schlagen Sie im Index unter CREATESTRUCT nach und klicken Sie auf den Link zu dem Parameter dxExStyle).

Anpassung über die Methode PreCreateWindow()

Stil	Bedeutung
WS_POPUP	Popup-Fenster (nicht in Kombination mit WS_CHILD).
WS_POPUPWINDOW	Kombination der Stile WS_BORDER, WS_POPUP und WS_SYSMENU. (Zu verwenden in Kombination mit WS_CAPTION).
WS_SIZEBOX	Entspricht WS_THICKFRAME.
WS_SYSMENU	Systemmenü (erfordert WS_CAPTION).
WS_TABSTOP	Steuerelement kann mit Tabulator-Taste angesteuert werden.
WS_THICKFRAME	Dicker Rahmen. Nur dieser Rahmen erlaubt das Verändern der Fenstergröße durch Ziehen des Rahmens.
WS_TILED	Entspricht WS_OVERLAPPED.
WS_TILEDWINDOW	Entspricht WS_OVERLAPPEDWINDOW.
WS_VISIBLE	Fenster anfangs sichtbar.
WS_VSCROLL	Vertikale Bildlaufleiste.

Tabelle 6.1:
Die Fensterstile (für cs.style)
(Fortsetzung)

Die Stilkonstanten können mit Hilfe der Bit-Operatoren hinzugefügt, gelöscht und kombiniert werden. Um beispielsweise den aktuellen Stil durch Einstellungen zu ersetzen, die ein typisches Rahmenfenster (WS_OVERLAPPEDWINDOW) mit vertikaler Bildlaufleiste (WS_VSCROLL) erzeugen, würde man schreiben:

```

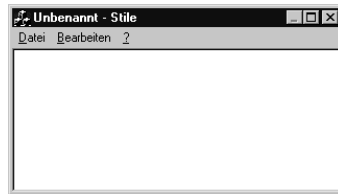
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // ZU ERLEDIGEN: Ändern Sie hier die Fensterklasse oder
    // das Erscheinungsbild, indem Sie
    // CREATESTRUCT cs modifizieren.
    cs.style = WS_OVERLAPPEDWINDOW | WS_VSCROLL;
    return TRUE;
}

```

Wenn Sie dagegen mehr daran interessiert sind, die Minimieren-Schaltfläche zu deaktivieren, schreiben Sie:

```
cs.style ^= WS_MINIMIZEBOX;
```

Bild 6.4:
Fenster mit
deaktivierter
Minimieren-
Schaltfläche



6.4 Anpassung über die Methode OnCreate()

Die Methode `OnCreate()` ist das Pendant zu der Methode `PreCreateWindow()`, insofern als `PreCreateWindow()` direkt vor der Erzeugung des Windows-Fensters und `OnCreate()` direkt nach der Erzeugung des Windows-Fensters (aber bevor das Fenster sichtbar wird) aufgerufen werden.

6.4.1 Symbolleiste und Statusleiste

Der Anwendungs-Assistent legt diese Methode automatisch an, wenn Sie sich im Schritt 4 des Assistenten für eine Symbolleiste (ToolBar) und/oder eine Statusleiste (StatusBar) entschieden haben. Die Implementierung der Methode sieht wie folgt aus:¹

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
        WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS
        | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Symbolleiste konnte nicht erstellt \
            werden\n");
        return -1;        // Fehler bei Erstellung
    }
}
```

¹ Beachten Sie, daß die Methode `OnCreate()` sowie die in der Methode verwendeten Elementvariablen `m_wndStatusBar` und `m_wndToolBar` zusätzlich in der Klasse `CMainFrame` deklariert werden müssen.

```

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;
}

```

Dieses Beispiel zeigt übrigens sehr schön, wie Fenster in MFC-Programmen erzeugt werden, denn Symbol- und Statusleiste sind aus Sicht von Windows Fenster. Die Einrichtung dieser Fenster im Quelltext erfolgt wie besprochen in zwei Schritten:

1. Zuerst wird ein Objekt der Fensterklasse erzeugt. In diesem Fall sind dies die Objekte `m_wndToolBar` und `m_wndStatusBar`, die als Elementvariablen in der Klasse des Rahmenfensters deklariert werden. Im Zuge der Erzeugung des Rahmenfensterobjekts werden auch diese eingebetteten Objekte erzeugt.
2. Danach wird für das Fensterklassenobjekt die Methode `Create()` aufgerufen (beziehungsweise `CreateEx()` für die Symbolleiste), die das zugehörige Windows-Fenster (also die eigentliche Symbol- oder Statusleiste) einrichtet und mit dem Fensterklassenobjekt verbindet, so daß wir im Programm auf dem Weg über das Fensterobjekt das wirkliche Windows-Fenster manipulieren können.

6.5 Das Anwendungssymbol

Unter Windows verfügt üblicherweise jedes Programm über ein Symbol (Icon) zur grafischen Präsentation. Windows verwendet dieses Symbol in verschiedenen Kontexten (Titel des Hauptfensters, Anzeige im Explorer, Task-Leiste) in jeweils verschiedenen Größen.

Aufgabe jeder ordentlichen Windows-Anwendung ist es daher, ein entsprechendes Symbol bereitzustellen. Prinzipiell genügt es dabei, ein Symbol in den Maßen 32x32 und 16 Farben anzulegen. Windows paßt dann das

Symbol automatisch an die Erfordernisse der verschiedenen Kontexte sowie Bildschirmauflösungen an. Gerade beim Verkleinern des Symbols (beispielsweise für die Anzeige im Explorer) kann es dabei aber zu Verzerrungen kommen. Um dies zu vermeiden, empfiehlt es sich, das Symbol zusätzlich mit den Maßen 16x16 (und vielleicht auch 20x20) bereitzustellen.

Übung 6-2: Eigenes Anwendungssymbol vorsehen

1. Legen Sie wie in Kapitel 5 mit Hilfe des MFC-Anwendungs-Assistenten ein neue SDI-Anwendung mit Unterstützung für Doc/View an, oder laden Sie eines der letzten Projekte, dessen Anwendungssymbol Sie ändern möchten.

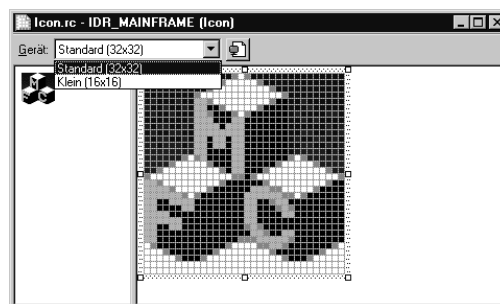


2. Führen Sie das Programm zum Test aus (**Strg** + **F5**), und achten Sie auf das Symbol der Anwendung in Titelleiste, Task-Leiste und Explorer. Es ist dies das Standardsymbol, das der Assistent den Anwendungen mitgibt.

Wir wollen dieses Standardsymbol nun durch ein eigenes Symbol ersetzen.

3. Wechseln Sie zur RESSOURCEN-Ansicht des Arbeitsbereichsfensters, und expandieren Sie den Knoten ICON. Doppelklicken Sie auf den Knoten mit der Bezeichnung IDR_MAINFRAME, um das Symbol in den Editor zu laden.

Bild 6.5:
Geräteabbilder
für Anwen-
dungssymbol



4. Klappen Sie jetzt bitte einmal das Listenfeld mit der Bezeichnung GERÄT auf. Wie Sie sehen können, hat der MFC-Anwendungs-Assistent für das Anwendungssymbol zwei Darstellungen in verschiedenen Größen angelegt. Sehr schön! Die Standardversion von 32x32 Pixel wollen wir jetzt übermalen.

Übrigens: Hätte der Assistent noch keine 16x16-Version eingerichtet, wie hätten wir dies nachholen können? Richtig, durch Anklicken der Schaltfläche NEUES GERÄTEABBILD neben dem Listenfeld.

- Überpinseln Sie mit Hilfe der Malwerkzeuge aus der Symbolleiste GRAFIKEN die Standardversion des Anwendungssymbols.

Wenn Sie sich dabei einmal vermalen, können Sie die letzten Zeichenoperationen mit Hilfe des Menübefehls BEARBEITEN/RÜCKGÄNGIG (**Strg** + **Z**) wieder zurücknehmen.

- Führen Sie das Programm erneut aus (**Strg** + **F5**), und achten Sie auf das Symbol der Anwendung in Titelleiste, Task-Leiste und Explorer. Es ist dies immer noch das Standardsymbol in den Ausmaßen 16x16.

Schauen Sie sich das Symbol im Explorer an (wird neben der EXE-Datei im Unterverzeichnis *Debug* angezeigt). Schalten Sie im Explorer die Ansicht mit den GROSSEN SYMBOLEN ein. Jetzt sollten Sie Ihr eigenes Anwendungssymbol sehen.

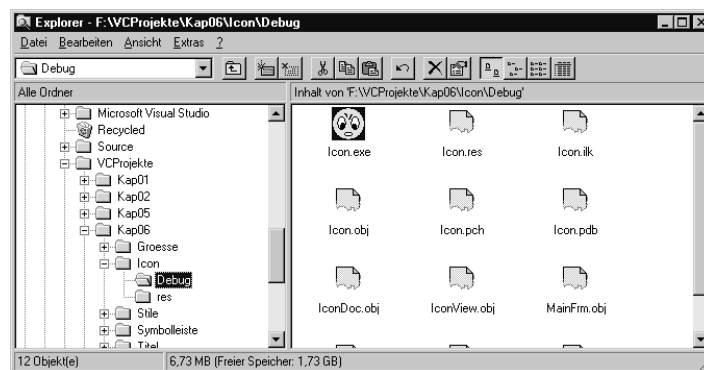
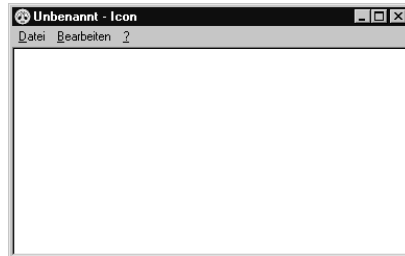


Bild 6.6:
Das große
Symbol im
Explorer

- Kehren Sie nochmals zurück zur IDE und zu Ihrem Anwendungssymbol.
- Löschen Sie auch das Geräteabbild mit den Maßen 16x16. Wählen Sie das Geräteabbild über das Listenfeld GERÄT aus, und rufen Sie dann den Menübefehl BILD/GERÄTEABBILD LÖSCHEN auf.
- Führen Sie das Programm erneut aus (**Strg** + **F5**), und achten Sie wieder auf das Symbol der Anwendung in Titelleiste, Task-Leiste und Explorer. Überall sehen Sie jetzt Ihr eigenes Symbol, das von Windows bei Bedarf automatisch verkleinert wird.

Bild 6.7:
Das verkleinerte Symbol
in der Titelleiste



6.6 Kommandozeilenargumente

Auch für Windows-Anwendungen ist es manchmal angebracht, Kommandozeilenargumente zu unterstützen. Die MFC sieht für bestimmte Argumente eine Standardverarbeitung vor, die mit wenigen Methodenaufrufen implementiert werden kann. Darüber hinaus kann man aber auch eigene Argumente definieren und verarbeiten.

Zum Einlesen und Abfragen der Kommandozeilenargumente sind in der MFC die Klasse `CCommandLineInfo` und die Methode `ParseCommandLine()` vorgesehen. Letztere ruft zur Verarbeitung der Kommandozeile die `CCommandLineInfo`-Methode `ParseParam()` auf, die man zur Verarbeitung eigener Kommandozeilenargumente überschreiben kann.



Die von der MFC standardmäßig unterstützten Kommandozeilenargumente werden automatisch in der `ParseParam()`-Methode der `CCommandLineInfo`-Klasse ausgewertet und können über einen Aufruf der `CWinApp`-Methode `ProcessShellCommand()` ihrer Verarbeitung zugeführt werden.

Im folgenden Programm soll das Hauptfenster je nach Kommandozeilenargument bei Programmstart in normaler Größe (kein Argument), minimiert (-min) oder maximiert (-max) angezeigt werden.

Übung 6-3: Kommandozeilenargumente unterstützen

1. Legen Sie wie in Kapitel 5 mit Hilfe des MFC-Anwendungs-Assistenten ein neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Kommandozeile« genannt.

Um die `CCommandLineInfo`-Methode `ParseParam()` überschreiben zu können, müssen wir zuerst eine eigene Klasse von `CCommandLineInfo` ableiten. Der Einfachheit halber legen wir dafür keine eigene Dateien an, sondern verwenden die Dateien des Anwendungsobjekts.

2. Öffnen Sie die Header-Datei, in der die Anwendungsklasse deklariert ist (in meinem Fall heißt die Datei *Kommandozeile.h*).
3. Deklarieren Sie über der Anwendungsklasse die abgeleitete CCommand-LineInfo-Klasse, in der Sie die Methode ParseParam() deklarieren.

```
// Kommandozeile.h : Haupt-Header-Datei für die Anwendung
...
////////////////////////////////////
// My_CCommandLineInfo:
class My_CCommandLineInfo : public CCommandLineInfo
{
public:
    virtual void ParseParam(const char* pszParam,
                           BOOL bFlag, BOOL bLast);
};
```

4. Öffnen Sie jetzt die zugehörige Quelltextdatei (*Kommandozeile.cpp*). Implementieren Sie hier die überschriebene Methode ParseParam().

Um die Darstellung des Hauptfensters beim Starten der Anwendung festzulegen, setzen wir das Datenelement `m_nCmdShow` des Anwendungsobjekts auf `SW_SHOWMINIMIZED` oder `SW_SHOWMAXIMIZED`. Das Datenelement `m_nCmdShow` übergeben wir später in Schritt 6 an die Methode `ShowWindow()`. Einen Zeiger auf das Anwendungsobjekt besorgen wir uns mit Hilfe der globalen MFC-Funktion `AfxGetApp()`.

Zu guter Letzt rufen wir noch die Implementierung der Methode aus der Basisklasse auf. Dies ist zwar nicht unbedingt notwendig, läßt uns aber die Möglichkeit offen, auch die von der MFC automatisch unterstützten Kommandozeilenargumente auszuwerten.

```
// Kommandozeile.cpp : Legt das Klassenverhalten für die
// Anwendung fest.
...
////////////////////////////////////
// Die Kommandozeile parsen
void My_CCommandLineInfo::ParseParam(const char* pszParam,
                                     BOOL bFlag, BOOL bLast)
{
    if(lstrcmpA(pszParam, "min") == 0)
        AfxGetApp()->m_nCmdShow = SW_SHOWMINIMIZED;
    else if (lstrcmpA(pszParam, "max") == 0)
        AfxGetApp()->m_nCmdShow = SW_SHOWMAXIMIZED;
```



```
CCommandLineInfo::ParseParam(pszParam, bFlag, bLast);
}
```

Der nächste Schritt besteht darin, die Auswertung der Kommandozeile in unserem Programm anzustoßen.

5. Scrollen Sie in der Datei nach unten, bis Sie in der `InitInstance()`-Methode zur Zeile mit der Anweisung »`CCommandLineInfo cmdInfo;`« kommen. Ersetzen Sie den Klassentyp `CCommandLineInfo` durch unsere abgeleitete Klasse:

```
BOOL CKommandozeileApp::InitInstance()
{
    ...
    My_CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
}
```

Jetzt müssen wir nur noch dafür sorgen, daß die Einstellungen, die wir in der Elementvariablen `m_nCmdShow` abgespeichert haben, auch berücksichtigt werden.

6. Scrollen Sie zum Ende der `InitInstance()`-Methode. Hier finden Sie den Aufruf der Methode `ShowWindow()`, der standardmäßig das Argument `SW_SHOW` übergeben wird. Ersetzen Sie das Argument `SW_SHOW` durch die Elementvariable `m_nCmdShow`.

```
BOOL CKommandozeileApp::InitInstance()
{
    ...
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
```

7. Führen Sie das Programm zum Test aus (`Strg` + `F5`), und testen Sie die verschiedenen Kommandozeilenargumente aus.

Um einer Anwendung beim Ausführen im Debugger Kommandozeilenargumente übergeben zu können, wechseln Sie im Dialogfenster PROJEKTEINSTELLUNGEN (Aufruf über PROJEKT/EINSTELLUNGEN) zur Seite DEBUG, und geben Sie Ihre Kommandozeilenargumente in das Eingabefeld PROGRAMMARGUMENTE ein. Beachten Sie dabei, daß den Argumenten ein Bindestrich vorangestellt wird.

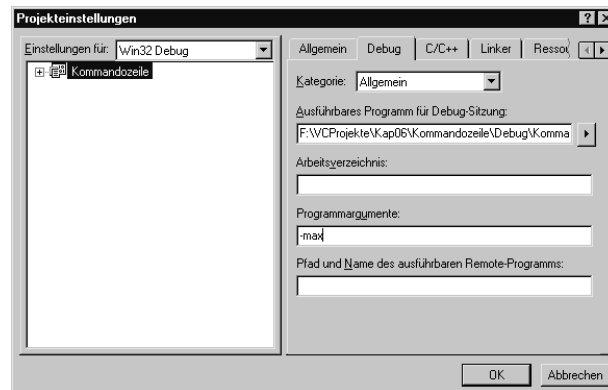


Bild 6.8:
Angabe von
Kommando-
zeilenargumen-
ten über den
Debugger

6.7 Zusammenfassung

Einer der ersten Schritte bei der Überarbeitung des MFC-Anwendungsgerüsts ist die Anpassung des Hauptfensters.

- ✗ Bereits bei der Einrichtung des Projekts mit dem Anwendungs-Assistenten können bestimmte Fensterstile und Fensterdekorationen (Symbolleiste, Statusleiste, Erscheinungsbild der Steuerelemente) festgelegt werden.
- ✗ Die anfängliche Position und Größe des Hauptfensters kann auf dem Weg über den `CREATESTRUCT`-Parameter der `PreCreateWindow()`-Methode festgelegt werden.
- ✗ In der gleichen Weise können Sie die Fensterstile konfigurieren.
- ✗ Das Anwendungssymbol kann zur Bearbeitung aus der Ressourcendatei in den Bildeditor geladen werden.

Kommandozeilenargumente können auch an Windows-Programme übergeben werden. Um die Argumente aus der Kommandozeile einlesen und bearbeiten zu können, muß man eine eigene Klasse von `CCommandLineInfo` ableiten.

Wichtig für das Verständnis des MFC-Anwendungsgerüsts ist es, die Beziehung zwischen der Instanz einer MFC-Fensterklasse und dem eigentlichen Fenster zu verstehen. Ein Fenster ist ein Windows-Objekt, das unter Windows erzeugt wird und von Windows angezeigt und verwaltet wird. Eine Instanz einer Fensterklasse ist erst einmal ein Programm-Objekt im objekt-orientierten Sinne, das dafür gedacht ist, im Programm eine Verbindung zu einem Windows-Fenster herzustellen und die Programmierung mit diesem

zu ermöglichen. Die Erzeugung eines Fensters in einem MFC-Programm läuft daher so ab, daß

- ✗ eine Instanz einer Fensterklasse eingerichtet wird (Konstruktoraufruf) und dann
- ✗ das eigentliche Windows erzeugt und mit der Instanz der Fensterklasse verbunden wird (Aufruf der `Create()`-Methode für die Instanz).

6.8 Fragen

1. Warum sieht man im Quelltext des vom MFC-Anwendungs-Assistenten eingerichteten Anwendungsgerüsts nichts von den `Create()`-Aufrufen zur Erzeugung der Fenster?
2. In welcher Beziehung stehen die Methoden `Create()` und `PreCreateWindow()` zueinander?
3. Wie kann ich die Anfangsgröße meines Hauptfensters vorgeben?
4. Wo finde ich eine Beschreibung der verschiedenen Fensterstile?
5. Wie erreiche ich, daß die Größe meines Hauptfensters nicht vom Anwender verändert werden kann?

6.9 Aufgaben

1. Welches sind die Standardkommandozeilenargumente, die vom MFC-Anwendungsgerüst bearbeitet werden? (TIP: Schlagen Sie in der Online-Hilfe unter `CCommandLineInfo` nach.)
2. Legen Sie eine Kopie des Anwendungsgerüsts aus der Aufgabe 3 des Kapitels 5 an. Legen Sie die Größe des Hauptfensters unveränderbar fest, und nehmen Sie eine Statusleiste in das Hauptfenster auf.

Um eine Kopie anzulegen, beginnen Sie mit einem leeren Win32-Anwendungsprojekt. Rufen Sie den Menübefehl PROJEKT/EINSTELLUNGEN auf, und wählen Sie auf der Seite ALLGEMEIN im Feld MICROSOFT FOUNDATION CLASSES eine der Optionen zur Verwendung der MFC aus. Kopieren Sie die Quelltextdatei (in meinem Beispiel *Applik.cpp*) und die Header-Datei (in meinem Beispiel *Applik.h*) in das Verzeichnis des Projekts, und nehmen Sie beide Dateien mit Hilfe des Befehls PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN in das Projekt auf.

Die Größe des Hauptfensters können Sie direkt beim Aufruf der `Create()`-Methode festlegen. Für die Statusleiste überschreiben Sie die `OnCreate()`-Methode des Rahmenfensters. Als Vorlage können Sie das Listing aus dem Abschnitt 6.4 verwenden. Leider reicht dies jedoch nicht. `OnCreate()` ist eine Behandlungsmethode zur Windows-Nachricht `WM_CREATE`. Im Anwendungsgerüst des Assistenten wird automatisch eine Verknüpfung dieser Nachricht mit der Methode `OnCreate()` eingerichtet. In Ihrem Anwendungsgerüst müssen Sie dies selbst tun.

Sie müssen eine Antworttabelle für die Rahmenfensterklasse einrichten. Rufen Sie dazu irgendwo in der Klassendeklaration das Makro `DECLARE_MESSAGE_MAP()` auf. In der Quelltextdatei definieren Sie die Antworttabelle mit den Makros `BEGIN_MESSAGE_MAP(CRahmenfenster, CFrameWnd)` und `END_MESSAGE_MAP()`. In der Tabelle rufen Sie das Makro `ON_WM_CREATE()` auf, das für Sie die Verbindung zwischen `WM_CREATE` und der `OnCreate()`-Methode herstellt.

```
BEGIN_MESSAGE_MAP(CRahmenfenster, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()
```

3. Legen Sie eine Kopie des Anwendungsgerüsts aus Aufgabe 2 an. Definieren Sie ein Anwendungssymbol für das Programm.

Legen Sie mit Hilfe des Befehls `PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU` eine Ressourcenskriptdatei (*Applik.rc*) an. Fügen Sie eine `ICON`-Ressource ein (Befehl `EINFÜGEN/RESSOURCE`), und bearbeiten Sie diese.

Um das Symbol aus der Ressource zu laden, verwenden Sie die `CWinApp`-Methode `LoadIcon()`. Doch wohin mit dem geladenen Symbol? Sie müssen im Konstruktor des Rahmenfensters eine Fensterklassenvariable vom Typ `LPCTSTR` deklarieren und dieser den Rückgabewert der Funktion `AfxRegisterWndClass()` zuweisen. Im Aufruf von `AfxRegisterWndClass()` laden Sie das Symbol. Die Fensterklasse wird dann bei Erzeugung des Rahmenfensters der Methode `Create()` übergeben.

Wie die beschriebenen Funktionen und Methoden aufgerufen werden, entnehmen Sie der Online-Hilfe.

6.10 Lösungen zu den Aufgaben

Zu 1: Standardkommandozeilenargumente

Tabelle 6.2:
Standardmäßig
verarbeitete
Kommando-
zeilenargu-
mente

Argument	Bearbeitung
app	Legt eine neue Datei an.
app Dateiname	Öffnet die spezifizierte Datei.
app /p Dateiname	Druckt die Datei über den Standarddrucker.
app /pt Dateiname Port	Druckt die Datei über den spezifizierten Drucker-Port.
app /dde	Wartet auf DDE-Befehl.
app /Automation	Startet als OLE-Automatisierungs-Server.
app /Embedding	Startet zur Vor-Ort-Bearbeitung eines OLE-Objekts.

Wenn Sie an der Verarbeitung der Standardargumente interessiert sind, rufen Sie die Methode `ProcessShellCommand()` mit ihrem `CCommandLineInfo`-Objekt als Argument auf. Beachten Sie, daß vor diesem Aufruf das Fenster vollständig erzeugt und ein Zeiger auf das Fenster an das Datenelement `m_pMainWnd` übergeben sein muß.

Zu 2: Anwendungsgerüst mit unveränderbarem Hauptfenster und Statusleiste

Die Header-Datei:

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

class CAnsicht : public CView {
public:
    CAnsicht(CFrameWnd *parent);
protected:
    afx_msg void OnDraw(class CDC *);
};

class CRahmenfenster : public CFrameWnd {
public:
    CRahmenfenster();
protected:
    CStatusBar m_wndStatusBar;
```

```

    int OnCreate(LPCREATESTRUCT cs);
    DECLARE_MESSAGE_MAP()
};
class CMyApp : public CWinApp {
public:
    virtual BOOL InitInstance();
};

```

Die Quelltextdatei:

```

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

// Anwendungs-Objekt erzeugen
CMyApp Anwendung;
static UINT indicators[] = {ID_SEPARATOR, };
// Ansichtsfenster
CAnsicht::CAnsicht(CFrameWnd *parent) {
    Create(0, 0, WS_CHILD | WS_VISIBLE,
          CRect(), parent, AFX_IDW_PANE_FIRST);
}
// muss überschrieben werden, da sonst abstrakte Klasse
void CAnsicht::OnDraw(class CDC *dc) {
    RECT rect;
    TEXTMETRIC tm;
    GetClientRect(&rect);
    dc->GetTextMetrics(&tm);
    dc->TextOut(10, (rect.bottom-tm.tmHeight)/2,
               "Dies ist die Ansichtsklasse!");
}
// Rahmenfenster
BEGIN_MESSAGE_MAP(CRahmenfenster, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()
CRahmenfenster::CRahmenfenster() {
    // Fenster erzeugen
    Create(0, "Rahmen mit View",
          WS_OVERLAPPEDWINDOW & ~WS_THICKFRAME,
          CRect(20, 20, 300, 200));
    // Ansicht erzeugen
    new CAnsicht(this);
}
// Eingebettete Rahmenfensterobjekte erzeugen
int CRahmenfenster::OnCreate(LPCREATESTRUCT cs) {

```

```

        if(CFrameWnd::OnCreate(cs) == -1)
            return -1;
        m_wndStatusBar.Create(this);
        m_wndStatusBar.SetIndicators(indicators,
                                     sizeof(indicators)/sizeof(UINT));

        return 0;
    }
    // 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;
    }

```

Zu 3: Anwendungsgerüst mit eigenem Symbol

Die Header-Datei bleibt unverändert.

Die Änderungen in der Quelltextdatei:

```

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

CRahmenfenster::CRahmenfenster() {
    LPCTSTR wndClass = AfxRegisterWndClass(NULL,
        AfxGetApp()->LoadCursor(IDC_ARROW),
        (HBRUSH) (COLOR_WINDOW+1),
        AfxGetApp()->LoadIcon(IDI_ICON1));

    // Fenster erzeugen
    Create(wndClass, "Rahmen mit View",
        WS_OVERLAPPEDWINDOW & ~WS_THICKFRAME,
        CRect(20, 20, 300, 200));

    // Ansicht erzeugen
    new CAnsicht(this);
}

```

Interaktivität durch Nachrichten

Die sichtbare Benutzerschnittstelle eines Windows-Programms sind seine Fenster. Über diese Fenster kann das Programm dem Anwender Informationen anzeigen (beispielsweise den Inhalt einer Datei oder das Ergebnis einer Berechnung). Doch es wäre ein rechtes Armutszeugnis, wenn diese Benutzerschnittstelle nur in einer Richtung (vom Programm zum Anwender) funktionieren würde. Ein ordentliches Programm muß schon in der Lage sein, seinerseits auch wieder Eingaben vom Anwender zu empfangen – sei es der Aufruf eines Menübefehls, ein Klick mit der Maus oder eine Tastatureingabe. Überlegt man sich, wie eine solche Schnittstelle, die Benutzereingaben akzeptiert, aussehen könnte, steht man vor einem Problem.

Unter DOS wird immer nur ein Programm gleichzeitig ausgeführt. Dieses Programm verfügt uneingeschränkt über alle Ressourcen des Systems. Wenn der Anwender also die Maus bewegt oder etwas über Tastatur eintippt, ist klar, daß diese Eingaben an das laufende Programm gerichtet sind.

Unter Windows werden aber mehrere Anwendungen gleichzeitig ausgeführt. Welche dieser Anwendungen ist verantwortlich, wenn der Anwender mit der Maus klickt oder auf der Tastatur herumhackt? An diesem Punkt schaltet sich das Betriebssystem ein.

Alle Aktionen des Anwenders (Klick oder Bewegung der Maus, Betätigen der Tastatur) werden zuerst einmal von Windows abgefangen. Gleichzeitig überwacht Windows, in welchem TopLevel-Fenster (Hauptfenster oder Dialog) der Anwender gerade arbeitet, das heißt, welches Fenster aktiv ist. (Das



aktive Fenster wird durch eine farbige Titelleiste gekennzeichnet – inaktive Fenster haben eine graue Titelleiste.) Windows ermittelt, welche Anwendung zu dem aktiven Fenster »gehört« und schickt dieser Anwendung eine Nachricht, die die Anwendung über das aufgetretene Ereignis informiert.

Wie die Anwendung die Nachricht entgegennimmt, und wie man den Code implementiert, der auf die Nachricht antworten soll – das wollen wir in diesem Kapitel klären. Wie so oft bei der MFC-Programmierung gilt dabei, daß die zugrundeliegende Theorie wesentlich komplizierter ist als die Programmierung.

Sie lernen in diesem Kapitel:

- ✗ Was Antworttabellen (MESSAGE_MAP) sind
- ✗ Wie man mit dem Klassen-Assistenten Behandlungsroutinen für Nachrichten einrichtet
- ✗ Wie man Mausereignisse abfängt und beantwortet
- ✗ Wie man sich Gerätekontexte zum Zeichnen beschafft
- ✗ Wie man Tastaturereignisse abfängt
- ✗ Wie man erreicht, daß der Inhalt eines Fensters automatisch aktualisiert wird
- ✗ Wie man seine Anwendungen mit einer internen Uhr ausstattet

7.1 Ereignisverarbeitung unter Windows und der MFC

In der Einleitung dieses Kapitels haben Sie bereits erfahren, wie sich Windows in die Kommunikation zwischen Anwender und Anwendung einschaltet, indem es alle Aktionen des Anwenders abfängt und in Form von Nachrichten an die betreffenden Anwendungen weiterleitet.

Betrachten wir dazu ein konkretes Beispiel.

7.1.1 Ereignisverarbeitung unter Windows

Der Anwender klickt mit der Maus in das Hauptfenster einer Anwendung. 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-Parametern (`wParam` und `lParam`) für zusätzliche Informationen (beispielsweise die Koordinaten des Mausklicks).

Diese Nachricht schickt Windows an die Nachrichtenwarteschlange der Anwendung (`Application Queue`). Jede Anwendung bekommt von Windows eine solche Nachrichtenwarteschlange zugeteilt.

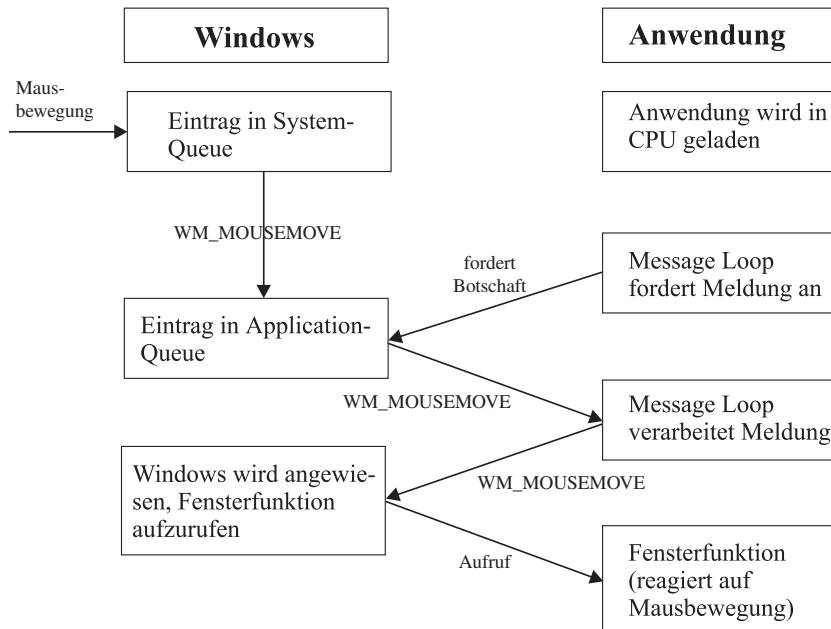
Um die eingetroffenen Nachrichten auszulesen, muß die Anwendung eine Schleife implementieren, in der sie ständig die Nachrichtenwarteschlange nach Nachrichten abfragt und diese gegebenenfalls ausliest. Dies ist die sogenannte »Message Loop«, die typischerweise wie folgt implementiert ist:

Die Message Loop

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

Sie ist allerdings nicht die Endstation der Nachrichtenverarbeitung, sondern lediglich eine Zwischenstation, denn das eigentliche Ziel einer Nachricht ist das Fenster, an das die Nachricht gerichtet ist. Hier geht es nicht mehr nur um das übergeordnete Fenster, das zur Identifizierung der Anwendung benötigt wurde. (Wenn der Anwender in ein untergeordnetes Fenster geklickt hat, beispielsweise ein Listefeld aus einem Dialog, ist das eigentliche Ziel-fenster das Listefeld-Steuerelement.) Die Verteilung der Nachrichten an die verschiedenen Fenster der Anwendung übernimmt die API-Funktion `DispatchMessage()`.

Bild 7.1:
Ereignisbe-
handlung un-
ter Windows



An diesem Punkt möchte ich erst einmal abbrechen. Es gäbe zwar noch einiges zur Nachrichtenverarbeitung unter Windows zu sagen, doch wir müssen uns nicht alle Grundlagen auf einmal aneignen. Zudem weichen die theoretischen Grundlagen ab dem Zeitpunkt, da die Nachrichten von der Anwendung entgegengenommen werden, mehr und mehr von der praktischen Erfahrung bei der MFC-Programmierung ab. Trotzdem werde ich im dritten Teil des Buches noch einmal auf die Grundlagen der Ereignisbehandlung zurückkommen und Sie weiter mit theoretischen Ausführungen und API-Implementierungen quälen. Schließlich sollen Sie sich nach Durcharbeiten dieses Buches als fortgeschrittener Windows-Programmierer bezeichnen dürfen, und dazu ist ein fundiertes Hintergrundwissen zu Windows einfach unerlässlich.

Lassen Sie sich nicht entmutigen, wenn Sie den bisherigen Ausführungen nicht in allem folgen konnten. Lesen Sie erst einmal weiter, und erfreuen Sie sich daran, wie leicht es Ihnen in den nachfolgenden Abschnitten fallen wird, in Ihren MFC-Programmen auf Benutzerereignisse und Nachrichten zu antworten. Schnuppern Sie danach schon einmal in das Kapitel 14 »Der von den Assistenten erzeugte Code« rein, und kehren Sie dann zu diesem Abschnitt zurück.

7.1.2 Jetzt übernimmt das Programm

Wir waren an dem Punkt stehengeblieben, wo die Nachrichten von der Anwendung aus der Nachrichtenwarteschlange (Application Queue) ausgefiltert und an die Fenster der Anwendung verteilt werden. In MFC-Programmen werden Fenster aber in Fensterklassen gekapselt. Die MFC-Klassen für die Fenster sind so implementiert, daß sie die Nachrichten entgegennehmen und entweder selbst beantworten oder an abgeleitete Fensterklassen – ja sogar an bestimmte Nicht-Fensterklassen des Anwendungsrüsts (Anwendungsklasse, Dokumentklasse) zur Verarbeitung weiterreichen.

Alles was wir tun müssen, ist, in unseren abgeleiteten Klassen Behandlungsmethoden zu implementieren und mit den Nachrichten zu verbinden. Da unser Code dabei mit bereits bestehendem Code aus den MFC-Klassen zusammenarbeiten muß, ist klar, daß wir bestimmte Konventionen beachten müssen. Diese Konventionen nehmen in unserem Programm die Gestalt einer sogenannten Antworttabelle (MESSAGE_MAP) an.

Zur Verdeutlichung ein Beispiel:

Wir wollen ein Programm schreiben, das, wo immer der Anwender mit der linken Maustaste in den Client-Bereich des Fensters klickt, eine Meldung ausgibt, die die Koordinaten des Mausklicks anzeigt.



- a) Der Client-Bereich des Hauptfensters wird von einem Ansichtsfenster ausgefüllt, also ist das Ansichtsfenster das Ziel des Mausereignisses.
- b) Wir setzen für die Klasse unseres abgeleiteten Ansichtsfensters eine neue Methode ein, die die aktuellen Mauskoordinaten bestimmt und ausgibt. Dies ist die Behandlungsmethode für unser Ereignis.
- c) Das Drücken der linken Maustaste löst unter Windows die Nachricht WM_LBUTTONDOWN aus. Daß diese Nachricht bei unserem Ansichtsfenster ankommt, dafür sorgen Windows und die Implementierung der MFC.

Was noch fehlt, ist eine Verbindung zwischen dem WM_LBUTTONDOWN-Ereignis und unserer Behandlungsmethode, damit jedesmal, wenn das Ansichtsfenster eine WM_LBUTTONDOWN-Nachricht empfängt, automatisch unsere Behandlungsmethode aufgerufen wird.

- d) Diese Verbindung wird durch einen Eintrag in einer Antworttabelle hergestellt.

7.1.3 Antworttabellen (MESSAGE_MAP)

In einer Antworttabelle wird festgehalten, welche Nachrichten eine Klasse bearbeitet und welche Methode für welche Nachricht aufgerufen werden soll.

Um eine Antworttabelle einzurichten, bedient man sich spezieller Makros.

- ✗ Zuerst deklariert man die Antworttabelle in der Deklaration der betreffenden Klasse.
- ✗ Dann definiert man die Antworttabelle in der Quelltextdatei (.cpp) der Klasse.

Um eine Behandlungsmethode für ein Ereignis einzurichten, geht man wie folgt vor:

- ✗ Zuerst deklariert man die Methode in der Deklaration der betreffenden Klasse.
- ✗ Dann definiert man die Methode in der Quelltextdatei (.cpp) der Klasse.

Schließlich trägt man die Quelltextdatei mit Hilfe bestimmter Makros in der Antworttabelle ein.

Sie wären dumm, wenn Sie all diese Arbeiten selbst ausführen würden. Wenn Sie Ihre Anwendung mit dem MFC-Anwendungs-Assistenten begonnen haben, sollten Sie jetzt den Klassen-Assistenten zur Einrichtung von Behandlungsmethoden nutzen. Dieser nimmt Ihnen die leidige Arbeit der Deklaration und Definition vollständig ab.



Für bestimmte Nachrichten sind in den Basisklassen der MFC bereits Behandlungsmethoden definiert. Sie erkennen diese Methoden daran, daß sie alle mit dem Präfix »On« beginnen. Beispielsweise ist für Ansichtsfenster die WM_PAINT-Nachricht mit der Methode OnDraw() verbunden. Wenn Sie die WM_PAINT-Nachricht selbst bearbeiten wollen, brauchen Sie nur die Methode OnDraw() zu überschreiben (siehe beispielsweise Abschnitt 5.5).

7.2 Der Klassen-Assistent ist eine große Hilfe

Um in einem mit dem Anwendungs-Assistenten erstellten Programm eine Behandlungsmethode für eine Windows-Nachricht einzurichten, bedient man sich üblicherweise des Klassen-Assistenten.

Der Klassen-Assistent ist eine große Hilfe

Rufen Sie dazu den Befehl ANSICHT/KLASSEN-ASSISTENT auf, und lassen Sie die Seite NACHRICHTENZUORDNUNGSTABELLEN anzeigen (eine Nachrichtenzuordnungstabelle ist das, was wir als Antworttabelle bezeichnen).

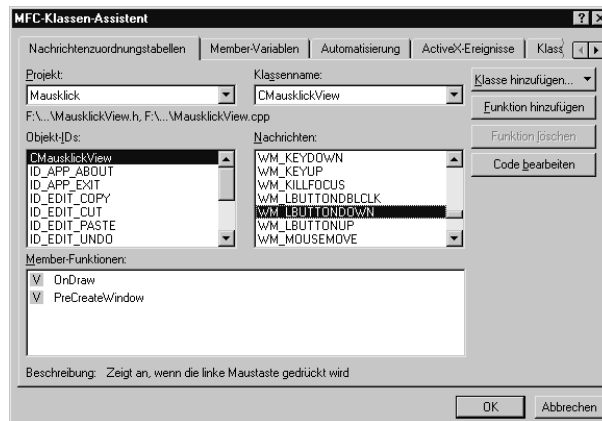


Bild 7.2:
Nachrichten
mit dem Klas-
sen-Assisten-
ten bearbeiten

Wenn Sie sich mit chinesischen Schriftzeichen beschäftigen, lernen Sie als erstes, daß man die Zeichen von links nach rechts und von oben nach unten schreibt. Ähnlich gehen Sie auf der Seite NACHRICHTENZUORDNUNGSTABELLEN des Klassen-Assistenten vor.

- ✗ Links oben wird Ihnen Ihr Projekt angezeigt. Das ist okay so.
- ✗ Rechts daneben werden im Listefeld KLASSENNAME die Klassen des Projekts aufgelistet. Hier wählt man die Klasse aus, in der man das Ereignis bearbeiten möchte.
- ✗ Darunter kommt zuerst das Feld OBJEKT-IDS. Hier werden die Objekte aufgeführt, für die Nachrichten bearbeitet werden können. Zuerst wird immer die Klasse aufgeführt. Darunter folgen Menübefehle, die man in der Klasse bearbeiten könnte (wobei gleich anzumerken ist, daß Menübefehle wie ID_APP_EXIT zum Beenden der Anwendung nicht in der Klasse des Ansichtsfensters, sondern passenderweise in der Klasse des Anwendungsobjekts behandelt werden sollten). Für Dialogklassen werden auch noch die Steuerelemente im Dialog aufgeführt. Wählen Sie das Objekt aus, für das Sie Ereignisse bearbeiten möchten.
- ✗ Daneben steht das Listefeld NACHRICHTEN. Hier werden die Nachrichten und Methoden angezeigt, die Sie für das zuvor ausgewählte Objekt bearbeiten können. Da wären zum einem die Windows-Nachrichten (erkenntlich am Präfix WM_), dann die bereits in den Basisklassen der MFC implementierten Behandlungsmethoden, die Sie überschreiben

können (wie zum Beispiel ONDRAW), und schließlich noch weitere virtuelle Methoden, die man ebenfalls überschreiben kann (wobei diese nichts mit der Nachrichtenbehandlung zu tun haben).

Zum Abschluß drücken Sie auf den Schalter FUNKTION HINZUFÜGEN. Der Klassen-Assistent legt Ihre Behandlungsmethode an und trägt sie in die Antworttabelle ein (die gegebenenfalls ebenso vom Assistenten angelegt wird). Ein Klick auf die Schaltfläche CODE BEARBEITEN führt Sie zur Definition der Methode, wo Sie sich nur noch um deren Implementierung zu kümmern brauchen. Das sollten wir uns jetzt auch mal in der Praxis anschauen.



Die Anzeige im Listenfeld NACHRICHTEN hängt auch von den Einstellungen im Feld FILTER FÜR NACHRICHTEN auf der Seite KLASSEN-INFO ab. Wenn Sie hier beispielsweise die Option FENSTER auswählen, bekommen Sie im Feld NACHRICHTEN wesentlich mehr Windows-Nachrichten angezeigt, als wenn die Option OBERSTER RAHMEN ausgewählt ist.

7.3 Mausereignisse

Die Maus ist ein äußerst vielseitiges Eingabegerät. Sie besitzt mehrere Tasten, die man drücken und wieder loslassen kann. Man kann die Maus bewegen, und man kann das Mausrad betätigen. Für alle diese Aktionen gibt es passende Windows-Nachrichten.

Tabelle 7.1:
Die wichtigsten Windows-Nachrichten für die Maus¹

Nachricht	Beschreibung
WM_LBUTTONDOWNBLCLK	linke Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_LBUTTONDOWN	linke Maustaste wurde gedrückt
WM_LBUTTONUP	linke Maustaste wurde losgelassen
WM_MBUTTONDOWNBLCLK	mittlere Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_MBUTTONDOWN	mittlere Maustaste wurde gedrückt

¹ Es gibt noch weitere Nachrichten für Mausclicks, die aber weniger interessant sind, beispielsweise Nachrichten wie WM_NCLBUTTONDOWN, die ausgelöst wird, wenn die linke Maustaste im Nicht-Clientbereich eines Rahmenfensters gedrückt wurde, oder WM_MBUTTONDOWN...-Nachrichten für die mittlere Maustaste.

Nachricht	Beschreibung
WM_MBUTTONDOWN	mittlere Maustaste wurde losgelassen
WM_MOUSEMOVE	Maus wurde bewegt
WM_MOUSEWHEEL	wird an das aktive Fenster gesendet, wenn das Mausrad rotiert
WM_RBUTTONDOWN	rechte Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_RBUTTONDOWN	rechte Maustaste wurde gedrückt
WM_RBUTTONUP	rechte Maustaste wurde losgelassen

*Tabelle 7.1:
Die wichtigsten Windows-Nachrichten für die Maus (Fortsetzung)*

Übung 7-1: Behandlung von Mausklicks

Um ein wenig Praxis zu bekommen, werden wir jetzt ein einfaches Programm implementieren, das Mausklicks im Client-Bereich seines Hauptfensters mit der Anzeige der Mauskoordinaten beantwortet.

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »Mausklick« an. Wie üblich verwenden wir eine SDI-Anwendung mit Doc/View-Unterstützung.
2. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ANSICHT), und lassen Sie die Seite NACHRICHTENZUORDNUNGSTABELLEN anzeigen.
3. Erweitern Sie Ihre Ansichts-klasse (CMAUSKlickVIEW) mit Hilfe des Klassen-Assistenten um eine Methode zur Behandlung des WM_LBUTTONDOWN-Ereignisses.

Wählen Sie in den Feldern KLASSENNAME und OBJEKT-IDS die Ansichts-klasse (CMAUSKlickVIEW) aus.

Im Feld NACHRICHTEN scrollen Sie bis zum Eintrag WM_LBUTTONDOWN, den Sie markieren. (Sollte die Nachricht nicht in der Liste angezeigt werden, kontrollieren Sie den Nachrichtenfilter auf der Seite KLASSEN-INFO.)

Drücken Sie den Schalter FUNKTION HINZUFÜGEN. Die Methode wird jetzt im Feld MEMBER-FUNKTIONEN hervorgehoben angezeigt.

Drücken Sie den Schalter CODE BEARBEITEN.

Im Editor wird Ihnen jetzt die folgende Definition der Behandlungsroutine angezeigt:


```

////////////////////////////////////
// CMAusklickView Nachrichten-Handler
void CMAusklickView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen

    CView::OnLButtonDown(nFlags, point);
}

```

In dieser Methode sollen wir den Code erzeugen, der die aktuellen Mauskoordinaten anzeigt. Dabei stellen sich zwei Probleme:

- ✗ Woher bekommen wir die aktuellen Mauskoordinaten?
- ✗ Wie gibt man Text aus?

Parameter von Behandlungsroutinen

Etliche Nachrichten transportieren nicht nur den Code, der das auslösende Ereignis beschreibt (beispielsweise `WM_LBUTTONDOWN` für einen Klick mit der linken Maustaste), sondern noch zusätzliche Informationen über das Ereignis. Die Nachrichten für Mausclickereignisse enthalten beispielsweise auch die Koordinaten der Maus und Flags, die anzeigen, ob bei Auslösung des Ereignisses eine der Maustasten oder die `[Strg]`- oder `[⇧]`-Taste gedrückt waren.

CPoint Diese zusätzlichen Informationen werden den Behandlungsmethoden zu den Nachrichten als Parameter übergeben. So werden die Mauskoordinaten für `WM_LBUTTONDOWN` der Behandlungsroutine `OnLButtonDown()` als *CPoint*-Parameter übergeben. Die x-Koordinate der Maus zum Zeitpunkt des Ereignisses steht in `point.x`, die y-Koordinate in `point.y`.

Meldungsfenster

Eine gute Möglichkeit, den Anwender über bestimmte Ereignisse, Fehler oder Probleme zu informieren, ist der Aufruf eines Meldungsfensters.

Meldungsfenster sind für die Programmierer ausgesprochen bequem zu handhaben, da sie bereits in Windows vordefiniert sind und durch den Aufruf einer einzigen Methode erzeugt und angezeigt werden können. Diese Methode ist:

```

int CWnd::MessageBox (LPCTSTR lpszText,
                      LPCTSTR lpszCaption = NULL,
                      UINT nType = MB_OK );

```

- ✗ `lpszText` ist der Text, den wir ausgeben möchten.
- ✗ `lpszCaption` ist der Text für den Titel des Meldungsfensters.
- ✗ `nType` ist eine Kombination aus Konstanten, die angeben, welche Schalter und welches Symbol im Meldungsfenster angezeigt werden soll.

Schalter	Symbole
MB_OK	MB_ICONEXCLAMATION
MB_OKCANCEL	MB_ICONINFORMATION
MB_RETRYCANCEL	MB_ICONQUESTION
MB_YES	MB_ICONSTOP
MB_YESNO	
MB_YESNOCANCEL	

Tabelle 7.2:
Konstanten für
den `nType`-
Parameter

Da `MessageBox()` eine Methode der Fensterklasse `CWnd` ist, ist sie nur in Methoden von Fensterklassen verfügbar. Um aus den Methoden anderer Klassen heraus Meldungsfenster anzuzeigen, verwendet man die globale Funktion `AfxMessageBox()`.



Übung 7-1: Fortsetzung

4. Setzen Sie den Code für die Behandlungsroutine `OnLButtonDown()` ein:

```
void CMausklickView::OnLButtonDown(UINT nFlags, CPoint point)
{
    char sKoord[100];
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    sprintf(sKoord, "Klick an Position: %d, %d",
        point.x, point.y);
    MessageBox(sKoord, "Linke Maustaste gedrückt",
        MB_OK | MB_ICONINFORMATION);

    CView::OnLButtonDown(nFlags, point);
}
```

5. Führen Sie das Programm aus (`[Strg] + [F5]`).

Bild 7.3:
Mausklick und
Meldungs-
fenster



7.4 Tastaturereignisse

Die Behandlung von Tastaturereignissen erfolgt ganz analog zur Behandlung der Mausereignisse.

Tabelle 7.3:
Die wichtig-
sten Tastatur-
ereignisse

Nachricht	Beschreibung
WM_KEYDOWN	Es wurde eine Taste gedrückt. (Die [Alt] -Taste wurde nicht gleichzeitig gedrückt.)
WM_KEYUP	Es wurde eine Taste losgelassen. (Die [Alt] -Taste wurde nicht gleichzeitig gedrückt.)

Übung 7-2: Behandlung von Tastaturereignissen

Das folgende Programm verhält sich, wie das obige Programm zu den Mausereignissen, nur daß das Meldungsfenster aufgerufen wird, wenn der Anwender die Taste »m« (oder **[⇧] + [M]**) drückt.

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »Tastatur« an. Wie üblich verwenden wir eine SDI-Anwendung mit Doc/View-Unterstützung.
2. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ANSICHT), und lassen Sie die Seite NACHRICHTENZUORDNUNGSTABELLEN anzeigen.
3. Erweitern Sie Ihre Ansichtsklasse (CTASTATURVIEW) mit Hilfe des Klassen-Assistenten um eine Methode zur Behandlung des WM_KEYDOWN-Ereignisses.

Wählen Sie in den Feldern KLASSENNAME und OBJEKT-IDS die Ansichtsklasse (CTASTATURVIEW) aus.

Im Feld NACHRICHTEN scrollen Sie bis zum Eintrag WM_KEYDOWN, den Sie markieren. (Sollte die Nachricht nicht in der Liste angezeigt

werden, kontrollieren Sie den Nachrichtenfilter auf der Seite KLASSEN-INFO.)

Drücken Sie den Schalter FUNKTION HINZUFÜGEN. Die Methode wird jetzt im Feld MEMBER-FUNKTIONEN hervorgehoben angezeigt.

Drücken Sie den Schalter CODE BEARBEITEN.

4. Setzen Sie den Code für die Behandlungsroutine `OnKeyDown()` auf:

```
void CTastaturView::OnKeyDown(UINT nChar, UINT nRepCnt,
                               UINT nFlags)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    if (nChar == 'M')
        MessageBox("Die Taste 'm' wurde gedrückt",
                    "Tastaturereignis",
                    MB_OK | MB_ICONINFORMATION);
    CView::OnKeyDown(nChar, nRepCnt, nFlags);
}
```



Bild 7.4:
Tastaturereignis und Meldungsfenster

7.5 Ganz wichtig: WM_PAINT

Die `WM_PAINT`-Nachricht wird von Windows an Fenster geschickt, um diesen mitzuteilen, daß sie sich selbst neu zeichnen sollen. Dies ist zum Beispiel der Fall, wenn ein Fenster vom Anwender aus dem Hintergrund wieder in den Vordergrund gehoben wird. Windows kann dann zwar den Fensterrahmen rekonstruieren, nicht aber den im Client-Bereich des Fensters angezeigten Text oder etwaige Grafiken.

Es schickt daher eine `WM_PAINT`-Nachricht an das betreffende Fenster. Aufgabe des Programmierers ist es, diese Nachricht abzufangen und mit einer Funktion zu verbinden, die den Fensterinhalt rekonstruiert (üblicherweise

geschieht dies, indem man Ausgaben in Fenster gleich in dieser Funktion vornimmt).

- ✗ In MFC-Anwendungen ohne Doc/View behandelt man die WM_PAINT-Nachricht in einer OnPaint()-Behandlungsmethode.
- ✗ In MFC-Anwendungen mit Doc/View wird die Nachricht von den Ansichts-klassen der MFC automatisch abgefangen und in der Methode OnDraw() behandelt. Diese Methode muß in abgeleiteten Klassen überschrieben werden. Der Anwendungs-Assistent tut dies automatisch beim Anlegen des Anwendungsgerüsts. Sie brauchen also nur noch zur Definition dieser Methode zu springen und Ihren Code hinzufügen.



Intern wird in den Ansichts-klassen des Doc/View-Gerüsts die WM_PAINT-Nachricht von einer OnPaint()-Methode abgefangen und bearbeitet. In dieser OnPaint()-Methode werden einige für das Zeichnen benötigte Vorarbeiten erledigt, dann wird die Methode OnDraw() aufgerufen, in die wir unseren Code einfügen.

Die folgenden zwei Beispiele sollen verdeutlichen, welche Bedeutung WM_PAINT für Ihre Anwendungen hat.

7.5.1 Außerhalb von OnDraw() zeichnen

Das folgende Programm soll als Antwort auf einen Mausklick an der Position der Maus einen Kreis einzeichnen. Die einzige Schwierigkeit dabei ist das Zeichnen.

Geräte-kontexte Zeichnenoperationen sind in Windows nur auf sogenannten Gerätekontexten möglich. Einen Gerätekontext stellt man sich dabei am besten als eine Art Leinwand vor. Bevor wir also in ein Ansichtsfenster zeichnen können, müssen wir uns einen Gerätekontext besorgen, der diesem Fenster entspricht. Für das Ansichtsfenster erreichen wir dies durch Instanziierung der Gerätekontextklasse CClientDC. Das Gerätekontextobjekt, das wir auf diesem Wege erhalten, enthält auch Methoden, die wir zum Zeichnen nutzen können – beispielsweise Ellipse().

Übung 7-3: Eine Scheibe zeichnen

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »WM_PAINT1« an. Wie üblich verwenden wir eine SDI-Anwendung mit Doc/View-Unterstützung.

Ganz wichtig: WM_PAINT

2. Rufen Sie den Klassen-Assistenten auf, und richten Sie in der Ansichtsklasse eine Behandlungsroutine für die WM_LBUTTONDOWN-Nachricht ein (siehe Übung 7.1).

3. Setzen Sie den folgenden Code für die Behandlungsroutine OnLButtonDown() ein:

```
void CWM_PAINT1View::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    CClientDC dc(this);
    dc.Ellipse(point.x-20, point.y-20,
               point.x+20, point.y+20);

    CView::OnLButtonDown(nFlags, point);
}
```

4. Führen Sie das Programm aus (**Strg** + **F5**).

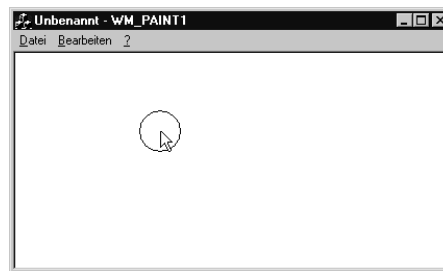


Bild 7.5:
In Nachrichten-
tenbehand-
lungsmethode
zeichnen

Spielen Sie ein wenig mit dem Programm herum. Wenn Sie mehrmals in das Fenster klicken, hinterlassen Sie eine kleine Folge von Kreisen. Wenn Sie aber das Fenster als Symbol in der Task-Leiste ablegen und wieder öffnen oder es kleiner und wieder größer machen oder mit anderen Fenstern verdecken und wieder in den Vordergrund bringen, verschwinden die Kreise.

Der Grund dafür ist, daß diese Fenstermanipulationen dazu führen, daß Windows das Fenster neu zeichnen muß. Da es nur den Fensterrahmen, nicht aber den Fensterinhalt rekonstruieren kann, schickt es eine WM_PAINT-Nachricht an das Fenster. Doch diese Nachricht wird in unserem Programm nicht beachtet – die Folge ist, daß das Fenster mit leerem Fensterinhalt rekonstruiert wird.

7.5.2 Innerhalb von OnDraw() zeichnen

Zum Vergleich wollen wir jetzt eine Scheibe in der OnDraw()-Methode zeichnen.

Übung 7-4: Eine zweite Scheibe zeichnen

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »WM_PAINT2« an. Wie üblich verwenden wir eine SDI-Anwendung mit Doc/View-Unterstützung.
2. Für die Ansichtsklasse wird die WM_PAINT-Nachricht bereits vom Anwendungsgerüst abgefangen und mit dem Aufruf der Methode OnDraw() verbunden. Gehen Sie also zur KLASSEN-Ansicht des Arbeitsbereichsfensters, expandieren Sie den Knoten der Ansichtsklasse, und doppelklicken Sie auf den Eintrag für OnDraw().
3. Setzen Sie Code für die Behandlungsroutine OnDraw() ein.

In der Methode OnDraw() müssen wir den Gerätekontext zum Zeichnen nicht selbst erzeugen, er wird uns bereits als Argument an pDC übergeben.

```
void CWM_PAINT2View::OnDraw(CDC* pDC)
{
    CWM_PAINT2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    pDC->Ellipse(50, 50, 70, 70);
}
```

4. Führen Sie das Programm aus (**Strg** + **F5**).

Spielen Sie ein wenig mit dem Programm herum. Minimieren Sie das Fenster, verdecken Sie es, verkleinern und vergrößern Sie es. Wenn Sie es wieder in den Vordergrund holen, wird jedesmal wieder der Kreis eingezeichnet.

Dafür mußten wir die Position des Kreises explizit festlegen. Schöner wäre es, wenn der Anwender mit der Maus in das Fenster klicken könnte und die Anwendung dann an dieser Stelle Kreise zeichnete, die auch beim Neuzeichnen des Fensters rekonstruiert würden. Überlegen Sie sich schon einmal, wie man dies bewerkstelligen könnte. In Kapitel 12 werden wir ein Beispiel dazu sehen.

7.6 Zeitgeber

Zum Schluß möchte ich Ihnen noch die Windows-Nachricht WM_TIMER vorstellen. Mit Hilfe dieser Nachricht können Sie eine Art Zeitgeber implementieren, der Ihr Programm in immer gleichen Abständen benachrichtigt. So könnten Sie Ihr Programm beispielsweise dazu bringen, alle 10 Sekunden einen Pieps auszustößen.

Wie geht man nun vor, wenn man einen bestimmten Code in regelmäßigen Abständen ausführen lassen will?

1. **Zuerst muß man den Zeitgeber in Gang setzen.** Dazu verwendet man die CWnd-Methode `SetTimer()`, die Windows mitteilt, daß das Fenster alle soundsoviel Millisekunden eine WM_TIMER-Nachricht erhalten soll.

```
UINT SetTimer(
    UINT nIDEvent,    // ID fuer den Zeitgeber
    UINT nElapse,     // Intervall in Millisekunden
    void (CALLBACK EXPORT* lpfnTimer) // Antwort-
                          (HWND, UINT, UINT, DWORD) // methode
);
```

Übergeben Sie dem ersten Parameter einen beliebigen Integer-Wert (allerdings dürfen keine zwei Zeitgeber die gleiche ID erhalten), `nElapse` übergeben Sie das Zeitintervall zwischen den WM_TIMER-Nachrichten (angegeben in Millisekunden). Dem dritten Parameter übergeben Sie den Wert `NULL`, damit die Nachricht in die Message Loop eingetragen und nicht direkt an eine Antwortmethode geschickt wird (reicht für unsere Zwecke).

2. **Richten Sie eine Bearbeitungsmethode zu dem WM_TIMER-Ereignis ein.**
3. **Löschen Sie den Zeitgeber** durch Aufruf der Methode `KillTimer()`.

```
BOOL KillTimer( int nIDEvent );
```

Übung 7-5: Ein Reaktionstestprogramm

Mit dem Programm, das wir jetzt erstellen werden, können Sie Ihre Reaktionsschnelligkeit testen. In Gang gesetzt wird der Reaktionstest durch einen Klick mit der rechten Maustaste. Das Programm wartet dann eine kurze Weile, wobei die Länge der Wartezeit zufällig variiert, damit der Anwender sich nicht auf die Wartezeit einstellen kann. Dann ertönt ein Piep, der an-

zeigt, daß es losgeht. Jetzt muß der Anwender so schnell es geht, mit der linken Maustaste in das Fenster klicken. In der Zwischenzeit empfängt das Programm WM_TIMER-Nachrichten (der Zeitgeber wurde eingerichtet, als der Anwender die rechte Maustaste drückte) und läßt für jede WM_TIMER-Nachricht eine rote Leuchtdiode aufleuchten (sprich wir zeichnen rote Ellipsen ein). An der Anzahl der aufleuchtenden Dioden kann der Anwender ablesen, wie schnell er reagiert hat.

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »Reaktion« an. Wie üblich verwenden wir eine SDI-Anwendung mit Doc/View-Unterstützung.
2. Geben Sie die anfängliche Fenstergröße vor. Ein schmales hohes Fenster paßt gut zu der Anwendung.

```

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.
    cs.x = 75;
    cs.y = 50;
    cs.cx = 200;
    cs.cy = 500;
    return TRUE;
}

```

3. In der Deklaration der Dokumentklasse CReaktionDoc richten Sie eine Elementvariable m_nSpots ein, die als Zähler für die anzuzeigenden Leuchtdioden dient.

```

class CReaktionDoc : public CDocument
{
    ....
    // Attribute
public:
    int m_nSpots;
    ...
}

```

4. Im Konstruktor setzen Sie den m_nSpots-Zähler auf 0.

```

CReaktionDoc::CReaktionDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion
    // einfügen
    m_nSpots = 0;
}

```

5. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für die WM_RBUTTONDOWN-Nachricht ein.

In der Methode richten Sie den Zeitgeber ein, warten ein paar Sekunden und lassen dann den Computer-Lautsprecher ertönen, auf den der Anwender reagieren soll.

```

void CReaktionView::OnRButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    CReaktionDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    SetTimer(1, 50, NULL);
    pDoc->m_nSpots = 0;
    Invalidate();
    UpdateWindow();
    // kurze Zeit warten
    srand((unsigned)time(NULL));
    int wait = (rand() % 7 + 3) * CLOCKS_PER_SEC;
    time_t start = clock();
    while(clock() < start+wait);
    Beep(1000, 100);

    CView::OnRButtonDown(nFlags, point);
}

```

6. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für die WM_LBUTTONDOWN-Nachricht ein.

Der Anwender hat reagiert! Löschen Sie den Zeitgeber!

```

void CReaktionView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    KillTimer(1);
}

```

```
CView::OnLButtonDown(nFlags, point);
}
```

7. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für die WM_TIMER-Nachricht ein.

Als Antwort auf das WM_TIMER-Ereignis inkrementieren Sie den Zähler `m_nSpots` und lassen das Fenster neu zeichnen.

```
void CReaktionView::OnTimer(UINT nIDEvent)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    CReaktionDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDoc->m_nSpots++;
    Invalidate();
    UpdateWindow();

    CView::OnTimer(nIDEvent);
}
```

8. Lassen Sie in der `OnDraw()`-Methode des Ansichtsfensters die Scheiben für die Leuchtdioden anzeigen.

```
void CReaktionView::OnDraw(CDC* pDC)
{
    CReaktionDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    CBrush *pBrush =
        pDC->SelectObject(new CBrush(0x000000FF));
    CRect rect;
    GetClientRect(&rect);
    int width = rect.Width();
    int height = rect.Height();
    for(int i=0; i < pDoc->m_nSpots; i++)
        pDC->Ellipse( width/2 - 10.0, 10 + i * height/20,
                     width/2 + 10, 10 + i * height/20 + 10);
    pDC->SelectObject(pBrush);
}
```

9. Führen Sie das Programm aus (**Strg** + **F5**).

7.7 Zusammenfassung

Alle Aktionen des Anwenders (Mausklicks, Aufruf von Menübefehlen, Aktivieren eines Fensters) werden von Windows abgefangen und in Form von Nachrichten an die betroffenen Anwendungen weitergeleitet. Aufgabe des Programms ist es, die Nachrichten abzufangen und die für das Programm interessanten Nachrichten einer passenden Methode zur Behandlung des Ereignisses zuzuführen. Zur Einrichtung der Behandlungsmethoden verwenden Sie den Klassen-Assistenten.

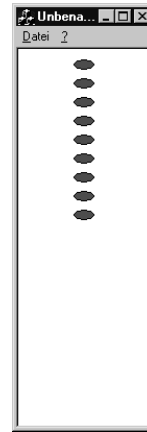


Bild 7.6:
Das Reaktions-
programm

Der Klassen-Assistent legt mit Hilfe der Makros `DECLARE_MESSAGE_MAP`, `BEGIN_MESSAGE_MAP` und `END_MESSAGE_MAP` Antworttabellen an. Antworttabellen sind immer bestimmten Klassen zugeordnet. Für jede behandelte Nachricht wird ein Eintrag in die Antworttabelle eingetragen, der eine Verbindung zwischen der `WM_`-Nachricht und der zugehörigen Behandlungsmethode herstellt. Der Klassen-Assistent richtet auch gleich die Behandlungsmethode ein und führt Sie bei Bedarf direkt zur Definition der Methode.

7.8 Fragen

1. Was ist die *Message Loop*?
2. Wie richtet man mit Hilfe des Klassen-Assistenten Behandlungsmethoden zu Nachrichten ein?
3. Nennen Sie einige interessante Mausnachrichten!
4. Wann wird die Nachricht `WM_PAINT` ausgelöst?
5. Wie behandelt man die Nachricht `WM_PAINT`?
6. Wie kann man in einem Windows-Programm in regelmäßigen Abständen bestimmte Arbeiten ausführen?

7.9 Aufgaben

1. Schreiben Sie eine Anwendung, die ein Meldungsfenster anzeigt, wenn der Anwender mit der linken Maustaste in das Fenster klickt, und ein zweites Meldungsfenster anzeigt, wenn der Anwender mit der linken Maustaste doppelklickt.
2. Informieren Sie sich im Anhang dieses Buches oder der Online-Hilfe über die verschiedenen Windows-Nachrichten.
3. Legen Sie eine Kopie des Anwendungsgerüsts aus der Aufgabe 3 des Kapitels 6 an. Beantworten Sie in der Anwendung das Drücken der linken Maustaste mit dem Einzeichnen eines Kreises an der Position des Mausclicks.

Um eine Kopie anzulegen, beginnen Sie mit einem leeren Win32-Anwendungsprojekt. Rufen Sie den Menübefehl PROJEKT/EINSTELLUNGEN auf, und wählen Sie auf der Seite ALLGEMEIN im Feld MICROSOFT FOUNDATION CLASSES eine der Optionen zur Verwendung der MFC aus. Kopieren Sie die Quelltextdateien und die Header-Datei einschließlich der Ressourcendateien (.rc, .ico) in das Verzeichnis des Projekts, und nehmen Sie die Quelldateien mit Hilfe des Befehls PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN in das Projekt auf.

Für die WM_LBUTTONDOWN-Nachricht müssen Sie eine Antworttabelle für die Ansichtsfensterklasse einrichten. Rufen Sie dazu irgendwo in der Klassendeklaration das Makro DECLARE_MESSAGE_MAP() auf. In der Quelltextdatei definieren Sie die Antworttabelle mit den Makros BEGIN_MESSAGE_MAP(CAnsicht, CView) und END_MESSAGE_MAP(). In der Tabelle rufen Sie das Makro ON_WM_LBUTTONDOWN() auf, das für Sie die Verbindung zwischen WM_LBUTTONDOWN und Ihrer OnLButtonDown()-Methode herstellt. (Der Name OnLButtonDown() ist vorgeschrieben und darf nicht verändert werden.)

```
BEGIN_MESSAGE_MAP(CAnsicht, CView)
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
```

7.10 Lösungen zu den Aufgaben

Zu 3: Anwendungsgerüst mit Nachrichtenbehandlung

Die Header-Datei:

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

class CAnsicht : public CView {
public:
    CAnsicht(CFrameWnd *parent);
protected:
    afx_msg void OnDraw(class CDC *);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    DECLARE_MESSAGE_MAP()
};

class CRahmenfenster : public CFrameWnd {
public:
    CRahmenfenster();
protected:
    CStatusBar m_wndStatusBar;
    int OnCreate(LPCREATESTRUCT cs);
    DECLARE_MESSAGE_MAP()
};

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

Die Quelltextdatei

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

// Anwendungs-Objekt erzeugen
CMyApp Anwendung;
static UINT indicators[] = {ID_SEPARATOR, };
// Ansichtsfenster
BEGIN_MESSAGE_MAP(CAnsicht, CView)
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
```

```

CAnsicht::CAnsicht(CFrameWnd *parent) {
    Create(0, 0, WS_CHILD | WS_VISIBLE,
           CRect(), parent, AFX_IDW_PANE_FIRST);
}
// muss überschrieben werden, da sonst abstrakte Klasse
void CAnsicht::OnDraw(class CDC *dc) {
}
void CAnsicht::OnLButtonDown(UINT nFlags, CPoint point) {
    CClientDC dc(this);
    dc.Ellipse(point.x-20, point.y-20, point.x+20, point.y+20);
}

// Rahmenfenster
BEGIN_MESSAGE_MAP(CRahmenfenster, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()
CRahmenfenster::CRahmenfenster() {
    LPCTSTR wndClass = AfxRegisterWndClass(NULL,
                                           AfxGetApp()->LoadCursor(IDC_ARROW),
                                           (HBRUSH) (COLOR_WINDOW + 1),
                                           AfxGetApp()->LoadIcon(IDI_ICON1));

    // Fenster erzeugen
    Create(wndClass, "Rahmen mit View",
           WS_OVERLAPPEDWINDOW & ~WS_THICKFRAME,
           CRect(20, 20, 300, 200));
    // Ansicht erzeugen
    new CAnsicht(this);
}
// Eingebettete Rahmenfensterobjekte erzeugen
int CRahmenfenster::OnCreate(LPCREATESTRUCT cs) {
    if(CFrameWnd::OnCreate(cs) == -1)
        return -1;
    m_wndStatusBar.Create(this);
    m_wndStatusBar.SetIndicators(indicators,
                                sizeof(indicators)/sizeof(UINT));

    return 0;
}
// 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;  
}
```


Menüs, Symbolleisten, Tastaturkürzel

Die meisten Anwendungen stellen ihre Befehle über eine Menüleiste zur Verfügung, die unter der Titelleiste des Hauptfensters der Anwendung angezeigt wird. Unterstützt wird die Menüleiste meist durch

- ✗ eine (oder mehrere) passende Symbolleiste(n), über die ausgesuchte Befehle der Menüleiste mit einem einzigen Mausklick aufgerufen werden können
- ✗ Tastaturkürzel, die den schnellen Aufruf von wichtigen Befehlen über die Tastatur erlauben
- ✗ eine Statusleiste, in der Hilfetexte zu den einzelnen Menübefehlen angezeigt werden

Umfangreichere Anwendungen ergänzen Ihr Menüsystem meist durch Kontextmenüs, die der Anwender durch Klick mit der rechten Maustaste aufrufen kann.

Menüs sind seit Urzeiten typischer Bestandteil von Windows-Programmen. Die Anwender haben sich an den Gebrauch der Menüs gewöhnt. Wenn es darum geht, eine große Zahl von Programmfunktionen als Befehle zur Verfügung zu stellen, sind Menüs praktisch unschlagbar. Doch es gibt auch Alternativen. Viele Spiele verwenden eine rein grafische Benutzeroberfläche ohne Menüs. Auch andere Programme, beispielsweise Lernsoftware für Kinder, setzen gelegentlich auf anklickbare Grafiken und Bilder statt auf hierarchisch strukturierte Menüstrukturen. Denkbar ist auch, daß wir in



Zukunft mehr Programme sehen, deren Benutzeroberflächen wie Webseiten (Navigation durch sequentielle Links) aufgebaut sind.

Unabhängig von Trends und Traditionen bleibt aber festzustellen, daß Menüs zu den wichtigsten Elementen der Benutzeroberflächengestaltung gehören und es daher verdienen, in einiger Ausführlichkeit besprochen zu werden.

Sie lernen in diesem Kapitel:

- ✗ Wie man mit Hilfe des MFC-Anwendungs-Assistenten in wenigen Minuten ein Anwendungsgerüst mit komplettem Menüsystem anlegt.
- ✗ Wie man Menüressourcen bearbeitet und welche Einstellmöglichkeiten man dabei hat
- ✗ Wie man Symbolleisten- und Tastaturkürzel-Ressourcen bearbeitet
- ✗ Wie man Behandlungsroutinen für Menübefehle implementiert
- ✗ Wie man Befehle deaktiviert
- ✗ Wie man Kontextmenüs einrichtet

8.1 Eine komplette Menüunterstützung

Wenn Sie Ihre Projekte mit dem MFC-Anwendungs-Assistenten beginnen, legt dieser automatisch eine Standardmenüleiste und optional auch eine Symbolleiste und Statusleiste für Sie an, die Sie dann ohne große Mühe an Ihre speziellen Bedürfnisse anpassen können.

Eine komplette Menüunterstützung

Übung 8-1: Anwendungsgerüst mit kompletter Menüunterstützung

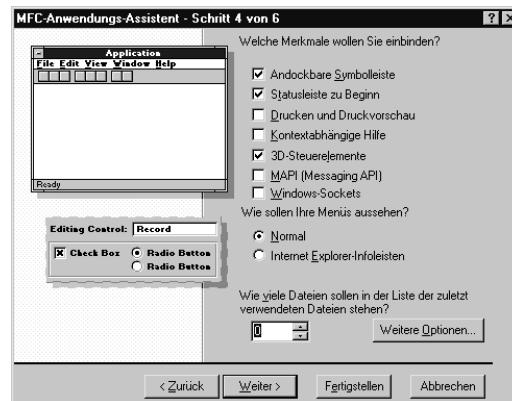


Bild 8.1:
Einstellungen
im Anwen-
dungs-
Assistent

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neues Projekt namens »Menue« an.

Im Schritt 1 entscheiden Sie sich wie üblich für eine SDI-Anwendung mit Doc/View-Unterstützung.

Im Schritt 4 lassen Sie die Optionen für die ANDOCKBARE SYMBOLLEISTE und die STATUSLEISTE aktiviert und schalten lediglich die Optionen für nicht gewünschte Menübefehle (Drucken, Dateiliste) aus.

2. Führen Sie das Programm aus (**Strg** + **F5**).

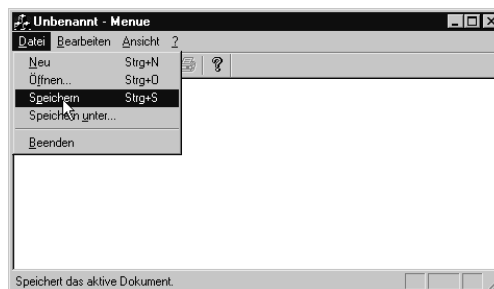


Bild 8.2:
Das vom Assi-
stenten er-
stellte Anwen-
dungsgerüst

Wie Sie selber feststellen können, ist das vom Assistenten eingerichtete Menüsystem schon ziemlich perfekt. Gut, die meisten Menübefehle sind nur teilweise oder gar nicht implementiert – so ruft der Befehl DATEI/ÖFFNEN zwar den Dialog zum Öffnen einer Datei auf, doch geladen wird die ausgewählte Datei nicht –, aber der Rest ist beeindruckend:

- ✗ Wir haben eine voll funktionsfähige Menüleiste mit mehreren Popup-Menüs und etlichen Menübefehlen in den Popup-Menüs.
- ✗ Die Menübefehle können über Alt-Tastenkombinationen (beispielsweise **[Alt] + [D]**, **[F]** für DATEI/ÖFFNEN) und zum Teil auch über Tastaturkürzel (beispielsweise **[Strg] + [S]** für DATEI/SPEICHERN) aufgerufen werden.
- ✗ Wir haben eine Symbolleiste mit Schaltflächen für die wichtigsten Befehle und Quickinfos.
- ✗ Wir haben eine Statusleiste am unteren Rand des Hauptfensters, in der Hilfetexte zu den einzelnen Menübefehlen angezeigt werden.

Was uns bleibt, ist zu staunen und uns zu wundern, wie der Anwendungs-Assistent all dies hingezaubert hat.

Lassen Sie sich nicht bluffen. Die Einrichtung des Menüsystems mit Symbol- und Statusleiste, Tastaturkürzel und Hilfetexten ist dem Assistenten gar nicht so schwer gefallen. Schauen wir ihm einmal auf die Finger.

Menü, Symbolleiste und Statusleiste sind Elemente, deren grundlegende Funktionalität bereits in Windows und den MFC-Klassen implementiert ist. Um ein Popup-Menü der Menüleiste aufzuklappen oder Menübefehle über **[Alt]**-Tastenkombinationen oder Tastaturkürzel aufzurufen oder Hilfetexte in die Statusleiste einzublenden, braucht der Anwendungs-Assistent keinen speziellen Code mehr aufzusetzen. Diese Funktionalität ist schon vollständig vorhanden. Die Informationen, wie das Menü und die Symbolleiste aufgebaut zu sein haben, wie die Menübefehle heißen und welche **[Alt]**-Tastenkombinationen, Tastaturkürzel und Hilfetexte es zu den Menübefehlen gibt, sind vollständig in Ressourcen abgelegt. Diese Ressourcen legt der Assistent nicht einmal selbst an; er entnimmt sie speziellen DLLs, die mit Visual C++ zusammen ausgeliefert werden. Alles, was der Assistent im Grunde zu tun hat, ist, auf der Grundlage dieser Ressourcen Menü, Symbolleiste und Statusleiste zu erzeugen.

Erzeugung des Menüs

Menüs, die auf der Grundlage von Ressourcen erstellt werden, bereiten praktisch keine Mühe. Alles was nötig ist, um das Menü zu erzeugen und mit dem Rahmenfenster zu verbinden, ist, die Ressourcen-ID an die Create()-Methode des Rahmenfensterobjekts zu übergeben. (Zur Erinnerung: Für Fensterobjekte wird erst eine Instanz der Fensterklasse erzeugt und dann die Create()-Methode des Objekts aufgerufen.)

----- Menü können auch mit Hilfe der Klasse CMenu erzeugt und manipuliert werden. Dies ist beispielsweise dann interessant, wenn man Menüs zur Laufzeit austauschen oder verändern möchte. -----



In MFC-Anwendungen wird die Create()-Methode für Rahmenfenster meist jedoch nicht direkt aufgerufen, sondern entweder über

✗ die Methode CFrameWnd::LoadFrame()

```
// IDR_MAINFRAME ist ID für Menü-Ressource
CMainFrame *pMainWnd = new CMainFrame;
pMainWnd->LoadFrame(IDR_MAINFRAME);
m_pMainWnd = pMainWnd;
```

oder

✗ die Erzeugung einer Dokumentvorlage für MFC-Anwendungen mit Dokument/Ansicht-Architektur.

```
// IDR_MAINFRAME ist ID für Menü-Ressource
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CMen1Doc),
    RUNTIME_CLASS(CMainFrame),
    RUNTIME_CLASS(CMen1View));
AddDocTemplate(pDocTemplate);
```

----- Die Tastaturkürzel werden zusammen mit dem Menü erzeugt – vorausgesetzt, die Tastaturkürzel-Ressource hat die gleichen Ressourcen-ID wie das Menü. -----



Erzeugung der Symbolleiste

Für die Symbolleiste muß zuerst ein Objekt der MFC-Klasse `CToolBar` eingerichtet werden. Dies geschieht einfach, indem man in der Klasse des Rahmenfensters ein Objekt der Klasse `CToolBar` deklariert.

```
class CMainFrame : public CFrameWnd
{
...
protected: // Eingebundene Elemente
    CToolBar m_wndToolBar;
...
};
```

Das `CToolBar`-Objekt wird dann automatisch im Zuge der Instanziierung des Rahmenfensterobjekts erzeugt.

In einem zweiten Schritt wird für das leere `CToolBar`-Objekt `m_wndToolBar` die eigentliche Symbolleiste erzeugt. Dies geschieht in der Rahmenfenster-methode `OnCreate()`.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
...

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD
        | WS_VISIBLE | CBR_S_TOP | CBR_S_GRIPPER
        | CBR_S_TOOLTIPS | CBR_S_FLYBY | CBR_S_SIZE_DYNAMIC)
        || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Symbolleiste konnte nicht erstellt \
werden\n");
        return -1; // Fehler bei Erstellung
    }
}
```

Die Methode `CreateEx()` erzeugt das Symbolleisten-Steuerelement, wobei ein Zeiger auf das übergeordnete Fenster (`this` verweist hier auf das Rahmenfenster) und eine Kombination von Konstanten für Symbolleistenstile (`TBSTYLE_`) und Symbolleistenschalterstile (`CBRS_`) übergeben werden. Interessant sind hierbei beispielsweise die Stile `CBRS_TOOLTIPS`, zur Unterstützung von Quickinfos, und `CBRS_FLYBY`, für Hilfetexte in der Statusleiste.

Mit Hilfe der Methode `LoadToolBar()` wird das Bitmap für die Symbolleiste geladen.

Erzeugung der Statusleiste

Analog zur Symbolleiste wird zuerst ein Objekt der MFC-Klasse `CStatusBar` eingerichtet.

```
class CMainFrame : public CFrameWnd
{
...
protected: // Eingebundene Elemente
    CStatusBar m_wndStatusBar;
...
};
```

Das `CStatusBar`-Objekt wird dann automatisch im Zuge der Instanziierung des Rahmenfensterobjekts erzeugt.

In einem zweiten Schritt wird für das leere `CStatusBar`-Objekt `m_wndStatusBar` die eigentliche Statusleiste erzeugt. Dies geschieht in der Rahmenfenstermethode `OnCreate()`:

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
...
    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
    }
...
}
```

Hinter dem Argument `indicators`, das der `CStatusBar`-Methode `SetIndicators()` übergeben wird, verbirgt sich ein Array mit Ressourcen-IDs, das global in der Quelltextdatei der Rahmenfensterklasse definiert ist:

```
static UINT indicators[] =
{
    ID_SEPARATOR, // Statusleistenanzeige
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

Die erste Ressourcen-ID, `ID_SEPARATOR`, sorgt dafür, daß im ersten Feld der Statusleiste die Hilfetexte zu den Menübefehlen angezeigt werden können.

Die weiteren Ressourcen-IDs erzeugen in der Statusleiste Felder, die den Anwender über den Zustand bestimmter Tasten (Großschreibung, Numerische Tastatur, Rollen) informiert.

Anpassung des Menüs

Die weitere Bearbeitung des vom Assistenten erzeugten Menüs erfolgt auf zwei Ebenen:

- ✕ Die Überarbeitung der Ressourcen
- ✕ Die Einrichtung der Behandlungsmethoden zu den Menübefehlen

8.2 Bearbeitung der zugehörigen Ressourcen

Selten wird man das Menü genauso übernehmen, wie es vom Assistenten angelegt wurde. Entweder wird man es um weitere Menübefehle erweitern, oder man wird Menübefehle löschen und durch andere Befehle ersetzen. Auch die Zusammensetzung der Symbolleiste und der Tastaturkürzel wird nicht jedem zusagen, und die Hilfetexte für die Quickinfos und die Statusleiste ändern sich natürlich, wenn sich die Menübefehle ändern.

All diese Anpassungen werden in den Ressourcen vorgenommen, und zwar in:

- ✕ der Menü-Ressource, die den Aufbau des Menüs definiert und die Menübefehle mit Ressourcen-IDs verbindet
- ✕ der Symbolleisten-Ressource, die das Bitmap für die Symbolleiste definiert und den einzelnen Schaltflächen in der Symbolleiste Ressourcen-IDs zuweist. (Schaltflächen, die zu Menübefehlen korrespondieren, erhalten die gleiche Ressourcen-ID wie der Menübefehl).
- ✕ der Stringtabelle, in der die Hilfetexte für die Menübefehle und Schaltflächen definiert werden.

8.2.1 Anpassung des Menüs

Im Menü-Editor werden Menüleisten Menü für Menü und Befehl für Befehl aufgebaut.

Um die folgenden Beschreibungen nachvollziehen zu können, sollten Sie die Menü-Ressource Ihrer *Menue*-Anwendung in den Menü-Editor laden. Öffnen Sie dazu die RESSOURCEN-Ansicht des Arbeitsbereichfensters, ex-

pandieren Sie den Ordner MENU, und doppelklicken Sie auf den Eintrag IDR_MAINFRAME.

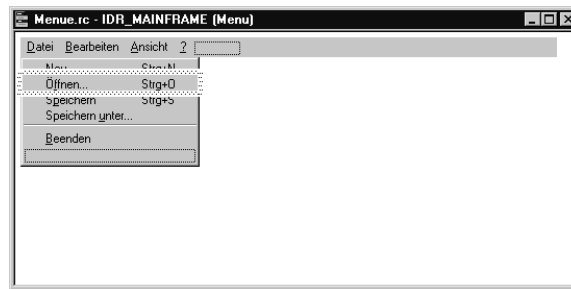


Bild 8.3:
Anzeige eines
Menüs im
Menü-Editor

Im Menü-Editor sehen Sie das Menü, so wie es sich später im Rahmen Ihrer Anwendung darstellen wird. Darüber hinaus können die einzelnen Drop-down-Menüs in der Menüleiste wie gewohnt mit der Maus geöffnet werden. Auch die Auswahl von Menüs und Menübefehlen kann wie bei einem echten Menü mit der Maus oder den Pfeiltasten der Tastatur erfolgen (allerdings nicht über Tastaturkürzel oder **[Alt]**-Tastenkombinationen).

Welche Möglichkeiten Sie haben, die Menüleiste anzupassen, können Sie Tabelle 8.1 entnehmen.

Aktion	Ausführung
Menüelement hinzufügen	Zum Hinzufügen neuer Menüelemente (Dropdown-Menüs, Menüeinträge, Trennlinien, Untermenüs) stehen Ihnen die leeren Schablonen am Ende der Menüleiste, am unteren Ende jedes Dropdown-Menüs und links von etwaigen Untermenüeinträgen zur Verfügung. Doppelklicken Sie auf die passende Schablone, und konfigurieren Sie in dem aufspringenden Dialogfeld das neue Menüelement.
Menüelement löschen	Markieren Sie das Menüelement, und drücken Sie die [Entf] -Taste.
Menüelement verschieben	Ziehen Sie das Menüelement einfach mit der Maus an die gewünschte Position.
Menüelement konfigurieren	Doppelklicken Sie auf das Menüelement, um das Eigenschaften-Dialogfeld zu dem Menüelement aufzurufen. Hier können Sie vom Titel des Menüeintrags bis zu den Hilfetexten alle wichtigen Einstellungen zum Menüelement vornehmen.

Tabelle 8.1:
Bearbeitung
von Menü-
leisten

Bild 8.4:
Eigenschaften-
Dialog für
Menüelemente



Welche Möglichkeiten Sie im Eigenschaften-Dialogfeld zur Konfiguration der Menüelemente haben, können Sie Tabelle 8.2 entnehmen.

Tabelle 8.2:
Konfiguration
von Menü-
elementen

Option	Beschreibung
ID	Ressourcen-ID für den Menübefehl; Dropdown-Menüs benötigen keine ID.
Titel	Titel des Menüeintrags. Menüelemente sollten auch durch Drücken der [Alt]-Taste und eines bestimmten Buchstabens aufgerufen werden können. Welcher Buchstabe für das Menüelement gedrückt werden soll, legen Sie fest, indem Sie im Titel des Menüeintrags dem Buchstaben ein Kaufmännisches Und (&) voranstellen. Im Menü wird der Buchstabe durch einen Unterstrich gekennzeichnet. Wenn Sie vorhaben, ein Tastaturkürzel für einen Menübefehl einzurichten, sollten Sie die Tastenkombination des Tastaturkürzels rechtsbündig im Titel des Menüeintrags anzeigen. Trennen Sie dazu den Titel und die Tastenkombination durch \t. (Im Gegensatz zu den Tastenkombinationen ist das Tastaturkürzel damit aber noch nicht funktionsfähig; dazu muß es in der Tastaturkürzel-Ressource eingetragen werden.)
Trennlinie	Das Menüelement ist eine Trennlinie.
Popup	Das Menüelement ist ein Dropdown-Menü.
Inaktiv	Das Menüelement ist deaktiviert (kann nicht aufgerufen werden).
Aktiviert	Das Menüelement wird mit einem Häkchen versehen.
Grau	Das Menüelement wird grau dargestellt.
Hilfe	Ordnet das Menüelement rechtsbündig an. Wurde früher verwendet, um das Hilfemenü rechtsbündig in der Menüleiste anzuzeigen.
Anhalten	Zum Umbrechen der Menüleiste.
Statuszeilentext	Der hier eingegebene Text wird in der Statuszeile eingeblendet, wenn das Menüelement ausgewählt ist. Zusätzlich können Sie eine Quickinfo-Text anhängen. Trennen Sie beide Hilfetexte durch \n.

Übung 8-2: Menü anpassen

1. Wenn Sie es noch nicht getan haben, laden Sie jetzt das IDR_MAINFRAME-Menü der *Menue*-Anwendung in den Menü-Editor.
2. Überarbeiten Sie das Menü.

Wir bescheiden uns mit zwei Dropdown-Menüs: *Programm* und *Text*. Löschen Sie die überflüssigen Dropdown-Menüs. Legen Sie das Menü *Text* zur Übung am besten ganz neu an.

Das Menü *Programm* soll nur den Menübefehl *Beenden* enthalten. Achten Sie darauf, daß dieser Menübefehl die Ressourcen-ID `ID_APP_EXIT` hat, damit bei Aufruf des Menübefehls die Anwendung geschlossen wird.

Das Menü *Text* soll die Menübefehle mit den Titeln *Anzeigen* und *Löschen* enthalten. Zwischen den beiden Menübefehlen fügen wir eine Trennlinie ein.

Achten Sie darauf, daß die Menüelemente mit passenden IDs und Hilfetexten für Statusleiste und Quickinfo ausgestattet sind. Alle Menüelemente sollten über `[Alt]`-Tastenkombinationen aufrufbar sein. Wenn Sie wollen, können Sie auch Tastaturkürzel vorsehen.

In den nachfolgenden Übungen werden Sie erfahren, wie Sie Tastaturkürzel, Symbolleiste und Stringtabelle an das überarbeitete Menü angleichen.

8.2.2 Anpassung der Tastaturkürzel

Öffnen Sie die RESSOURCEN-Ansicht des Arbeitsbereichsfensters, expandieren Sie den Ordner ACCELERATOR, und doppelklicken Sie auf den Eintrag `IDR_MAINFRAME`.

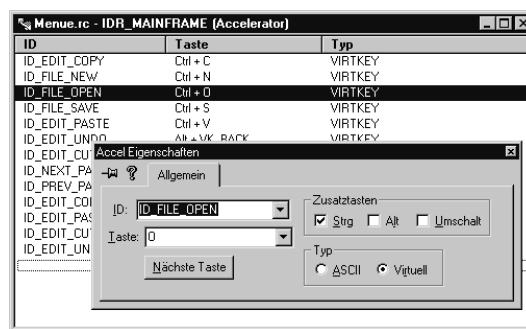


Bild 8.5:
Eigenschaften-
Dialog des
Tastaturkürzel-
Editors

Welche Möglichkeiten Sie im Tastaturkürzel-Editor haben, können Sie Tabelle 8.3 entnehmen.

Tabelle 8.3:
Bearbeitung
von Tastatur-
kürzeln

Aktion	Ausführung
Tastaturkürzel hinzufügen	Doppelklicken Sie auf die leere Schablone am unteren Ende der Tabelle, und richten Sie in dem aufspringenden Dialogfeld das neue Tastaturkürzel ein.
Tastaturkürzel löschen	Markieren Sie das Tastaturkürzel und drücken Sie die [Entf] -Taste.
Tastaturkürzel konfigurieren	Doppelklicken Sie auf das Tastaturkürzel, um das Eigenschaften-Dialogfeld zu dem Tastaturkürzel aufzurufen. Wählen Sie im Feld ID die Ressourcen-ID des Menübefehls aus, der über das Tastaturkürzel aufgerufen werden soll. Geben Sie die gewünschte Tastenkombination ein. Drücken Sie dazu den Schalter NÄCHSTE TASTE. Danach können Sie die Tastenkombination direkt über die Tastatur eingeben (so wie sie später ausgelöst wird), und der Tastaturkürzel-Editor setzt die entsprechenden Optionen im Dialogfeld. Schließen Sie das Eigenschaften-Dialogfeld.

Übung 8-3: Tastaturkürzel anpassen

1. Wenn Sie es noch nicht getan haben, laden Sie jetzt die IDR_MAINFRAME-Tastaturkürzelressource der Menue-Anwendung in den Tastaturkürzel-Editor.
2. Löschen Sie alle Tastaturkürzel, die Sie nicht benötigen, und richten Sie statt dessen Tastaturkürzel für Ihre Menübefehle ein (sofern Sie in Übung 8.2 Tastaturkürzel für Menübefehle angegeben haben).

8.2.3 Anpassung der Symbolleiste

Öffnen Sie die Ressourcen-Ansicht des Arbeitsbereichsfensters, expandieren Sie den Ordner TOOLBAR, und doppelklicken Sie auf den Eintrag IDR_MAINFRAME.

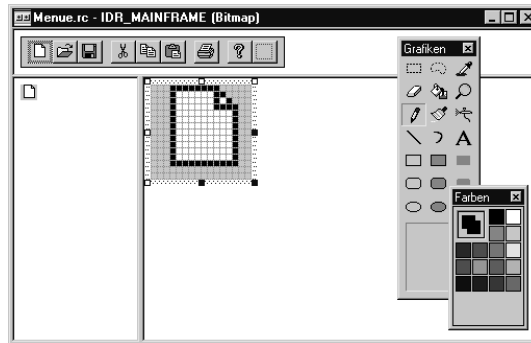


Bild 8.6:
Der Symbolleis-
ten-Editor

Im Symbolleisten-Editor bearbeiten Sie das Bitmap für die Symbolleisten der Anwendung.

Abgesehen von den üblichen Grafikbefehlen zum Zeichnen der Schaltflächen stehen Ihnen dabei die folgenden Bearbeitungsmöglichkeiten zur Verfügung:

Aktion	Ausführung
Neue Schaltfläche anlegen	Klicken Sie in die leere Schablone, die im oberen Teilfenster der Symbolleiste angezeigt wird. Sowie Sie dann in einem der unteren Fenster mit der Bearbeitung beginnen, wird im oberen Fenster eine neue leere Schablone eingefügt.
Schaltflächen bearbeiten	Klicken Sie einfach im oberen Fenster auf die zu bearbeitende Schaltfläche.
Schaltflächen umordnen	Verschieben Sie die Schaltflächen in der Symbolleiste im oberen Fenster mit der Maus.
Schaltflächen löschen	Nehmen Sie die Schaltfläche im oberen Fenster mit der Maus auf, und ziehen Sie die Schaltfläche aus der Symbolleiste heraus.
Leerräume einfügen	Um vor einer Schaltfläche einen Leerraum einzufügen, schieben Sie die Schaltfläche einfach um etwa eine halbe Breite über die nachfolgende Schaltfläche.
Maße festlegen	Schaltflächensymbole haben eine feste Größe (standardmäßig beträgt diese 16x15 Pixel). Sie können die Maße selbst festlegen, indem Sie für die Symbolleiste den Befehl ANSICHT/EIGENSCHAFTEN aufrufen und auf der Seite ALLGEMEIN Breite und Höhe angeben. (Beachten Sie, daß die Darstellungen aller Schalter in einem gemeinsamen Bitmap abgelegt werden. Die Breitenangabe für die Schaltflächen gibt daher auch an, wie die Bitmap intern in einzelne Schaltflächendarstellungen zu unterteilen ist. Daraus folgt, daß alle Schaltflächen in der Symbolleiste die gleichen Maße haben müssen).

Tabelle 8.4:
Bearbeitung
der Symbol-
leiste

Tabelle 8.4:
Bearbeitung
der Symbol-
leiste
(Fortsetzung)

Aktion	Ausführung
Ressourcen-ID definieren	Für Schaltflächen, die zu Menübefehlen korrespondieren, wählen Sie die ID des Menübefehls aus der Liste aus. Für Schaltflächen, die Aktionen auslösen sollen, zu denen es keine Entsprechung gibt, geben Sie eine neue ID ein.
Hilfetexte definieren	Wenn Sie möchten, daß ein beschreibender Hilfetext in die Statuszeile eingeblendet wird, wenn die betreffende Schaltfläche ausgewählt wird, rufen Sie das Dialogfeld EIGENSCHAFTEN auf (über den Befehl ANSICHT/EIGENSCHAFTEN), und geben Sie den Text in das Feld STATUSZEILENTEXT ein. Um ein Quickinfo zu einem Schalter einzurichten, hängen Sie an den Statuszeilentext das Zeilenumbruchzeichen (\n) und den Quickinfo-Text an. Wenn die Schaltfläche eine ID besitzt, zu der bereits ein Hilfetext erzeugt wurde (beispielsweise die ID eines Menübefehls), wird dieser Hilfetext angezeigt.

Übung 8-4: Symbolleiste anpassen

1. Wenn Sie es noch nicht getan haben, laden Sie jetzt die IDR_MAIN-FRAME-Symbolleiste der Menue-Anwendung in den Symbolleisten-Editor.
2. Löschen Sie bis auf zwei Symbole alle Schaltflächen (für den *Beenden*-Befehl brauchen wir keine Schaltfläche).

Laden Sie die Symbole in den Editor, und übermalen Sie sie gegebenenfalls.

Verbinden Sie die Schaltflächen mit den IDs der Menübefehle Ihrer neuen Menü-Ressource.

8.2.4 Anpassung der Stringtabelle

Öffnen Sie die RESSOURCEN-Ansicht des Arbeitsbereichsfensters, expandieren Sie den Ordner STRING TABLE, und doppelklicken Sie auf den Eintrag ZEICHENFOLGENTABELLE.

Wenn Sie Statuszeilen- und Quickinfo-Text zu den Menübefehlen bereits im EIGENSCHAFTEN-Dialog des Menü-Editors eingegeben haben, sind diese schon in der Stringtabelle eingetragen.

Übung 8-5: Programm testen

1. Führen Sie das Programm aus (**Strg**+**F5**), und prüfen Sie, ob Symbolleiste, **Alt**-Tastenkombinationen und Hilfetexte korrekt eingerichtet wurden.

Wundern Sie sich nicht, wenn bis auf den *Beenden*-Befehl alle Menübefehle und auch die Schaltflächen in der Symbolleiste grau dargestellt und deaktiviert sind. Dies liegt einfach daran, daß die Elemente noch nicht mit Behandlungsroutinen verbunden sind. Wir werden dies nun nachholen.

8.3 Methoden für Menübefehle einrichten

Was nützt dem Anwender das schönste Menü, wenn die Befehle des Menüs keine Aktionen auslösen? Nichts, und deshalb werden wir uns jetzt daranmachen, die nötigen Behandlungsmethoden für unsere Menübefehle einzurichten. Da dies, dank des Klassen-Assistenten, wiederum ganz einfach ist, nutze ich die Gelegenheit, Ihnen noch einmal ein bißchen über Windows und die Nachrichtenverarbeitung unter Windows zu erzählen.

8.3.1 WM_COMMAND

Anwenderaktionen werden unter Windows vom Betriebssystem abgefangen und in Form von speziellen Nachrichten an die betroffenen Programme geschickt. So löst beispielsweise das Drücken der linken Maustaste im Client-Bereich eines Fensters eine `WM_LBUTTONDOWN` aus, die zur Bearbeitung an das Fenster geschickt wird. Andere Nachrichten, die wir bereits kennengelernt haben, waren `WM_KEYDOWN` und `WM_PAINT`. Es sollte klar sein, daß unter Windows auch der Aufruf von Menübefehlen Nachrichten auslöst, die an die betreffende Anwendung geschickt werden. Doch welche Nachricht soll Windows denn auslösen, wenn der Anwender in unserem Programm den Befehl `TEXT/ANZEIGEN` aufruft? `WM_TEXT_ANZEIGEN`? Nein, unmöglich, denn die `WM`-Codes der Nachrichten müssen alle vordefiniert sein, sie können nicht von Windows nach Bedarf zusammengesetzt werden. Da Windows nicht wissen kann, welche Menübefehle eine Anwendung zur Verfügung stellt (dies liegt ja ganz im Ermessen des Programmierers), definiert es einfach eine `WM`-Nachricht für alle Menübefehle: `WM_COMMAND`.

Wann immer der Anwender also einen Menübefehl aufruft (oder eine Symbolschaltfläche drückt oder ein Tastaturkürzel eintippt), wird eine `WM_COMMAND`-Nachricht an die Anwendung geschickt. Doch diese Erkenntnis

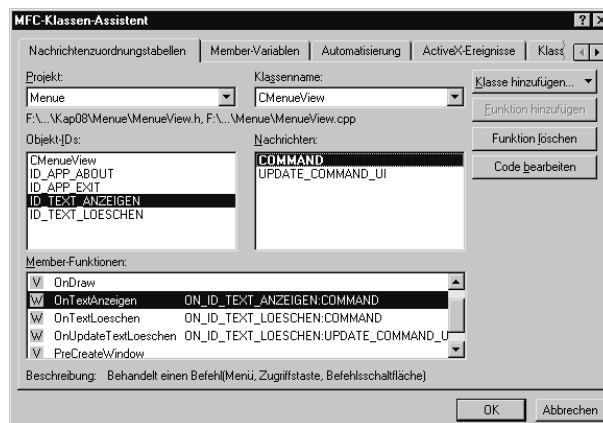
wirft bei uns gleich eine weitere Frage auf: Wenn es nur eine Nachricht für alle möglichen Menübefehle gibt, wie kann die Anwendung dann unterscheiden, welcher Menübefehl eine empfangene WM_COMMAND-Nachricht ausgelöst hat? Die Beantwortung dieser Frage ist gar nicht so schwer, denn schließlich verfügt jeder Menübefehl über eine eindeutige Ressourcen-ID. Diese wird der WM_COMMAND-Nachricht als Parameter mitgegeben und von unserer Anwendung ausgewertet. Das Ganze geschieht vollautomatisch, ohne daß wir etwas dazu tun müßten. Unsere Aufgabe ist es, die durch ihre Ressourcen-IDs identifizierten Menübefehle mit Behandlungsmethoden zu verbinden. Dazu rufen wir jetzt den Klassen-Assistenten auf.

8.3.2 Einsatz des Klassen-Assistenten



Wenn Sie noch den Menü-Editor geöffnet haben, können Sie den Klassen-Assistenten aus dem Kontextmenü des Menü-Editors heraus aufrufen. Klicken Sie dazu mit der rechten Maustaste in das Fenster des Menü-Editors, und wählen Sie den Befehl KLASSEN-ASSISTENT. Der Klassen-Assistent zeigt dann beim Start automatisch die Seite NACHRICHTENZUORDNUNGSTABELLEN an und hat im Feld KLASSENNAME Ihre Rahmenfensterklasse ausgewählt.

Bild 8.7:
Behandlungsmethoden für
Menübefehle
einrichten



Um eine Behandlungsroutine für einen Menübefehl einzurichten, gehen Sie wie folgt vor:

1. **Wählen Sie die Klasse aus, in der die Behandlungsmethode definiert werden soll** (beispielsweise die Rahmenfensterklasse).
2. **Wählen Sie im Feld OBJEKT-IDS die Ressourcen-ID des Menübefehls aus.**

Wenn Sie den Ressourcen-IDs sinnvolle Namen gegeben haben, sollte es Ihnen leicht fallen, aus dem Ressourcenbezeichner auf den Menübefehl zurückzuschließen. Der Anwendungs-Assistent fügt beispielsweise dem Präfix ID_ den Namen des Menüs und den Namen des Menübefehls an.

Wenn Sie sich nicht mehr sicher sind, welche Ressourcen-ID zu welchem Menübefehl gehört, kehren Sie zurück zum Menü-Editor, und doppelklicken Sie auf die einzelnen Menübefehle. Im EIGENSCHAFTEN-Dialog können Sie dann nachsehen, welche ID ein bestimmter Befehl hat.

3. **Wählen Sie im Feld NACHRICHTEN den Eintrag COMMAND aus.**
4. **Drücken Sie den Schalter FUNKTION HINZUFÜGEN.** Die Methode wird eingerichtet und im Feld MEMBER-FUNKTIONEN angezeigt.
5. **Drücken Sie auf den Schalter CODE BEARBEITEN,** und setzen Sie im Texteditor den Code für die Behandlungsmethode auf.

Sicherlich haben Sie mit diesem Prozedere keine Schwierigkeiten mehr. Probleme könnte bestenfalls die Auswahl der geeigneten Klasse im Feld KLASSENNAME bereiten.

8.3.3 Die Klassen des Anwendungsgerüsts

Menüleisten werden immer im Rahmen des Rahmenfensters angezeigt¹, doch das bedeutet nicht, daß die Befehle und einzelne Dropdown-Menüs in der Menüleiste nur die Befehle des Rahmenfensters präsentieren. Auch allgemeine Befehle und Befehle des Client-Fensters werden in das Menü aufgenommen. Damit stellt sich die Frage, welche Befehle von welchen Klassen des Anwendungsgerüsts (Anwendungsklasse, Rahmenfensterklasse, Ansichtsklasse und sogar die Dokumentklasse) behandelt werden sollen.

¹ In Ausnahmefällen auch in Dialogfenstern.

Tabelle 8.5:
Menübefehle
und die Klas-
sen des An-
wendungs-
gerüsts

Klasse	Menübefehle
Anwendungsklasse ¹	Allgemeine Befehle werden üblicherweise von der Anwendungsklasse behandelt. Hierzu gehören beispielsweise der Befehl zum Beenden der Anwendung (ID_APP_EXIT), sowie Befehle, die weder einzelne Dokumente noch die Anzeige der Daten betreffen (ID_FILE_NEW, ID_FILE_OPEN, ID_APP_ABOUT).
Rahmenfensterklasse	Zur Behandlung von Befehlen, die das Rahmenfenster betreffen. Hierzu gehören die Befehle zum Aktivieren und Deaktivieren von Symbolleisten und Statusleiste sowie die Befehle im Fenster-Menü (soweit vorhanden).
Ansichtsklasse	Zur Behandlung von Befehlen, die die Anzeige im Ansichtsfenster betreffen (beispielsweise die Befehle für die Zwischenablage).
Dokumentklasse ²	Zur Behandlung von Befehlen, die die Verwaltung der Daten betreffen (beispielsweise die Befehle zum Speichern und Schließen von Dateien).

Vordefinierte Nachrichtenbehandlung des Anwendungsgerüsts

Beachten Sie, daß für viele der Menübefehle des vom MFC-Anwendungs-Assistenten eingerichteten Menüs bereits passende Behandlungsmethoden in den MFC-Klassen vorgesehen sind. Die Anwendungsklasse definiert beispielsweise die Methode `OnAppExit()`, die mit der Menübefehls-ID `ID_APP_EXIT` verbunden ist und die Anwendung beendet.

Um die bereits eingerichtete Nachrichtenbehandlung nicht zu stören, ist es wichtig, daß Sie weder die Menübefehl-IDs verändern (also ja nicht `ID_APP_EXIT` in `ID_APP_BEENDEN` umbenennen), noch die betreffenden Nachrichten selbst abfangen (also nicht noch einmal eine Behandlungsmethode für `ID_APP_EXIT` einrichten). Leider nutzt uns die Anzeige im Klassen-Assistenten hier recht wenig. Im Feld `MEMBER-FUNKTIONEN` werden zwar die Nachrichtenbehandlungsmethoden angezeigt, die für die im Feld `KLASSENNAME` ausgewählte Klasse bereits implementiert sind (die zugehörige Nachricht wird im Feld `NACHRICHTEN` durch Fettdruck hervorgehoben),

¹ Ja, auch die Anwendungsklasse und die Dokumentklasse können Nachrichten empfangen und bearbeiten. Dies ist insofern ungewöhnlich, als Nachrichten unter Windows nur an Fenster geschickt werden. In MFC-Anwendungen werden die Nachrichten aber, nachdem sie empfangen wurden, in der Klassenhierarchie weiterverteilt und können dabei auch von Nicht-Fensterklassen bearbeitet werden.

² siehe 1.

doch hilft dies nichts, wenn die Ereignisbehandlung wie im Falle der `ID_APP_EXIT`-Nachricht in MFC-Basisklassen versteckt ist.

Ein wenig Hilfe bietet die Technische Notiz Nr. 22, die Sie über die Online-Hilfe aufrufen können, indem Sie nach dem Indexeintrag »standard commands« suchen. Doch die Informationen dieser Notiz dürften für Programmierer, die sich gerade erst in die Materie einarbeiten, eher verwirrend als hilfreich sein.

Mein Rat ist, daß Sie – bevor Sie zu einem Menübefehl eine eigene `WM_COMMAND`-Behandlungsmethode einrichten – zuerst das Anwendungsgerüst ausführen und prüfen, inwieweit der Befehl bereits mit Code verbunden ist. So werden Sie beispielsweise bei Auswahl des Menübefehls `DATEI/ÖFFNEN` feststellen, daß der `ÖFFNEN`-Dialog aufgerufen wird. Dies bedeutet, daß die `WM_COMMAND`-Nachricht zu diesem Befehl bereits irgendwo im verborgenen Code des Anwendungsgerüsts abgefangen wird. (Näheres dazu entnehmen Sie der Technischen Notiz Nr. 22). In einem solchen Fall sollten Sie für den Menübefehl auf keinen Fall eine eigene Behandlungsmethode einrichten. Schauen Sie lieber nach, ob Sie im Feld `NACHRICHTEN` des Klassen-Assistenten eine passende On-Methode finden, die Sie überschreiben können. Für den Menübefehl `DATEI/ÖFFNEN` (ID: `ID_FILE_OPEN`) bietet die Dokumentklasse beispielsweise die Methode `OnOpenDocument()` an.

Übung 8-6: Behandlungsmethoden für Menübefehle einrichten

1. Rufen Sie den Klassen-Assistenten auf (Befehl `ANSICHT/KLASSEN-ASSISTENT`), und lassen Sie die Seite `NACHRICHTENZUORDNUNGSTABELLEN` anzeigen.
2. Richten Sie eine Behandlungsmethode für den Befehl `TEXT/ANZEIGEN` ein.

Wählen Sie im Feld `KLASSENNAME` die Ansichtsfensterklasse aus.

Wählen Sie im Feld `OBJEKT-IDS` die Ressourcen-ID des Menübefehls aus.

Wählen Sie im Feld `NACHRICHTEN` den Eintrag `COMMAND` aus.

Drücken Sie den Schalter `FUNKTION HINZUFÜGEN`.

Drücken Sie den Schalter `CODE BEARBEITEN`.

3. In der Behandlungsmethode zeichnen wir einen kurzen Text an zufälliger Position ins Fenster.

```
void CMenuView::OnTextAnzeigen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
}
```

```

CClientDC dc(this);
RECT rect;
int x, y;
GetClientRect(&rect);
x = (int) ((rand() * (rect.right - rect.left)) / RAND_MAX);
y = (int) ((rand() * (rect.bottom - rect.top)) / RAND_MAX);
dc.TextOut(x, y, "Hallo, hallo");
}

```

Die Textausgabe sieht so aus, daß wir uns zuerst einen Gerätekontext für den Client-Bereich des Ansichtsfensters besorgen (Instanziierung von `dc`) und dann die Ausmaße des Client-Bereichs des Ansichtsfensters abfragen (`GetClientRect()`). Dann berechnen wir die Koordinaten für die Ausgabe.

Die C-Funktion `rand()` liefert einen Integer-Wert zwischen 0 und der C-Konstanten `RAND_MAX`. Indem wir diesen Wert mit der Breite des Client-Bereichs multiplizieren und das Ergebnis durch `RAND_MAX` teilen, erhalten wir einen Wert zwischen 0 und der Breite des Client-Bereichs. Für `x` und `y` durchgeführt, ergibt das eine zufällige Position für den auszugebenden Text.

Zum Schluß wird der Text mit Hilfe der Methode `TextOut()` ausgegeben.

4. Richten Sie eine Behandlungsmethode für den Befehl TEXT/LÖSCHEN ein.

```

void CMenuView::OnTextLoeschen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    Invalidate();
    UpdateWindow();
}

```

Statt die Ausgaben einzeln zu löschen (was gar nicht mehr möglich wäre, da wir die Koordinaten der Texte nicht abgespeichert haben), erklären wir einfach den ganzen Fensterinhalt für ungültig (Methode `Invalidate()`) und lassen das Fenster neu zeichnen (Aufruf von `UpdateWindow()`).

8.4 Menübefehle deaktivieren

Obwohl der Anwender mittlerweile daran gewöhnt ist (oder gewöhnt sein sollte), daß sich die Menüstruktur eines Programms zur Laufzeit ändern kann (durch Ein- und Ausblenden einzelner Menüs oder Menübefehle), sollte die Menüstruktur doch vornehmlich statisch angelegt sein, damit der Anwender den Überblick darüber behält, welche Befehle ihm zur Verfügung stehen.

Doch nicht immer sollen in jeder Situation alle Befehle aufrufbar sein. Nehmen wir zum Beispiel den Menübefehl TEXT/LÖSCHEN aus dem vorangehenden Abschnitt. Solange kein Text ausgegeben wurde, ist dieser Befehl unsinnig. Ruft der Anwender den Befehl in einer solchen Situation auf, wird er sich wundern, warum nichts geschieht. Man könnte nun eine boolesche Elementvariable einrichten, die anzeigt, ob Text ausgegeben wurde oder ob das Fenster leer ist, und eine Meldung ausgeben, wenn der Anwender versucht, den Inhalt eines leeren Fensters zu löschen. Einfacher und für den Anwender klarer ist es allerdings, den Befehl zu deaktivieren.

Diese Möglichkeit zur Befehlsaktivierung bietet die UPDATE_COMMAND_UI-Nachricht zu den Menübefehlen. Mit ihrer Hilfe können Sie eine Behandlungsmethode einrichten, die immer dann aufgerufen wird, wenn das Oberflächenelement zu einem COMMAND-Befehl (Menübefehl oder Schaltfläche) aktualisiert werden muß.

Übung 8-7: Menübefehle aktivieren und inaktivieren

1. Rufen Sie den Klassen-Assistenten auf (Befehl ANSICHT/KLASSEN-ASSISTENT), und lassen Sie die Seite NACHRICHTENZUORDNUNGSTABELLEN anzeigen.
2. Richten Sie eine UPDATE_COMMAND_UI-Behandlungsmethode für den Befehl TEXT/LÖSCHEN ein.

Wählen Sie im Feld KLASSENNAME die Rahmenfensterklasse aus.

Wählen Sie im Feld OBJEKT-IDS die Ressourcen-ID des Menübefehls aus.

Wählen Sie im Feld NACHRICHTEN den Eintrag UPDATE_COMMAND_UI aus.

Drücken Sie den Schalter FUNKTION HINZUFÜGEN.

Drücken Sie den Schalter CODE BEARBEITEN.

3. Implementieren Sie die Methode.

Über den Parameter `pCmdUI` der Methode können Sie auf das zu aktualisierende Element zugreifen. Über die `Enable()`-Methode der Klasse `CCmdUI` können Sie das Oberflächenelement aktivieren oder deaktivieren.

```
void CMenuView::OnUpdateTextLoeschen(CCmdUI* pCmdUI)
{
    // TODO: Code für die Befehlsbehandlungsroutine zum
    // Aktualisieren der Benutzeroberfläche hier einfügen
    if(m_bText)
        pCmdUI->Enable(true);
    else
        pCmdUI->Enable(false);
}
```

Diese Behandlungsmethode wird automatisch ausgeführt, wann immer eine Aktualisierung der Menüstruktur erforderlich ist. In Abhängigkeit von der Elementvariablen `m_bText` wird der Menübefehl TEXT/LÖSCHEN dann aktiviert oder deaktiviert. Doch wo kommt die Elementvariable `m_bText` her? Nirgends, wir müssen Sie noch deklarieren und setzen.

4. Deklarieren Sie die Schaltervariable `m_bText` als Elementvariable der Ansichtsklasse.

```
class CMenuView : public CView
{
protected: // Nur aus Serialisierung erzeugen
    CMenuView();
    DECLARE_DYNCREATE(CMenuView)
// Attribute
public:
    CMenuDoc* GetDocument();
    bool m_bText;
```

5. Setzen Sie die Schaltervariable im Konstruktor und in der Methode `OnTextLoeschen()` auf `false`, damit der Befehl TEXT/LÖSCHEN deaktiviert wird, wenn das Fenster leer ist.

```
// Konstruktor
CMenuView::CMenuView()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen,
    m_bText = false;
}
```

und

```
void CMenuView::OnTextLoeschen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    Invalidate();
    UpdateWindow();
    m_bText = false;
}
```

6. Setzen Sie die Schaltervariable der Methode `OnTextAnzeigen()` auf `true`, damit der Befehl `TEXT/LÖSCHEN` aktiviert wird, sobald Text ausgegeben wird.



Bild 8.8:
Deaktivierter
Menübefehl

8.5 Kontextmenüs

Als Kontextmenüs bezeichnet man die Popup-Menüs, die aufspringen, wenn der Anwender mit der rechten Maustaste auf bestimmte Fensterelemente oder -bereiche klickt. Kontextmenüs nennt man sie, weil die in ihnen angezeigten Befehle auf das Element abgestimmt sind, für das sie aufgerufen wurden.

Um unsere Anwendungen mit Kontextmenüs auszustatten, müssen wir auf die Klasse `CMenu` zurückgreifen. Doch der Umgang mit der Klasse ist gar nicht so kompliziert. Zuvor aber müssen wir eine Menü-Ressource für das Kontextmenü erzeugen und eine Ereignisbehandlungsmethode für die Windows-Nachricht `WM_CONTEXTMENU` einrichten, die uns informiert, wenn der Anwender versucht, ein Kontextmenü zu öffnen.

Übung 8-8: Kontextmenüs einrichten

Bild 8.9:
Kontextmenü
im Menü-
Editor



1. Erstellen Sie die Menü-Ressource für das Kontextmenü. Sie gehen dabei ganz wie beim Erzeugen einer Menüleiste vor, nur daß Sie nur ein einziges Dropdown-Menü einrichten. Der Titel des Dropdown-Menüs wird später im Programm nicht mehr angezeigt.

Achten Sie darauf, den Menübefehlen die Ressourcen-IDs der Menübefehle aus der Menüleiste zuzuweisen.

Wenn Sie möchten, können Sie im Kontextmenü des Menü-Editors den Befehl ALS KONTEXTMENÜ ANZEIGEN aufrufen, um das Menü als einzelnes Popup-Menü statt als Teil einer Menüleiste anzeigen zu lassen.

2. Rufen Sie den Klassen-Assistenten auf, und richten Sie die Behandlungsmethode für die Windows-Nachricht WM_CONTEXTMENU ein.
3. In dieser Methode erzeugen Sie das Menü auf der Grundlage Ihrer Menü-Ressource und rufen die Methoden `GetSubMenu()` und `TrackPopupMenu()` auf, die das Kontextmenü anzeigen und überwachen, welcher Menübefehl ausgewählt wird.

```
void CMenuView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen
    CMenu menu;
    menu.LoadMenu(IDR_MENU1);
    menu.GetSubMenu(0)->TrackPopupMenu(
        TPM_LEFTALIGN | TPM_RIGHTBUTTON,
        point.x, point.y, this);
}
```

Zuerst erzeugen wir eine Instanz von `CMenu`.

Dann wird das eigentliche Menü auf der Grundlage der Menü-Ressource erstellt und mit dem `CMenu`-Objekt verbunden.

Das Objekt `menu` weiß nichts davon, daß es nur ein Kontextmenü und nicht eine vollwertige Menüleiste ist. Für `menu` ist unser Kontextmenü einfach das erste Dropdown-Menü in seiner Menüleiste (Index 0). Um auf das Kontextmenü zuzugreifen, rufen wir daher die Methode `GetSubMenu()` mit dem Index 0 auf.

Der Rückgabewert von `GetSubMenu()` ist ein Zeiger auf das Untermenü, für das wir die Methode `TrackPopupMenu()` aufrufen, die überwacht, welchen Befehl der Anwender im Kontextmenü aufruft.

4. Führen Sie die Anwendung aus (`(Strg) + (F5)`).

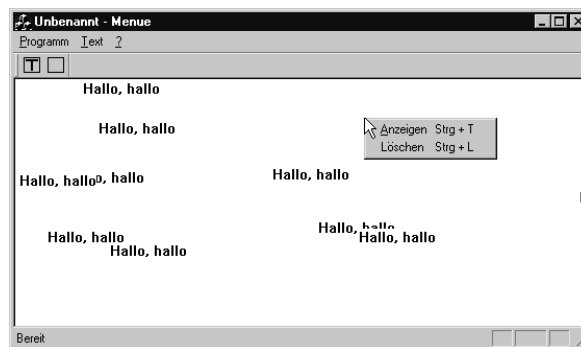


Bild 8.10:
Das Kontextmenü

Leider wird der Menübefehl LÖSCHEN im Kontextmenü nicht deaktiviert.



8.6 Zusammenfassung

Mit Hilfe des MFC-Anwendungs-Assistenten kann man, ohne eine Zeile Code zu schreiben, eine komplette Menüleiste mit zugehörigen Tastaturkürzeln, einer passenden Symbolleiste und Hilfetexten für Quickinfo und Statusleiste einrichten lassen. Sie brauchen nur noch die Ressourcen anzupassen und Behandlungsmethoden für die Menübefehle zu erzeugen.

Zur Einrichtung der Behandlungsmethoden für die Menübefehle verwendet man den Klassen-Assistent und geht wie bei der Einrichtung von Behandlungsmethoden für Windows-Nachrichten vor, nur daß man im Feld `OBJEKT-IDS` nicht den Klassennamen, sondern die Ressourcen-ID des Menübefehls und als Nachricht den Eintrag `COMMAND` auswählt.

Durch Bearbeitung der Nachricht `UPDATE_COMMAND_UI` für einen Menübefehl kann man steuern, unter welchen Bedingungen der Menübefehl aktiviert und deaktiviert werden soll.

Kontextmenüs werden wie Menüleisten im Menü-Editor erzeugt. Zur Behandlung von Kontextmenüs wird die Windows-Nachricht `WM_CONTEXTMENU` abgefangen, die ausgelöst wird, wenn der Anwender versucht, ein Kontextmenü zu öffnen.

8.7 Fragen

1. Wie richtet man `[Alt]`-Tastenkombinationen zum Aufruf von Menübefehlen ein?
2. Wie richtet man Tastaturkürzel für Menübefehle ein?
3. Wie richtet man Quickinfos für Menübefehle ein?
4. Wozu braucht man die Klasse `CMenu`?

8.8 Aufgaben

1. Legen Sie eine Kopie des Anwendungsgerüsts aus der Aufgabe 3 des Kapitels 7 an. Richten Sie für das Programm eine Menüleiste mit einem Menü `DATEI` und den Menübefehlen `DATEI/NEU` und `DATEI/BEENDEN` ein. Bei Aufruf des Menübefehls `DATEI/NEU` soll der Fensterinhalt gelöscht werden, bei Aufruf des Menübefehls `DATEI/BEENDEN` soll die Anwendung beendet werden.

Um eine Kopie anzulegen, beginnen Sie mit einem leeren Win32-Anwendungsprojekt. Rufen Sie den Menübefehl `PROJEKT/EINSTELLUNGEN` auf, und wählen Sie auf der Seite `ALLGEMEIN` im Feld `MICROSOFT FOUNDATION CLASSES` eine der Optionen zur Verwendung der MFC aus. Kopieren Sie die Quelltextdateien und die Header-Datei, einschließlich der Ressourcendateien (`.rc`, `.ico`), in das Verzeichnis des Projekts, und nehmen Sie die Quelldateien mit Hilfe des Befehls `PROJEKT/DEM PROJEKT HINZUFÜGEN/DATEIEN` in das Projekt auf.

Klickt der Anwender in das Fenster der Anwendung, zeichnet die Anwendung an der Stelle des Mausklicks einen Kreis. Da die Kreise nur in der Behandlungsmethode der `WM_LBUTTONDOWN`-Nachricht und nicht in der `OnDraw()`-Methode gezeichnet werden, genügt es bei Aufruf des Menübefehls `DATEI/NEU` zum Löschen der Kreise, das Fenster neu zeichnen zu lassen (Methodenaufrufe `Invalidate()` und `UpdateWindow()`).

Um das Programm zu beenden, schickt man dem Hauptfenster mit Hilfe der Methode `SendMessage()` eine `WM_CLOSE`-Nachricht, die es auffordert, sich selbst zu schließen.

Das Menü selbst kann der `Create()`-Methode des Rahmenfensters übergeben werden oder mit Hilfe der `CFrameWnd`-Methode `LoadFrame()` bei der Einrichtung der Anwendung geladen und mit dem Hauptfenster verbunden werden (in letzterem Fall müssen Sie die `Create()`-Methode aus dem Konstruktor des Rahmenfensters löschen, da diese intern von `LoadFrame()` aufgerufen wird).

8.9 Lösung zur Aufgabe

Die Header-Datei:

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

class CAnsicht : public CView {
public:
    CAnsicht(CFrameWnd *parent);
protected:
    afx_msg void OnDraw(class CDC *);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    void OnNeu();
    DECLARE_MESSAGE_MAP()
};

class CRahmenfenster : public CFrameWnd {
public:
    CRahmenfenster();
protected:
    CStatusBar m_wndStatusBar;
    int OnCreate(LPCREATESTRUCT cs);
    void OnBeenden();
    DECLARE_MESSAGE_MAP()
};

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

Die Quelltextdatei:

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

// Anwendungs-Objekt erzeugen
CMyApp Anwendung;
static UINT indicators[] = {ID_SEPARATOR, };
// Ansichtsfenster
BEGIN_MESSAGE_MAP(CAnsicht, CView)
    ON_COMMAND(ID_DATEI_NEU, OnNeu)
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
CAnsicht::CAnsicht(CFrameWnd *parent) {
    Create(0, 0, WS_CHILD | WS_VISIBLE,
        CRect(), parent, AFX_IDW_PANE_FIRST);
}
// muss überschrieben werden, da sonst abstrakte Klasse
void CAnsicht::OnDraw(class CDC *dc) {
}
void CAnsicht::OnLButtonDown(UINT nFlags, CPoint point) {
    CClientDC dc(this);
    dc.Ellipse(point.x-20, point.y-20, point.x+20, point.y+20);
}
void CAnsicht::OnNeu() {
    Invalidate();
    UpdateWindow();
}

// Rahmenfenster
BEGIN_MESSAGE_MAP(CRahmenfenster, CFrameWnd)
    ON_WM_CREATE()
    ON_COMMAND(ID_DATEI_BEENDEN, OnBeenden)
END_MESSAGE_MAP()
CRahmenfenster::CRahmenfenster() {
    LPCTSTR wndClass = AfxRegisterWndClass(NULL,
        AfxGetApp()->LoadCursor(IDC_ARROW),
        (HBRUSH) (COLOR_WINDOW + 1),
        AfxGetApp()->LoadIcon(IDI_ICON1));
```

```
// Fenster erzeugen
Create(wndClass, "Rahmen mit View",
      WS_OVERLAPPEDWINDOW & ~WS_THICKFRAME,
      CRect(20, 20, 300, 200), 0,
      MAKEINTRESOURCE(IDR_MENU1));
// Ansicht erzeugen
new CAnsicht(this);
}
// Eingebettete Rahmenfensterobjekte erzeugen
int CRahmenfenster::OnCreate(LPCREATESTRUCT cs) {
    if(CFrameWnd::OnCreate(cs) == -1)
        return -1;
    m_wndStatusBar.Create(this);
    m_wndStatusBar.SetIndicators(indicators,
                                sizeof(indicators) / sizeof(UINT));
    return 0;
}
void CRahmenfenster::OnBeenden() {
    SendMessage(WM_CLOSE);
}
// 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;
}
```


Steuerelemente

Als Steuerelemente bezeichnet man die typischen Oberflächenelemente, die die meisten Anwender aus den Dialogfeldern kennen: Textfelder, Eingabefelder, Listenfelder, Schalter etc. Steuerelemente kann man aber nicht nur in Dialogfeldern verwenden, man kann sie ebenso gut in Rahmen- oder Ansichtsfenster einbauen. Dies ist zwar eher unüblich, doch lassen sich auf diese Weise durchaus attraktive Benutzeroberflächen erstellen, die eher an Web-Seiten als an Windows-Programme erinnern.

Schade ist nur, daß Visual C++ hinsichtlich des Aufbaus von Benutzeroberflächen mit Steuerelementen mit zweierlei Maß mißt:

- ✗ Wenn Sie Steuerelemente in Dialogfelder aufnehmen wollen, steht Ihnen in Form des Dialog-Editors ein leistungsfähiges grafisches Design-Tool zur Verfügung. Mit der Maus können Sie Steuerelemente aus einer Symbolleiste auswählen und in den Dialog aufnehmen. Sie können die Steuerelemente mit der Maus verschieben und vergrößern. Und Sie können über den Befehl ANSICHT/EIGENSCHAFTEN ein Dialogfeld aufrufen, in dem Sie die einzelnen Steuerelemente konfigurieren können. Zudem brauchen Sie sich keine Sorgen um die Erzeugung der Steuerelemente zu machen – dies geschieht automatisch, wenn der Dialog erzeugt wird (siehe nachfolgendes Kapitel).
- ✗ Wenn Sie Steuerelemente dagegen in ein Rahmen- oder Ansichtsfenster aufnehmen, verweigert Ihnen die IDE jegliche Unterstützung. Sie sind selbst für die Erzeugung der Steuerelemente verantwortlich, Sie müssen die Steuerelemente per Hand konfigurieren, und – das Schlimmste überhaupt – Sie müssen die Steuerelemente von Hand, also durch Angabe von Koordinaten im Quelltext, plazieren.



Kein Wunder also, daß man Steuerelemente doch eher selten in Fenstern zu Gesicht bekommt. Trotzdem möchte ich Ihnen zeigen, wie man Steuerelemente in Fenster aufnimmt, und Ihnen bei dieser Gelegenheit die wichtigsten Standardsteuerelemente von Windows vorstellen. Zum Abschluß werden wir dann einen einarmigen Banditen als Beispiel für ein Programm mit Webseiten-ähnlicher Oberfläche erstellen.

Sie lernen in diesem Kapitel:

- ✗ Wie man allgemein vorgeht, wenn man Steuerelemente in Fenster aufnimmt
- ✗ Wie man mit Textfeldern umgeht
- ✗ Wie man mit Eingabefeldern umgeht
- ✗ Wie man mit Schaltflächen umgeht
- ✗ Wie man mit Kontrollkästchen und Optionsfeldern umgeht
- ✗ Wie man mit Listen- und Kombinationsfeldern umgeht
- ✗ Wie man Zufallszahlen erzeugt

9.1 Steuerelemente in Fenster integrieren

Um ein Steuerelement in ein Fenster zu integrieren, gehen Sie wie folgt vor:

1. In der Klasse des übergeordneten Fensters deklarieren Sie eine Elementvariable vom Typ der MFC-Steuerelementklasse.

```
class CHauptfenster : public CFrameWnd
{
    ...
protected:
    CStatic m_Textfeld;
```

2. In der Fensterklasse richten Sie eine Ereignisbehandlungsroutine für die WM_CREATE-Nachricht ein. (Sofern diese nicht schon eingerichtet ist.)

```
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
    ON_WM_CREATE()
END_MESSAGE_MAP()
```

3. In dieser Methode erzeugen Sie das Steuerelement.

```
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
{
    m_Textfeld.Create("Ich bin Dr. Jekyll !",
                      WS_CHILD | WS_VISIBLE | SS_CENTER,
                      CRect(20, 20, 300, 200), this);
    return CFrameWnd::OnCreate(lpCS);
}
```

4. Nutzen Sie die Methoden der Steuerelementklasse, um das Steuerelement zu manipulieren.

```
// Steuerelement manipulieren
void CHauptfenster::OnLButtonDown(UINT nFlags, CPoint
point)
{
    m_Textfeld.SetWindowText("Ich bin Mr. Hyde !");
}
```

Die Beispielprogramme für die nachfolgenden Unterkapitel wurden nicht mit dem Anwendungs-Assistenten erzeugt, sondern beruhen auf dem Projekttyp Win32-Anwendung. Wenn Sie diese Projekte nachprogrammieren wollen, vergessen Sie nicht, auf der Seite ALLGEMEIN der Projekteinstellungen die MFC einzubinden.



Der abgedruckte Quelltext ist nicht vollständig, sondern auf die wichtigen Teile beschränkt. (Es fehlt das Anwendungsobjekt.)

9.2 Statische Textfelder

Statische Textfelder sind Textfelder, die einen Text anzeigen, der vom Anwender nicht verändert werden kann (wohl aber von der Anwendung).

In der MFC sind die statischen Textfelder in der Klasse CStatic gekapselt.

```
BOOL Create (
    LPCTSTR lpszText,      // der anzuzeigende Text
    DWORD dwStyle,         // Fensterstil
    const RECT& rect,      // Umgebendes Rechteck
    CWnd* pParentWnd,      // Elternfenster
    UINT nID = 0xffff );  // Ressourcen-ID
```



```
CStatic m_Textfeld;
m_Textfeld.Create("Ich bin Dr. Jekyll !",
                  WS_CHILD | WS_VISIBLE | SS_CENTER,
                  CRect(20, 20, 300, 200), this);
```

Tabelle 9.1:
Stile für CStatic¹, die zusätzlich zu WS-Stilen vergeben werden können

Stil	Beschreibung
SS_BITMAP	Für Steuerelemente, die neben Text auch ein Bitmap anzeigen.
SS_BLACKRECT	Das Rechteck des Steuerelements wird in der Farbe für Windows-Fensterrahmen ausgefüllt (üblicherweise Schwarz).
SS_CENTER	Der Text im Steuerelement wird zentriert.
SS_GRAYRECT	Das Rechteck des Steuerelements wird in der Farbe des Windows-Bildschirmhintergrunds ausgefüllt (üblicherweise Grau).
SS_LEFT	Der Text im Steuerelement wird linksbündig ausgerichtet.
SS_NOPREFIX	Zeigt an, daß das Kaufmännische Und im Text kein Hinweis auf eine [Alt]-Tastenkombination, sondern ein normales Zeichen sein soll.
SS_RIGHT	Der Text im Steuerelement wird rechtsbündig ausgerichtet.
SS_SIMPLE	Ein nicht veränderliches, einzeliges Textfeld
SS_WHITERECT	Das Rechteck des Steuerelements wird in der Farbe des Windows-Fensterhintergrunds ausgefüllt (üblicherweise Weiß).

Die wichtigsten Methoden sind:

```
int GetWindowText( LPTSTR lpszStringBuf, int nMaxCount ) const;
void GetWindowText( CString& rString ) const;
```

Liefert den Text im Steuerelement zurück.

```
void SetWindowText( LPCTSTR lpszString );
```

Ersetzt den Text im Steuerelement durch den neuen Text.

```
HBITMAP SetBitmap( HBITMAP hBitmap );
```

Lädt ein Bitmap in die linke obere Ecke des Steuerelements.

1 Für eine vollständige Liste aller Stile siehe die Online-Hilfe zur Create()-Methode.

```
class CHauptfenster : public CFrameWnd
{
public:
    CHauptfenster();
protected:
    CStatic m_Textfeld;
    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
    void OnLButtonDown(UINT nFlags, CPoint point);
DECLARE_MESSAGE_MAP();
};
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
CHauptfenster::CHauptfenster()
{
    Create(0, "Statische Textfelder", WS_OVERLAPPEDWINDOW);
}
// Steuerelement erzeugen
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
{
    m_Textfeld.Create("Ich bin Dr. Jekyll !",
                      WS_CHILD | WS_VISIBLE | SS_CENTER,
                      CRect(20, 20, 300, 200), this);
    return CFrameWnd::OnCreate(lpCS);
}
// Steuerelement manipulieren
void CHauptfenster::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_Textfeld.SetWindowText("Ich bin Mr. Hyde !");
}
```



Bild 9.1:
Statisches
Textfeld

9.3 Eingabefelder

Eingabefelder sind ein- oder mehrzeilige Textfelder, in die der Anwender Text eintippen und an die Anwendung übergeben kann.

In der MFC sind die Eingabefelder in der Klasse `CEdit` gekapselt.

```
BOOL Create(
    DWORD dwStyle,          // Fensterstil
    const RECT& rect,        // Umgebendes Rechteck
    CWnd* pParentWnd,        // Elternfenster
    UINT nID);              // Ressourcen-ID
```



```
CEdit m_Eingabefeld;
m_Eingabefeld.Create(
    WS_CHILD | WS_VISIBLE | WS_BORDER,
    CRect(20, 70, 300, 100), this,
    ID_EDIT_EINGABE);
```

Tabelle 9.2:
Stile für
`CEdit`¹, die
zusätzlich zu
WS-Stilen
vergeben
werden
können

Stil	Beschreibung
ES_AUTOHSCROLL	Der eingegebene Text im Eingabefeld wird automatisch gescrollt, wenn der Anwender mehr Text eingibt, als in das Eingabefeld paßt.
ES_MULTILINE	Erzeugt ein mehrzeiliges Eingabefeld
ES_PASSWORD	Für Paßwortabfragen. Im Eingabefeld werden Sternchen statt Buchstaben angezeigt.
ES_READONLY	Nicht editierbares Eingabefeld.
ES_WANTRETURN	Das Drücken der Eingabetaste in mehrzeiligen Eingabefeldern soll die Zeile umbrechen (und nicht zum Beenden des Dialogs führen, sofern sich das Steuerelement in einem Dialog befindet).

Die wichtigsten Methoden sind:

```
int GetWindowText( LPTSTR lpszStringBuf, int nMaxCount ) const;
void GetWindowText( CString& rString ) const;
```

Liefert den Text im Steuerelement zurück.

1 Für eine vollständige Liste aller Stile siehe die Online-Hilfe zur `Create()`-Methode.

Eingabefelder

```
void GetSel( int& nStartChar, int& nEndChar ) const;
```

Gibt an, welche Zeichen im Eingabefeld markiert sind. nStartChar ist die Position des ersten markierten Zeichens, nEndChar die Position des ersten nicht mehr markierten Zeichens.

```
BOOL Undo( );
void Clear( );
void Copy( );
void Cut( );
void Paste( );
```

Methoden zur Unterstützung der Zwischenablage.

```
#define ID_EDIT_EINGABE 101
class CHauptfenster : public CFrameWnd
{
public:
    CHauptfenster();
protected:
    CStatic m_Textfeld;
    CEdit m_Eingabefeld;
    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
    void OnLButtonDown(UINT nFlags, CPoint point);
DECLARE_MESSAGE_MAP();
};
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
CHauptfenster::CHauptfenster()
{
    Create(0, "Eingabefelder", WS_OVERLAPPEDWINDOW);
}
// Steuerelement erzeugen
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
{
    m_Textfeld.Create("Dies ist ein statischer Text!",
        WS_CHILD | WS_VISIBLE | SS_CENTER,
        CRect(20, 20, 300, 50), this);
```

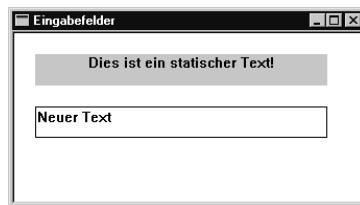


```

m_Eingabefeld.Create(
    WS_CHILD | WS_VISIBLE | WS_BORDER,
    CRect(20, 70, 300, 100), this,
    ID_EDIT_EINGABE);
return CFrameWnd::OnCreate(lpCS);
}
// Steuerelement manipulieren
void CHauptfenster::OnLButtonDown(UINT nFlags, CPoint point)
{
    CString eingabe;
    m_Eingabefeld.GetWindowText(eingabe);
    m_Textfeld.SetWindowText(eingabe);
}

```

Bild 9.2:
Eingabefeld



9.4 Schaltflächen

Schaltflächen sind Steuerelemente, die nur auf Klickereignisse reagieren. Meist ist mit einem Schalter eine bestimmte Aktion verbunden, die ausgeführt wird, wenn der Anwender den Schalter anklickt.

In der MFC sind die Schaltflächen in der Klasse CButton gekapselt.

```

BOOL Create (
    LPCTSTR lpszCaption, // Titel
    DWORD dwStyle,       // Fensterstil
    const RECT& rect,     // Umgebendes Rechteck
    CWnd* pParentWnd,    // Elternfenster
    UINT nID);           // Ressourcen-ID

```



```

CButton m_Schalter;
m_Schalter.Create("Klick mich !",
    WS_CHILD | WS_VISIBLE,
    CRect(20, 70, 300, 100), this, ID_BUTTON);

```

Stil	Beschreibung
BS_AUTOCHECKBOX	Erzeugt ein Kontrollkästchen. Die Markierung des Kästchens wird von Windows übernommen.
BS_AUTORADIOBUTTON	Erzeugt ein Optionsfeld. Die Markierung des Kästchens wird von Windows übernommen.
BS_CHECKBOX	Erzeugt ein Kontrollkästchen.
BS_GROUPBOX	Erzeugt ein Gruppenfeld.
BS_RADIOBUTTON	Erzeugt ein Optionsfeld
BS_PUSHBUTTON	Erzeugt einen Schalter, der ein WM_COMMAND-Ereignis auslöst.

Tabelle 9.3:
Stile für CButton¹, die zusätzlich zu WS-Stilen vergeben werden können

Die wichtigsten Methoden sind:

```
int GetWindowText( LPTSTR lpszStringBuf, int nMaxCount ) const;
void GetWindowText( CString& rString ) const;
```

Liefert den Titel des Schalters zurück.

```
void SetWindowText( LPCTSTR lpszString );
```

Weist dem Schalter einen neuen Titel zu.

```
ON_BN_CLICKED(ID_BUTTON, OnClicked)
```

Ereignisbehandlung für gedrückte Schalter.

```
#define ID_BUTTON 101
class CHauptfenster : public CFrameWnd
{
public:
    CHauptfenster();
protected:
    CButton m_Schalter;
    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
    void OnClicked();
    DECLARE_MESSAGE_MAP();
};
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
    ON_WM_CREATE()
    ON_BN_CLICKED(ID_BUTTON, OnClicked)
END_MESSAGE_MAP()
```



¹ Für eine vollständige Liste aller Stile siehe die Online-Hilfe zur Create()-Methode.


```
CHauptfenster::CHauptfenster()
{
    Create(0, "Schaltflächen", WS_OVERLAPPEDWINDOW);
}
// Steuerelement erzeugen
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
{
    m_Schalter.Create("Klick mich !",
                      WS_CHILD | WS_VISIBLE,
                      CRect(20, 70, 300, 100), this, ID_BUTTON);
    return CFrameWnd::OnCreate(lpCS);
}
// Steuerelement manipulieren
void CHauptfenster::OnClicked()
{
    m_Schalter.SetWindowText("Danke");
}
```

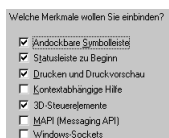
Bild 9.3:
Schaltflächen



9.5 Kontrollkästchen und Optionsfelder

Kontrollkästchen und Optionsfelder sind Varianten von Schaltern.

Kontrollkästchen



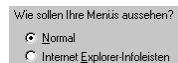
Kontrollkästchen sind Schalter mit einem rechteckigen Kästchen und einem Beschriftungsfeld. Der Anwender kann die Kästchen beliebig markieren und die Markierung wieder aufheben. Kontrollkästchen werden üblicherweise verwendet, um dem Anwender verschiedene Optionen zur Auswahl anzubieten.

Kontrollkästchen werden mit dem Stil BS_AUTOCHECKBOX (oder BS_CHECKBOX) erzeugt:

```
CButton m_Kontroll1;  
m_Kontroll1.Create("Hat einen Buckel",  
                  WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,  
                  CRect(50, 250, 250, 280), this, 3);
```

Optionsfelder

Optionsfelder sind Schalter mit einem runden Kästchen und einem Beschriftungsfeld. Der Anwender kann die Felder beliebig markieren und die Markierung wieder aufheben. Innerhalb einer Gruppe von Optionsfeldern kann aber immer nur ein einziges Optionsfeld markiert werden. Die Gruppierung richtet sich nach der Erzeugungsreihenfolge; neue Gruppen beginnt man, indem man vor den Optionsfeldern der nächsten Gruppe ein Gruppenfeld (Schalter mit dem Stil BS_GROUPBOX) erzeugt.



Optionsfelder werden mit dem Stil BS_AUTORADIOBUTTON (oder BS_RADIOBUTTON) erzeugt:

```
CButton m_Option11;  
m_Option11.Create("Ein Auge",  
                  WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,  
                  CRect(50, 50, 150, 75), this, 11);
```

Die wichtigsten Methoden für Kontrollkästchen und Optionsfelder sind:

```
int GetCheck( ) const;
```

Liefert 1, wenn das Steuerelement markiert ist, und 0, wenn es nicht markiert ist.

```
void SetCheck( int nCheck );
```

Wird als Argument 1 übergeben, wird das Steuerelement markiert. Wird 0 übergeben, wird eine etwaige Markierung aufgehoben.

```
#define ID_BUTTON 101
```

```
class CHauptfenster : public CFrameWnd  
{  
public:  
    CHauptfenster();  
protected:  
    CButton m_Schalter;  
    CButton m_Kontroll1, m_Kontroll2;  
    CButton m_Gruppe1, m_Gruppe2;
```



```

        CButton m_Option11, m_Option12, m_Option13,
                m_Option21, m_Option22;
        afx_msg int OnCreate(LPCREATESTRUCT lpCS);
        void OnClicked();
    DECLARE_MESSAGE_MAP();
};
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
    ON_WM_CREATE()
    ON_BN_CLICKED(ID_BUTTON, OnClicked)
END_MESSAGE_MAP()
CHauptfenster::CHauptfenster()
{
    Create(0, "Frankensteins Monsterkatalog - Schritt 3",
          WS_OVERLAPPEDWINDOW);
}
// Steuerelement erzeugen
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
{
    // 1. Gruppe Optionsfelder
    m_Gruppe1.Create("Anzahl Augen",
                     WS_CHILD | WS_VISIBLE |
                     WS_GROUP | BS_GROUPBOX,
                     CRect(20, 20, 200, 200), this, 1);
    m_Option11.Create("Ein Auge",
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                      CRect(50, 50, 150, 75), this, 11);
    m_Option12.Create("Zwei Augen",
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                      CRect(50, 100, 150, 125), this, 12);
    m_Option13.Create("Drei Augen",
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                      CRect(50, 150, 150, 175), this, 13);
    // 2. Gruppe Optionsfelder
    m_Gruppe2.Create("Geschlecht",
                     WS_CHILD | WS_VISIBLE |
                     WS_GROUP | BS_GROUPBOX,
                     CRect(250, 20, 450, 200), this, 2);
    m_Option21.Create("weiblich",
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
                      CRect(280, 75, 390, 100), this, 21);
}

```

```

m_Option22.Create("männlich",
    WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON,
    CRect(280, 125, 390, 150), this, 22);
// Kontrollkästchen
m_Kontroll11.Create("Hat einen Buckel",
    WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
    CRect(50, 250, 250, 280), this, 3);
m_Kontroll12.Create("Zieht ein Bein nach",
    WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
    CRect(50, 300, 250, 330), this, 4);
// Schalter
m_Schalter.Create("Weiter",
    WS_CHILD | WS_VISIBLE,
    CRect(300, 270, 450, 310), this,
    ID_BUTTON);
return CFrameWnd::OnCreate(lpCS);
}
// Steuerelement manipulieren
void CHauptfenster::OnClicked()
{
    if(m_Option21.GetCheck())
        AfxMessageBox("Weibliches Monster wird erzeugt");
    if(m_Option22.GetCheck())
        AfxMessageBox("Männliches Monster wird erzeugt");
}

```



Bild 9.4:
Verschiedene
Arten von
Schaltern

9.6 Listenfelder und Kombinationsfelder

Listenfelder bieten eine Liste von Einträgen zur Auswahl an. Kombinationsfelder bestehen aus einer Kombination aus Listenfeld und Eingabefeld, d.h., der Anwender kann nicht nur ein Element aus der Liste auswählen, sondern auch eigene Eingaben machen.

In der MFC sind die Listenfelder in der Klasse `CListBox` gekapselt und die Kombinationsfelder in der Klasse `CComboBox`.

```
BOOL Create (
    DWORD dwStyle,          // Fensterstil
    const RECT& rect,       // Umgebendes Rechteck
    CWnd* pParentWnd,       // Elternfenster
    UINT nID = 0xffff );   // Ressourcen-ID
```



```
CListBox m_Listenfeld;
CComboBox m_Kombinationsfeld;
m_Listenfeld.Create(
    WS_CHILD | WS_VISIBLE | WS_BORDER,
    CRect(50, 50, 250, 200), this, 1);
m_Kombinationsfeld.Create(
    WS_CHILD | WS_VISIBLE,
    CRect(350, 50, 550, 200), this, 2);
```

Tabelle 9.4:
Stile für
`CListBox`
(LBS) und
`CComboBox`
(CBS)¹, die zu-
sätzlich zu WS-
Stilen verge-
ben werden
können

Stil	Beschreibung
LBS_EXTENDEDSEL	Es können mit Hilfe der -Taste mehrere Einträge ausgewählt werden.
LBS_SORT	Einträge werden alphabetisch sortiert.
CBS_DROPDOWN	Liste wird erst auf Anwenderanforderung aufgeklappt.
CBS_SIMPLE	Liste wird angezeigt; der aktuell markierte Eintrag wird im Eingabefeld angezeigt.
CBS_SORT	Einträge werden alphabetisch sortiert.

Die wichtigsten Methoden sind:

```
int AddString( LPCTSTR lpszString );
```

Trägt einen String in die Liste ein.

1 Für eine vollständige Liste aller Stile siehe die Online-Hilfe zur `Create()`-Methode.

```
int GetCurSel( ) const;
```

Liefert die Indexnummer des aktuell ausgewählten Elements zurück.

```
int FindString( int nStartAfter, LPCTSTR lpszString ) const;
```

Sucht einen Eintrag mit dem Präfix *lpszString* und liefert dessen Indexnummer zurück.

```
// nur für Listenfelder
```

```
int GetText( int nIndex, LPTSTR lpszBuffer ) const;
```

```
void GetText( int nIndex, CString& rString ) const;
```

```
// nur für Kombinationsfelder
```

```
int GetLBText( int nIndex, LPTSTR lpszText ) const;
```

```
void GetLBText( int nIndex, CString& rString ) const;
```

Kopiert den n-ten Eintrag aus der Liste in das übergebene Stringargument.

```
int DeleteString( UINT n );
```

Löscht das n-te Element aus der Liste.

```
class CHauptfenster : public CFrameWnd
```

```
{
```

```
public:
```

```
    CHauptfenster();
```

```
protected:
```

```
    CButton m_Schalter;
```

```
    CListBox m_Listenfeld;
```

```
    CComboBox m_Kombinationsfeld;
```

```
    afx_msg int OnCreate(LPCREATESTRUCT lpCS);
```

```
    void OnClicked();
```

```
DECLARE_MESSAGE_MAP();
```

```
};
```

```
BEGIN_MESSAGE_MAP (CHauptfenster, CFrameWnd)
```

```
    ON_WM_CREATE()
```

```
    ON_BN_CLICKED(ID_BUTTON, OnClicked)
```

```
END_MESSAGE_MAP()
```

```
CHauptfenster::CHauptfenster()
```

```
{
```

```
    Create(0, "Schaltflächen", WS_OVERLAPPEDWINDOW);
```

```
}
```

```
// Steuerelement erzeugen
```

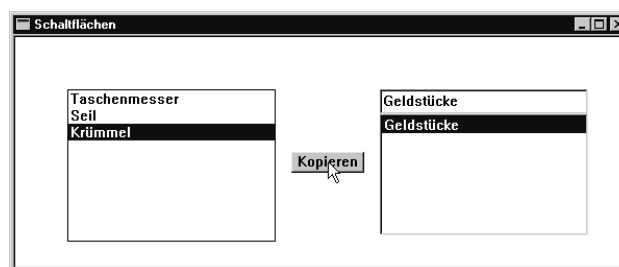
```
int CHauptfenster::OnCreate(LPCREATESTRUCT lpCS)
```



```
{
    m_Listenfeld.Create(
        WS_CHILD | WS_VISIBLE | WS_BORDER,
        CRect(50, 50, 250, 200), this, 1);
    m_Schalter.Create("Kopieren",
        WS_CHILD | WS_VISIBLE,
        CRect(265, 110, 335, 130), this,
        ID_BUTTON);

    m_Kombinationsfeld.Create(
        WS_CHILD | WS_VISIBLE,
        CRect(350, 50, 550, 200), this, 2);
    m_Listenfeld.AddString("Taschenmesser");
    m_Listenfeld.AddString("Seil");
    m_Listenfeld.AddString("Geldstücke");
    m_Listenfeld.AddString("Krümmel");
    return CFrameWnd::OnCreate(lpCS);
}
// Steuerelement manipulieren
void CHauptfenster::OnClicked()
{
    CString str;
    int i = 0;
    i = m_Listenfeld.GetCurSel();
    if ( i != LB_ERR)
    {
        m_Listenfeld.GetText(i, str);
        m_Listenfeld.DeleteString(i);
        m_Kombinationsfeld.AddString(str);
    }
}
```

Bild 9.5:
Listenfeld und
Kombina-
tionsfeld



9.7 Ein einarmiger Bandit

Als letztes Beispiel werden wir aus einem MFC-Anwendungsgerüst einen Einarmigen Banditen machen. Sie wissen schon – diese netten Groschengräber, deren Hebel man drücken muß, um das Spiel in Gang zu setzen. Wie bei einem echten Glücksspielautomaten soll unser Einarmiger Bandit drei Anzeigefelder haben, in denen jeweils das gleiche Symbol (in unserem Fall eine Zahl) erscheinen muß, damit der Spieler gewonnen hat. Und wie bei einem echten Spielautomaten soll dieses Ereignis fast nie eintreten.

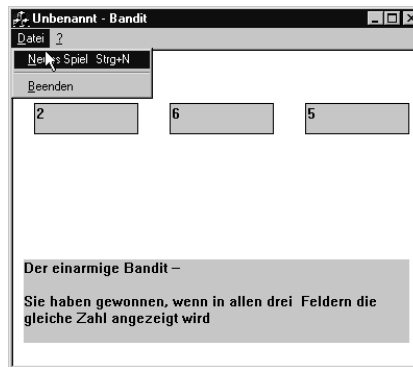


Bild 9.6:
Das Haupt-
fenster des
einarmigen
Banditen

Zufallszahlen

Zufallszahlen kann man mit Hilfe der C-Funktion `rand()` erzeugen. Die Funktion `rand()` liefert einen Integer-Wert zwischen 0 und der C-Konstanten `RAND_MAX` zurück. Allerdings ist zu beachten, daß `rand()` bei jeder Programmausführung genau die gleiche Folge zufälliger Zahlen erzeugt – was vor allem für das Debuggen der Programme wichtig ist, da auf diese Weise der Programmablauf wiederholbar ist.

Um bei jedem Programmstart eine neue Folge von Zufallszahlen zu erhalten, muß man den Zufallsgenerator, den `rand()` benutzt, mit unterschiedlichen Werten initialisieren. Am einfachsten übergibt man dazu der C-Funktion `srand()`, die den Zufallsgenerator initialisiert, den Rückgabewert der C-Funktion `clock()`, die die abgelaufene Prozessorzeit in Ticks zurückliefert.

Um mit Hilfe der Funktion `rand()` Zahlen zwischen 0 und `Max` zu erzeugen, multipliziert man den Rückgabewert von `rand()` mit `MAX` und teilt das Ergebnis durch `RAND_MAX`. So erzeugt beispielsweise der Aufruf

```
int zufall = (int) (rand() * 7) / RAND_MAX;
```

Zahlen zwischen 0 und 7.

Übung 9-1: Steuerelemente im MFC-Anwendungsgerüst

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe das Projekt für diese Übung »Bandit« genannt.
2. Laden Sie die Menü-Ressource IDR_MAINFRAME in den Menü-Editor, und löschen Sie alle Befehle bis auf DATEI/NEU und DATEI/BEENDEN. Ändern Sie den Befehl DATEI/NEU in DATEI/NEUES SPIEL, und weisen Sie ihm eine neue ID zu (beispielsweise ID_FILE_NEU).

Vergessen Sie nicht, die Tastaturkürzel anzugleichen.

3. Deklarieren Sie in der Ansichtsklasse drei Eingabefelder und ein statisches Textfeld.

```
class CBanditView : public CView
{
...
public:
    CStatic m_Text;
    CEdit m_Feld[3];
```

4. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für die WM_CREATE-Nachricht ein, die ausgelöst wird, wenn das Fenster erzeugt wird.
5. Erzeugen Sie in der Methode OnCreate() des Ansichtsfensters die Steuerelemente.

Weisen Sie den Eingabefeldern den Stil ES_READONLY zu, damit deren Wert nicht vom Anwender verändert werden kann.

```
int CBanditView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Speziellen Erstellungscode hier einfügen
    m_Feld[0].Create(
        WS_CHILD | WS_VISIBLE | WS_BORDER | ES_READONLY,
        CRect(20, 50, 120, 80), this, 3);
    m_Feld[0].SetWindowText("0");
    m_Feld[1].Create(
        WS_CHILD | WS_VISIBLE | WS_BORDER | ES_READONLY,
        CRect(150, 50, 250, 80), this, 4);
    m_Feld[1].SetWindowText("0");
```

```

        m_Feld[2].Create(
            WS_CHILD | WS_VISIBLE | WS_BORDER | ES_READONLY,
            CRect(280, 50, 380, 80), this, 5);
        m_Feld[2].SetWindowText("0");
        m_Text.Create("Der einarmige Bandit -- \n\nSie haben
gewonnen, wenn in allen drei Feldern die gleiche Zahl
angezeigt wird",
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            CRect(10, 200, 380, 280), this, 1);
        return 0;
    }

```

6. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für den Menübefehl DATEI/NEUES SPIEL ein.
7. Um das Drehen der Trommeln des Einarmigen Banditen zu simulieren, erzeugen wir drei Zufallszahlen und lassen diese in den Eingabefeldern anzeigen.

```

void CBanditView::OnFileNeu()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    int zufall;
    char str[100];

    srand(clock());
    for (int i = 0; i < 3; i++)
    {
        zufall = (int) (rand() * 7) / RAND_MAX;
        sprintf(str, "%d", zufall);
        m_Feld[i].SetWindowText(str);
    }
}

```

8. Passen Sie die Größe des Hauptfensters an das Layout des Ansichtsfensters an, und sorgen Sie dafür, daß die Größe nicht verändert werden kann. Springen Sie dazu in die Methode PreCreateWindow() (vergleiche Kapitel 6).

Legen Sie Breite und Höhe des Rahmenfensters fest, und löschen Sie die Konstante WS_THICKFRAME aus den Fensterstilen, damit der Anwender das Fenster nicht vergrößern oder verkleinern kann.

```

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.
    cs.cx = 400;
    cs.cy = 350;
    cs.style ^= WS_THICKFRAME;
    return TRUE;
}

```

9. Führen Sie das Programm aus (**Strg** + **F5**).

9.8 Zusammenfassung

Steuerelemente sind Windows-Fenster. Eine ganze Reihe von Steuerelementen sind in Windows vordefiniert und als Teil des Betriebssystems implementiert. Zu diesen Standardsteuerelementen gehören beispielsweise auch

- ✗ das statische Textfeld
- ✗ das Eingabefeld
- ✗ die Schaltfläche
- ✗ die Kontrollkästchen und Optionsfelder
- ✗ die Listen- und Kombinationsfelder

Die Programmierung mit diesen Steuerelementen ist nicht kompliziert, da die MFC Klassen bereitstellt, die diese Steuerelemente kapseln. Für alle wichtigen Arbeiten im Umgang mit Steuerelementen sind in diesen Klassen passende Methoden definiert.

9.9 Fragen

1. Wie heißt die MFC-Klasse für Schalter?
2. Wie heißt die MFC-Klasse für Optionsfelder?
3. Was ist ein Kombinationsfeld?
4. Kann der Text statischer Textfelder verändert werden?
5. Wie richtet man Eingabefelder für Paßwortabfragen ein?

9.10 Aufgaben

1. Informieren Sie sich in der Online-Hilfe darüber, welche Standardsteuerelemente in Windows definiert sind. (TIP: Starten Sie mit dem Indexeintrag *Steuerelemente, Übersicht*.)

Dialogfelder

Die Kommunikation zwischen Anwender und Programm beschränkt sich nur in den seltensten Fällen auf Mausklicks und den Aufruf von Menübefehlen. Gerade Aktionen, die über Menübefehle ausgelöst werden, bedürfen meist näherer Angaben durch den Anwender. Um diese und andere Informationen abzufragen, setzt man Dialoge ein.

Das Erstellen der Dialoge ist – wie ich es Ihnen bereits in der Einleitung zum vorangehenden Kapitel versprochen habe – dank des Dialog-Editors recht einfach. Schwieriger gestaltet sich die Kommunikation zwischen Dialog und Anwendung. Zum einen möchte die Anwendung den Steuerelementen im Dialog vor dessen Aufruf bestimmte Anfangswerte zuweisen, zum anderen muß die Anwendung nach Beendigung des Dialogs natürlich darüber informiert werden, welche Eingaben der Anwender im Dialog vorgenommen hat. Glücklicherweise stehen uns die MFC und der Klassen-Assistent dabei, wie so oft, hilfreich zur Seite.

Wir werden uns in diesem Kapitel ganz auf die technische Seite konzentrieren. Wichtig ist, daß Sie ein wenig Praxis im Umgang mit dem Dialogeditor bekommen und verstehen, wie die Kommunikation zwischen Dialog und Anwendung funktioniert.

Sie lernen in diesem Kapitel:

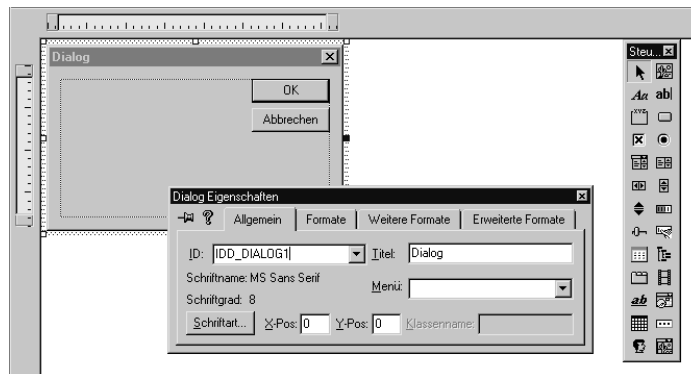
- ✕ Wie man Dialogfelder im Dialogeditor bearbeitet
- ✕ Wie Sie der Klassen-Assistent bei der Einrichtung von Dialogen unterstützt



- ✗ Wie man Dialogfelder initialisiert
- ✗ Wie man Dialogfelder aufruft
- ✗ Wie man Eingaben aus Dialogfeldern abfragt
- ✗ Worin der Unterschied zwischen modalen und nicht-modalen Dialogfeldern liegt

10.1 Dialog-Ressourcen erstellen

Bild 10.1:
Bearbeitung
von Dialog-
feldern



Den Dialog-Editor kennen Sie bereits aus Kapitel 2. Auch ist die Arbeit mit dem Dialog-Editor nicht allzu schwierig, so daß ich mich darauf beschränke, kurz die wichtigsten Arbeitsschritte vorzustellen.

Steuerelemente in Dialog einfügen

Wählen Sie durch Klick mit der Maus in der Symbolleiste **STEUERELEMENTE** aus, welche Art von Steuerelement Sie benötigen. (Wenn Sie nicht wissen, welches Steuerelement sich hinter einem bestimmten Symbol verbirgt, warten Sie, bis das Quickinfo-Fenster zu dem Symbol aufspringt.) Wenn Sie danach mit der Maus in den Dialog klicken, wird das Steuerelement an der Stelle des Mausklicks eingefügt.

Wenn Sie die **[Strg]**-Taste gedrückt halten, während Sie in der Symbolleiste auf ein Symbol klicken, können Sie mehrere Steuerelemente des ausgewählten Typs durch aufeinanderfolgende Mausklicks einfügen. Durch Drücken der **[Esc]**-Taste beenden Sie den Einfügemodus.

Steuerelemente verschieben

Klicken Sie das Steuerelement mit der linken Maustaste an, und halten Sie die Maustaste gedrückt. Verschieben Sie das Steuerelement jetzt bei gedrückt gehaltener Maustaste.

Steuerelemente markieren

- ✗ Um ein einzelnes Element auszuwählen, klicken Sie einfach mit der linken Maustaste auf das Element.
- ✗ Um gleichzeitig mehrere Elemente auszuwählen, ziehen Sie einen Rahmen um die Elemente, die Sie markieren wollen (Dialoghintergrund anklicken und bei gedrückter Maustaste den Rahmen aufspannen).
- ✗ Um Elemente einzeln der Gruppe zuzuweisen oder auszusondern, halten Sie die -Taste gedrückt und klicken die entsprechenden Elemente an (durch zweites Anklicken können Sie die Markierung eines Elements aufheben).
- ✗ Um alle Elemente zu markieren, rufen Sie den Befehl BEARBEITEN/ ALLES MARKIEREN auf.

Steuerelemente löschen

Markieren Sie das Steuerelement, und drücken Sie die -Taste.

Steuerelemente plazieren und ausrichten

- ✗ Am einfachsten plaziert man die Steuerelemente durch Verschieben mit der Maus.
- ✗ Feiner geht es, indem man das Steuerelement markiert und dann mit Hilfe der Pfeiltasten pixelweise verrückt.
- ✗ Möchte man mehrere Steuerelemente zueinander ausrichten (so daß die Steuerelemente linksbündig abschließen oder gleichen vertikalen Abstand haben) ruft man die entsprechenden Befehle im Menü LAYOUT auf (beispielsweise AUSRICHTEN und GLEICHMÄSSIG VERTEILEN). Das Element mit den dunklen Markierungspunkten dient dabei als Referenz (siehe Abbildung 10.2).

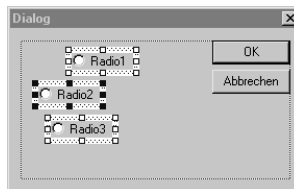


Bild 10.2:
Bearbeitung
von Dialog-
feldern

Wenn Sie also drei Optionsfelder linksbündig nach dem mittleren Optionsfeld ausrichten wollen, markieren Sie die Optionsfelder, indem Sie die -Taste gedrückt halten und dann die Optionsfelder anklicken. Klicken Sie das



mittlere Optionsfeld zuletzt an, damit es als Referenzfeld verwendet wird, und rufen Sie dann den Befehl LAYOUT/AUSRICHTEN/LINKS auf.

Größe der Steuerelemente festlegen

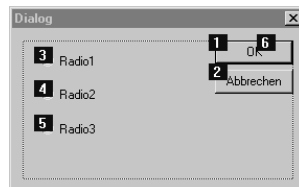
- ✗ Am einfachsten verändert man die Größe von Steuerelementen, indem man sie anklickt und dann den Markierungsrahmen auf- oder zuzieht.
- ✗ Feiner geht es, indem man das Steuerelement markiert, die Umschalt-Taste gedrückt hält und dann die Größe mit Hilfe der Pfeiltasten pixelweise anpaßt.
- ✗ Sollen mehrere Steuerelemente die gleiche Breite oder Höhe haben, ruft man den entsprechenden Befehl im Untermenü LAYOUT/GLEICHE GRÖSSE auf. Das Element mit den dunklen Markierungspunkten dient dabei als Referenz (siehe oben).

Steuerelemente konfigurieren

Dies geschieht über den Befehl EIGENSCHAFTEN aus dem Kontextmenü des Steuerelements. In dem zu diesem Befehl erscheinenden Dialogfeld stehen Ihnen alle wichtigen Optionen zur Konfiguration des Steuerelements zur Verfügung: Vergabe einer eigenen Ressourcen-ID, Angabe eines Titels, Festlegung des Anfangszustandes, Tastenkombination für schnellen Zugriff etc. – die einzelnen Optionen variieren, je nach markiertem Steuerelement.

Tabulatorreihenfolge

Bild 10.3:
Bearbeitung
von Dialog-
feldern



Dieser Befehl erlaubt es Ihnen, durch einfaches Anklicken die Reihenfolge vorzugeben, in der Elemente mit der -Taste erreichbar sind. (Damit ein Element überhaupt mit der -Taste angesprochen werden kann, muß im Dialogfeld EIGENSCHAFTEN die Option TABSTOPP des Steuerelements aktiviert sein.)

Unabhängig davon, ob für ein Steuerelement die Option TABSTOPP aktiviert ist oder nicht, ist jedes Steuerelement in die Tabulator-Reihenfolge eingeordnet (wenn auch nicht mit der -Taste erreichbar). Die Tabulator-

Reihenfolge bestimmt nicht nur die Abfolge, in der die Steuerelemente, für die die Option TABSTOPP gesetzt ist, aktiviert werden, sondern beispielsweise auch die Aufteilung der Steuerelemente in Gruppen (s.u.).

Optionsfelder gruppieren

Üblicherweise dient die Gruppierung der Elemente dazu, funktionell zusammengehörende Elemente für den Benutzer als Gruppe zu kennzeichnen. Diese rein visuelle Gruppierung ist von der internen Gruppierung der Elemente zu unterscheiden, die insbesondere für die Optionsfelder von Bedeutung ist. Von den Optionsfeldern einer Gruppe kann nämlich immer nur eines ausgewählt sein (Windows sorgt dafür automatisch).

- ✗ Zur rein visuellen Kennzeichnung dient vor allem das Steuerelement GRUPPENFELD.
- ✗ Die interne Gruppierung wird bestimmt durch die Definition der jeweils ersten Elemente einer Gruppe und die Tabulator-Reihenfolge der Elemente. Um ein Element als erstes Element einer Gruppe festzulegen, rufen Sie das Kontextmenü des Steuerelements auf, wählen Sie den Befehl EIGENSCHAFTEN aus, und aktivieren Sie die Option GRUPPE. Alle Steuerelemente, die in der Tabulator-Reihenfolge auf dieses Element folgen, bilden dann eine Gruppe, die bis zum nächsten ersten Element einer nachfolgenden Gruppe reicht.

Für das erste Optionsfeld einer Gruppe muß im Dialogfeld EIGENSCHAFTEN die Option GRUPPE gesetzt werden.

Schaltflächen zum Verlassen des Dialogs

Standardmäßig werden Ihre Dialoge mit den Schaltern OK und ABBRECHEN ausgestattet sein.

- ✗ Der Schalter OK dient dazu, den Dialog zu verlassen und die Einstellungen im Dialog zu übernehmen.
- ✗ Der Schalter ABBRECHEN dient dazu, den Dialog ohne Übernahme der vom Anwender vorgenommenen Einstellungen zu verlassen.

Auf welche Art und Weise ein Schalter einen Dialog beendet, hängt aber natürlich nicht von seinem Titel ab. Wichtig ist die Ressourcen-ID des Schalters. Dem OK-Schalter weist man die Kennung IDOK und dem ABBRECHEN-Schalter die Kennung IDCANCEL zu.

Übung 10-1: Aufsetzen einer Dialog-Ressource

Um ein wenig Praxis zu gewinnen – und auch um einen Beispieldialog für später zu haben –, werden wir jetzt einen Dialog erstellen:

- ✗ mit den Schaltern OK und ABBRECHEN zum Verlassen des Dialogs,
- ✗ einem Eingabefeld für einzeiligen Text und
- ✗ einer Gruppe von Optionsfeldern zur Auswahl einer Hintergrundfarbe.

Bild 10.4:
Bearbeitung
von Dialog-
feldern



1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Dialoge« genannt.
2. Legen Sie eine neue Dialog-Ressource an. Rufen Sie dazu den Befehl EINFÜGEN/RESSOURCE auf. Markieren Sie in dem aufspringenden Dialogfeld den Eintrag DIALOG, und drücken Sie dann auf den Schalter NEU.
Visual C++ legt für Sie die Ressource an und lädt sie in den Dialog-Editor.
3. Passen Sie den Titel des Dialogs an. Klicken Sie mit der rechten Maustaste in den Dialog, und rufen Sie den Befehl EIGENSCHAFTEN auf. In dem erscheinenden Dialogfeld können Sie einen TITEL für Ihren Dialog eingeben (beispielsweise »Hintergrundfarbe«).
4. Schalter ausrichten. Der Dialog-Editor beginnt mit einem Dialog, der bereits die beiden Standardschalter OK und ABBRECHEN enthält. Um diese am unteren, statt am rechten Rand auszurichten, gehen Sie wie folgt vor:

Markieren Sie beide Schalter, indem Sie mit der Maus einen Rahmen um diese aufziehen.

Rufen Sie dann den Befehl LAYOUT/SCHALTFLÄCHEN ANORDNEN/UNTEN auf.

5. Optionsfelder einrichten. Richten Sie untereinander drei Optionsfelder ein, über die der Anwender eine Hintergrundfarbe auswählen kann.

Fügen Sie ein GRUPPENFELD-Steuerelement in den Dialog ein, das in etwa die linke Hälfte über den Schaltern einnimmt. Geben Sie über das Dialogfeld EIGENSCHAFTEN als Titel für das Gruppenfeld »Hintergrundfarbe« ein.

Nehmen Sie die Optionsfelder in das Gruppenfeld auf. Drücken Sie die -Taste, und klicken Sie in der Steuerelement-Symbolleiste auf die Schaltfläche OPTIONSFELD. Klicken Sie drei Optionsfelder in das Gruppenfeld. Klicken Sie dann auf das Auswählen-Symbol in der Steuerelement-Symbolleiste. Nutzen Sie die Befehle im Menü LAYOUT, um die Optionsfelder gleichmäßig im Gruppenfeld auszurichten.

Geben Sie als Beschriftungen für die Optionsfelder »Weiß«, »Rot« und »Gelb« ein (über die Seite ALLGEMEIN des Dialogfelds EIGENSCHAFTEN). Aktivieren Sie für das erste Optionsfeld auch das Kontrollkästchen TABSTOPP, damit der Anwender mit der -Taste zu den Optionen springen kann (innerhalb der Gruppe der Optionsfelder kann man den Fokus mit Hilfe der Pfeiltasten verschieben). Des weiteren müssen Sie die Option GRUPPE setzen. Es gibt zwar ehemals nur eine Gruppe von Optionsfeldern in dem Dialog, doch wenn Sie vergessen, für das erste Optionsfeld die Option GRUPPE zu setzen, können Sie später nach Ausführung des Dialogs nicht abfragen, welche Einstellungen der Anwender in der Gruppe der Optionsfelder vorgenommen hat.

6. Nehmen Sie ein EINGABEFELD auf. Über dieses soll der Anwender einen Titel eingeben.

Nehmen Sie zuerst ein statisches TEXTFELD auf, das Sie rechts von dem Gruppenfeld platzieren.

Darunter legen Sie das Eingabefeld ab.

Das Textfeld dient als Beschriftung für das Eingabefeld; geben Sie daher als Titel für das Textfeld »Text eingeben« ein (im EIGENSCHAFTEN-Dialogfeld).

7. Testen Sie den Dialog. Rufen Sie den Befehl LAYOUT/TESTEN auf, und überprüfen Sie, ob Sie mit der -Taste zu den Farboptionen gelangen können und ob von diesen immer nur eine ausgewählt werden kann.
8. Speichern Sie die Ressourcendatei.

10.2 Erstellung von Dialogen auf der Grundlage von Ressourcen

In Übung 10-1 haben wir eine Dialog-Ressource erstellt. Um daraus einen Dialog zu erstellen und anzuzeigen, ist noch einige Arbeit erforderlich.

10.2.1 Dialogklasse anlegen

Dialoge sind Fenster. Es sind irgendwo besondere Fenster, denn sie können auf der Grundlage von Ressourcen erstellt werden, doch es sind Fenster. In einem API-Programm würden wir jetzt »einfach« auf der Grundlage der Dialog-Ressource das Dialogfenster erzeugen. In MFC-Programmen kapseln wir das Dialogfenster dagegen noch in einer eigenen Dialogklasse, die uns in der Folge die Programmierung mit dem Dialog vereinfacht.

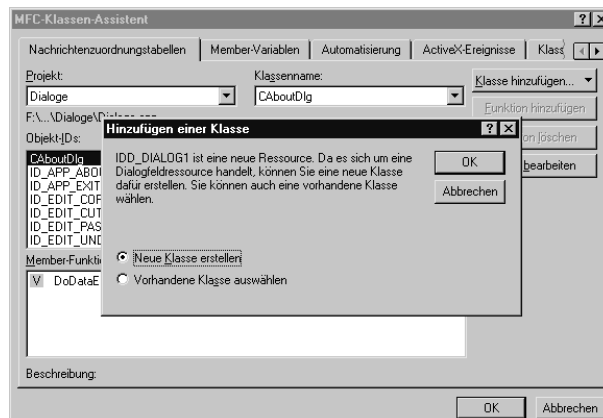
Doch zuerst einmal müssen wir die Klasse haben.

Die MFC implementiert für Dialoge die Klasse CDialog, und von dieser Klasse müssen wir eine eigene Dialogklasse ableiten. Doch bevor Sie jetzt darangehen, eine neue Header- und eine Quelltextdatei in Ihr Projekt aufzunehmen und sich die Finger beim Erzeugen der Klassendeklaration wund tippen, delegieren Sie die Arbeit doch lieber an den Klassen-Assistenten.

Übung 10-2: Erstellen der Dialogklasse

1. Laden Sie die Dialog-Ressource in den Dialog-Editor.
2. Klicken Sie mit der rechten Maustaste in den Hintergrund des Dialog-Editors, und rufen Sie im erscheinenden Kontextmenü den Befehl KLASSEN-ASSISTENT auf.

Bild 10.5:
Dialogklasse
anlegen lassen



3. Wird der Klassen-Assistent für eine neue Dialog-Ressource aufgerufen, wird der in Abbildung 10.5 dargestellte Dialog HINZUFÜGEN EINER NEUEN KLASSE geöffnet. Wählen Sie dort die Option NEUE KLASSE ERSTELLEN, und betätigen Sie anschließend die Schaltfläche OK.

In dem nun angezeigten Dialog NEUE KLASSE (Abbildung 10.6) können Sie den Namen des Dialogs angeben und weitere Optionen setzen, wie zum Beispiel den Dateinamen, den Ressourcenbezeichner oder Automationseinstellungen. Sie können die Klasse außerdem der Komponentengalerie hinzufügen, so daß diese später von anderen Anwendungen verwendet werden kann.

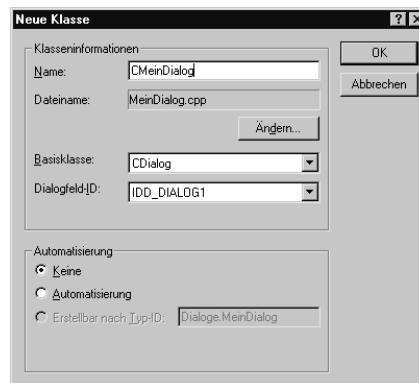


Bild 10.6:
Dialogklasse
konfigurieren

4. Geben Sie der neuen Klasse einen Namen, zum Beispiel *CMeinDialog*.
5. Lassen Sie die von dem Klassen-Assistenten generierten Dateinamen *MeinDialog.h* und *MeinDialog.cpp* für unser Beispiel unverändert. Basisklasse und Ressourcen-ID dürfen Sie nicht verändern.
6. Betätigen Sie die Schaltfläche OK, um die neue Klasse zu erstellen und in den Dialog des Klassen-Assistenten zu springen.

Sollten Sie den Dateinamen ändern, den der Klassen-Assistent für die neuen Header- und Implementierungsdateien Ihrer Klasse vorschlägt? Sollten Sie einen 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 Anwendungs-Assistent handelt anders. 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.



Die Quelltextdatei der neuen Dialogklassen ist wenig interessant, aber die Deklaration der Dialogklasse in der Header-Datei sollten wir uns unbedingt anschauen:¹

```

1:  // MeinDialog.h : Header-Datei
2:  //
3:
4:  //////////////////////////////////////
5:  // Dialogfeld CMeinDialog
6:
7:  class CMeinDialog : public CDialog
8:  {
9:  // Konstruktion
10: public:
11:     CMeinDialog(CWnd* pParent = NULL);
12:
13: // Dialogfelddaten
14:     //{AFX_DATA(CMeinDialog)
15:     enum { IDD = IDD_DIALOG1 };
16:     // HINWEIS: Der Klassen-Assistent fügt hier
17:     // Datenelemente ein
18:     //}AFX_DATA
19:
20: // Überschreibungen
21:     // Vom Klassen-Assistenten generierte virtuelle
22:     // Funktionsüberschreibungen
23:     //{AFX_VIRTUAL(CMeinDialog)
24:     protected:
25:     virtual void DoDataExchange(CDataExchange* pDX);
26:     //}AFX_VIRTUAL
27:
28: // Implementierung
29:     protected:
30:     // Generierte Nachrichtenzuordnungsfunktionen
31:     //{AFX_MSG(CMeinDialog)
32:     // HINWEIS: Der Klassen-Assistent fügt hier
33:     // Member-Funktionen ein
34:     //}AFX_MSG
35:     DECLARE_MESSAGE_MAP()
36: };

```

¹ Einige der für uns weniger interessanten Zeilen habe ich weggelassen.

In Zeile 7 wird unsere Dialogklasse von `CDialog` abgeleitet – okay, das wußten wir schon.

Wo steht die Information, auf welcher Dialogressource die Dialogklasse basiert? Ah, in Zeile 15. Wenn später mit Hilfe der `Create()`-Methode das Dialogfenster erzeugt wird (ich meine nicht ein Objekt der Klasse `CMeinDialog`, sondern das echte Windows-Dialogfenster, das von dem `CMeinDialog`-Objekt gekapselt wird), wird die ID der Dialogressource als Argument an die `Create()`-Methode übergeben.

Vielversprechend klingt auch die Methode `DoDataExchange()`, die in Zeile 24 deklariert ist. Offensichtlich scheint diese Methode etwas mit der Kommunikation – sprich dem Datenaustausch – zwischen Anwendung und Dialog zu tun zu haben. Schaut man aber in der Quelltextdatei des Dialogs nach, wie diese Methode implementiert ist, wird man enttäuscht: Für die Methode wurde noch kein Code erzeugt.

10.2.2 Zugriff auf die Steuerelemente

Wie Sie bereits wissen, sind die Steuerelemente, die wir in den Dialog aufgenommen haben, selbst wieder Fenster. Wenn wir davon sprechen, daß wir mit dem Dialog Daten austauschen wollen, meinen wir eigentlich gar nicht den Dialog, sondern die einzelnen Steuerelemente im Dialog. Der Dialog ist nur eine Art Behälter oder eine Transportunterlage für die Steuerelemente.

Was wir brauchen, ist daher eine Form von Zugriff auf die Steuerelemente im Dialog. Dabei unterstützt uns wieder der Klassen-Assistent.

Übung 10-3: Zugriff auf die Dialog-Steuerelemente einrichten

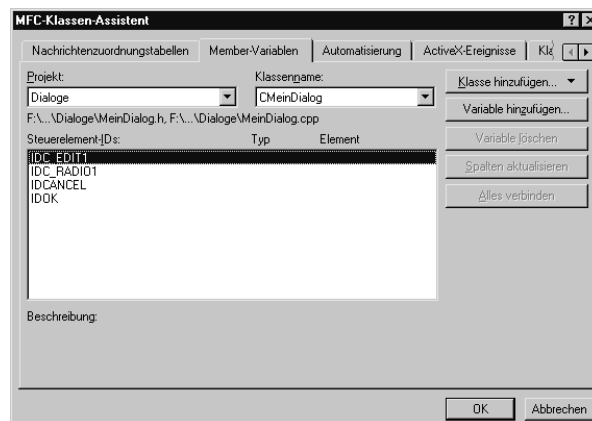


Bild 10.7:
Die Seite
Member-Vari-
ablen des Klas-
sen-Assisten-
ten

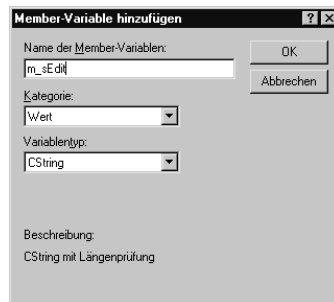
1. Wechseln Sie im Klassen-Assistenten zur Seite MEMBER-VARIABLEN.
2. Wählen Sie im Feld KLASSENNAME die Dialogklasse aus (in unserem Beispiel CMEINDIALOG).
3. Um den Zugriff auf ein Steuerelement des Dialogs einzurichten, wählen Sie jetzt im Feld STEUERELEMENT-IDS die Ressourcen-ID des betreffenden Steuerelements aus.

Wählen Sie als erstes die Ressource für das Eingabefeld aus: ID_EDIT1.

Klicken Sie dann auf den Schalter VARIABLE HINZUFÜGEN.

Es erscheint das Dialogfeld MEMBER-VARIABLE HINZUFÜGEN.

Bild 10.8:
Member-Variablen für den Zugriff auf Dialogsteuerelemente



Der Klassen-Assistent wird jetzt zweierlei tun:

- ✗ er wird der Dialogklasse eine Elementvariable für das ausgewählte Steuerelement hinzufügen, und
- ✗ er wird für diese Elementvariable Code in die `DoDataExchange()`-Methode aufnehmen, der den Datenaustausch zwischen Anwendung und Steuerelement (in beide Richtungen) übernimmt.

Was wir tun müssen, ist dem Klassen-Assistenten mitzuteilen, wie die Elementvariable heißen soll und ob wir nur an dem Eingabewert des Steuerelements interessiert sind oder vollständigen Zugriff auf das Steuerelement haben wollen.

- ✗ Im ersten Fall wählt man im Feld KATEGORIE den Eintrag WERT und gibt im Feld VARIABLENTYP den Datentyp der Elementvariablen an.
- ✗ Im zweiten Fall wählt man im Feld KATEGORIE den Eintrag CONTROL und gibt im Feld VARIABLENTYP den Datentyp der Elementvariablen an. Der Zugriff auf die Steuerelemente erfolgt dann über die Methoden der zugehörigen MFC-Klasse, siehe Übung aus Kapitel 2 und Kapitel 9.

4. Geben Sie als Namen *m_sEdit* ein, und behalten Sie die Kategorie WERT und den Datentyp CSTRING bei. Schicken Sie das Dialogfeld mit OK ab.
5. Analog richten Sie eine Elementvariable *m_iOption* vom Typ INT ein, die uns den Index des ausgewählten Optionsfelds liefert.
6. Schließen Sie den Klassen-Assistenten.

Schauen wir uns das Ergebnis im Quelltext an. Die Deklaration der Dialogklasse wurde um die Deklaration der Elementvariablen erweitert:

```
// MeinDialog.h : Header-Datei
//
...
class CMeinDialog : public CDialog
{
...
// Dialogfelddaten
//{{AFX_DATA(CMeinDialog)
enum { IDD = IDD_DIALOG1 };
CString    m_sEdit;
int        m_iOption;
//}}AFX_DATA
```

In der Quelltextdatei wurden der Konstruktor und die Methode DoDataExchange() ausgebaut:

```
// MeinDialog.cpp: Implementierungsdatei
//
...
CMeinDialog::CMeinDialog(CWnd* pParent /*=NULL*/)
: CDialog(CMeinDialog::IDD, pParent)
{
//{{AFX_DATA_INIT(CMeinDialog)
m_iOption = -1;
m_sEdit = _T("");
//}}AFX_DATA_INIT
}
void CMeinDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    {{{AFX_DATA_MAP(CMeinDialog)
    DDX_Text(pDX, IDC_EDIT1, m_sEdit);
```

```
DDX_Radio(pDX, IDC_RADIO1, m_iOption);
//}}AFX_DATA_MAP
}
```

Die DoDataExchange()-Methode für den Datenaustausch

DoDataExchange() ist die Methode, die den Datenaustausch zwischen den Elementvariablen und den Dialogfeld-Steuerelementen 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 durch die Elementvariable m_bSaveAndValidate des CDataExchange-Objekts bestimmt, auf das der Parameter pDX verweist.

Tabelle 10.1:
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	
Sonstige		DDX_Control	

Die DDV_-Funktionen dienen der Überprüfung des Datenaustauschs.

Ein Beispiel für eine Dialog-Datenüberprüfungsfunktion ist `DDV_MaxChars()`, die die Länge einer Zeichenfolge in einem Eingabe-Steuerelement überprüft. Möchten Sie zum Beispiel sicherstellen, daß 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.



10.3 Dialogfelder initialisieren

Welche Einstellungen werden bei Aufruf des Dialogs angezeigt? Welches der Optionsfelder wird ausgewählt sein, welcher Text wird im Eingabefeld stehen?

Ganz einfach: Die Steuerelemente im Dialog spiegeln die Werte wider, die in den Elementvariablen zu den Steuerelementen gespeichert sind, denn bei Aufruf des Dialogs wird automatisch die Methode `DoDataExchange()` aufgerufen, die die Werte aus den Elementvariablen auf die Steuerelemente überträgt.

Wir brauchen also nur dafür zu sorgen, daß vor Aufruf des Dialogs die gewünschten Werte in den Elementvariablen stehen.

Die meisten Anwendungen handhaben dies:

- ✗ entweder so, daß Sie bei jedem Aufruf des Dialogs mit den gleichen Einstellungen starten. (Um dies zu implementieren, genügt es, wenn wir die Anfangswerte im Konstruktor der Dialogklasse setzen.)
- ✗ oder so, daß sie beim ersten Aufruf Standardeinstellungen vorgeben, während bei nachfolgenden Aufrufen des Dialogs die Einstellungen vom letzten Aufruf angezeigt werden. (Hierfür müßte man die Einstellungen speichern und im Konstruktor in die Elementvariablen kopieren.)

Übung 10-4: Starteinstellungen für Dialog festlegen

1. Überarbeiten Sie die Implementierung des Konstruktors, und weisen Sie den Elementvariablen für die Dialogsteuerelemente eigene Werte zu.

```
CMeinDialog::CMeinDialog(CWnd* pParent /*=NULL*/)
: CDialog(CMeinDialog::IDD, pParent)
{
   //{{AFX_DATA_INIT(CMeinDialog)
    m_iOption = 0;           // erste Option gesetzt
    m_sEdit = _T("");       // leeres Eingabefeld
    //}}AFX_DATA_INIT
}
```



Optionsfelder werden über einen nullbasierten Index angesprochen, d.h., das erste Optionsfeld in einer Gruppe hat den Index 0. Ein Indexwert von -1 führt dazu, daß kein Optionsfeld ausgewählt ist.

10.4 Dialogfelder aufrufen

Der nächste Schritt ist, den Dialog anzuzeigen. Dies geschieht in den für Fenster typischen zwei Schritten:

1. Es wird ein Objekt der Dialogklasse erzeugt.
2. Es wird das eigentliche Fenster erzeugt und mit dem Dialogobjekt verbunden.

Für Dialoge (genauer gesagt, für modale Dialoge – dazu später mehr) wird der zweite Schritt durch einen Aufruf der Methode `DoModal()` ausgeführt.

```
int CDialog::DoModal( );
```

Die Methode `DoModal()` nimmt die Ressourcen-ID des Dialogobjekts, erzeugt daraus ein Dialogfenster, verbindet dieses mit dem Dialogobjekt und zeigt den Dialog an.

- ✘ Wird der Dialog später durch Drücken des `IDOK`-Schalters (oder Drücken der Eingabetaste) beendet, liefert die Methode den Wert `IDOK` zurück.
- ✘ Wurde der Dialog durch Drücken des Abbrechen-Schalters beendet (oder Drücken der `[Esc]`-Taste), liefert die Methode den Wert `IDCANCEL` zurück.

Anhand des Rückgabewerts kann man also prüfen, ob der Anwender möchte, daß seine Eingaben im Dialog ausgewertet oder verworfen werden sollen.

Die übliche Anweisungsfolge zum Aufruf eines Dialogs sieht damit wie folgt aus:

```
CMeinDialog meinDialog;
// Elementvariablen werden im Konstruktor initialisiert
// Jetzt können die Element noch manipuliert werden
if (meinDialog.DoModal() == IDOK)
{
    // Eingaben des Anwenders verarbeiten
}
```

Übung 10-5: Dialog aufrufen

1. Erweitern Sie Ihre Menüressource um einen Menübefehl ANSICHT/DIALOG (siehe Kapitel 8).

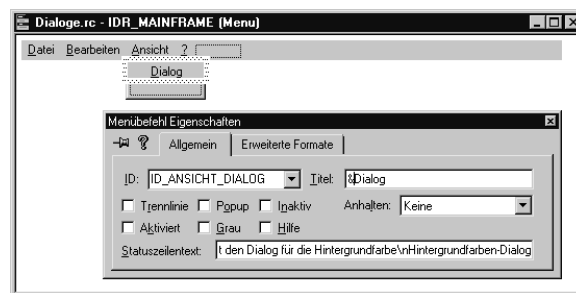


Bild 10.9:
Menübefehl
zum Aufruf
des Dialogs
einrichten

2. Richten Sie mit Hilfe des Klassen-Assistenten eine Behandlungsmethode zu diesem Menübefehl ein.
3. Implementieren Sie in dieser Methode den Code zum Anzeigen des Dialogfensters.

```
void CMainFrame::OnAnsichtDialog()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    CMeinDialog meinDialog;
    if (meinDialog.DoModal() == IDOK)
    {
    }
}
```

4. Nehmen Sie eine `#include`-Anweisung für die Header-Datei des Dialogs in die Quelltextdatei auf, in der Sie die Behandlungsmethode für den Befehl ANSICHT/DIALOG definiert haben.

10.5 Benutzereingaben auswerten

Wenn der Dialog durch Drücken der Eingabetaste oder des OK-Schalters beendet wurde, möchte er, daß das Programm seine Ausgaben auswertet und irgendwie verarbeitet.

In unserem Beispiel werden wir

- ✗ das Ansichtsfenster in der gewünschten Hintergrundfarbe einfärben und
- ✗ darüber den Text aus dem Eingabefeld ausgeben.

```

1: void CMainFrame::OnAnsichtDialog()
2: {
3:     // TODO: Code für Befehlsbehandlungsroutine hier
    // einfügen
4:     CMeinDialog meinDialog;
5:     if (meinDialog.DoModal() == IDOK)
6:     {
7:         CClientDC dc(this);
8:         RECT rect;
9:
10:        GetClientRect(&rect);
11:        switch(meinDialog.m_iOption)
12:        {
13:            case 0:
14:                dc.FillRect(&rect,
                           new CBrush(RGB(255, 255, 255)));
15:                break;
16:            case 1:
17:                dc.FillRect(&rect,
                           new CBrush(RGB(255, 0, 0)));
18:                break;
19:            case 2:
20:                dc.FillRect(&rect,
                           new CBrush(RGB(255, 255, 0)));
21:                break;
22:        }
23:        dc.TextOut(30, 30, meinDialog.m_sEdit);
24:    }
25: }
```



Bild 10.10:
Das Hauptfenster nach Ausführung des Dialogs

Wurde der Dialog durch Drücken des OK-Schalters geschlossen, besorgt sich die Methode in Zeile 7 einen Gerätekontext für den Client-Bereich des Fensters (siehe auch Kapitel 12), ermittelt die Größe des Client-Bereichs (Zeile 10) und füllt diesen dann in der vom Anwender ausgewählten Farbe aus.

10.6 Modale und nicht-modale Dialoge

Modale Dialoge sind Dialoge, die den Eingabefokus nicht abgeben. Dies bedeutet zum einen, daß der Anwender erst dann wieder mit dem Hauptprogramm weiterarbeiten kann, nachdem er den Dialog geschlossen hat, zum anderen, daß die Methode `DoModal()` zum Anzeigen des Dialogs erst dann beendet wird, wenn der Dialog geschlossen wird.

Ein nicht-modaler Dialog kann dagegen auch während der weiteren Arbeit mit einem Programm geöffnet bleiben (beispielsweise Suchen/Ersetzen-Dialog von Windows-Programmen).

Zum Erzeugen eines nicht-modalen Dialogs rufen Sie zuerst den Konstruktor auf (ohne Parameter). Zum Anzeigen des Dialogs rufen Sie die Methode `Create()` mit der ID der Dialog-Ressource auf. Bei nicht modalen Dialogen müssen Sie darauf achten, daß das Dialogobjekt (die Instanz Ihrer Dialogklasse) nicht aufgelöst wird, bevor der Dialog vom Anwender geschlossen wurde!



10.7 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.
- ✗ Nutzen Sie den Klassen-Assistenten, um der Klasse Nachrichtenbehandlungsmethoden hinzuzufügen, sofern erforderlich.
- ✗ Fügen Sie Ihrer Anwendung Programmcode hinzu, der ein Dialogobjekt erstellt, dieses aufruft (mit der `DoModal()`-Methode) und das Ergebnis des Aufrufs ermittelt.

Die Steuerelemente eines Dialogs werden häufig durch Elementvariablen in der entsprechenden Dialogklasse repräsentiert. Der Dialog-Datenaustausch, der vom Klassen-Assistenten für die Elementvariablen eingerichtet wird, ermöglicht den Datenaustausch zwischen den Steuerelementen des Dialogobjekts 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.

Modale Dialoge sind Dialoge, die geschlossen werden müssen, bevor man mit der Anwendung weiterarbeiten kann. Nicht-modale Dialoge sind dagegen Dialoge, die während der weiteren Arbeit mit der Anwendung geöffnet bleiben können.

10.8 Fragen

1. Wie kann man Steuerelemente am geschicktesten zueinander ausrichten?
2. Wie kann man Steuerelemente pixelweise im Dialog verschieben?
3. Wie gruppiert man Optionsfelder?
4. Von welcher Basisklasse werden Dialoge abgeleitet?
5. Wie richtet man Dialogklassen am bequemsten ein?
6. Wozu dient die Methode `DoDataExchange()`?
7. Nennen Sie einen typischen nicht-modalen Dialog, der sich in recht vielen Windows-Programmen findet!

10.9 Aufgaben

1. Implementieren Sie einen Dialog zur Abfrage eines Paßworts. (Hinweis: Schauen Sie sich die Optionen im EIGENSCHAFTEN-Dialog des Eingabefeldes an.)

Text und Dateien

Ich muß gestehen, daß die Behandlung von Texteingaben und -ausgaben nicht gerade zu meinen Lieblingsthemen gehört. Trotzdem kann man an einem so wichtigen Thema natürlich nicht einfach vorbeigehen.

Sie lernen in diesem Kapitel:

- ✗ Welche Möglichkeiten es gibt, Text ein- oder auszugeben
- ✗ Wie man Meldungsfenster anzeigt
- ✗ Wie man mit Dateien umgeht
- ✗ Wie das MFC-Konzept der Serialisierung funktioniert
- ✗ Wie man ohne Mühe einen Texteditor implementiert

11.1 Ein Problem – viele Lösungen

Die Textverarbeitung gehört mit zu den wichtigsten und am häufigsten benötigten Aufgaben bei der Windows-Programmierung. Entsprechend vielseitig ist das Angebot an unterstützenden Klassen und Methoden in der MFC.

11.1.1 Meldungsfenster

Der einfachste Weg, einen kurzen Text auszugeben, ist die Anzeige eines Meldungsfensters. Anwendungen verwenden Meldungsfenster, um den Anwender auf Probleme hinzuweisen oder wichtige Informationen anzuzeigen.



Die Autorenedition von Visual C++ stattet beispielsweise alle mit dieser Edition erstellten Programme mit einem Meldungsfenster aus, das beim Start des Programms angezeigt wird und den Benutzer des Programms darauf hinweist, daß dieses Programm nicht mit einer lizenzierten Compiler-Version erstellt wurde. Vertraut sind auch die Meldungsfenster, die uns Programmierer auf Fehler in der Speicheradressierung unseres Programms hinweisen.

Meldungsfenster sind für die Programmierer ausgesprochen bequem zu handhaben, da sie bereits in Windows vordefiniert sind und durch den Aufruf einer einzigen Methode erzeugt und angezeigt werden können. Tatsächlich sind sie so problemlos zu verwenden, daß viele Programmierer sie auch zum Debuggen verwenden und sich Variablen und Informationen zum Programmablauf über Meldungsfenster anzeigen lassen.

CWnd::MessageBox()

Wenn Sie in einer Methode einer MFC-Fensterklasse ein Meldungsfenster aufrufen wollen, können Sie dazu die Methode `MessageBox()` verwenden:

```
int CWnd::MessageBox (LPCTSTR lpszText,
                     LPCTSTR lpszCaption = NULL,
                     UINT nType = MB_OK );
```

- ✗ `lpszText` ist der Text, den wir ausgeben möchten.
- ✗ `lpszCaption` ist der Text für den Titel des Meldungsfensters.
- ✗ `nType` ist eine Kombination aus Konstanten, die angeben, welche Schalter und welches Symbol im Meldungsfenster angezeigt werden sollen.

Tabelle 11.1:
Konstanten für
den `nType`-
Parameter

Schalter	Symbole
MB_OK	MB_ICONEXCLAMATION
MB_OKCANCEL	MB_ICONINFORMATION
MB_RETRYCANCEL	MB_ICONQUESTION
MB_YES	MB_ICONSTOP
MB_YESNO	
MB_YESNOCANCEL	
MB_ABORTRETRYIGNORE	

AfxMessageBox()

Um aus den Methoden anderer Klassen heraus Meldungsfenster anzuzeigen, verwendet man die globale Funktion `AfxMessageBox()`.

```
int AfxMessageBox( LPCTSTR lpszText,
                  UINT nType = MB_OK,
                  UINT nIDHelp = 0 );
```

Der Rückgabewert der Methoden zeigt an, welcher Schalter zum Schließen des Meldungsfensters gedrückt wurde:

Rückgabewert	Zeigt an:
IDABORT	Die Abbrechen-Schaltfläche wurde gedrückt.
IDCANCEL	Die Abbrechen-Schaltfläche wurde gedrückt.
IDIGNORE	Die Ignorieren-Schaltfläche wurde gedrückt.
IDNO	Die Nein-Schaltfläche wurde gedrückt.
IDOK	Die OK-Schaltfläche wurde gedrückt.
IDRETRY	Die Wiederholen-Schaltfläche wurde gedrückt.
IDYES	Die Ja-Schaltfläche wurde gedrückt.

Tabelle 11.2:
Konstanten für
den *nType*-
Parameter

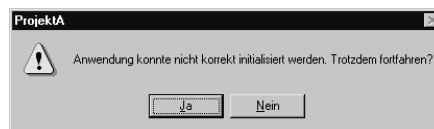


Bild 11.1:
Meldungs-
fenster

`AfxMessageBox()`:

```
BOOL CProjektAApp::InitInstance()
{
    ....
    if (AfxMessageBox("Anwendung konnte nicht korrekt \
        initialisiert werden. Trotzdem fortfahren?",
        MB_YESNO | MB_ICONEXCLAMATION ) == IDYES)
    {
        ...
    }
}
```



`CWnd::MessageBox()`:

```
CProjektAView::CProjektAView()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen,
    MessageBox("Ansicht wird erzeugt", "Debug", MB_OK );
}
```

11.1.2 Text zeichnen

Eine andere Möglichkeit ist, Text in ein Ansichtsfenster (oder ein Rahmenfenster) zu zeichnen. Wie Sie vielleicht aus Beispielen der vorangehenden Kapitel schon wissen, benötigt man zum Zeichnen in ein Fenster immer einen Gerätekontext.



Was ein Gerätekontext ist und wie man mit Gerätekontexten arbeitet, werden wir uns im Kapitel 12 zu den Grafiken noch ganz genau anschauen.

Hat man sich erst einmal einen passenden Gerätekontext zu einem Fenster beschafft, stehen einem zwei Gerätekontextmethoden für die Textausgabe zur Verfügung:

- ✗ CDC::TextOut() und
- ✗ CDC::DrawText()

TextOut()

Mit Hilfe der Methode TextOut() kann man einen Text in ein Fenster zeichnen. Die Stelle der Textausgabe wird durch seine Fensterkoordinaten spezifiziert. Um die Methode ohne große Typumwandlung auch mit CString-Strings aufrufen zu können, ist sie für CString-Objekte überladen.

```
virtual BOOL TextOut( int x, int y, LPCTSTR lpszString,
                    int nCount );
BOOL TextOut( int x, int y, const CString& str );
```

- ✗ x und y. Die Koordinaten der Stelle, an der der Text ausgegeben wird. Beide Werten werden in Pixel angegeben. Der Ursprung des Koordinatensystems (0, 0) ist die obere linke Ecke des Fensters.
- ✗ lpszString oder str. Der auszugebende Text; entweder ein C-String oder ein CString-Objekt.
- ✗ nCount. Für C-Strings muß die Länge des Strings angegeben werden (kann mit Hilfe der C-Methode strlen() ermittelt werden).



```
void CProjektAView::OnDraw(CDC* pDC)
{
    CProjektADoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen
    // Daten hinzufügen
    pDC->TextOut(30, 30, "Hallo!"); // verwendet 2. Version
}
```

DrawText()

Mit Hilfe der Methode `DrawText()` kann man einen Text in ein Fenster zeichnen. `DrawText()` bietet im Vergleich zu `TextOut()` weitreichende Möglichkeiten zur Positionierung und Formatierung des Strings. So kann man mit Hilfe von `DrawText()` Text beispielsweise ohne große Berechnungen zentriert ausgeben.

```
virtual int DrawText( LPCTSTR lpszString, int nCount,
                    LPRECT lpRect, UINT nFormat );
int DrawText( const CString& str, LPRECT lpRect,
             UINT nFormat );
```

- ✗ `lpszString` oder `str`. Der auszugebende Text; entweder ein C-String oder ein `CString`-Objekt.
- ✗ `nCount`. Für C-Strings muß die Länge des Strings angegeben werden (kann mit Hilfe der C-Methode `strlen()` ermittelt werden). Ist der String nullterminiert (endet mit »\0«), können Sie -1 übergeben und die Methode die Stringlänge selbst berechnen lassen.
- ✗ `lpRect`. Rechteckbereich, in den der Text ausgegeben wird.
- ✗ `nFormat`. Eine Reihe von Flags, die festlegen, wie der Text formatiert und im Ausgaberechteck zu positionieren ist.

Rückgabewert	Zeigt an:
DT_BOTTOM	Der Text wird am unteren Rand des Rechtecks ausgerichtet (nur in Kombination mit DT_SINGLELINE).
DT_CALCRECT	Berechnet die Breite des Ausgaberechtecks neu, so daß der Text in das Rechteck paßt. Der Text wird nicht ausgegeben.
DT_CENTER	Zentriert den Text horizontal im Rechteck.
DT_LEFT	Der Text wird im Rechteck linksbündig ausgerichtet.
DT_RIGHT	Der Text wird im Rechteck rechtsbündig ausgerichtet.
DT_SINGLELINE	Erzeugt eine einzeilige Ausgabe. Wichtig für andere Formatkonstanten.
DT_TOP	Der Text wird am oberen Rand des Rechtecks ausgerichtet (nur in Kombination mit DT_SINGLELINE).
DT_VCENTER	Zentriert den Text vertikal im Rechteck (nur in Kombination mit DT_SINGLELINE).
DT_WORDBREAK	Trennt Wörter, damit Zeilen nicht über die Breite des Rechtecks hinausgehen.

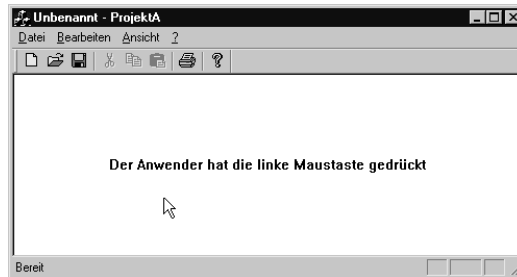
*Tabelle 11.3:
Ausgewählte
Konstanten für
den `nFormat`-
Parameter¹*

¹ Eine Auflistung sämtlicher `nFormat`-Konstanten finden Sie in der Online-Hilfe zu dem Indexeintrag `DrawText`.



Häufig wird für das Ausgaberechteck der Client-Bereich des Fensters ermittelt. So auch in dem folgenden Beispiel, das den auszugebenden Text im Fenster zentriert.

Bild 11.2:
Zentrierte
Ausgabe mit
DrawText()



```
void CProjektAView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für
    // Nachrichten hier einfügen und/oder Standard aufrufen
    CClientDC dc(this);
    RECT rect;
    CString str("Der Anwender hat die linke Maustaste \
                gedrückt");
    GetClientRect(&rect);
    dc.DrawText(str, &rect, DT_CENTER | DT_VCENTER
               | DT_SINGLELINE);

    CView::OnLButtonDown(nFlags, point);
}
```

11.1.3 Steuerelemente

Steuerelemente können für die Ein- und Ausgabe von Text verwendet werden. Die Arbeit mit Steuerelementen ist recht bequem, da die grundlegende Funktionalität des Steuerelements bereits in Windows implementiert ist.

CStatic-Textfeld

Windows-Steuerelement zur Anzeige von einzelligem Text. Das Programm kann zur Laufzeit den Text im Feld ändern, der Anwender nicht. Wird üblicherweise in Dialogfeldern zur Beschriftung anderer Steuerelemente oder zur Anzeige von erklärenden Beschreibungen genutzt.

CEdit-Eingabefeld

Windows-Steuerelement zur Anzeige und zum Editieren von einzeiligem oder mehrzeiligem Text. Die Klasse `CEdit` kann vielseitig eingesetzt werden, beispielsweise in

- ✗ Dialogfeldern zur Abfrage von Anwendereingaben,
- ✗ zu Paßwortabfragen oder
- ✗ zur Implementierung einfacher Texteditoren, wobei die `CEdit`-Klasse intern bereits die Zwischenablage, die Möglichkeit des Schreibschutzes und das Markieren von Text unterstützt.

CRichEdit-Eingabefeld

Windows-Steuerelement zur Anzeige und zum Editieren einzeiligen oder mehrzeiligen, formatierten Textes. Die Klasse `CRichEdit` unterstützt über die Funktionalität des `CEdit`-Eingabefeldes hinaus Zeichen- und Absatzformatierung.

11.1.4 Spezielle Ansichtsklassen

Steuerelemente werden üblicherweise in Dialogfeldern eingesetzt. Die Steuerelemente `Edit` und `RichEdit` wären aber auch für die Implementierung von Texteditoren interessant, nur daß man nicht unbedingt ein `Edit`-Steuerelement in seinem Fenster haben möchte – jedenfalls nicht solange das Steuerelement als solches vom Anwender zu erkennen ist. Aus diesem Grunde stellt uns die MFC zwei abgeleitete View-Klassen zur Verfügung, die das Erscheinungsbild und Verhalten eines normalen Ansichtsfensters mit der Funktionalität eines Eingabe-Steuerelements verbinden.

CEditView

Von `CView` abgeleitete View-Klasse, die ein `CEdit`-Objekt kapselt. Über die Methode `GetEditCtrl()` kann man auf das zugrundeliegende `CEdit`-Steuerelement und dessen Funktionalität zugreifen. Zudem unterstützt `CEditView` das Laden und Speichern von Dateien sowie das Suchen und Ersetzen von Text.

Wie man mit Hilfe von `CEditView` und dem MFC-Anwendungs-Assistenten einen Texteditor implementiert, erfahren Sie im letzten Abschnitt dieses Kapitels.



CRichEditView

Von CView abgeleitete View-Klasse, die ein CRichEdit-Objekt kapselt. Über die Methode GetRichEditCtrl() kann man auf das zugrundeliegende CRichEdit-Steurelement und dessen Funktionalität zugreifen. Zudem unterstützt CRichEditView das Suchen und Ersetzen von Text.

11.1.5 Weitere nützliche Klassen

CString

MFC-Stringklasse, die ähnlich wie die ANSI-C++-Klasse basic_string und deren Spezialisierungen string und wstring die Handhabung von Zeichenketten (insbesondere die damit verbundene dynamische Speicherverwaltung) erleichtert.



Sammlungen von Strings können in den MFC-Klassen CStringArray oder CStringList verwaltet werden.

CFile

MFC-Basisklasse für Dateioperationen. CFile selbst unterstützt nur binäre, ungepufferte Operationen. Die abgeleitete Klasse CStdioFile ermöglicht dagegen gepuffertes Schreiben und Lesen von Binär- wie Textdateien. Eine andere Möglichkeit ist die Verbindung mit einem CArchive-Objekt (siehe Abschnitt zur Serialisierung).



Als Alternative können Sie auch die ANSI-C++-Datenstreamklassen verwenden.

11.2 Dateien

Die meisten Programme dienen der Verarbeitung von Daten. Dabei entsteht zwangsläufig irgendwann das Bedürfnis, diese Daten in Dateien zu speichern oder aus Dateien zu laden. In C/C++-Programmen benutzt man dazu normalerweise die in den Standardbibliotheken definierten Dateibehandlungsfunktionen (fopen(), fprintf() etc.) oder Streams (fstream). Diese besitzen auch noch in Windows-Programmen ihre Gültigkeit, doch Windows

stellt auch eigene Funktionen für die Arbeit mit Dateien zur Verfügung. In der MFC sind diese in der Klasse `CFile` gekapselt.

Wir werden uns der Dateibehandlung in MFC-Programmen auf zwei Weisen nähern:

- ✗ Hier und im nächsten Abschnitt werden wir die Klasse `CFile` unter die Lupe nehmen und uns anschauen, wie eine Unterstützung von Dateioperationen mit den Klassen der MFC grundsätzlich aufgebaut wird.
- ✗ Im letzten Abschnitt dieses Kapitels werden wir uns anschauen, wie man mit Hilfe des Anwendungs-Assistenten einen Texteditor mit Dateiunterstützung erzeugt, und im Kapitel 12 werden wir im Aufgabenteil die Dateiunterstützung für einen Grafik-Editor implementieren.

11.2.1 Die Klasse `CFile`

`CFile` ist die Basisklasse für MFC-Dateidienste. `CFile` unterstützt das Lesen und Schreiben in Dateien – allerdings per se nur für binäre Daten. Das soll uns aber nicht stören. Schauen wir uns lieber die wichtigsten Dateioperationen an.

Dateien öffnen

Das Erstellen eines `CFile`-Objekts geschieht entweder in einem oder in mehreren Schritt(en). Dabei ist wiederum die Dualität vieler MFC-Objekte zu beachten, die wir schon von den Fensterklassen her kennen: Ein `CFile`-Objekt ist nicht mit einem Dateiobjekt unter Windows identisch. Es kann aber mit einem Dateiobjekt verbunden werden und repräsentiert dieses dann.

Um eine Datei in einem Schritt zu öffnen, rufen Sie den Konstruktor auf und übergeben ihm einen Handle auf eine bereits geöffnete oder den Namen einer zu öffnenden Datei.

```
CFile( int hFile );  
CFile( LPCTSTR lpszFileName, UINT nOpenFlags );
```

Der zweiten Version können Sie verschiedene Flags übergeben, die festlegen, in welchem Modus die Datei geöffnet werden soll – beispielsweise, ob die Datei neu angelegt werden soll, wenn sie noch nicht existiert.

Tabelle 11.4:
Einige Kon-
stanten für
den `nOpen`-
Flags-
Parameter¹

Konstante	Bedeutung
<code>CFile::modeCreate</code>	Datei wird bei Bedarf neu angelegt. Der Inhalt bestehender Dateien wird beim Öffnen gelöscht.
<code>CFile::modeRead</code>	Die Datei kann nur gelesen werden.
<code>CFile::modeReadWrite</code>	Die Datei wird zum Lesen und Schreiben geöffnet.
<code>CFile::modeWrite</code>	Die Datei wird nur zum Schreiben geöffnet.



```
CFile datei("c:\\demo.txt", CFile::modeCreate |
CFile::modeWrite );
```

Alternativ dazu können Sie

1. den Konstruktor `CFile()` verwenden, dem keine Parameter übergeben werden, und
2. danach die `Open()`-Methode aufrufen.

Dateien schließen

Zum Schließen rufen Sie die Methode `Close()` auf.



```
datei.Close();
```

Lesen aus und Schreiben in ein CFile-Objekt

Das Lesen und Schreiben wird für ein `CFile`-Objekt mit den Methoden `Read()` und `Write()` ausgeführt.



Natürlich muß die Datei in dem entsprechenden Modus geöffnet sein, damit die Operation erfolgreich durchgeführt werden kann.

Lesen

```
virtual UINT Read( void* lpBuf, UINT nCount );
```

- ✗ `lpBuf` bezeichnet einen Speicherbereich (Puffer), in den die Daten eingelesen werden.
- ✗ `nCount` ist die Anzahl an Bytes, die maximal eingelesen werden (um einen Überlauf des Puffers zu verhindern).

¹ Eine Auflistung sämtlicher `nOpenFlag`-Konstanten finden Sie in der Online-Hilfe zu dem Indexeintrag `CFile::CFile`.

```
char str[100];
CFile datei("c:\\demo.txt", CFile::modeRead );
datei.Read(str, sizeof(str));
```



Schreiben

```
virtual void Write( const void* lpBuf, UINT nCount );
```

- ✗ `lpBuf` ist ein Zeiger auf den Speicherbereich (Puffer), in dem die auszugebenden Daten stehen.
- ✗ `nCount` ist die Anzahl an Bytes, die ausgegeben werden.

```
char str[100];
CFile datei("c:\\demo.txt", CFile::modeCreate |
                        CFile::modeWrite );
datei.Write(str, sizeof(str));
```



Fehlerbehandlung

Dateioperationen können inkorrekt ausgeführt werden. Während einige `CFile`-Methoden (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:

```
try
{
    CFile datei("c:\\demo.txt", CFile::modeCreate
                | CFile::modeWrite );
    datei.Write(str, sizeof(str));
    datei.Close();
}
catch (CFileException *e)
{
    if (e->m_cause == CFileException::diskFull)
        printf("Festplatte voll\n");
    e->Delete();
}
```

Wie Sie sehen, können Sie über das Element `m_cause` des `catch`-Parameters `e` herausfinden, welches Problem die Ausnahme ausgelöst hat.

Tabelle 11.5:
Einige Kon-
stanten für
m_cause¹

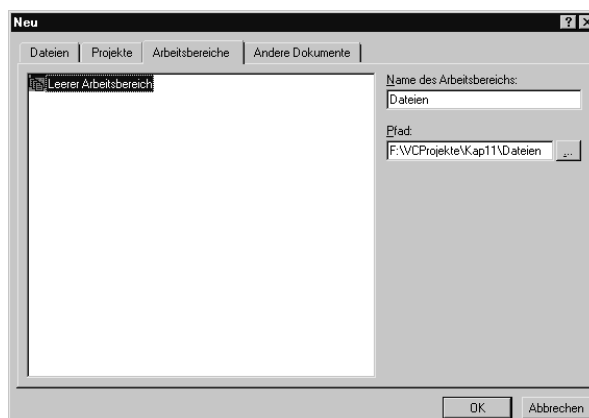
Konstante	Bedeutung
fileNotFound	Die Datei wurde nicht gefunden.
accessDenied	Der Zugriff auf die Datei wurde verweigert.
diskFull	Die Festplatte ist voll.
endOfFile	Das Dateieinde wurde erreicht.

11.2.2 Dateien schreiben

Wie man mit der Klasse `CFile` programmiert, schauen wir uns an zwei Win32-Konsolenanwendungen an. Die Projekte zu den Programmen sollen *Schreiben* und *Lesen* heißen und in einem gemeinsamen Anwendungsbereich liegen. Wir beginnen daher mit dem Anlegen eines leeren Anwendungsbereichs namens *Dateien*.

Übung 11-1: Leeren Anwendungsbereich anlegen

Bild 11.3:
Leeren Ar-
beitsbereich
anlegen



1. Legen Sie einen neuen Arbeitsbereich an. Rufen Sie dazu den Befehl `DATEI/NEU` auf, und wechseln Sie im Dialogfeld `NEU` zur Seite `ARBEITSBEREICHE`.

Wie für Projekte geben Sie einen Namen und ein übergeordnetes Verzeichnis an.

¹ Eine Auflistung sämtlicher `m_cause`-Konstanten finden Sie in der Online-Hilfe zu `CFileException::m_cause`.

Übung 11-2: In Dateien schreiben

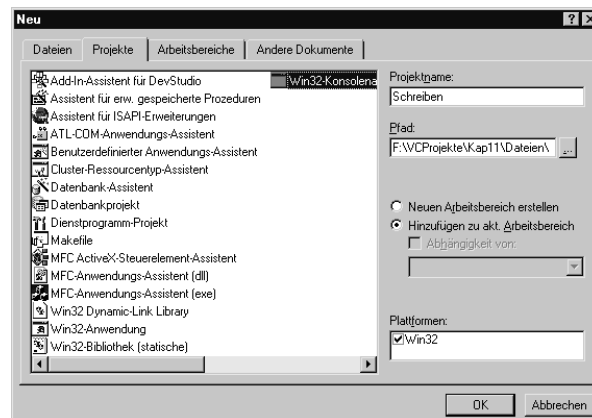


Bild 11.4:
Neues Projekt
in Arbeitsbe-
reich einfügen

1. Legen Sie ein neues Projekt an (Befehl DATEI/NEU, Seite PROJEKTE).
Geben Sie als Projektnamen *Schreiben* an, und aktivieren Sie die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH.
Wählen Sie als Projekttyp WIN32-KONSOLEANWENDUNG.
2. Im nachfolgenden Dialogfeld des »Konsolen-Assistenten« wählen Sie die Option EINE ANWENDUNG, DIE MFC UNTERSTÜTZT.
3. Laden Sie die Quelltextdatei *Schreiben.cpp* in den Editor (Doppelklick auf den Dateiknoten in der DATEIEN-Ansicht des Arbeitsbereichsfensters).

Der folgende Quelltext wird angezeigt:¹

```

1:  // Schreiben.cpp
2:  //
3:
4:  #include "stdafx.h"
5:  #include "Schreiben.h"
6:
7:
8:  //////////////////////////////////////
9:  // Das einzige Anwendungsobjekt
10:
11: CWinApp theApp;
```

¹ Ich habe den Code im Ausdruck leicht gekürzt


```

12: using namespace std;
13:
14: int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
15: {
16:     int nRetCode = 0;
17:
18:     // MFC initialisieren, Fehlermeldung bei Fehlern
19:     if (!AfxWinInit(::GetModuleHandle(NULL), NULL,
                     ::GetCommandLine(), 0))
20:     {
21:         // ZU ERLEDIGEN: Fehlercode bei Bedarf ändern
22:         cerr << _T("Fatal Error: MFC initialization \
                     failed") << endl;
23:         nRetCode = 1;
24:     }
25:     else
26:     {
27:         // ZU ERLEDIGEN: Anwendungsverhalten hier
                // festlegen.
28:         CString strHello;
29:         strHello.LoadString(IDS_HELLO);
30:         cout << (const TCHAR*)strHello << endl;
31:     }
32:
33:     return nRetCode;
34: }

```

4. Den Code im else-Block (Zeilen 27 bis 30) ersetzen wir durch unseren Quelltext:

```

char str[100];
// Text über Konsole einlesen
CStdioFile ausgabestream(stdout);
ausgabestream.WriteString("Tippen Sie einen Text ein: \
                          \n\n\t");
CStdioFile eingabestream(stdin);
eingabestream.ReadString(str, 99);

// Text in Datei schreiben
try
{
    CFile datei("c:\\demo.txt", CFile::modeCreate
                | CFile::modeWrite );

```

```

        datei.Write(str, sizeof(str));
        datei.Close();
    }
    catch (CFileException *e)
    {
        if (e->m_cause == CFileException::diskFull)
            printf("Festplatte voll\n");
        e->Delete();
    }

```

Wir verwenden `CStudioFile`-Objekte, um in das Konsolenfenster zu schreiben und aus der Konsole zu lesen, doch wir hätten genauso gut die üblichen C-Funktionen oder C++-Streams verwenden können. `CStudioFile` ist eine von `CFile` abgeleitete Klasse, die extra für den Zugriff auf die Konsole implementiert wurde. So verfügt `CStudioFile` beispielsweise über die Methoden `ReadString()` und `WriteString()`, mit denen man einzelne Zeilen (die mit einem Zeilenumbruch abschließen) einlesen und ausgeben kann.

CStudioFile

Im nachfolgenden try-Block wird der eingelesene String in eine Datei geschrieben. Ich glaube, dazu gibt es nichts mehr zu sagen. Sollte Ihnen etwas unklar sein, lesen Sie noch einmal in den vorangehenden Abschnitten nach.

Wie gut sind Ihre C-Kenntnisse? Wissen Sie, warum in der Dateiangabe für den Verzeichniswechsel zwei Backslashes benötigt werden? Ein einzelner Backslash in einem String leitet ein Sonderzeichen ein (beispielsweise `\n` für den Zeilenumbruch, `\t` für den Tabulator). Ein Backslash direkt vor einem zweiten Backslash zeigt an, daß der nachfolgende Backslash kein Sonderzeichen einleitet, sondern als ganz normaler Buchstabe angesehen werden soll.



5. Führen Sie das Programm aus (`Strg` + `F5`).



Bild 11.5:
Das Programm
liest einen
String von der
Konsole ein

11.2.3 Dateien lesen

Oben haben wir ein Programm erstellt und ausgeführt, das einen Text von der Konsole einlas und in eine Datei schrieb. Jetzt wollen wir ein Programm schreiben, das diese Datei öffnet, den Inhalt einliest und auf die Konsole ausgibt.

Übung 11-3: Dateien lesen

1. Legen Sie ein neues Projekt an (Befehl DATEI/NEU, Seite PROJEKTE), ohne zuvor den Arbeitsbereich aus Übung 11-2 zu schließen.

Geben Sie als Projektnamen *Lesen* an, und aktivieren Sie die Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH.

Wählen Sie als Projekttyp WIN32-KONSOLENANWENDUNG.

2. Im nachfolgenden Dialogfeld des »Konsolen-Assistenten« wählen Sie die Option EINE ANWENDUNG, DIE MFC UNTERSTÜTZT.
3. Laden Sie die Quelltextdatei *Lesen.cpp* in den Editor (Doppelklick auf den Dateiknoten in der DATEIEN-Ansicht des Arbeitsbereichsfensters).
4. Den Code im `else`-Block ersetzen Sie durch folgenden Quelltext:

```
char str[100];
// Text aus Datei lesen
try
{
    CFile datei("c:\\demo.txt", CFile::modeRead );
    datei.Read(str, sizeof(str));
    datei.Close();
}
catch (CFileException *e)
{
    if (e->m_cause == CFileException::fileNotFound)
        printf("Datei nicht gefunden\n");
    e->Delete();
}

// Text auf Konsole ausgeben
CStdioFile ausgabestream(stdout);
ausgabestream.WriteString(str);
```

5. Führen Sie das Programm aus (**Strg** + **F5**).

Wenn Sie abwechselnd die beiden Projekte im Arbeitsbereich Dateien bearbeiten wollen, rufen Sie den Befehl PROJEKT/AKTIVES PROJEKT FESTLEGEN auf, wenn Sie das Projekt wechseln wollen.



11.3 Was bedeutet »Serialisierung«?

Ein in der MFC-Bibliothek wichtiges Konzept ist die Serialisierung.

Als Serialisierung bezeichnet man das Lesen und Schreiben von Objekten, genauer gesagt von CObject-Objekten.

CObject ist die oberste Basisklasse der MFC-Klassen. Alle Objekte von Klassen (MFC-Klassen wie vom Programmierer selbst definierte Klassen) können mit Hilfe der Serialisierung in Dateien geschrieben oder aus Dateien gelesen werden.¹

Um Objekte per Serialisierung in CFile-Dateien zu schreiben oder umgekehrt aus Dateien einzulesen, braucht man einen Mittler: die CArchive-Klasse.

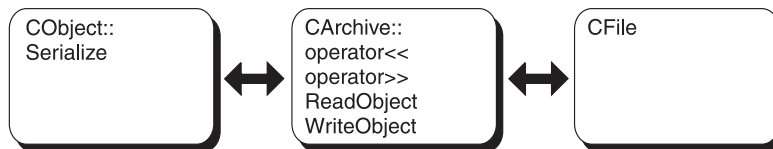


Bild 11.6: Beziehung zwischen CObject, CArchive und CFile

11.3.1 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 allgemeine Hüllklasse 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, das heißt, sie delegiert die Aufgabe des Speicherns an die Objekte, die dazu eine eigene Serialize()-Methode definieren müssen.

¹ Nicht nur das! Die Serialisierung unterstützt auch die Zwischenablage und OLE.

Serialisierbare Klassen

Damit die Objekte einer Klasse serialisierbar sind, müssen drei Dinge gegeben sein:

- ✗ Die Klasse muß von `CObject` abgeleitet sein.
- ✗ Die Klasse muß die Makros `DECLARE_SERIAL(CKlasse)` (in der Klassendeklaration) und `IMPLEMENT_SERIAL(CKlasse, CObject, 1)` (in der Quelltextdatei) aufrufen.
- ✗ Die Klasse muß eine eigene `Serialize()`-Methode definieren.

Das folgende Beispiel zeigt eine Klasse `CKoord`, in deren Objekten zweidimensionale Koordinaten abgespeichert werden und die serialisierbar ist.

```
// Koord.h    Header-Datei
class CKoord : public CObject
{
    DECLARE_SERIAL(CKoord)
public:
    CKoord();
    CKoord(int x_koord, int y_koord);
    int x;
    int y;
    virtual void Serialize(CArchive& ar);
};

// Koord.cpp    Quelltextdatei
#include "stdafx.h"
#include "Koord.h"
IMPLEMENT_SERIAL(CKoord, CObject, 1)
CKoord::CKoord()
{
    x = 1;
    y = 1;
}
CKoord::CKoord(int x_koord, int y_koord)
{
    x = x_koord;
    y = y_koord;
}
void CKoord::Serialize(CArchive& ar)
{

```

```

if (ar.IsStoring())
{
    // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
    ar << x << y;
}
else
{
    // ZU ERLEDIGEN: Hier Code zum Laden einfügen
    ar >> x >> y;
}
}

```

Objekte serialisierbarer Klassen können einfach mit den Streamoperatoren (<<, >>) an CArchive-Objekte übergeben werden.

11.3.2 Lesen und Schreiben über CArchive

Die folgenden Programme können Sie in einem Texteditor erzeugen und über die Kommandozeile kompilieren (siehe Kapitel 1):

```

cl /MT writeListe.cpp
cl /MT readListe.cpp

```

Sie müssen nicht bis ins Detail verstehen, was in diesen Programmen passiert. Schauen Sie sich einfach nur den Quelltext an, und achten Sie auf das Zusammenspiel von CFile, CArchive und den zu serialisierenden Objekten.

Interessanter ist für uns letztendlich, wie man die Serialisierung in MFC-Windows-Anwendungen nutzen kann, und diesem Thema werden wir uns im nächsten Kapitel bei der Implementierung des Mäuse-Editors eingehender widmen.

Schreiben

Das folgende 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()-Methode. Sie unterstützt jedoch nicht die Operatoren << und >>. Dazu muß man erst eine passende operator<<-Funktion definieren. Diese Art der Deklaration führen wir ebenfalls für Objekte vom Typ CList<WORD und WORD> aus.

```

// writeListe.cpp
#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;
    cout << "Liste wird angelegt: ";
    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;
}
```

Ausgabe Liste wird angelegt: 41 18467 6334 26500 19169 15724 11478 29358
26962 24464

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.

Einlesen

Auch zum Einlesen der Liste benötigt man nur eine Zeile.

```
// readListe.cpp
#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 << "Eingelesene Liste: ";
    while (pos)
    {
        int n = myList.GetNext(pos);
        cout << n << ' ';
    }
}
```

Eingelesene Liste: 41 18467 6334 26500 19169 15724 11478 29358 26962 24464 *Ausgabe*

11.4 Ein einfacher Texteditor

Zum Schluß werden wir noch ganz schnell einen kleinen Texteditor erstellen, mit dem man ASCII-Textdateien laden, anzeigen, bearbeiten, speichern und drucken kann und der auch die Zwischenablage unterstützt. Was schätzen Sie, wie lange wir dafür brauchen werden? Sie haben sich schon verschätzt, es geht wesentlich schneller.

Übung 11-4: Grundgerüst für einen Texteditor

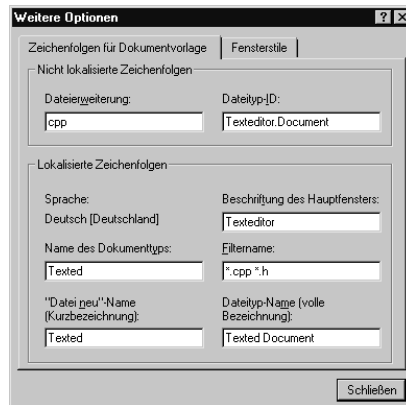
1. Legen Sie ein neues Projekt an (Befehl DATEI/NEU, Seite PROJEKTE).

Geben Sie als Namen »Texteditor« ein. Wählen Sie ein passendes übergeordnetes Verzeichnis für das Projekt, und lassen Sie einen neuen Arbeitsbereich für das Projekt anlegen.

Links wählen Sie den MFC-Anwendungs-Assistenten aus.

2. Im ersten Schritt des Assistenten entscheiden Sie sich für eine SDI-Anwendung mit Dokument/Ansicht-Architektur.

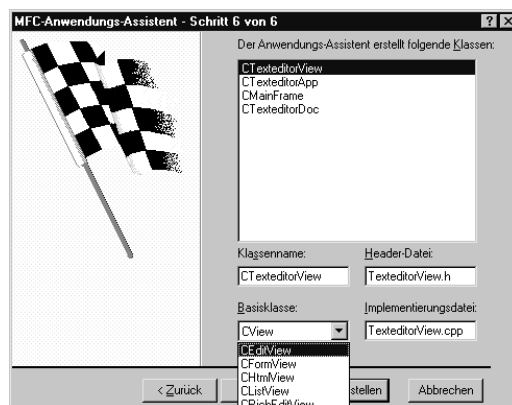
Bild 11.7:
Einstellungen
für die Datei-
Dialoge



- Im vierten Schritt behalten Sie die Voreinstellungen bei und klicken auf den Schalter WEITERE. Auf der Seite ZEICHENFOLGEN FÜR DOKUMENTVORLAGE geben Sie als Dateierweiterung »cpp« an (diese Extension wird automatisch beim Speichern angehängt, wenn der Anwender keine Extension für einen Dateinamen angibt).

Geben Sie im Feld FILTERNAME die Extensionen an, nach denen die Dialoge zum Öffnen und Speichern die Verzeichnisse durchsuchen sollen.

Bild 11.8:
Eigene Basis-
klasse für die
Ansicht
vorgeben



- Jetzt kommt die wichtigste Einstellung überhaupt! Im sechsten Schritt klicken Sie oben auf die Klasse für das Ansichtsfenster und wählen als Basisklasse nicht mehr CVIEW, sondern CEDITVIEW aus.
- Lassen Sie das Projekt jetzt fertigstellen und ausführen (**Strg** + **F5**).

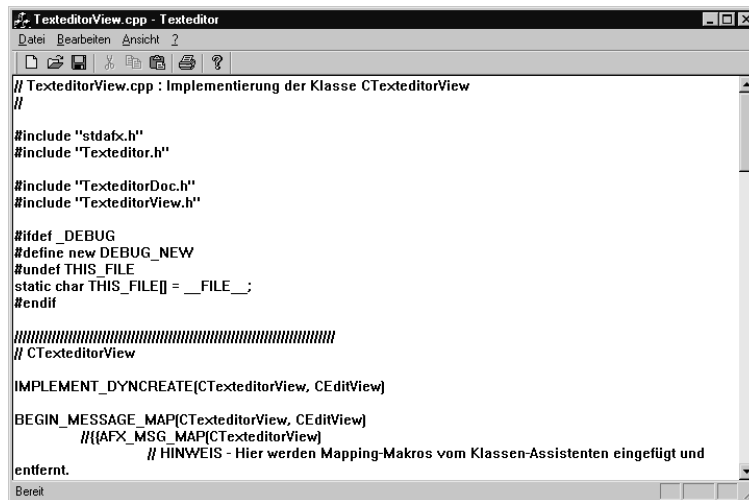


Bild 11.9:
Der fertige
Texteditor

Sie waren bisher ganz zufrieden mit Visual C++ und der Arbeit der Assistenten? Dann dürften Sie jetzt begeistert sein. Der Assistent freut sich auch, denn endlich durfte er das machen, was er am besten kann: ein Anwendungsgerüst für einen Texteditor aufsetzen.

11.5 Zusammenfassung

Zur Textverarbeitung kann man sich verschiedener Wege bedienen:

- ✗ Meldungsfenster aufrufen
- ✗ Text in ein Fenster zeichnen
- ✗ Eingabefelder (CEdit oder CRichEdit) verwenden
- ✗ Ansichtsfenster verwenden, die eines der Eingabefelder-Steuerelemente kapseln.

Zur Arbeit mit Dateien nutzt man die MFC-Klasse CFile, die alle wichtigen Methoden zur Verwaltung und zum Schreiben und Lesen von Dateien enthält. Allerdings ist die Klasse auf die Bearbeitung binärer Dateien spezialisiert. In MFC-Anwendungen, die mit dem Anwendungs-Assistenten erstellt wurden, arbeitet man selten direkt mit der Klasse CFile. Vielmehr nutzt man den integrierten Mechanismus der Serialisierung.

Serialisierung ist ein Mechanismus, der den von CObject abgeleiteten Klassen das Schreiben oder Lesen der eigenen Daten in und aus einem beständigen Speicher ermöglicht (beispielsweise eine Datei).

Klassen, deren Objekte serialisiert werden sollen, müssen die `CObject`-Methode `Serialize()` überschreiben und mit dem `DECLARE_SERIAL`-Makro deklariert und mit `IMPLEMENT_SERIAL` implementiert werden.

Der Anwendungsrahmen stellt eine Standardimplementierung der Menübefehle `DATEI/ÖFFNEN` und `DATEI/SPEICHERN` zur Verfügung. Diese Implementierungen rufen die `SERIALIZE()`-Methode Ihrer Dokumentklasse auf. Die Funktion muß von Ihnen implementiert werden und sollte alle beständigen Daten des Dokuments serialisieren.

11.6 Fragen

1. Wie zeigt man am schnellsten einen Text an?
2. Wie kann man für die Textausgabe mit `TextOut()` oder `DrawText()` eine bestimmte Schriftart auswählen?
3. Muß man in MFC-Anwendungen die Klasse `CString` für Strings verwenden, oder kann man auch die C++-Klasse `string` verwenden?
4. Welche MFC-Klasse dient der Unterstützung von Dateioperationen?
5. Kann man in Windows-Anwendungen auch die C/C++-Funktionen und -Klassen zur Dateibehandlung verwenden (`fopen()`, `fprintf()`, `fstream`)?
6. Welche Bedingungen müssen Klassen erfüllen, deren Objekte serialisierbar sein sollen?

11.7 Aufgaben

1. Schreiben Sie ein Programm, das Meßwerte aus einer Datei einliest und daraus den Mittelwert berechnet. Ob Sie ein Konsolenprogramm oder eine Windows-Anwendung oder eine auf dem Anwendungsgerüst des Assistenten aufbauende Anwendung implementieren, bleibt Ihnen überlassen. (Als Vorlage für die Serialisierung eigener Daten in MFC-Anwendungsgerüsten können Sie die Lösung der 2. Aufgabe aus Kapitel 12 heranziehen.)

Zeichnen

Kommen wir nun zu meinem Lieblingsthema: der Grafikausgabe. Ich habe das Thema auf zwei Kapitel aufgeteilt:

- ✕ In diesem Kapitel zeige ich Ihnen, wie man in Fenster zeichnet. Dabei werden Ihnen Ihre Erfahrungen im Umgang mit dem Bildeditor von Visual C++ (oder anderen Rastergrafikeditoren) sehr zugute kommen, denn das Zeichnen mit den Klassen der MFC verläuft analog zum Zeichnen mit den Werkzeugen des Bildeditors.
- ✕ Im nächsten Kapitel schauen wir uns dann an, wie man fertige Bitmaps aus BMP-Dateien in ein Programm lädt, manipuliert und ausgibt.

Bevor Sie sich jedoch in die Arbeit stürzen und beginnen, das Kapitel durcharbeiten, muß ich Sie warnen. So einfach wie bei der Textausgabe wird es nicht. Die Grafikausgabe ist an bestimmte Formalismen und Konzepte gebunden, die beachtet sein wollen. Zudem können wir nicht auf die gleiche Unterstützung wie bei der Textbehandlung vertrauen: Weder gibt es Standardsteuerelemente, die Grafikeingaben unterstützen, noch können wir den Anwendungs-Assistenten einen fertigen Grafikeditor für uns implementieren lassen. Solide Handarbeit ist gefragt. Die nötigen Grundlagen dazu lernen Sie in diesem und dem nachfolgenden Kapitel.

Vergessen Sie nicht die Aufgaben zu diesem Kapitel! Sie sind diesmal besonders interessant.



Sie lernen in diesem Kapitel:

- ✗ Wie man grundsätzlich vorzugehen hat, wenn man in ein Fenster zeichnen möchte
- ✗ Was Gerätekontexte sind und welche Vorteile sie haben
- ✗ Welche Zeichenmethoden uns zur Verfügung stehen
- ✗ Wie man mit einfachen Zeichenoperationen faszinierende Grafiken aufbaut
- ✗ Wie man Stift und Pinsel führt
- ✗ Wie man Farben auswählt
- ✗ Wie man Fraktale programmiert
- ✗ Wie man einzelne Pixel einfärbt

12.1 Das Arbeitsmaterial des Künstlers

Wenn Sie einen Text erstellen, sei es ein Brief, eine kurze Notiz oder ein Essay, greifen Sie sich einfach ein Blatt Papier und einen Stift und schreiben drauf los.

Ganz anders verhält es sich, wenn Sie eine Zeichnung oder ein Bild, beispielsweise ein Ölgemälde, anfertigen wollen.

- ✗ **DIE LEINWAND** (Gerätekontext). Zuerst müssen Sie sich um den passenden Untergrund kümmern. Für Kohle- oder Bleistiftzeichnungen reicht noch einfaches weißes (oder auch gefärbtes) Papier, für Aquarelle brauchen Sie spezielles saugfähiges Papier, für chinesische Tuschzeichnungen werden Sie Reis- oder Seidenpapier verwenden, und für Ölbilder schließlich brauchen Sie eine grundierte Leinwand. Für Grafikausgaben am Computer brauchen Sie einen Gerätekontext, der zu dem Fenster korrespondiert, in das Sie zeichnen wollen.
- ✗ **STIFTE** (CPen) und **PINSEL** (CBrush). Als nächstes müssen Sie sich um Ihre Zeichenwerkzeuge kümmern. Meist verwendet man zarte Kohle- oder Bleistifte zum Vorzeichnen der Konturen und Pinsel unterschiedlicher Struktur und Breite zum Auftragen der Farbe. In Ihren Programmen verwenden Sie ein CPen-Objekt für Konturen und ein CBrush-Objekt zum Füllen von Figuren (Rechtecke, Ellipsen, Polygone).

- ✗ **DIE FARBEN (RGB).** Die Farbe selbst ist mit die größte Herausforderung für den Künstler. Eine bestimmte Auswahl an Farben gibt es zu kaufen. Der Vorteil dieser Farben ist, daß sie immer den gleichen Ton und die gleiche Sättigung haben. Darüber hinaus hat der Maler aber auch die Möglichkeit, eigene Farben anzumischen und zu verwenden. Die Farben, mit denen er arbeiten möchte, stellt er dann auf seiner Malerpalette zusammen. In Ihren Programmen »tauchen« Sie Ihre Zeichenwerkzeuge vor dem Zeichnen in die gewünschte Farbe ein.

Leinwände, Stifte und Pinsel, Farben und Paletten – das sind also die Zeichnungsmittel, mit denen Sie in jedem herkömmlichen Grafikprogramm arbeiten, und es sind auch die Objekte, mit denen wir es bei der Grafikprogrammierung zu tun haben.

Wie geht man dabei vor?

Einfache Zeichenausgaben

1. Man beschafft sich einen Gerätekontext als Leinwand.
2. Man ruft zum Zeichnen die entsprechenden Methoden des Gerätekontextes auf.

```
CClientDC dc(this);
dc.Ellipse(100, 100, 200, 200);
```



Zeichenausgaben mit eigenen Zeichenwerkzeugen

1. Man beschafft sich einen Gerätekontext als Leinwand.
2. Man richtet die Zeichenwerkzeuge (beispielsweise einen farbigen Pinsel) her.
3. Man lädt die Zeichenwerkzeuge in den Gerätekontext.
4. Man ruft zum Zeichnen die entsprechenden Methoden des Gerätekontextes auf.
5. Man nimmt die Zeichenwerkzeuge wieder aus dem Gerätekontext heraus.
6. Man löscht die Zeichenwerkzeuge.

```
CClientDC dc(this);
CBrush roterPinsel(RGB(255, 0, 0));
CBrush *p_alterPinsel = dc.SelectObject(&roterPinsel);
dc.Ellipse(100, 100, 200, 200);
dc.SelectObject(p_alterPinsel);
// lokal definiertes Zeichenwerkzeug wird am Ende der
// Methode aufgelöst
```



12.2 Gerätekontexte

Wenn Sie Text oder Grafiken ausgeben wollen, sprechen Sie das Ausgabe-
gerät (sei es ein Drucker oder einfach ein Fenster der Anwendung) nicht di-
rekt, sondern über einen Gerätekontext an. Der Umweg über den Geräte-
kontext hat für den Programmierer den Vorteil, daß er von der jeweils
installierten Hardware weitgehend unabhängig ist. Statt abfragen zu müs-
sen, wie die angeschlossenen Geräte ausgestattet sind, schreibt er in den
Gerätekontext und überläßt Windows die Übergabe an die Hardware.

12.2.1 Gerätekontextklassen

In welchen Gerätekontext Sie schreiben, hängt davon ab, über welche
Schnittstelle Ihre Informationen ausgegeben werden sollen. Die MFC stellt
dabei für die verschiedenen Ausgabegeräte spezielle Klassen zur Verfügung.

Tabelle 12.1:
Gerätekon-
textklassen
für Fenster

Klasse	Ausgabeeinheit
CDC	Basisklasse aller Gerätekontextklassen mit reicher Auswahl an Zeichen- und Ausgabeoperationen.
CPaintDC	Spezieller Fenster-DC zur Implementierung von Behandlungsmethoden zur WM_PAINT-Nachricht. Die Fensterklassen der MFC implementieren standardmäßig bereits OnPaint()-Methoden zur Behandlung der WM_PAINT-Nachricht. In diesen wird eine CPaintDC-Instanz gebildet und an die virtuelle Methode OnDraw() übergeben, die Sie zur Ausgabe überschreiben sollten.
CClientDC	Gerätekontext für die Ausgabe in den Client-Bereich eines Fensters.
CWindowDC	Gerätekontext für den gesamten Bereich eines Fensters (einschließlich Rahmen, Titel etc.)

Wie kommt man nun an einen passenden Gerätekontext zu dem Fenster, in das man zeichnen möchte?

Dies hängt ganz davon ab, in welcher Methode Sie Ihre Zeichenoperationen ausführen. Wie Sie bereits aus Kapitel 7 wissen, speichert Windows keine Informationen über Fensterinhalte. Konkret bedeutet dies, daß bei bestimmten Fensteroperationen (wie Verkleinern, Vergrößern, Minimieren, in den Vordergrund holen) der Fensterinhalt nicht von Windows rekonstruiert werden kann. Windows schickt daher in solchen Fällen eine WM_PAINT-Nach-

richt ab, um das Fenster darüber zu informieren, daß es seinen Inhalt selbst neu zeichnen soll. Ihr MFC-Anwendungsgerüst fängt diese Nachricht ab und ruft die virtuelle Methode `OnDraw()` auf. In diese sollten Sie alle Anweisungen schreiben, die zum Rekonstruieren des Fensters (üblicherweise die gesamte Grafik- oder Textausgabe) erforderlich sind.

- ✗ Für Zeichenausgaben, die Sie in `OnDraw()` vornehmen, übergibt Ihnen die MFC einen passenden Gerätekontext als Argument an `OnDraw()`.
- ✗ Für Zeichenausgaben, die Sie in anderen Methoden vornehmen, müssen Sie sich einen Gerätekontext als Instanz einer passenden Gerätekontextklasse erzeugen.

Für beide Verfahren wollen wir uns jetzt ein Beispiel anschauen. Wir werden dazu ein Programm namens *Kreise* aufsetzen. Wenn der Anwender mit der linken Maus in den Client-Bereich des Hauptfensters des Programms klickt, wird dort ein Kreis von zufälligem Radius eingezeichnet.

12.2.2 Gerätekontexte selbst erzeugen

Wenn Sie wie in dem folgenden Programm Mausklicke des Anwenders mit dem Einzeichnen von Kreisen beantworten wollen, können Sie nicht die `OnDraw()`-Methode zum Zeichnen verwenden, sondern müssen die Ausgabe direkt in der Behandlungsmethode zur Windows-Nachricht `WM_LBUTTONDOWN` vornehmen. Dies bedeutet, daß Sie sich Ihren eigenen Gerätekontext für die Ausgabe in Ihr Fenster erzeugen müssen.

Üblicherweise wird man nur in den Client-Bereich eines Fensters zeichnen wollen, weswegen man den Gerätekontext als Instanz der Klasse `CClientDC` erzeugt. Dabei übergibt man dem Konstruktor einen Zeiger auf das Fenster. Spielt sich das Ganze in einer Methode der Fensterklasse des Fensters ab, was meist der Fall sein wird, kann man einfach den `this`-Zeiger übergeben:

```
void CDemoView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
```

Sollte dies einmal nicht der Fall sein, muß man den Fensterzeiger als Argument an die Methode übergeben.

```
void EineFunktion( LPVOID pWnd)
{
    CClientDC dc( CWnd*) pWnd);
```


Übung 12-1: Kreise an Mausclickposition zeichnen

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Kreise« genannt.
2. Richten Sie mit Hilfe des Klassen-Assistenten in der Klasse des Ansichtsfensters eine Nachrichtenbehandlungsmethode für WM_LBUTTONDOWN ein.
3. Zeichnen Sie mit Hilfe der CDC-Methode `Ellipse()` einen Kreis mit zufälligem Radius an der Position des Mausclicks.

```
void CKreiseView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    int b = (int) (100.0 * rand() / RAND_MAX);
    dc.Ellipse(point.x-b, point.y-b, point.x+b, point.y+b);

    CView::OnLButtonDown(nFlags, point);
}
```



Sie können zum Einrichten von Gerätekontexten auch die API-Funktionen verwenden – beispielsweise `GetDC()`, die es auch als Methode der Klasse `CWnd` gibt. Beachten Sie dann, daß Sie derart erzeugte Gerätekontexte selbst durch Aufruf der Methode `ReleaseDC()` löschen müssen.

12.2.3 OnDraw()

Die `OnDraw()`-Methode dient der Rekonstruktion des Fensterinhalts und wird automatisch ausgeführt, wenn Windows eine `WM_PAINT`-Nachricht an das Fenster schickt, um ihm mitzuteilen, daß es seinen Fensterinhalt neu zeichnen soll. Die `OnDraw()`-Methode wird auf einem Umweg aufgerufen. Die `WM_PAINT`-Nachricht wird von den Fensterklassen der MFC in einer `OnPaint()`-Methode abgefangen. Diese bereitet einen passenden Gerätekontext für die Zeichenausgabe vor und übergibt diesen an die Methode `OnDraw()`.

Wenn wir also `OnDraw()` in unseren Fensterklassen überschreiben, können (ja müssen!) wir uns dieses Gerätekontextes bedienen.

```
void CKreiseView::OnDraw(CDC* pDC)
{
    CKreiseDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
```

```

// ZU ERLEDIGEN: Hier Code zum Zeichnen der
// ursprünglichen Daten hinzufügen
pDC->Ellipse(100, 100, 140, 140);
}

```

Beachten Sie den Aufruf von `GetDocument()`. Da hinter dem Doc/View-Modell die Trennung von Datenverwaltung und Datenanzeige steht, ist die Ansichtsklasse üblicherweise gezwungen, sich die anzuzeigenden Daten von dem zugehörigen Dokument zu besorgen. Zu diesem Zweck besitzt die Methode `OnDraw()` standardmäßig einen Zeiger `pDoc` auf das Dokument. Über diesen Zeiger kann man dann auf `public`-Methoden und -Elemente der Dokumentklasse zugreifen.

Übung 12-2: Kreise rekonstruieren

Wir wollen das Programm *Kreise* so ausbauen, daß die eingezeichneten Kreise rekonstruiert werden, wenn der Anwender das Fenster minimiert oder in den Hintergrund schickt und wieder hervorholt.

Dazu werden wir in der Dokumentklasse einen Zähler und ein `CPoint`-Array einrichten, in dem wir die Mittelpunkte der Kreise abspeichern. (Die Kreisradien speichern wir nicht; die Kreise werden mit neuen Radien gezeichnet.)

1. Deklarieren Sie in der Dokumentklasse einen `int`-Zähler und ein `CPoint`-Array für 100 Kreismittelpunkte.

```

class CKreiseDoc : public CDocument
{
protected: // Nur aus Serialisierung erzeugen
    CKreiseDoc();
    DECLARE_DYNCREATE(CKreiseDoc)
// Attribute
public:
    int m_nZaehler;
    CPoint m_Kreise[100];
}

```

2. Initialisieren Sie den Zähler im Konstruktor der Dokumentklasse mit dem Wert 0.

```

CKreiseDoc::CKreiseDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion
    // einfügen
    m_nZaehler = 0;
}

```

3. Speichern Sie in der `OnLButtonDown()`-Methode die Kreismittelpunkte, und inkrementieren Sie den Zähler.

```
void CKreiseView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    CKreiseDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CClientDC dc(this);
    int b = (int) (100.0 * rand() / RAND_MAX);
    dc.Ellipse(point.x-b, point.y-b, point.x+b, point.y+b);
    if (pDoc->m_nZaehler < 100)
        pDoc->m_nZaehler++;
    pDoc->m_Kreise[pDoc->m_nZaehler - 1] = point;

    CView::OnLButtonDown(nFlags, point);
}
```

4. Rekonstruieren Sie die Kreise in der `OnDraw()`-Methode.

```
void CKreiseView::OnDraw(CDC* pDC)
{
    CKreiseDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    for(int i = 0; i < pDoc->m_nZaehler; i++)
    {
        int b = (int) (100.0 * rand() / RAND_MAX);
        pDC->Ellipse(pDoc->m_Kreise[i].x - b,
                    pDoc->m_Kreise[i].y - b,
                    pDoc->m_Kreise[i].x + b,
                    pDoc->m_Kreise[i].y + b);
    }
}
```

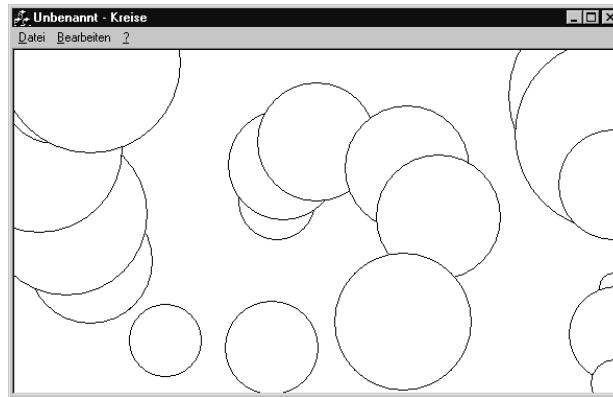


Bild 12.1:
In einen Gerä-
tekontext für
ein Fenster
zeichnen

Invalidate() und UpdateWindow()

Im obigen Programm zeichnen wir sowohl in der `OnLButtonDown()`- als auch der `OnDraw()`-Methode. Genausogut könnten wir die gesamte Grafikausgabe aber in der `OnDraw()`-Methode erledigen. Doch wie sieht der Anwender dann, wie die Kreise für seine Mausklicks liegen? Muß er nach jedem Mausklick das Fenster minimieren und wiederherstellen, um das Neuzeichnen des Fensters anzustoßen?

Nein, natürlich nicht. Wir können das Neuzeichnen des Fensters auch aus dem Programm anstoßen, indem wir einfach selbst eine `WM_PAINT`-Nachricht auslösen. Dazu brauchen wir nur die `CWnd`-Methode `UpdateWindow()` aufzurufen.

Zur Sicherheit sollte man zuvor noch den Fensterinhalt als ungültig erklären. Sonst kann es passieren, daß der Fensterinhalt trotz `WM_PAINT`-Nachricht nicht neu gezeichnet wird, weil das Anwendungsgerüst der Meinung ist, daß ein Neuzeichnen nicht notwendig ist. Rufen Sie also vor `UpdateWindow()` die `CWnd`-Methode `Invalidate()` auf.

```
void CKreiseView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    ...
    Invalidate();
    UpdateWindow();

    CView::OnLButtonDown(nFlags, point);
}
```

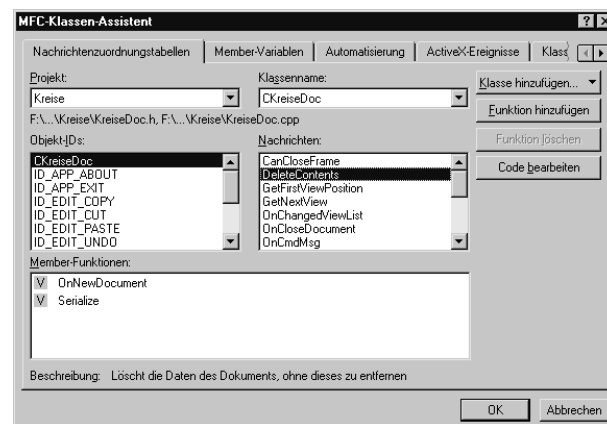


Übung 12-3: Fenster löschen und neu zeichnen lassen

In unserem Beispiel wäre es unsinnig, das Fenster als Antwort auf jeden Mausklick neu zeichnen zu lassen. Dagegen wäre es ganz schön, wenn der Anwender die Möglichkeit hätte, den Fensterinhalt auch wieder zu löschen.

1. Laden Sie die Menü-Ressource IDR_MAINFRAME in den Menü-Editor.
2. Löschen Sie alle Menübefehle bis auf DATEI/NEU und DATEI/BEENDEN.

Bild 12.2:
Überschreiben
der Delete-
Contents()-
Methode



3. Sie können mit Hilfe des Klassen-Assistenten in der Dokumentklasse die Methode DeleteContents() überschreiben. Diese Methode wird auf dem Umweg über etliche Methoden des Anwendungsgerüsts aufgerufen, wenn der Befehl DATEI/NEU ausgewählt wird. In der Methode brauchen Sie nur den Zähler m_nZaehler auf 0 zu setzen. Das nachfolgende Neuzeichnen des Fensters wird vom Anwendungsgerüst veranlaßt.

```
void CKreisDoc::DeleteContents()
{
    // TODO: Speziellen Code hier einfügen und/oder
    // Basisklasse aufrufen
    m_nZaehler = 0;
    CDocument::DeleteContents();
}
```

12.3 Die Zeichenmethoden

Die eigentlichen Zeichenoperationen werden mit Hilfe der Methoden des Gerätekontext ausgeführt. In der Basisklasse für die Gerätekontexte, CDC, sind daher eine ganze Reihe von Methoden zum Zeichnen definiert. All diese Methoden können Sie auch in den abgeleiteten Gerätekontextklassen CPaintDC und CClientDC verwenden.

12.3.1 Übersicht

Methoden	Ausgabeeinheit
Linien	
CPoint MoveTo(int x, int y);	Die Linienfunktionen nutzen das Konzept der »aktuellen Zeichenposition«. Mit dieser Methode können Sie die aktuelle Zeichenposition auf eine bestimmte Koordinate im Gerätekontext setzen.
CPoint MoveTo(POINT point);	
	Üblicherweise ruft man diese Methode auf, um den Anfangspunkt einer Linie festzulegen.
	Liefert die letzte aktuelle Zeichenposition zurück.
BOOL LineTo(int x, int y);	Zeichnet eine Linie von der aktuellen Zeichenposition bis zu der als Argument übergebenen Koordinate.
BOOL LineTo(POINT point);	
	Die übergebene Koordinate ist danach die neue aktuelle Zeichenposition
Bögen	
BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);	Zeichnet einen elliptischen Bogen. Der Bogen wird spezifiziert durch ein umgebendes Rechteck, sowie einen Anfangs- und Endpunkt (die nicht unbedingt auf dem Bogen liegen müssen).
BOOL Arc(LPCRECT lpRect, POINT ptStart, POINT ptEnd);	

Tabelle 12.2:
Auswahl an
Zeichen-
methoden¹

¹ Für eine vollständige Beschreibung aller Zeichenmethoden der Klasse CDC schlagen Sie bitte in der Online-Hilfe unter dem Indexeintrag CDC, class members nach.

Tabelle 12.2:
Auswahl an
Zeichen-
methoden
(Fortsetzung)

Methode	Ausgabeeinheit
Rechtecke	Position und Abmessung des zu zeichnenden Rechtecks werden durch Angabe der oberen linken Ecke sowie Breite und Höhe oder durch Übergabe eines CRect- oder RECT-Objekts spezifiziert.
void Draw3dRect(LPCRECT lpRect, COLORREF clrTopLeft, COLORREF clrBottomRight);	Zeichnet ein dreidimensionales Rechteck. Übergeben wird ein umgebendes Rechteck, die Farbe für den oberen und linken Teil des Rahmens und die Farbe für den unteren und rechten Teil des Rahmens).
void Draw3dRect(int x, int y, int cx, int cy, COLORREF clrTopLeft, COLORREF clrBottomRight);	
void FillRect(LPCRECT lpRect, CBrush* pBrush);	Zeichnet ein Rechteck und füllt es in der Farbe und dem Muster des übergebenen Pinsel-Objekts. Der linke und obere Rahmen werden ebenfalls eingefärbt.
void FillSolidRect(LPCRECT lpRect, COLORREF clr);	Zeichnet ein Rechteck und füllt es in der angegebenen Farbe (ohne Muster).
void FillSolidRect(int x, int y, int cx, int cy, COLORREF clr);	
void FrameRect(LPCRECT lpRect, CBrush* pBrush);	Zeichnet einen Rahmen um das angegebene Rechteck. Das Pinsel-Objekt bestimmt Farbe und Muster des Rahmens.
BOOL Rectangle(int x1, int y1, int x2, int y2);	Zeichnet ein Rechteck. Der Rahmen wird mit dem Stift-Objekt, der Inhalt des Rechtecks mit dem Pinsel-Objekt des Gerätekontextes gezeichnet.
BOOL Rectangle(LPCRECT lpRect);	
BOOL RoundRect(int x1, int y1, int x2, int y2, int x3, int y3);	Zeichnet ein Rechteck mit abgerundeten Ecken. Die Abrundung wird durch Breite und Höhe einer Ellipse definiert.
BOOL RoundRect(LPCRECT lpRect, POINT point);	

Methode	Ausgabeeinheit
Kreise	
<pre>BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);</pre> <pre>BOOL Chord(LPCRECT lpRect, POINT ptStart, POINT ptEnd);</pre>	Zeichnet ein Ellipsensegment – die Schnittfigur einer Ellipse mit einer Linie. Übergeben werden das umschließende Rechteck der Ellipse sowie Start- und Endpunkt der schneidenden Linie.
<pre>BOOL Ellipse(int x1, int y1, int x2, int y2);</pre> <pre>BOOL Ellipse(LPCRECT lpRect);</pre>	Zeichnet eine Ellipse, die das übergebene Rechteck ausfüllt. Ist das Rechteck ein Quadrat, ist die Ellipse ein Kreis.
<pre>BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);</pre> <pre>BOOL Pie(LPCRECT lpRect, POINT ptStart, POINT ptEnd);</pre>	Zeichnet ein Tortenstück aus einer Ellipse. Übergeben werden das umschließende Rechteck der Ellipse sowie Start- und Endpunkt des Bogens.
Polygone	
<pre>BOOL Polygon(LPPOINT lpPoints, int nCount);</pre>	Zeichnet ein Polygon. Übergeben werden die Punkte, die zu verbinden sind. Der letzte Punkt wird automatisch mit dem ersten Punkt verbunden.

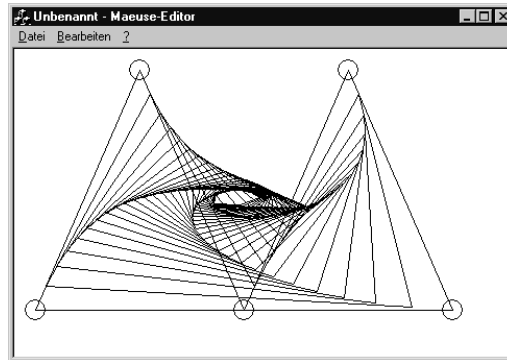
*Tabelle 12.2:
Auswahl an
Zeichen-
methoden
(Fortsetzung)*

Wir haben schon etliche Male in früheren Beispielen auf diese Methoden zurückgegriffen – vor allem auf die Methode `Ellipse()`.

Jetzt werden wir ein Programm schreiben, das mit Hilfe der Methoden `MoveTo()` und `LineTo()` Liniengrafiken auf den Bildschirm zaubert. Ich nenne das Programm den »Mäuse-Editor«.

12.3.2 Der Mäuse-Editor

Bild 12.3:
Mit dem
Mäuse-Editor
erzeugte
Grafik



Der Mäuse-Editor ist ein Grafikprogramm, das Liniengrafiken erzeugt (siehe Abbildung 12.3). Dahinter steht ein spezieller Algorithmus, der diese Grafiken aufbaut, und diesen Algorithmus möchte ich Ihnen vorab beschreiben.

Der Mäuse-Algorithmus

Den Algorithmus, der hinter diesem Programm steht, können Sie sich am besten als Fangspiel denken.

Die Startpositionen der Fänger werden vorgegeben. Auf diesen Startpositionen sitzen ... beispielsweise Mäuse. Dann fällt der Startschuß, und jede Maus läuft in Richtung auf die nächste Maus los. Nach einer kurzen Strecke, die von der Geschwindigkeit der Mäuse abhängt, bleiben die Mäuse stehen. In der nächsten Iteration laufen sie wieder los, aber da jede Maus mittlerweile ihre Position verändert hat, ändern sich auch die Richtungen, in die die Mäuse nun laufen. Wenn Sie die Linien den Positionen der Mäuse nachziehen, entsteht so etwas wie eine Spirale, da die Mäuse immer näher aufeinander zukommen.

Übung 12-4: Erste Version des Mäuse-Editors

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Maeuse« genannt.

Klicken Sie im vierten Schritt auf den Schalter WEITERE OPTIONEN, und geben Sie im Feld DATEIERWEITERUNG an, welche Extension die Dateien des Editors erhalten sollen (wird für Aufgabe 2 benötigt). Geben Sie beispielsweise »mic« für mice an.

Zum Abspeichern der Koordinaten der Mauspositionen legen wir eine eigene Klasse namens `CKoord` an. Wir hätten auch einfach die vordefinierte Klasse `CPoint` nehmen können, doch unterstützt diese keine Serialisierung. Wir werden aber in einer der Aufgaben zu diesem Kapitel das Programm dahingehend erweitern, daß die Mauszeichnungen in Dateien abgespeichert werden können, und dazu müssen die Koordinaten der Startpositionen serialisierbar sein.

Damit die Klasse serialisierbar ist, muß sie von `CObject` abgeleitet sein und die Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` verwenden. Die Methode `Serialize()` werden wir erst in der entsprechenden Aufgabe definieren.

2. Richten Sie mit Hilfe des Befehls `PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU` eine Header-Datei namens *Koord.h* ein, und deklarieren Sie darin die Klasse `CKoord`.

```
// Koord.h
class CKoord : public CObject
{
    DECLARE_SERIAL(CKoord)
public:
    CKoord();
    CKoord(int x_koord, int y_koord);
    int x;
    int y;
};
```

3. Richten Sie mit Hilfe des Befehls `PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU` eine Quelltextdatei namens *Koord.cpp* ein, und definieren Sie darin die Konstruktoren der Klasse `CKoord`.

```
// Koord.cpp
#include "stdafx.h"
#include "Koord.h"
IMPLEMENT_SERIAL(CKoord, CObject, 1)
CKoord::CKoord()
{
    x = 1;
    y = 1;
}
CKoord::CKoord(int x_koord, int y_koord)
{
    x = x_koord;
```

```
y = y_koord;
}
```

Abgespeichert werden die Startpositionen ... natürlich in unserer Dokumentklasse. Wir deklarieren dazu ein Objekt der Container-Klasse `CObArray`.

Container-Klassen

Die MFC definiert eine Reihe von Container-Klassen, in denen man Daten und Objekte speichern und verwalten kann. Sie stehen damit in Konkurrenz zu den Container-Klassen der STL (Teil der neuen ANSI C++-Bibliothek). Die Container-Klassen der MFC haben aber den Vorteil, daß sie den Prozeß der Serialisierung unterstützen.

Die `CObArray`-Klasse enthält genau die Funktionalität, die wir für die Verwaltung der Mauspositionen benötigen:

- ✗ Mit Hilfe der Methode `Add()` können neue Objekte der Klasse `CKoord` eingefügt werden.
- ✗ Mit Hilfe der Methode `GetSize()` kann bei Bedarf abgefragt werden, wie viele Objekte in dem Container enthalten sind.
- ✗ Mit Hilfe des Operators `[]` kann man gezielt auf einzelne Mauspositionen zugreifen.

Übung 12-4: Fortsetzung

4. Deklarieren Sie in der Dokumentklasse ein `CObArray`-Element.

```
class CMaeuseDoc : public CDocument
{
...
// Attribute
public:
    CObArray m_Maeuse;    // Maeusekoordinaten
```

5. Initialisieren Sie die `CObArray`-Elementvariable im Konstruktor der Dokumentklasse. Wir erzeugen dazu fünf `CKoord`-Objekte und fügen Zeiger auf diese Objekte in den Container ein.

```
CMaeuseDoc::CMaeuseDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion
    // einfügen
    m_Maeuse.Add(new CKoord(20, 20));
```

```

        m_Maeuse.Add(new CKoord(70, 150));
        m_Maeuse.Add(new CKoord(300, 100));
        m_Maeuse.Add(new CKoord(200, 200));
        m_Maeuse.Add(new CKoord(400, 30));
    }

```

6. Da wir auf die Klasse CKoord zugreifen, müssen wir noch die Header-Datei der Klasse CKoord in die Liste der Include-Dateien von *MaeuseDoc.cpp* aufnehmen:

```

// MaeuseDoc.cpp : Implementierung der Klasse CMaeuseDoc
#include "stdafx.h"
#include "Maeuse.h"
#include "Koord.h"

```

Soviel zu den Vorarbeiten. Jetzt nähern wir uns dem eigentlichen Algorithmus, den wir vollständig in der Methode OnDraw() der Ansichtsklasse erzeugen.

7. Da wir in dem Algorithmus mit CKoord-Objekten arbeiten, müssen wir auch in die Quelltextdatei der Ansichtsklasse die Header-Datei zu CKoord einbinden:

```

// MaeuseView.cpp : Implementierung der Klasse CMaeuseView
#include "stdafx.h"
#include "Maeuse.h"
#include "Koord.h"

```

Wenn die Mäuse aufeinander zulaufen, tun sie dies mit einer definierten Geschwindigkeit. Wir könnten diesen Wert im Algorithmus festlegen (als Konstante oder lokale Variable), aber es ist besser, wir richten dafür eine Elementvariable in der Ansichtsklasse ein. (In späteren Versionen des Programms könnte man diesen Wert beispielsweise über ein Dialogfeld vom Anwender festlegen lassen.)

8. Deklarieren Sie in der Ansichtsklasse eine Elementvariable für die Geschwindigkeit der Mäuse.

```

class CMaeuseView : public CView
{
    ...
    // Attribute
public:
    CMaeuseDoc* GetDocument();
    double m_Geschwindigkeit;

```

9. Initialisieren Sie die Geschwindigkeit im Konstruktor der Ansichtsklasse.

```
CMaeuseView::CMaeuseView()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen
    m_Geschwindigkeit = 0.1;
}
```

10. Überschreiben Sie die Methode OnDraw() der Ansichtsklasse.

```
1: void CMaeuseView::OnDraw(CDC* pDC)
2: {
3:     CMaeuseDoc* pDoc = GetDocument();
4:     ASSERT_VALID(pDoc);
5:     CObArray maeuse;
6:     CKoord *pAktMaus, *pNextMaus;
7:     int loop2;
8:
9:     // Punkte kopieren
10:    for(int loop = 0; loop < pDoc->m_Maeuse.GetSize();
        loop++)
11:    {
12:        CKoord *pKoord = new CKoord;
13:        pKoord->x = ((CKoord*) pDoc->m_Maeuse[loop])->x;
14:        pKoord->y = ((CKoord*) pDoc->m_Maeuse[loop])->y;
15:        maeuse.Add(pKoord);
16:        pDC->Ellipse(pKoord->x - 10, pKoord->y - 10,
            pKoord->x + 10, pKoord->y + 10);
17:    }
18:
19:    // Iteriere 30 mal
20:    for(int loop1 = 1; loop1 <= 30; loop1++)
21:    {
22:        // Zeichne Linien
23:        for(loop2 = 0; loop2 < maeuse.GetSize();
            loop2++)
24:        {
25:            pAktMaus = (CKoord*) maeuse[loop2];
26:            pDC->MoveTo(pAktMaus->x, pAktMaus->y);
27:
28:            pNextMaus = (CKoord*) maeuse[ (loop2 + 1) %
                (maeuse.GetSize()) ];
29:            pDC->LineTo(pNextMaus->x, pNextMaus->y);
30:        }
```

```

31:
32:         // Berechne Punkte für nächste Iteration
33:         for(loop2 = 0; loop2 < maeuse.GetSize() - 1;
           loop2++)
34:         {
35:             pAktMaus = (CKoord*) maeuse[loop2];
36:             pNextMaus = (CKoord*) maeuse[loop2 + 1];
37:             pAktMaus->x += (pNextMaus->x - pAktMaus->x)
                           * m_Geschwindigkeit;
38:             pAktMaus->y += (pNextMaus->y - pAktMaus->y)
                           * m_Geschwindigkeit;
39:         }
40:         pAktMaus = (CKoord*) maeuse[maeuse.GetSize()-1];
41:         pNextMaus = (CKoord*) maeuse[0];
42:         pAktMaus->x += (pNextMaus->x - pAktMaus->x)
                       * m_Geschwindigkeit;
43:         pAktMaus->y += (pNextMaus->y - pAktMaus->y)
                       * m_Geschwindigkeit;
44:
45:     }
46: }

```

In den Zeilen 10 bis 17 werden die Startpositionen der Mäuse aus `pDoc->m_Maeuse` in das lokale `CObArray`-Objekt `maeuse` kopiert. Dies muß sein, denn wir werden im Laufe des Algorithmus die Mäusepositionen von Iteration zu Iteration verändern. Um nicht die Originalstartpositionen zu verlieren (die ja beim nächsten Neuzeichnen des Fensters wieder benötigt werden), legen wir die lokale Kopie an.

Dann lassen wir die Mäuse 30 Mal loslaufen. In jeder Iteration

- ✕ verbinden wir zuerst alle Mäuse mit Linien und
- ✕ berechnen dann die neuen Mauspositionen.

Die neuen Mauspositionen ermitteln wir einfach nach den Regeln der Vektoraddition. Wir berechnen getrennt für die x- und y-Koordinate die Differenz zwischen der Maus und ihrem Ziel (sprich den Abstand zwischen der aktuellen Maus und der Maus, auf die sie zuläuft) und multiplizieren diesen Betrag mit der Geschwindigkeit (die einen Wert kleiner 1 haben muß).

11. Führen Sie das Programm aus (**Strg** + **F5**).

12.4 Die Zeichenwerkzeuge

Nachdem Sie einen Gerätekontext erzeugt haben, benötigen Sie ein Zeichenwerkzeug: einen Zeichenstift, einen Pinsel oder vielleicht auch die Schriftart für eine Textausgabe.

Diese Zeichenwerkzeuge werden GDI-Objekte genannt (GDI steht für *Graphics Device Interface*). Jedem GDI-Objekt entspricht eine eigene Klasse, beispielsweise `CFont` oder `CBrush` für eine Schriftart oder ein Pinselobjekt. Die GDI-Objekte werden bei ihrer Instanzbildung konfiguriert und danach in den Gerätekontext geladen, wobei von jeder Art GDI-Objekt (`CPen`, `CBrush`, etc.) genau eine Instanz in einem Gerätekontext vorhanden ist. Wenn Sie in den Gerätekontext zeichnen, wird dann, je nachdem welche Zeichenoperation (CDC-Methoden) Sie aufrufen, automatisch das entsprechende Zeichenwerkzeug benutzt.

12.4.1 Überblick

Tabelle 12.3:
GDI-Objekte

Klasse	Ausgabeeinheit
<code>CBitmap</code>	Auf dem Umweg über Instanzen dieser Klasse können Bitmaps in Gerätekontexte geladen, bearbeitet und ausgegeben werden.
<code>CBrush</code>	Für Pinsel-Objekte. Das aktuelle Pinsel-Objekt des Gerätekontextes wird beispielsweise zum Füllen von geometrischen Figuren (CDC-Methoden <code>Ellipse()</code> , <code>Rectangle()</code> etc.) benutzt.
<code>CFont</code>	Für Schriftarten. Das aktuelle Font-Objekt des Gerätekontextes wird für die Textausgaben (CDC-Methode <code>TextOut()</code>) benutzt.
<code>CPalette</code>	Für Paletten (von 256 Farben).
<code>CPen</code>	Für Stift-Objekte. Das aktuelle Stift-Objekt des Gerätekontextes wird zum Zeichnen von Linien (<code>LineTo()</code>) sowie für die Umrisse geometrischer Figuren (CDC-Methoden <code>Ellipse()</code> , <code>Rectangle()</code> etc.) benutzt.
<code>CRgn</code>	Für Zeichenbereiche. Diese Regionen können zum Beispiel zur Definition von Clipping-Bereichen verwendet werden.

12.4.2 Vordefinierte GDI-Objekte

Jeder Gerätekontext ist standardmäßig mit einem Satz vordefinierter GDI-Objekte ausgestattet (den sogenannten »Stock-Objects«). Wenn Sie beispielsweise – ohne zuvor einen Stift oder einen Pinsel in den Gerätekontext geladen zu haben – eine Linie oder ein Rechteck zeichnen, wird für die

Linie und den Rahmen des Rechtecks das Standard-Stiftobjekt verwendet, welches dünne schwarze Linien zieht. Ausgemalt wird das Rechteck mit dem Standardpinsel, der weiß und ohne Muster ist.

Wenn Ihnen diese Vorgaben nicht zusagen oder Sie eher mit Farben arbeiten wollen, müssen Sie

- ✗ CDC-Methoden verwenden, die Farben oder Pinsel-Objekte als Argumente übernehmen, oder
- ✗ die GDI-Objekte des Gerätekontextes ersetzen (siehe nachfolgenden Abschnitt).

12.4.3 GDI-Objekte einrichten

Um ein GDI-Objekt in einen Gerätekontext zu laden, verwendet man die CDC-Methode `SelectObject()`.

```
CBrush roterPinsel(RGB(255, 0, 0));
pDC->SelectObject(&roterPinsel);
```

Wie Sie sehen, ist das Laden an sich kein großes Problem. Probleme bereitet dagegen die Speicherbereinigung.

Wenn Sie ein GDI-Objekt in einen Gerätekontext laden (beispielsweise ein `CPen`-Objekt), wird das alte Stiftobjekt, das zuvor im Gerätekontext war, aus dem Gerätekontext entfernt. Handelt es sich dabei um ein von Windows vordefiniertes Objekt, brauchen Sie sich um das Löschen dieses Objekts nicht zu kümmern – dies übernimmt Windows für Sie. Handelt es sich dagegen um ein von Ihnen definiertes GDI-Objekt, sollten Sie es unbedingt löschen (statt darauf zu warten, daß es mit Beendigung des Programms aus dem Speicher entfernt wird).

Wo wir gerade beim Löschen sind: Was passiert mit meinen selbstdefinierten GDI-Objekten, die im Gerätekontext enthalten sind? Ganz einfach. Wenn Sie diese löschen wollen, müssen die GDI-Objekte zuvor aus dem Gerätekontext entfernt werden. Dies geschieht

- ✗ entweder automatisch, wenn der Gerätekontext selbst aufgelöst wird,
- ✗ oder indem man ein neues GDI-Objekt lädt, das das alte GDI-Objekt überschreibt und damit löscht.¹

¹ Beachten Sie, daß ein `CPen`-Objekt natürlich nur das Stiftobjekt des Gerätekontextes ersetzt, ein `CBrush`-Objekt nur das Pinselobjekt des Gerätekontextes und so weiter.

Wenn Sie ganz sichergehen wollen, daß Sie bei der Speicherbereinigung keine Fehler machen, gehen Sie einfach wie folgt vor:

1. **Erzeugen Sie das GDI-Objekt.** Rufen Sie dazu den Konstruktor auf, dem Sie alle wichtigen Parameter zur Konfiguration des Objekts übergeben können (beispielsweise die Farbe und Strichstärke des Stifts).

Als Alternative dazu können Sie das Objekt auch mit Hilfe der `Create()`-Methode des GDI-Objekts initialisieren. Für die GDI-Objekte `CFont` und `CRgn` ist der Aufruf der `Create()`-Methode unerlässlich.

2. **Laden Sie das GDI-Objekt mit der Methode `SelectObject()` in den Gerätekontext.** Speichern Sie dabei das alte GDI-Objekt, das zuvor verwendet wurde und von der Methode `SelectObject()` zurückgeliefert wird.

`CPalette`-Objekte werden nicht mit `SelectObject()` in den Gerätekontext geladen.

3. **Entfernen Sie das GDI-Objekt aus dem Gerätekontext.** Selbstdefinierte GDI-Objekte können nicht gelöscht werden, solange sie in einen Gerätekontext geladen sind. Sofern Sie die selbstdefinierten GDI-Objekte vor dem Gerätekontext löschen wollen, entfernen Sie die Objekte aus dem Gerätekontext, indem Sie die originalen GDI-Objekte in den Gerätekontext laden.



```
CBrush neuerPinzel(RGB(255, 0, 0));
CBrush *p_alterPinzel = pDC->SelectObject(&roterPinzel);
// Zeichenoperationen ausführen
pDC->SelectObject(p_alterPinzel);
```

Übung 12-5: Flecken an Mausclickposition zeichnen

Das folgende Programm funktioniert ganz ähnlich wie das `Kreise`-Programm, zeichnet aber farbige Flecken statt Kreise.

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten ein neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Spots« genannt.
2. Deklarieren Sie in der Dokumentklasse einen `int`-Zähler und ein `CPoint`-Array für 100 Kreismittelpunkte.

```
class CSpotsDoc : public CDocument
{
...
}
```

```
// Attribute
public:
    int m_nZaehler;
    CPoint m_Kreise[100];
```

3. Initialisieren Sie den Zähler im Konstruktor der Dokumentklasse mit dem Wert 0.

```
CSpotsDoc::CSpotsDoc()
{
    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion
    // einfügen
    m_nZaehler = 0;
}
```

4. Richten Sie mit Hilfe des Klassen-Assistenten in der Klasse des Ansichtsfensters eine Nachrichtenbehandlungsmethode für WM_LBUTTONDOWN ein.
5. Zeichnen Sie mit Hilfe der CDC-Methode Ellipse() eine Scheibe mit zufälligem Radius an der Position des Mausklicks, und speichern Sie die Kreismittelpunkte.

```
void CSpotsView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    CSpotsDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CClientDC dc(this);
    CBrush roterPinzel(RGB(255, 0, 0));
    CBrush *p_alterPinzel = dc.SelectObject(&roterPinzel);
    int b = (int) (100.0 * rand() / RAND_MAX);
    dc.Ellipse(point.x-b, point.y-b, point.x+b, point.y+b);
    if (pDoc->m_nZaehler < 100)
        pDoc->m_nZaehler++;
    pDoc->m_Kreise[pDoc->m_nZaehler - 1 ] = point;
    dc.SelectObject(p_alterPinzel);

    CView::OnLButtonDown(nFlags, point);
}
```

6. Rekonstruieren Sie die Kreise in der OnDraw()-Methode.

```
void CSpotsView::OnDraw(CDC* pDC)
{
    CSpotsDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
```

```
// ZU ERLEDIGEN: Hier Code zum Zeichnen der
// ursprünglichen Daten hinzufügen
CBrush roterPinzel(RGB(255, 0, 0));
CBrush *p_alterPinzel = pDC->SelectObject(&roterPinzel);
for(int i = 0; i < pDoc->m_nZaehler; i++)
{
    int b = (int) (100.0 * rand() / RAND_MAX);
    pDC->Ellipse(pDoc->m_Kreise[i].x - b,
                pDoc->m_Kreise[i].y - b,
                pDoc->m_Kreise[i].x + b,
                pDoc->m_Kreise[i].y + b);
}
pDC->SelectObject(p_alterPinzel);
}
```

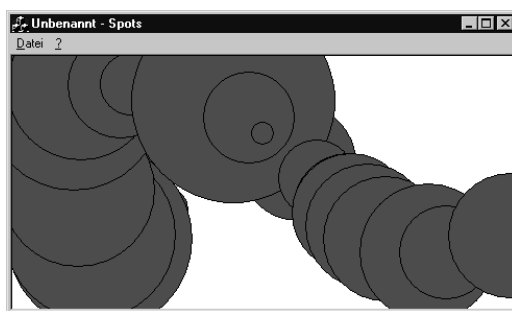
7. Löschen Sie alle Menübefehle bis auf DATEI/NEU und DATEI/BEENDEN.
8. Überschreiben Sie mit Hilfe des Klassen-Assistenten in der Dokument-klasse die Methode `DELETECONTENTS()`.

```
void CSpotsDoc::DeleteContents()
{
    // TODO: Speziellen Code hier einfügen und/oder
    // Basisklasse aufrufen
    m_nZaehler = 0;

    CDocument::DeleteContents();
}
```

9. Führen Sie das Programm aus (**Strg** + **F5**).

Bild 12.4:
Rote Flecken



12.5 Farbige Fraktale

Zum Abschluß werfen wir noch einen kurzen Blick auf die Definition von Farben nach dem RGB-Modell und testen die erworbenen Fähigkeiten an der Erzeugung einer Julia-Menge.

Das RGB-Modell

Die meisten Methoden, denen man Farben übergeben kann, erwarten einen `COLORREF`-Wert – so zum Beispiel der Konstruktor für das Pinselobjekt `CBrush`:

```
CBrush( COLORREF crColor );
```

Hinter `COLORREF` verbirgt sich ein 32-Bit-Wert, der eine Farbe nach dem RGB-Modell spezifiziert:

```
0x00bbggrr
```

Das RGB-Modell beruht auf dem Effekt, daß man durch Variation der Farbintensitäten aus den drei Lichtfarben Rot, Grün und Blau sämtliche Farben mischen kann. Werden beispielsweise rotes, grünes und blaues Licht in voller Intensität ausgestrahlt und gemischt, erhält man Weiß. Ist die Intensität aller drei Farben gleich Null (d.h., es wird kein Licht ausgestrahlt), erhält man in der Summe Schwarz. Sie kennen dies sicher bereits von Bühnenbeleuchtungen, Prismen oder auch von Ihrem Monitor.

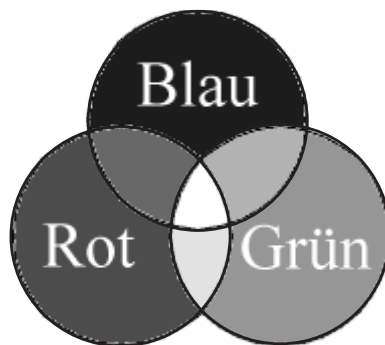


Bild 12.5:
Farbmischung
für Lichtfarben

In einem `COLORREF`-Wert codiert das unterste Byte den Rotanteil, das zweite Byte den Grünanteil und das dritte Byte den Blauanteil. Da ein Byte Werte zwischen 0 (hexadezimal 00) und 255 (hexadezimal FF) annehmen kann, können die einzelnen Farbanteile in 256 Schritten abgestuft werden. Die endgültige Farbe ergibt sich durch die Zusammenmischung der Farbanteile.

Tabelle 12.4:
RGB-Farben

COLORREF-Wert	Farbe
0x00000000	Schwarz
0x00FFFFFF	Weiß
0x000000FF	Rot
0x00FF60FF	Rosa

Statt die Farbwerte direkt anzugeben, benutzt man meist die `RGB()`-Funktion, der man die Farbanteile als Integer-Werte zwischen 0 und 255 übergeben kann.

```
CBrush roterPinzel(RGB(255, 0, 0));
```

Übung 12-6: Julia-Menge zeichnen

Zur Berechnung der Julia-Menge benötigt man komplexe Zahlen. Der Grund hierfür sind die speziellen Rechenregeln, die für komplexe Zahlen gelten. Man könnte dies zwar auch mit einfachen reellen Zahlen simulieren, die Rechenausdrücke wären dann aber weniger überschaubar.

Wenn Sie nicht mit den Rechenregeln für komplexe Zahlen vertraut sind, brauchen Sie deswegen nicht enttäuscht zu sein, denn die Operationen, die auf den Zahlen ausgeführt werden, sind prinzipiell die gleichen, wie für reelle Zahlen (Addition, Multiplikation, Betrag), lediglich die Ausführung dieser Rechenoperationen hat sich geändert. Glücklicherweise stellt die neue C++-Laufzeitbibliothek eine eigene Klasse zur Kapselung der komplexen Zahlen zur Verfügung, so daß Sie sich um die interne Implementierung der Rechenoperationen keine Gedanken zu machen brauchen. Was für Sie allerdings noch interessant sein dürfte, ist, daß komplexe Zahlen aus zwei Werten (Realteil und Imaginärteil) zusammengesetzt sind, was sie für die Abbildung auf die Punkte einer Ebene bestens geeignet macht (Realteil wird zu x-Koordinate, Imaginärteil zu y-Koordinate).

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe mein Projekt für diese Übung »Fraktale« genannt.
2. Laden Sie das Menü des Programms in den Menü-Editor. Löschen Sie alle Befehle bis auf den Befehl DATEI/BEENDEN. Richten Sie einen neuen Menübefehl DATEI/FRAKTAL ein.

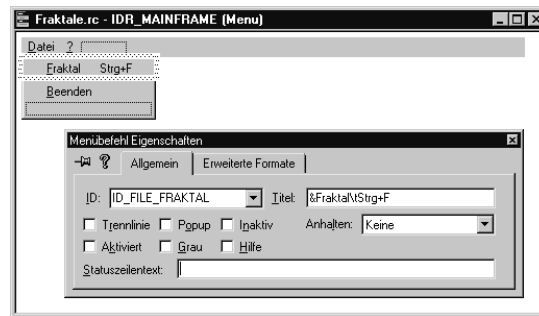


Bild 12.6:
Menü des
Programms

Passen Sie auch die Tastaturkürzel an das Programm an (ACCELERATOR-Ressource).

3. Richten Sie mit Hilfe des Klassen-Assistenten in der Ansichtsklasse eine Behandlungsmethode für den Befehl DATEI/FRAKTAL ein. In dieser Methode wird das Fraktal berechnet und ausgegeben.
4. Da wir in der Behandlungsmethode die C++-Klasse `complex` verwenden werden, müssen wir zuerst die zugehörige Header-Datei aufnehmen und den Namensbereich `std` einschalten.

```
// FraktaleView.cpp : Implementierung der Klasse
// CFraktaleView
...
#include "FraktaleDoc.h"
#include "FraktaleView.h"
#include <complex>
using namespace std;
```

5. Setzen Sie jetzt den Code für die Behandlungsmethode ein.

```
1: void CFraktaleView::OnFileFraktal()
2: {
3:     // TODO: Code für Befehlsbehandlungsroutine hier
      // einfügen
4:     CClientDC dc(this);
5:
6:     RECT rect;
7:     GetClientRect(&rect);
8:
9:     complex<double> c(-0.012, 0.74);
10:    for(int i = rect.left; i<rect.right; i++)
```

```

11:      for(int j = rect.top; j<rect.bottom; j++)
12:      {
13:          complex<double> x(0.0001 * i, 0.0001 * j);
14:          for(int n = 0; n <100; n++)
15:          {
16:              if(abs(x) > 100)
17:                  break;
18:              x = pow(x, 2) + c;
19:          }
20:          if(abs(x) < 1)
21:              dc.SetPixel(i, j, RGB(0, 0, 255));
22:          else
23:              dc.SetPixel(i, j, RGB(2 * abs(x), 255, 255));
24:      }
25:  }

```

Der Algorithmus funktioniert nun so, daß als Konstante die komplexe Zahl

$$c = -0.012 + 0.74i$$

vorgegeben wird. Diese Zahl wird in Zeile 9 instantiiert, indem die Werte für den Real- und den Imaginärteil an den Konstruktor der Klasse `complex` übergeben werden.

Als nächstes werden zwei verschachtelte Schleifen angelegt, so daß die Schleifenvariablen `i` und `j` alle Pixel im Client-Bereich des Fensters durchlaufen. (Die Abmessung des Client-Bereichs wurden zuvor in Zeile 7 ermittelt.) Für jedes Pixel wird die komplexe Zahl `x` instantiiert. Dabei werden `i` und `j` bei der Festlegung des Real- und Imaginärteils berücksichtigt. Anschließend beginnt die eigentliche Iteration:

$$x = x^2 + c$$

Durch die Quadrierung gehen Zahlen, deren Betrag kleiner Eins ist, gegen Null, während Zahlen, deren Betrag größer als Eins ist, gegen Unendlich gehen (Zahlen, deren Betrag gleich Eins ist, bleiben vom Betrag her unverändert).

Nach einer bestimmten Anzahl von Iterationen kann man dann darangehen, zu untersuchen, in welche Richtung sich die Zahl `x` für eine Pixelposition bewegt hat, und das Pixel entsprechend einfärben (Zeilen 20 bis 23).

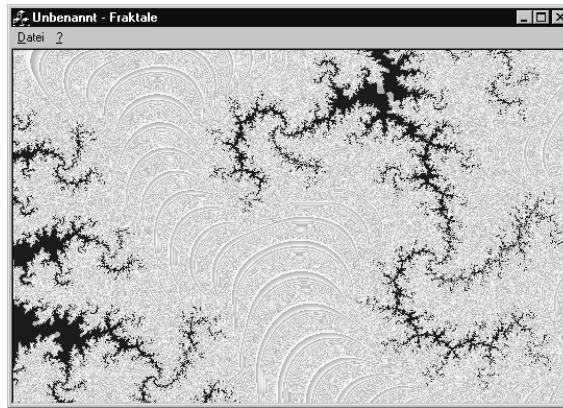


Bild 12.7:
Eine Julia-
Menge

Die Berechnung des Fraktals dauert recht lange und kann nicht abgebrochen werden. Bevor Sie den Befehl DATEI/FRAKTAL das erste Mal aufrufen, sollten Sie daher das Fenster verkleinern.



12.6 Zusammenfassung

Die GDI-Funktionalität (Graphics Device Interface) von Windows wird in der MFC in den Klassen CDC (repräsentiert Gerätekontexte) und CGdiObject (repräsentiert GDI-Objekte) sowie den abgeleiteten Klassen gekapselt.

Die wichtigsten von CDC abgeleiteten Klassen sind

- ✗ CClientDC (repräsentiert den Client-Bereich eines Fensters),
- ✗ CWindowDC (repräsentiert ein Fenster) und
- ✗ CPaintDC (repräsentiert einen Gerätekontext während der Bearbeitung einer WM_PAINT-Nachricht).

Die CDC-Klasse umfaßt die grundlegende GDI-Zeichenfunktionalität. Dazu zählen einfache Methoden, die Linien, Figuren, Text und Bitmaps zeichnen, aber auch fortgeschrittene Methoden für Clipping, Bildlauf, Bereiche und Abbildungsmodi.

Die meisten Zeichenoperationen nutzen Zeichenwerkzeuge (GDI-Objekte), die mit SelectObject() oder SelectStockObject() in den Gerätekontext geladen werden. 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. Um Speicherlecks zu vermeiden, sollte man darauf achten, GDI-Objekte, die man selbst in einen Gerätekontext geladen hat, nach der Zeichenausgabe aus dem Gerätekontext zu entfernen und einer korrekten Speicherbereinigung zuzuführen.

12.7 Fragen

1. Welche Elemente benötigt man, um in ein Fenster zu zeichnen?
2. Muß man eigene Zeichenwerkzeuge (GDI-Objekte) definieren und in den Gerätekontext laden?
3. Wie kann man sich einen Gerätekontext für ein Fenster selbst erzeugen?
4. Wie löscht man GDI-Objekte aus einem Gerätekontext?
5. Wie zeichnet man Linien?
6. Was ist ein umschließendes Rechteck?
7. Wie werden Farben an MFC-Methoden übergeben?

12.8 Aufgaben

1. Überlegen Sie sich, wie man den Mäuse-Editor ausbauen könnte, so daß der Anwender die Startpositionen in beliebiger Anzahl per Mausklick festlegen kann.

Wenn Sie sich selbst darin versuchen wollen, legen Sie sich eine Kopie des Projekts *Maeuse* an. Erstellen Sie dazu im Windows Explorer ein neues Verzeichnis (beispielsweise *Maeuse2*), öffnen Sie das Verzeichnis *Maeuse*, in dem die Dateien des alten Projekts stehen, und kopieren Sie diese Dateien samt Unterverzeichnissen in das neue Verzeichnis *Maeuse2*. Über den Befehl DATEI/ARBEITSBEREICH ÖFFNEN können Sie das Projekt in die IDE laden.

2. Überlegen Sie sich, wie man mit Hilfe der Serialisierung den Mäuse-Editor so erweitern könnte, daß man die Startpositionen der Mäuse in Dateien speichern kann. Wenn Sie ein wenig über das Problem nachgedacht haben, schauen Sie sich direkt die Lösung an.
3. Überlegen Sie sich, wie man Linien einzeichnet. Überlegen Sie sich, wie man ein Programm aufbauen müßte, welches dem Anwender erlaubt, Linien zu zeichnen.

4. Schauen Sie sich in der Online-Hilfe die Methoden der Klasse `CDC` an. Halten Sie Ausschau nach einer Methode, mit deren Hilfe Sie im Programm zum Einarmigen Bandit aus Kapitel 9 den Hintergrund einfärben könnten.

12.9 Lösungen zu den Aufgaben

Zu 1: Mäuse-Editor mit anwenderdefinierten Startpositionen

Mein Vorschlag ist, daß Sie im Menü `BEARBEITEN` zwei Menübefehle, `STARTPOSITIONEN FESTLEGEN` und `ZEICHNEN`, einrichten.

In der Ereignisbehandlungsmethode zum Befehl `STARTPOSITIONEN FESTLEGEN`, setzen Sie eine boolesche Steuervariable, die Sie in der Dokumentklasse deklariert haben und die im Konstruktor mit dem Wert `false` initialisiert wird, auf `true`. Nur wenn diese Steuervariable `true` ist, werden wir dem Anwender erlauben, durch Mausklicks neue Startpositionen festzulegen. In der Behandlungsmethode werden als nächstes die alten Startpositionen gelöscht. Zum Schluß wird die Methode `UpdateAllViews()` aufgerufen, die das Ansichtsfenster darüber informiert, daß sich die Daten im Dokument geändert haben und das Ansichtsfenster daher seinen Inhalt aktualisieren soll.

```
void CMaeuseDoc::OnEditPositionen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    // Mausklicks beachten
    m_bPunkteSetzen = true;
    // alte Positionen löschen
    for(int loop = m_Maeuse.GetSize() - 1; loop >= 0; loop--)
    {
        CKoord *pKoord = new CKoord;
        pKoord = (CKoord*) m_Maeuse[loop];
        m_Maeuse.RemoveAt( loop );
        delete pKoord;
    }
    UpdateAllViews(NULL);
}
```

In der Ereignisbehandlungsmethode zum Befehl `ZEICHNEN` wird das Ansichtsfenster zum Neuzeichnen aufgefordert, und die Steuervariable wird auf `false` gesetzt – in der Annahme, daß der Anwender zuvor per Mausklick neue Startpositionen festgelegt hat.

```
void CMaeuseDoc::OnEditZeichnen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    UpdateAllViews(NULL);
    m_bPunkteSetzen = false;
}
```

Schließlich wird die WM_LBUTTONDOWN-Nachricht abgefangen. In der Methode wird geprüft, ob zuvor der Befehl STARTPOSITIONEN FESTLEGEN aufgerufen wurde. Wenn ja, wird die Koordinate des Mausklicks als Startposition eingetragen.

```
void CMaeuseView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CMaeuseDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: Code für die Behandlungsroutine für Nachrichten
    // hier einfügen und/oder Standard aufrufen
    if (pDoc->m_bPunkteSetzen)
    {
        CKoord *pKoord = new CKoord(point.x, point.y);
        pDoc->m_Maeuse.Add(pKoord);
    }

    CView::OnLButtonDown(nFlags, point);
}
```

Zu 2: Mäuse-Editor mit Dateiunterstützung

Legen Sie sich eine Kopie des Projekts *Maeuse2* an (zum Kopieren von Projekten siehe Aufgabe 1).

Deklarieren Sie in der Klasse CKoord die Methode `Serialize()`. Diese Methode wird später bei der Serialisierung automatisch aufgerufen werden und muß korrekt implementiert sein.

```
class CKoord : public CObject
{
    DECLARE_SERIAL(CKoord)
public:
    CKoord();
    CKoord(int x_koord, int y_koord);
    int x;
    int y;
```

```
virtual void Serialize(CArchive& ar);
};
```

Implementieren Sie die Methode `Serialize()`. Kopieren Sie sich das Grundgerüst aus Ihrer Dokumentklasse, und erweitern Sie es um die Befehle zum Ausgeben und Einlesen der Koordinatenwerte.

```
void CKoord::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
        ar << x << y;
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
        ar >> x >> y;
    }
}
```

Das Anwendungsgerüst ist so eingerichtet, daß beim Speichern oder Laden die `Serialize()`-Methode der Dokumentklasse aufgerufen wird. In dieser Methode müssen Sie nur dafür sorgen, daß die `Serialize()`-Methode des `CObArray`-Objekts `m_Maeuse` aufgerufen wird. Diese Methode ist bereits in `CObArray` definiert und so implementiert, daß sie die `Serialize()`-Methode der Elemente im Array aufruft (in unserem Falle also die `Serialize()`-Methode von `CKoord`).

Wenn Sie zum Laden und Speichern den gleichen Code verwenden können, dürfen Sie die `if-else`-Anweisung ignorieren.



```
void CMaeuseDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
    }
    else
    {
        // ZU ERLEDIGEN: Hier Code zum Laden einfügen
    }
}
```

```
m_Maeuse.Serialize(ar);
}
```

Sorgen Sie dafür, daß der Fensterinhalt aktualisiert wird, wenn der Anwender über den Befehl DATEI/ÖFFNEN eine neue Mäusedatei lädt. Überschreiben Sie dazu mit Hilfe des Klassen-Assistenten die CDocument-Methode OnOpenDocument(), und rufen Sie in der Methode UpdateAllViews(0) auf.

```
BOOL CMaeuseDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if (!CDocument::OnOpenDocument(lpszPathName))
        return FALSE;

    // TODO: Speziellen Erstellungscode hier einfügen
    UpdateAllViews(0);

    return TRUE;
}
```

Wenn der Anwender den Befehl DATEI/NEU aufruft, soll der Fensterinhalt gelöscht werden. Für das Aktualisieren des Ansichtsfensters sorgt bereits das Anwendungsgerüst. Wir müssen nur mit Hilfe des Klassen-Assistenten die Methode DeleteContents() überschreiben und darin alle Mauspositionen löschen.

Wenn Sie möchten, können Sie auch die überflüssig gewordene Vorgabe der Mauspositionen im Konstruktor der Dokumentklasse löschen.

```
void CMaeuseDoc::DeleteContents()
{
    // TODO: Speziellen Code hier einfügen und/oder
    // Basisklasse aufrufen
    // alte Positionen löschen
    for(int loop = m_Maeuse.GetSize() - 1; loop >= 0; loop--)
    {
        CKoord *pKoord = new CKoord;
        pKoord = (CKoord*) m_Maeuse[loop];
        m_Maeuse.RemoveAt( loop );
        delete pKoord;
    }
    CDocument::DeleteContents();
}
```

Zu 3: Linien zeichnen lassen

Um Linien auszugeben, setzt man mit Hilfe der Methode `MoveTo()` die aktuelle Zeichenposition auf den Anfangspunkt der Linie. Dann zeichnet man mit Hilfe der Methode `LineTo()` eine Linie von dort bis zum Endpunkt der Linie.

Wenn ein Anwender eine Linie zeichnet, klickt er üblicherweise mit der Maus auf den Anfangspunkt und bewegt dann die Maus mit gedrückter Maustaste zum Endpunkt der Linie. Hier läßt er die Maus los.

Um die Linie zu zeichnen, rufen wir beim Herabdrücken der linken Maustaste (`WM_LBUTTONDOWN`) die Methode `MoveTo()` auf und setzen eine boolesche Variable auf `true`, die anzeigt, daß der Anwender die Maus gedrückt hat und jetzt die Linie zieht. (Statt einer booleschen Variablen könnte man auch einen Zeiger auf einen Gerätekontext verwenden, der in der Ansichtsklasse definiert ist und den man beim Drücken der Maustaste mit einem Gerätekontext für das Ansichtsfenster verbindet und beim Loslassen der Maustaste auf `NULL` setzt.) Beim Loslassen der Maustaste (`WM_LBUTTONUP`) schalten wir die boolesche Variable um und zeichnen die Linie mit Hilfe der Methode `LineTo()`. Bei diesem Vorgehen kann man den Verlauf der zukünftigen Linie auch während des Ziehens der Maus mit gedrückter Maustaste anzeigen. Fangen Sie die Nachricht `WM_MOUSEMOVE` ab, und kontrollieren Sie, ob die Maustaste gedrückt ist (Wert der booleschen Variable). Zieht der Anwender gerade eine Linie auf, lassen Sie das Fenster neu zeichnen und zeichnen dann mit Hilfe von `LineTo()` eine Linie zu der aktuellen Position.

Zu 4: Fensterhintergrund einfärben

Um den Fensterhintergrund einzufärben, bestimmen Sie die Größe des Fensters (`GetClientRect()`) und rufen eine der Methoden `FillSolidRect()` oder `FillRect()` auf.

Bitmaps

So mächtig die Zeichenmethoden der Klasse `CDC` auch sein mögen, so mühsam ist es doch, mit Hilfe dieser Methoden Bilder zu zeichnen. Für professionelle Grafikanwendungen, Animationen oder fotorealistische Darstellungen werden Sie also einen anderen Weg beschreiben. Statt die Bilder durch Zeichenoperationen im Programm zu erstellen,

- ✗ zeichnen Sie die Bilder vorab in einem leistungsfähigen Grafikprogramm oder
- ✗ scannen die Bilder ein,

um sie dann als fertige Bitmap-Ressourcen in Ihr Programm zu laden. Wie dies geht und was man so alles mit Bitmaps machen kann, wird Gegenstand dieses Kapitels sein.

Sie lernen in diesem Kapitel:

- ✗ Wie man Bitmap-Ressourcen anlegt.
- ✗ Wie man Bitmaps aus Ressourcen lädt
- ✗ Wie man Bitmaps in Fenster zeichnet.
- ✗ Wie man Bitmaps als Fensterhintergründe verwendet
- ✗ Wie man Bitmaps pixelweise manipuliert
- ✗ Wie man aus Bitmaps Liniengrafiken erzeugt



13.1 Bitmap-Ressourcen anlegen

Bevor wir eine Bitmap-Ressource erstellen können, brauchen wir ein Bitmap in Form einer BMP-Datei. Besorgen Sie sich also zuerst einmal für die nächsten Übungen ein oder zwei Bitmaps.

Quellen für Bitmaps

Lassen Sie Windows Ihre Festplatte nach BMP-Dateien durchsuchen (Befehl EXTRAS/SUCHEN im Windows Explorer).

- ✗ Grafikprogramme werden oftmals mit einer Sammlung von nützlichen und teilweise auch hochwertigen Bildern und Fotos ausgeliefert. Selbst wenn diese nicht im BMP-Format auf der CD-ROM vorliegen, kann man sie meist in das Grafikprogramm laden und dann im BMP-Format abspeichern.
- ✗ Durchsuchen Sie das Internet nach Bitmaps. Beachten Sie aber, daß Sie diese ohne vorherige Genehmigung des Copyright-Inhabers nur privat nutzen dürfen.
- ✗ Oder verwenden Sie die Bitmaps von der Begleit-CD (die allerdings auch dem Copyright-Schutz unterliegen).

Bitmaps als Ressourcen

Wenn Sie ein Bitmap parat haben, können Sie dieses einem bestehenden Programm als Ressource einverleiben, indem Sie

1. die Bitmap-Datei in das Unterverzeichnis *RES* des Projekts kopieren,
2. den IDE-Befehl EINFÜGEN/RESSOURCE aufrufen, links den Eintrag BITMAP auswählen und dann den Schalter IMPORTIEREN drücken.

Ihre Ressourcenskriptdatei wird dann um einen Verweis auf die Bitmapdatei erweitert, und dem Bitmap wird eine Ressourcen-ID zugewiesen, über die sie geladen werden kann.

Übung 13-1: Bitmap als Ressource in ein Projekt aufnehmen

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe das Projekt für diese Übung einfach »Bitmap« genannt.



Bild 13.1:
Bitmap aus
Datei impor-
tieren

2. Rufen Sie den IDE-Befehl EINFÜGEN/RESSOURCE auf (oder wechseln Sie zur RESSOURCEN-Ansicht des Arbeitsbereichfensters, und rufen Sie das Kontextmenü auf), wählen Sie links den Eintrag BITMAP aus, und drücken Sie den Schalter IMPORTIEREN. Wählen Sie das gewünschte Bitmap aus.

In Ihrer Ressourcenskriptdatei finden Sie jetzt einen Verweis auf die Bitmapdatei:

```
//////////////////////////////////////////
//
// Bitmap
IDB_BITMAP1      BITMAP  DISCARDABLE      "res\\Butterfly.bmp"
```

Um die Ressourcenskriptdatei (.rc) Ihres Projekts als Textdatei zu öffnen, rufen Sie den Befehl DATEI/ÖFFNEN auf, und wählen Sie im Feld ÖFFNEN ALS die Option TEXT aus.



13.2 Bitmaps laden und anzeigen

Bitmaps gehören zu den GDI-Objekten, die man in Gerätekontexte laden kann. Um jedoch ein Bitmap in ein Fenster auszugeben, kann man das Bitmap nicht einfach in den Gerätekontext des Fensters laden. Man muß vielmehr einen Umweg über einen Speicherkontext gehen und dann mit Hilfe der Methode `BitBlt()` das Bitmap aus dem Speicherkontext in den Fensterkontext kopieren.

Insgesamt läuft das Laden und Anzeigen von Bitmaps stets nach dem folgenden Muster ab:

Bitmap-Ressource in ein CBitmap-Objekt laden

1. Erzeugen Sie eine CBitmap-Instanz.

```
CBitmap bitmap;
```

2. Laden Sie das Bitmap mit Hilfe der Methode LoadBitmap() und der Ressourcen-ID in das CBitmap-Objekt.

```
bitmap.LoadBitmap(IDB_BITMAP2);
```

Abmessung des Bitmaps bestimmen

Um die Abmessung zu bestimmen, müssen Sie aus dem CBitmap-Objekt eine BITMAP-Struktur gewinnen.

3. Deklarieren Sie eine BITMAP-Struktur

```
BITMAP bm;
```

4. Rufen Sie die CBitmap-Methode GetObject() auf, um die Bitmap-Informationen in die BITMAP-Struktur einzulesen.

```
bitmap.GetObject(sizeof(bm), &bm);
```

Bitmap in einen Speicherkontext laden

5. Bilden Sie eine CDC-Instanz für den Speicherkontext.

```
CDC speicherDC;
```

6. Initialisieren Sie den Speicherkontext als kompatiblen Kontext zum eigentlichen Ausgabekontext. Rufen Sie dazu die CDC-Methode CreateCompatibleDC() auf.

```
speicherDC.CreateCompatibleDC(pDC);
```

7. Laden Sie das Bitmap mit Hilfe der SelectObject()-Methode in den Speicherkontext.

```
speicherDC.SelectObject(&bitmap);
```

Bitmap in das Fenster zeichnen

8. Kopieren Sie das GDI-Bitmap in den Gerätekontext des Fensters. Zum Kopieren können Sie eine der Methoden BitBlt() oder StretchBlt() aufrufen.

```
pDC->BitBlt(
    10, 10, bm.bmWidth, bm.bmHeight,
    &speicherDC, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
```

Beiden Methoden übergeben Sie sowohl die Abmessung des Zielbereichs als auch die Abmessung des Quellbereichs – jeweils in Form der Koordinaten der linken oberen Ecke sowie Breite und Höhe (`BitBlt()` erwartet für den Quellbereich nur die Koordinaten der linken oberen Ecke). Der Zielkontext ist durch das Objekt gegeben, für das die Methode aufgerufen wird, der Quellkontext wird ebenfalls als Parameter übergeben (5. Parameter, vor den Angaben zum kopierenden Bereich). Als letzten Parameter übergibt man eine Konstante für den Zeichenmodus – üblicherweise `SRCCOPY`.

Übung 13-2: Bitmap laden und anzeigen

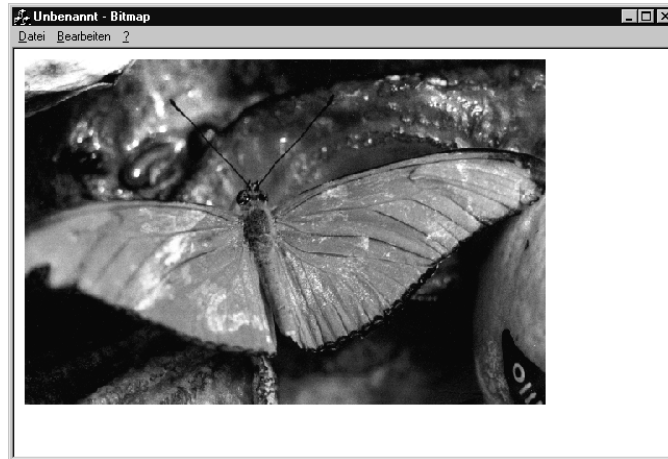
In dieser Übung wollen wir die Bitmap, die wir in Übung 13-1 als Ressource unserem Projekt einverleibt haben, im Ansichtsfenster anzeigen.

1. Springen Sie im Quelltexteditor zur `OnDraw()`-Methode der Ansichtsklasse. Am einfachsten klickt man dazu im Arbeitsbereichsfenster, Seite **KLASSEN**, auf den `CBitmapView`-Knoten für `ONDRAW()`.
2. Fügen Sie den Code zum Anzeigen des Bitmaps ein.

```
void CBitmapView::OnDraw(CDC* pDC)
{
    CBitmapDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    CBitmap bitmap;
    bitmap.LoadBitmap(IDB_BITMAP1);

    BITMAP bm;
    bitmap.GetObject(sizeof(bm), &bm);
    CDC speicherDC;
    speicherDC.CreateCompatibleDC(pDC);
    speicherDC.SelectObject(&bitmap);
    pDC->BitBlt(
        10, 10, bm.bmWidth, bm.bmHeight,
        &speicherDC, 0, 0, SRCCOPY);
}
```

Bild 13.2:
Anzeige eines
Bitmaps mit
BitBlt()



GDI-Bitmaps sind geräteabhängige Bitmaps (also DDBs und keine DIBs).

13.3 Bitmaps als Fensterhintergründe

BitBlt()

```

BOOL BitBlt(
    int x, int y,           // Zielausschnitt:
    int nWidth,            // linke obere Ecke
    int nHeight,           // Breite
    CDC* pSrcDC,           // Höhe
    int xSrc, int ySrc,    // Quellausschnitt
    DWORD dwRop );        // Speicherkontext
                        // linke obere Ecke
                        // Kopiermodus
    
```

Die Methode BitBlt() kopiert ein Bitmap pixelweise aus dem Speicherkontext in den Teil des Fensters, der der Methode BitBlt() in Form der oberen linken Ecke, der Breite und der Höhe angegeben wurde.

Wenn der Zielausschnitt kleiner ist als das Bitmap, werden nur so viele Pixel des Bitmaps kopiert, wie in den Ausschnitt passen (wobei man durch die Anfangskoordinaten des Quellausschnitts angeben kann, welcher Teil des Bitmaps kopiert werden soll). Umgekehrt füllt das Bitmap den Zielausschnitt nicht aus, wenn der Zielausschnitt breiter und höher ist als das Bitmap.

Mit einem Wort: `BitBlt()` paßt das Bitmap nicht an den Zielausschnitt an.

Genau diese Option braucht man aber, wenn man ein Bitmap als Hintergrundbild verwenden möchte. Und genau diese Möglichkeit bietet uns die Methode `StretchBlt()`.

StretchBlt()

```
BOOL StretchBlt(
    // Zielausschnitt:
    int x, int y,          // linke obere Ecke
    int nWidth,           // Breite
    int nHeight,          // Höhe
    // Quellausschnitt
    CDC* pSrcDC,          // Speicherkontext
    int xSrc, int ySrc,   // linke obere Ecke
    int nWidth,           // Breite
    int nHeight,          // Höhe
    DWORD dwRop );       // Kopiermodus
```

`StretchBlt()` paßt den exakt durch linke obere Ecke, Breite und Höhe angegebenen Quellausschnitt in den Zielausschnitt ein.

Definiert man als Zielausschnitt den Client-Bereich des Fensters, wird das Bitmap in das Fenster eingepaßt.

Übung 13-3: Bitmap als Hintergrundbild anzeigen

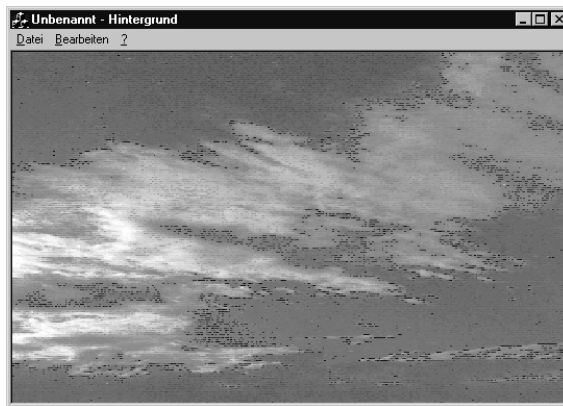
1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe das Projekt für diese Übung einfach »Hintergrund« genannt.
2. Rufen Sie den IDE-Befehl EINFÜGEN/RESSOURCE auf, wählen Sie links den Eintrag BITMAP aus, und drücken Sie den Schalter IMPORTIEREN. Wählen Sie das gewünschte Bitmap für den Fensterhintergrund aus.
3. Springen Sie im Quelltexteditor zur `OnDraw()`-Methode der Ansichts-klasse. Am einfachsten klickt man dazu im Arbeitsbereichsfenster, Seite KLASSEN, auf den `CHINTERGRUNDVIEW`-Knoten für `ONDRAW()`.
4. Fügen Sie den Code zum Anzeigen des Bitmaps ein.

```
void CHintergrundView::OnDraw(CDC* pDC)
{
    CHintergrundDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
```

```
// ZU ERLEDIGEN: Hier Code zum Zeichnen der
// ursprünglichen Daten hinzufügen
CBitmap bitmap;
bitmap.LoadBitmap(IDB_BITMAP1);

BITMAP bm;
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);
}
```

Bild 13.3:
Anzeige eines
Hintergrund-
bitmaps mit
StretchBlt()



13.4 Bitmaps manipulieren

Zuletzt wollen wir uns anschauen, wie man Bitmaps pixelweise bearbeiten und verändern kann. Das folgende Programm wandelt ein Bitmap in eine Liniengrafik um (zumindest sieht das Bitmap nach der Bearbeitung wie eine Liniengrafik aus).

Die Technik dazu funktioniert wie folgt:

- ✗ Zuerst wird das Bitmap in den Speicherkontext geladen.

- ✗ Mit Hilfe der CDC-Methode `GetPixel()` kann man dann abfragen, welche Farbe ein bestimmtes Pixel hat, und mit Hilfe der Methode `SetPixel()` kann man die Farbe eines Pixels verändern (wobei die einzelnen Pixel über ihre x,y-Koordinaten identifiziert werden).
- ✗ Dann kopiert man das manipulierte Bitmap in das Fenster.

Für die Bearbeitung des Bitmaps definieren wir in der Ansichtsklasse eine eigene Methode. (Hinweis: Sauberer wäre es, eine eigene Klasse von CDC abzuleiten und in dieser die Methode zu definieren, doch wir wollen uns hier auf die Pixelmanipulation konzentrieren.)

```
void Linien_erzeugen(CDC *pDC, BITMAP bm)
```

In dieser Methode werden die Pixel des Bitmaps Zeile für Zeile von links nach rechts durchgegangen. Das einzige, was uns dabei interessiert, ist, wie stark sich die Intensität der Farbkomponenten von Pixel zu Pixel ändert. Ändert sich die Farbe von Schwarz nach Weiß, bedeutet dies einen großen Intensitätsunterschied, einen hohen Kontrast; eine Linie wurde erkannt! Letzten Endes haben wir es also mit einem Kontrastdetektor zu tun. Hohe Kontraste führen dazu, daß die betroffenen Pixel in dem gefilterten Bild hell erscheinen, geringe Kontraste bewirken, daß die Pixel kaum hervortreten.

```
void CLiniengrafikView::Linien_erzeugen(CDC *pDC, BITMAP bm)
{
    // Die Intensität kann variiert werden.
    int Intensitaet = 2;
    // Die erste Zeile ist nicht definiert; grau färben.
    for( int i = 0; i < bm.bmWidth; i++ )
        pDC->SetPixel(i, 0, 0xff808080);
    for( int y = 1; y < bm.bmHeight; y++ )
        for( int x = 0; x < bm.bmWidth; x++ )
        {
            // Pixel links des aktuellen Pixels.
            int c = pDC->GetPixel(x - 1, y);
            int r1 = ( c & 0x00ff0000 ) >> 16;
            int g1 = ( c & 0x0000ff00 ) >> 8;
            int b1 = ( c & 0x000000ff );
            // Pixel über dem aktuellen Pixel.
            c = pDC->GetPixel(x, y - 1 );
            int r2 = ( c & 0x00ff0000 ) >> 16;
            int g2 = ( c & 0x0000ff00 ) >> 8;
            int b2 = ( c & 0x000000ff );
            // Aktuelles Pixel.
            c = pDC->GetPixel(x, y);
```



```

int r = ( c & 0x00ff0000 ) >> 16;
int g = ( c & 0x0000ff00 ) >> 8;
int b = ( c & 0x000000ff );
r = min( ( abs( r2 - r ) + abs( r1 - r ) )
        * Intensitaet, 255 );
g = min( ( abs( g2 - g ) + abs( g1 - g ) )
        * Intensitaet, 255 );
b = min( ( abs( g2 - b ) + abs( b1 - b ) )
        * Intensitaet, 255 );
pDC->SetPixel(x, y, 0x00000000 | ( r << 16 )
              | ( g << 8 ) | b);
    }
}

```

Der Algorithmus ist nicht ganz perfekt, erzeugt aber interessante Effekte, die durch Zuweisung verschiedener Werte an die Variable `Intensitaet` variiert werden können.

Übung 13-4: Bitmap pixelweise verändern

1. Legen Sie mit Hilfe des MFC-Anwendungs-Assistenten eine neue SDI-Anwendung mit Unterstützung für Doc/View an. Ich habe das Projekt für diese Übung »Liniengrafik« genannt.
2. Rufen Sie den IDE-Befehl EINFÜGEN/RESSOURCE auf, wählen Sie links den Eintrag BITMAP aus, und drücken Sie den Schalter IMPORTIEREN. Wählen Sie das gewünschte Bitmap aus.
3. Springen Sie im Quelltexteditor zur `OnDraw()`-Methode der Ansichts-klasse.
4. Fügen Sie den Code zum Anzeigen des Bitmaps ein. Der Einfachheit halber zeichnen und bearbeiten wir das Bild gleich in `OnDraw()`, das heißt, wir zeichnen das Originalbild, rufen dann die Methode `Linien-grafik_erzeugen()` auf und zeichnen danach das Bild noch einmal neben sich selbst. (Dies ist nicht gerade effektiv, denn zeitaufwendige Bild-manipulationen in `OnDraw()` können das Neuzeichnen des Fensters stark verzögern. Besser wäre es, eine Kopie des Bildes anzulegen (das Bild ein zweites Mal laden), die Kopie einmalig zu bearbeiten und in `OnDraw()` neben dem Original auszugeben.)

```

////////////////////////////////////
// CLiniengrafikView Zeichnen
void CLiniengrafikView::OnDraw(CDC* pDC)
{

```

```

CLiniengrafikDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
// ZU ERLEDIGEN: Hier Code zum Zeichnen der
// ursprünglichen Daten hinzufügen
CBitmap bitmap;
bitmap.LoadBitmap(IDB_BITMAP1);

BITMAP bm;
bitmap.GetObject(sizeof(bm), &bm);
CDC speicherDC;
speicherDC.CreateCompatibleDC(pDC);
speicherDC.SelectObject(&bitmap);
pDC->BitBlt(
    10, 10, bm.bmWidth, bm.bmHeight,
    &speicherDC, 0, 0, SRCCOPY);
Linien_erzeugen(&speicherDC, bm);
pDC->BitBlt(
    20 + bm.bmWidth, 10, bm.bmWidth, bm.bmHeight,
    &speicherDC, 0, 0, SRCCOPY);
}

```

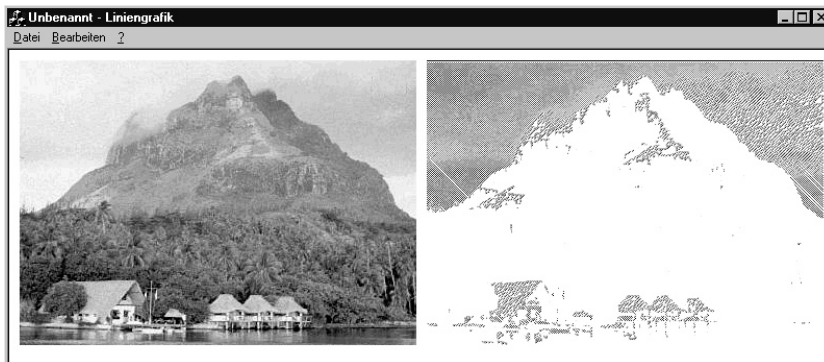


Bild 13.4:
Erzeugen einer
Liniengrafik

13.5 Zusammenfassung

Bilder werden in der MFC durch die Klasse `CBitmap` repräsentiert. `CBitmap`-Objekte können als GDI-Objekte in einen Speicherkontext geladen und von dort in einen kompatiblen Fenstergerätekontext gezeichnet werden. Die beiden Methoden, mit denen man Bitmaps kopieren kann, sind `BitBlt()` und `StretchBlt()`.

Will man ein Bitmap verändern, muß man es zuerst in den Speichergerätekontext laden. Dort kann man dann mit Hilfe der CDC-Methoden, beispielsweise auch der Methoden `GetPixel()` und `SetPixel()` auf das Bitmap zugreifen und in das Bitmap malen.

13.6 Fragen

1. Enthalten Gerätekontexte auch Standardbitmaps (ähnlich wie die Standardobjekte für Stifte, Pinsel und Font)?
2. Welche Methode verwendet man, um ein Bitmap in einen vorgegebenen Rahmen einzupassen?
3. Kann man Bitmaps auch aus Dateien laden?

13.7 Aufgaben

1. Implementieren Sie das Programm zum Einarmigen Bandit aus Kapitel 9 mit einem Hintergrundbitmap.
2. Schreiben Sie ein Programm, das in seiner `OnDraw()`-Methode ein eigenes Bitmap erzeugt und anzeigt.

Deklarieren Sie ein `CBitmap`-Objekt, und definieren Sie ein Byte-Array, in das Sie die Farbwerte für die einzelnen Pixel speichern.

Erzeugen Sie mit Hilfe der `CBitmap`-Methode `CreateCompatibleBitmap()` ein Bitmap-Objekt (statt dieses aus einer Ressource zu laden).

Kopieren Sie mit Hilfe der `CBitmap`-Methode `SetBitmapBits()` die Farbwerte aus dem Byte-Array in das Bitmap.

Laden Sie das Bitmap dann wie gewohnt in einen Speichergerätekontext, und zeigen Sie es an.

13.8 Lösungen zu den Aufgaben

Zu 1: Einarmiger Bandit mit Hintergrundbitmap

Erstellen Sie im Windows Explorer ein neues Verzeichnis (beispielsweise *Bandit2*), öffnen Sie das Verzeichnis *Bandit*, in dem die Dateien des alten Projekts stehen, und kopieren Sie diese Dateien samt Unterverzeichnissen in das neue Verzeichnis *Bandit*.

Kopieren Sie ein hübsches Hintergrundbitmap in das *RES*-Verzeichnis des neuen Projekts, und laden Sie das Projekt mit Hilfe des Befehls *DATEI/ARBEITSBEREICH ÖFFNEN* in die IDE.

Gehen Sie dann wie in Übung 13-3 vor, um die Bitmap-Ressource einzurichten und das Bitmap anzuzeigen. Den Code für die *OnDraw()*-Methode können Sie direkt übernehmen, Sie müssen schlimmstenfalls die ID für das Bitmap und den Namen der Dokumentklasse anpassen.

Um die Steuerelemente brauchen Sie sich keine Gedanken zu machen. Diese werden automatisch über dem Hintergrundbitmap angezeigt.



Bild 13.5:
Einarmiger
Bandit mit
Hintergrund-
bild

Zu 2: Eigenes Bitmap erzeugen

Der folgende Code erzeugt ein Bitmap aus 16 Pixeln, das ein schwarzes Karree in einem weißen Rahmen ausgibt. Allerdings wird bei dieser Definition der Farben für die Pixel davon ausgegangen, daß jede Farbe durch genau ein Byte spezifiziert wird. Um das schwarze Karree zu sehen, müssen Sie also die Farbauflösung Ihres Bildschirms auf 256 Farben einstellen (über *EINSTELLUNGEN/SYSTEMSTEUERUNG/ANZEIGE*).

```
void CBunglView::OnDraw(CDC* pDC)
{
    CBunglDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // ZU ERLEDIGEN: Hier Code zum Zeichnen der
    // ursprünglichen Daten hinzufügen
    BITMAP bmp;

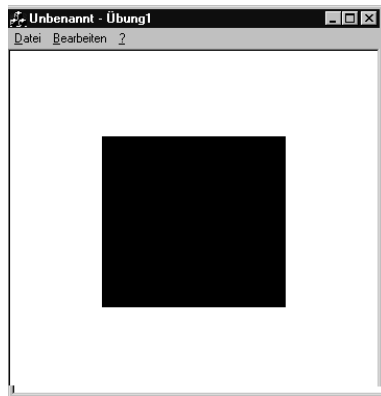
    CBitmap bitmap;
```

```

BYTE bits[16] = { 255, 255, 255, 255,
                  255,  0,  0, 255,
                  255,  0,  0, 255,
                  255, 255, 255, 255};
// Breite und Höhe in Pixeln angeben
bitmap.CreateCompatibleBitmap(pDC, 4, 4);
// Anzahl Bytes in Bits
bitmap.SetBitmapBits(16, &bits);
bitmap.GetObject(sizeof(bmp), &bmp);
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, bmp.bmWidth, bmp.bmHeight,
    SRCCOPY);
}

```

*Bild 13.6:
Das selbst
erzeugte
Bitmap bei
Auflösung mit
256 Farben*





TEIL 3

Verstehen



Der von den Assistenten erzeugte Code

Was soll ich Ihnen jetzt noch von der Windows-Programmierung erzählen. Ich denke, alles Wichtige ist besprochen, das Fundament ist gelegt, jetzt ist es an Ihnen, auszuziehen, Erfahrungen zu sammeln und das eigene Wissen auszubauen.

Doch ohne noch einmal auf Windows und die Grundlagen der Windows-Programmierung zurückzukommen, möchte ich Sie nicht ziehen lassen. Gut, Sie wissen mittlerweile, wie das MFC-Anwendungsgerüst aufgebaut ist, wie man es erweitert, wie man mit Hilfe der MFC-Klassen eigene Anwendungsgerüste erstellt oder wie man in der Doc/View-Architektur Datenverwaltung und Datenanzeige trennt. Sie wissen auch, daß man zwischen der Instanz einer MFC-Fensterklasse und dem eigentlichen mit `Create()` erzeugten Fensterobjekt unterscheiden muß. Sie wissen, daß Benutzerereignisse, wie z.B. Mausklicks, als Nachrichten an die Message Loop der Anwendung geschickt werden und die Verbindung zwischen Nachrichten und Behandlungsmethoden in Antworttabellen organisiert wird. Sie wissen, daß man nicht direkt in Fenster, sondern in Gerätekontexte zeichnet, die Fenster repräsentieren, und Sie wissen schließlich, wie man sich Gerätekontexte für Fenster beschafft und durch Laden eigener Zeichenwerkzeuge konfiguriert. All dies wurde bereits angesprochen und sicherlich auch schon halbwegs von Ihnen verinnerlicht.

Doch damit wollen wir uns nicht zufrieden geben. In diesem Kapitel werden wir die wichtigsten Punkte noch einmal zusammentragen. Manche Punkte, wie die Windows-Fensterklassen und die MFC-Antworttabellen, werden wir uns genauer anschauen, als dies in den vorangehenden Kapiteln der Fall war.



Doch das Wichtigste ist, daß wir alle diese Konzepte zusammenfassen und einen klaren Überblick gewinnen. Zu diesem Zweck werden wir peu à peu ein kleines Windows-API-Programm aufbauen, an dessen Beispiel wir sehen werden, wie Windows-Programme gestartet werden, wie sie ihre Hauptfenster erzeugen, wie sie von Windows Nachrichten empfangen und wie sie auf diese Nachrichten reagieren.

Sie lernen in diesem Kapitel:

- ✗ Wie Windows-Programme von Windows aufgerufen werden
- ✗ Was Fensterklassen sind
- ✗ Wie unter Windows Fenster erzeugt werden
- ✗ Wie unter Windows Nachrichten verschickt werden
- ✗ Wie Windows-Programme Nachrichten empfangen
- ✗ Was eine Fensterfunktion ist
- ✗ Wo die oben genannten Konzepte in unseren MFC-Programmen versteckt sind

14.1 Wie funktionieren Windows-Programme?

Um diese Frage zu klären, werden wir die Ausführung eines einfachen API-Programms vom Aufruf des Programms bis zur Textausgabe im Fenster des Programms verfolgen.

14.1.1 Die Eintrittsfunktion WinMain()

Der erste Punkt ist, daß die Eintrittsfunktion für Windows-Programme – also die Funktion, die vom Betriebssystem zum Starten des Programms aufgerufen wird – nicht `main()`, sondern `WinMain()` heißt:

```
int APIENTRY WinMain( HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int nCmdShow )
```

Wie gesagt, wird diese Funktion von Windows aufgerufen, und Windows übernimmt auch die Aufgabe, den Parametern der Funktion passende Argumente zu übergeben – als da wären:

- ✗ **hInstance.** Windows muß alle im System existierenden Objekte (Anwendungen, Fenster, Gerätekontexte, Dateien etc.) verwalten. Dazu muß es diese Objekte natürlich auch irgendwie identifizieren und ansprechen können. Zu diesem Zweck weist Windows den Objekten sogenannte Handles zu – so auch jeder neu aufgerufenen Anwendung. Und damit die Anwendung selbst auch weiß, was für ein Handle ihr zugewiesen wurde, übergibt Windows der Anwendung ihren eigenen Handle als Argument an **hInstance**.

Unter Win16 (Windows 3.x) war es sehr wichtig, daß die Handles eindeutig waren, da alle Programme in einem Adreßraum abliefen.

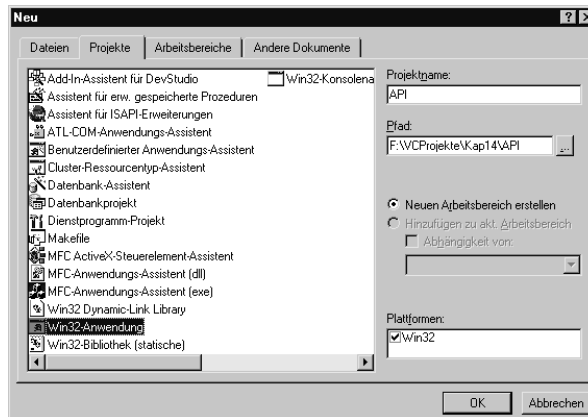


Unter Win32 (Windows 95, Windows NT) kann eine Verwechslung mit anderen Prozessen nicht vorkommen, da jeder Prozeß seinen eigenen Prozeßraum zugewiesen bekommt. Der Instanz-Handle entspricht im übrigen der Ladeadresse des Programms und liegt üblicherweise bei 0x0040 0000.

- ✗ **hPrevInstance.** Ein Relikt aus alten Win16-Tagen. Unter Win16 wurde diesem Parameter der Handle der vorherigen Instanz des Programms übergeben (für den Fall, daß das Programm mehrfach unter Windows ausgeführt wird). Unter Win32 wird stets NULL übergeben.
- ✗ **lpCmdLine.** Dieser Parameter ist schon interessanter, denn ihm wird die Kommandozeile des Programmaufrufs übergeben.
- ✗ **nCmdShow.** Dieser Parameter legt das Erscheinungsbild des Anwendungsfensters auf dem Desktop fest – beispielsweise, ob das Fenster in normaler Größe, als Vollbild oder als Symbol aufgerufen werden soll (siehe `CWnd::ShowWindow()` in Online-Hilfe).

Übung 14-1: Win-API-Projekt anlegen

Bild 14.1:
Win32-Anwen-
dungsprojekt
anlegen



1. Rufen Sie den Befehl DATEI/NEU auf, und wechseln Sie zur Seite PROJEKTE des Dialogfelds NEU. Geben Sie wie üblich Namen und Verzeichnis des Projekts an, achten Sie darauf, daß ein neuer Arbeitsbereich angelegt wird, und wählen Sie WIN32-ANWENDUNG als Projekttyp.
2. Entscheiden Sie sich im nachfolgenden Dialogfeld für EINE EINFACHE WIN32-ANWENDUNG, damit gleich eine Quelltextdatei für das Programm eingerichtet wird.
3. Rufen Sie die Quelltextdatei auf, und schauen Sie sich den Aufruf der WinMain()-Funktion an.

```
#include "stdafx.h"
int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR     lpCmdLine,
                    int       nCmdShow )
{
    // ZU ERLEDIGEN: Fügen Sie hier den Code ein.
    return 0;
}
```

14.1.2 Erzeugung des Hauptfensters

Unsere Anwendung wurde jetzt also gestartet, doch man sieht nichts von ihr, weil sie über kein Fenster verfügt.

Aus der MFC-Programmierung wissen Sie, daß Fenster durch einen Aufruf der `Create()`-Methode erzeugt werden. Hinter dieser MFC-Methode steht eine API-Funktion namens `CreateWindow()`:

```
HWND CreateWindow(
    LPCTSTR lpClassName,    // Zeiger auf registr. Fensterklasse
    LPCTSTR lpWindowName,   // Zeiger auf Fenstername (Titel)
    DWORD dwStyle,          // Fensterstil
    int x,                  // Horizontale Position
    int y,                  // Vertikale Position
    int nWidth,             // Breite des Fensters
    int nHeight,            // Höhe des Fensters
    HWND hWndParent,        // Übergeordnetes Fenster/Besitzer
    HMENU hMenu,            // Handle des Menüs
    HANDLE hInstance,       // Handle der Anwendung
    LPVOID lpParam           // Zeiger auf Fensterdaten
);
```

Möchte man nun mit Hilfe dieser Funktion ein Fenster erzeugen, stößt man gleich beim ersten Parameter auf Probleme: Eine registrierte Fensterklasse wird verlangt.

Fensterklassen

Unter Windows ist eine Fensterklasse eine Struktur, denn Windows selbst ist nicht objektorientiert. Dennoch steht hinter dieser Fensterklasse das gleiche Konzept wie hinter den Klassen der objektorientierten Programmierung. Die Klasse ist eine allgemeine Beschreibung, auf deren Grundlage bestimmte individuelle Objekte, in diesem Falle eben Fenster, erzeugt werden können.

Wenn Sie in C++ mit Koordinaten arbeiten wollen (siehe Kapitel 12, Übung 12-4), deklarieren Sie zuerst eine passende Klasse (in Übung 12-4 `CKoord`), in der festgelegt ist, über welche Attribute Objekte dieser Klasse verfügen sollen (x- und y-Werte). Später können Sie dann für jedes Koordinatenpaar eine Instanz dieser Klasse erzeugen und den Instanzen Werte für die Attribute der Klasse zuweisen. Ähnlich werden unter Windows Fenster erzeugt. Allerdings ist die Klassendeklaration bereits in Form der Struktur `WNDCLASS` vorgegeben.

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

Für diese Struktur muß nun eine Variable erzeugt werden. Dies ist dann die Fensterklasse. Dabei werden den verschiedenen Elementen der Struktur Werte zugewiesen. So sieht man für die Fensterklasse beispielsweise ein Symbol (*hIcon*), einen Mauszeiger (*hCursor*), eine Hintergrundfarbe (*hbrBackground*) – und eine Fensterfunktion (*lpfnWndProc*) vor. Die Fensterfunktion ist dabei besonders wichtig, denn sie legt fest, wie das Fenster auf Nachrichten reagiert (dazu im nächsten Abschnitt mehr). Der letzte Schritt ist, die Fensterklasse unter Windows anzumelden. Danach können auf der Basis dieser Fensterklasse Fenster erzeugt werden. Alle Fenster einer Fensterklasse teilen sich die Elemente, die in der Fensterklasse definiert sind, also Symbol, anfängliche Hintergrundfarbe, Fensterfunktion etc.



Die Fensterklasse ist also letztlich eine Strukturvariable. Auf diese Weise können in C mehrere Fensterklassen auf der Grundlage einer gemeinsamen Definition erzeugt werden. In C++ hätte man statt dessen eine abstrakte Basisklasse definiert und auf deren Grundlage für jeden Fenstertyp eine eigene abgeleitete Klasse erzeugt. Aber Windows ist halt nicht objektorientiert, und so muß man sich irgendwie behelfen.

Einige Fensterklassen sind in Windows vordefiniert und registriert – etwa die Klassen für die Standardsteuerelemente. Die Klasse für die Schaltflächen heißt z.B. *BUTTON*, und auf ihrer Grundlage kann man direkt eigene Fenster erstellen:

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

Wenn man aber nicht gerade eine Schaltfläche als Hauptfenster haben möchte, muß man eine eigene Fensterklasse definieren.

Fenster werden auf der Basis von Fensterklassen erzeugt. Bevor man auf der Grundlage einer Fensterklasse ein Fenster erzeugen kann, muß die Fensterklasse unter Windows registriert werden. Alle Fenster, die auf eine Fensterklasse zurückgehen, verwenden die gleiche Fensterfunktion zur Nachrichtenverarbeitung.



Fensterklasse definieren

Der folgende Quelltextauszug definiert eine Fensterklasse für unser Hauptfenster.

```
WNDCLASS WinClass;          // Fensterklasse
// Speicher reservieren und Variable einrichten
memset(&WinClass, 0, sizeof(WNDCLASS));
WinClass.style = CS_HREDRAW | CS_VREDRAW;
WinClass.lpfnWndProc = WndProc;
WinClass.hInstance = hInstance;
WinClass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
WinClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WinClass.lpszClassName = "Windows-Programm";
```

Bis auf WndProc sind alle Werte, die wir den Feldern der Fensterklasse zuweisen, vordefiniert. WndProc müssen wir aber selbst implementieren.

WndProc ist ein Zeiger auf die Fensterfunktion der Klasse. Unter C kann man auf der rechten Seite von Zuweisungen statt eines Zeigers auf eine Funktion einfach den Namen der Funktion übergeben. Wir brauchen also nur noch eine Funktion WndProc() mit passenden Parametern zu definieren:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMessage,
                          WPARAM wParam, LPARAM lParam)
{
    return 0;
}
```

Da wir noch keine genaue Vorstellung davon haben sollen, was diese Funktion machen soll, lassen wir sie einfach den Wert 0 zurückliefern.

Fensterklasse registrieren

Als nächstes wird die Fensterklasse unter Windows angemeldet. Dazu verwendet man die API-Funktion `RegisterClass()`:

```
if(!RegisterClass(&WinClass))
    return(FALSE);
```

Fenster erzeugen

Jetzt, nachdem die Fensterklasse registriert ist, können wir auf der Grundlage der Fensterklasse ein Fenster erzeugen und anzeigen.

```
HWND    hWindow;           // Fenster-Handle
// erstelle Hauptfenster der Anwendung
hWindow = CreateWindow("Windows-Programm",
                      "API-Programm",
                      WS_OVERLAPPEDWINDOW, 10, 10,
                      400, 300, NULL, NULL, hInstance, NULL);
ShowWindow(hWindow, nCmdShow);
UpdateWindow(hWindow);
```

Übung 14-2: Hauptfenster für Anwendung erzeugen

1. Übertragen Sie den oben aufgeführten Code zur Erzeugung eines Fensters in Ihr API-Programm.
2. Führen Sie das Programm aus (`[Strg] + [F5]`).

Das Hauptfenster des Programms flackert kurz auf und verschwindet gleich wieder, weil die Anwendung sofort nach dem Aufruf wieder beendet wird. Was fehlt, ist eine Warteschleife, in der die Anwendung auf Windows-Nachrichten wartet und diese verarbeitet.

14.1.3 Eintritt in die Nachrichtenverarbeitung

Wie Sie bereits aus Kapitel 7 wissen, fängt Windows alle Benutzerereignisse (Eingaben über Maus und Tastatur) ab und schickt sie in Form von Nachrichten an die betreffenden Anwendungen, genauer gesagt an eine spezielle Warteschlangenstruktur, die Windows für jede laufende Anwendung einrichtet. (Noch genauer: Da Win32 Multithreading unterstützt, erhält jeder Thread eine eigene Warteschlange für einkommende Nachrichten.)

Die Message Loop

Um die eingetroffenen Nachrichten auszulesen, muß die Anwendung eine Schleife implementieren, in der sie ständig die Nachrichtenwarteschlange nach Nachrichten abfragt und diese gegebenenfalls ausliest. Dies ist die sogenannte »Message Loop«, die typischerweise wie folgt implementiert ist:

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

Sie ist allerdings nicht die Endstation der Nachrichtenverarbeitung, sondern lediglich eine Zwischenstation, denn das eigentliche Ziel einer Nachricht ist das Fenster, an das die Nachricht gerichtet ist – oder, um ganz exakt zu sein, die Fensterfunktion der Fensterklasse des Fensters. Die Aufgabe, die einkommenden Nachrichten an die Fensterfunktionen der verschiedenen Fenster der Anwendung zu verteilen (hierzu zählt nicht nur das Hauptfenster, sondern beispielsweise auch Steuerelemente), übernimmt die Funktion `DispatchMessage()`, die die Nachricht zu diesem Zweck wieder an Windows zurückgibt.

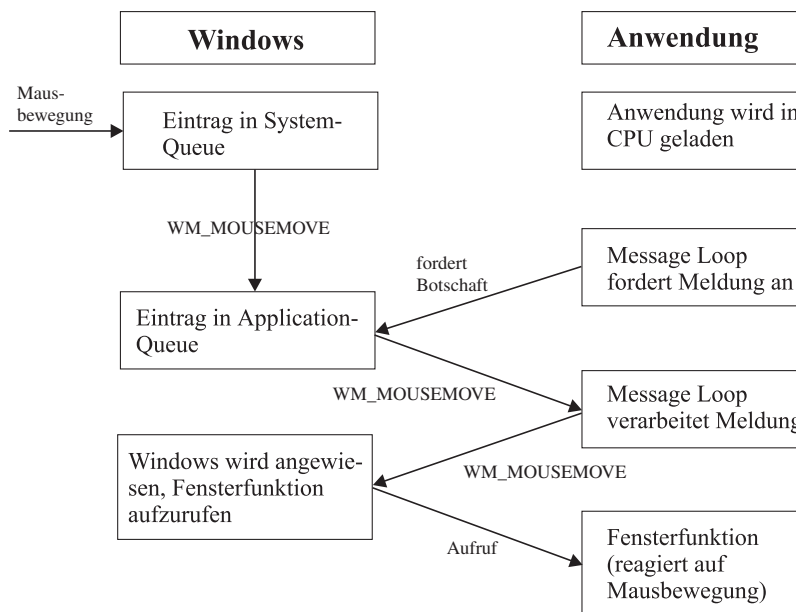


Bild 14.2:
Nachrichten-
verarbeitung
unter Windows

Übung 14-3: Message Loop implementieren

1. Implementieren Sie am Ende der WinMain()-Funktion eine Message Loop.

```
...
ShowWindow(hWindow, nCmdShow);
UpdateWindow(hWindow);

// Message loop
while (GetMessage (&Message, NULL, 0, 0) )
{
    TranslateMessage(&Message);
    DispatchMessage(&Message);
}
return (Message.wParam);
}
```



Führen Sie das Programm so bitte noch nicht aus. Es könnte nämlich wegen der fehlenden Implementierung der Fensterfunktion nicht korrekt beendet werden.

Die Fensterfunktion

Der letzte Schritt der Nachrichtenverarbeitung besteht darin, daß Windows die Fensterfunktion des Fensters aufruft, an das die Nachricht gerichtet ist, und die Nachricht als Argument an die Parameter der Funktion übergibt.

Aufgabe des Programmierers ist es, in der Funktion abzufragen, welche Nachricht empfangen wurde und dann auf die jeweilige Nachricht passend zu reagieren. Zu diesem Zweck richtet man in der Fensterfunktion eine switch-Verzweigung ein und für jede zu behandelnde Nachricht einen case-Block.

Aber natürlich kann man nicht von einem Programmierer erwarten, daß er für alle der über 200 Windows-Nachrichten Code zur Beantwortung bereitstellt – zumal das Gros der Windows-Nachrichten für die Anwendung ganz uninteressant ist. Aus diesem Grunde ist in Windows die Funktion DefWindowProc() definiert, der man im default-Block alle Nachrichten, die man nicht selbst behandeln möchte, weiterreicht.

Eine Nachricht muß man aber auf jeden Fall behandeln – WM_DESTROY. Die Nachricht WM_DESTROY wird versendet, wenn der Anwender das Hauptfenster schließt, um die Anwendung zu beenden. Das Schließen des Hauptfen-

sters geschieht automatisch, doch damit auch die Anwendung beendet wird, muß die Message Loop der Anwendung beendet werden. Dies geschieht, indem man als Antwort auf die WM_DESTROY-Nachricht die API-Funktion PostQuitMessage() aufruft. Diese schickt ihrerseits eine WM_QUIT-Nachricht an die Anwendung, und diese beendet die Message Loop.

Jetzt wollen wir die Fensterfunktion aber auch zur eigenen Nachrichtenbehandlung nutzen. Wie Sie bereits wissen, heißt die Windows-Nachricht für das Drücken der linken Maustaste WM_LBUTTONDOWN. Wir werden also in der Fensterfunktion eine switch-Anweisung implementieren, in der überprüft wird, welche Windows-Nachricht gerade übergeben wurde. Für die WM_LBUTTONDOWN-Nachricht legen wir einen eigenen case-Block an, in dem wir den Code zur Ausgabe eines kurzen Mitteilungstextes aufsetzen.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMessage,
                          WPARAM wParam, LPARAM lParam)
{
    char str[30] = "Hier erfolgte Mausklick";
    HDC dc;

    // beantworte Nachrichten mit entsprechenden Aktionen
    switch(uiMessage)
    {
        case WM_LBUTTONDOWN:
            dc = GetDC(hWnd);
            TextOut(dc, LOWORD(lParam),
                    HIWORD(lParam),
                    str, strlen(str));

            // Gerätekontext freigeben
            ReleaseDC(hWnd, dc);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hWnd, uiMessage,
                                  wParam, lParam);
    }
}
```

Übung 14-4: Fensterfunktion implementieren

1. Setzen Sie den Code für die Fensterfunktion ein und führen Sie das Programm dann aus.

Hier der vollständige Quelltext:

```
// API.cpp : Definiert den Einsprungpunkt für die Anwendung.
//
#include "stdafx.h"
// Vorwärtsdeklaration
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
// Eintrittsfunktion, die Argumente werden von Windows an
// die Funktion übergeben
int APIENTRY WinMain( HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int nCmdShow )
{
    // ZU ERLEDIGEN: Fügen Sie hier den Code ein.
    HWND hWindow; // Fenster-Handle
    MSG Message; // Strukturvariable für Nachrichten
    WNDCLASS WinClass; // Fensterklasse
    // erste Instanz
    memset(&WinClass, 0, sizeof(WNDCLASS));
    WinClass.style = CS_HREDRAW | CS_VREDRAW;
    WinClass.lpfWndProc = WndProc;
    WinClass.hInstance = hInstance;
    WinClass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    WinClass.hCursor = LoadCursor(NULL, IDC_ARROW);
    WinClass.lpszClassName = "Windows-Programm";
    // Fensterklasse anmelden
    if(!RegisterClass(&WinClass))
        return(FALSE);
    // erstelle Hauptfenster der Anwendung
    hWindow = CreateWindow("Windows-Programm",
                          "API-Programm",
                          WS_OVERLAPPEDWINDOW, 10, 10,
                          400, 300, NULL, NULL, hInstance, NULL);
    ShowWindow(hWindow, nCmdShow);
    UpdateWindow(hWindow);
    // Message loop
```

```

while (GetMessage (&Message, NULL, 0, 0) )
{
    TranslateMessage(&Message);
    DispatchMessage(&Message);
}
return (Message.wParam);
}

// Fensterfunktion WinProcedure
LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMessage,
                          WPARAM wParam, LPARAM lParam)
{
    char str[30] = "Hier erfolgte Mausklick";
    HDC dc;

    // beantworte Nachrichten mit entsprechenden Aktionen
    switch(uiMessage)
    {
        case WM_LBUTTONDOWN:
            dc = GetDC(hWnd);
            TextOut(dc, LOWORD (lParam),
                    HIWORD (lParam) ,
                    str, strlen(str));

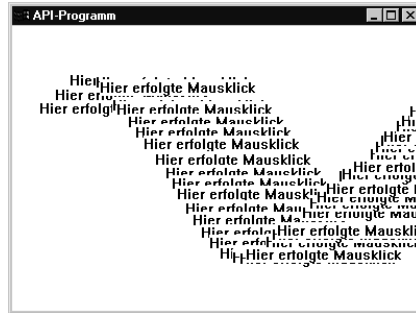
            // Gerätekontext freigeben
            ReleaseDC(hWnd, dc);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hWnd, uiMessage,
                                   wParam, lParam);
    }
}

```

Bild 14.3:
Ein echtes API-
Programm



Nachrichtenverarbeitung unter Umgehung der Message Loop

Nicht alle Windows-Nachrichten laufen wie oben gezeigt über die Message Loop der Anwendung. Viele Nachrichten werden von Windows unter Umgehung der Message Loop direkt an die Fensterfunktion des Zielfensters geschickt – und dies aus gutem Grund.

Viele interne Windows-Nachrichten dienen dazu, eine Anwendung darüber zu informieren, was augenblicklich auf dem System vor sich geht. Um diese Nachrichten über die Message Loop laufen zu lassen, müßte Windows die Nachrichten in die Nachrichtenwarteschlange der Anwendung stellen. Für die Nachrichten in der Warteschlange gilt aber, daß die zuerst eintreffenden Nachrichten auch zuerst bearbeitet werden. Dies kann unerwünschte Verzögerungen bedingen, und wenn die Nachricht endlich aus der Warteschlange ausgelesen wird, ist sie vielleicht bereits längst überholt. Fast alle internen Windows-Nachrichten werden daher direkt an die Fensterfunktionen geschickt, damit sie verzögerungsfrei verarbeitet werden können.

Für Benutzereingaben (Mausklicks, Tastaturereignisse, Menübefehlsaufrufe) ist die chronologische Abarbeitung der Nachrichten in der Warteschlange aber unbedingt notwendig. Schließlich möchte der Anwender, wenn er beim Aufsetzen eines Textes in einen bestimmten Absatz klickt und dort ein neues Wort einfügt, daß zuerst der Mausklick und dann die Tastatureingaben bearbeitet werden und nicht umgekehrt.

Nachrichten selbst senden

Fenster können nicht nur Nachrichten empfangen, sie können auch Nachrichten schicken: an sich selbst oder an andere Fenster, über die Message Loop oder direkt an eine Fensterfunktion.

PostMessage()

Zum Abschicken von Botschaften über die MessageLoop verwendet man die Funktionen der PostMessage-Familie. Diese Funktionen tragen die zu verschickende Botschaft in die gewünschte Nachrichtenwarteschlange ein 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 Nachricht in der empfangenden Anwendung bereits bearbeitet wurde!



Die Methode PostMessage() der Klasse CWnd erwartet als Parameter lediglich die Kennziffer der Nachricht und die mitzuliefernden Parameter:

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

Die Nachricht wird dann in die zu dem aufrufenden CWnd-Objekt gehörende Message Loop geschrieben.

Möchte man die Botschaft an ein anderes Fenster schicken, benutzt man die API-Version von PostMessage():

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

der man als zusätzlichen Parameter den Handle des empfangenden Fensters übergibt.

SendMessage()

Um Nachrichten direkt an bestimmte Fenster zu schicken, verwendet man die Funktionen der SendMessage()-Familie. Diese Funktionen senden die zu übermittelnde Nachricht 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 Nachricht verarbeitet hat. Die SendMessage()-Funktionen sind daher vor allem für die schnelle und synchronisierte Verarbeitung von Nachrichten bestens geeignet!

Die Methode `SendMessage()` der Klasse `CWnd` erwartet als Parameter lediglich die Kennziffer der Nachricht und die mitzuliefernden Parameter:

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

Die Nachricht wird direkt an die zu dem aufrufenden `CWnd`-Objekt gehörende Fensterfunktion geschickt.

Möchte man die Nachricht an ein anderes Fenster schicken, benutzt man die API-Version von `SendMessage()`:

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

der man als zusätzlichen Parameter den Handle des empfangenden Fensters übergibt.



Die MFC-Methode `UpdateWindow()` macht beispielsweise nichts anderes, als mittels `SendMessage()` eine `WM_PAINT`-Nachricht zu versenden.

14.2 Von der API zur MFC

Was uns die API-Programmierung lehrt, ist die eigentliche Funktionsweise eines Windows-Programms. Die API-Programmierung ist viel näher an Windows dran als die MFC. Dies erschwert die Programmierung, verschafft dem Programmierer aber auch Einblicke und Einsichten, die ihm verschlossen bleiben, wenn er sich nur mit der MFC befaßt. Doch da wir nicht mit der API, sondern mit der MFC programmieren wollen, helfen uns alle Erkenntnisse über die API-Programmierung wenig, wenn wir diese Einsichten nicht wieder auf die MFC-Programmierung zurückprojizieren können.

Schauen wir uns also an, wie die Konzepte der API-Programmierung in der MFC gekapselt sind.

14.2.1 WinMain() und Anwendungsobjekt

MFC-Programme definieren keine `WinMain()`-Eintrittsfunktion. Statt dessen wird eine globale Instanz der Anwendungsklasse definiert. Die `WinMain()`-Eintrittsfunktion ist im Code der MFC versteckt (Modul *AppModul.cpp*) und wird bei Verwendung der MFC-Bibliotheken automatisch mit eingebunden. Die `WinMain()`-Funktion ruft die globale MFC-Funktion `AfxWinMain()` auf, und diese greift auf das Anwendungsobjekt des Programms zu und ruft des-

sen `InitInstance()`-Methode auf, in der schließlich alle wichtigen Initialisierungsarbeiten von der Auswertung der Kommandozeilenargumente bis zur Erzeugung des Hauptfensters vorgenommen werden.

14.2.2 Message Loop und Run()

Die Message Loop der API-Programme ist in der `Run()`-Methode der MFC-Klasse `CWinApp` gekapselt. Aufgerufen wird die Methode in der MFC-Funktion `AfxWinMain()`.

14.2.3 Erzeugung der Fenster

In API-Programmen erzeugt man Fenster durch Einrichtung und Registrierung einer Fensterklasse und der nachfolgenden Erzeugung eines Fensters von dieser Fensterklasse.

In der MFC erzeugt man zuerst ein Objekt einer MFC-Fensterklasse und ruft dann die Methode `Create()` auf, die das eigentliche Fenster erzeugt und mit dem Objekt verbindet. Der Methode `Create()` kann man wie der API-Funktion `CreateWindow()` eine eigene Fensterklasse übergeben, doch ist es nicht nötig, in der MFC-Klasse `CWnd` automatisch eine passende Fensterklasse einzurichten. Übergibt man der `Create()`-Methode als erstes Argument den Wert `NULL`, wird die Standardfensterklasse von `CWnd` verwendet.

In vielen Fällen braucht man für das Hauptfenster der Anwendung die Methode `Create()` nicht selbst aufzurufen. Man kann die Methode `LoadFrame()` verwenden oder – im Falle von Doc/View-Anwendungen – die ganze Arbeit vom Konstruktor der Dokumentvorlagenklasse erledigen lassen.

Um bei allem Komfort nicht die Kontrolle über die Erzeugung und Konfiguration der Fenster aus der Hand zu geben, kann man die Methoden `PreCreateWindow()` und `OnCreate()` überschreiben (siehe Kapitel 6).

Der Parameter `nCmdShow` der `WinMain()`-Funktion entspricht der MFC-Variablen `m_nCmdShow`, die in der Methode `InitInstance()` an die Methode `ShowWindow()` übergeben werden kann.

14.2.4 Fensterfunktion und Antworttabellen

Nachrichten werden in API-Programmen in Fensterfunktionen behandelt. Fensterfunktionen enthalten eine `switch`-Anweisung, in der für alle zu behandelnden Nachrichten `case`-Blöcke vorgesehen werden. In dem `case`-Block steht dann der Code, der als Antwort auf die Nachricht ausgeführt

wird. Ist dieser Code umfangreicher, kann man ihn auch in eine eigene Funktion auslagern, die dann im `case`-Block aufgerufen wird. Würde man dies für alle behandelten Nachrichten machen, wäre die Fensterfunktion genauso aufgebaut wie eine Antworttabelle.

In MFC-Programmen werden Nachrichten an Antworttabellen weitergeleitet. In der Antworttabelle ist festgehalten, für welche Nachricht welche Methode zur Beantwortung aufgerufen werden soll.

Die `ON_`-Makros der Antworttabelle erfüllen dabei die gleiche Aufgabe wie die `case`-Blöcke der Fensterfunktion:

<pre> LRESULT CALLBACK WndProc(HWND hWnd, UINT uiMessage, WPARAM wParam, LPARAM lParam) { switch(uiMessage) { case WM_LBUTTONDOWN: OnMausklick(); return 0; case WM_SIZE: OnNeueGroesse(0); return 0; case WM_COMMAND: switch(LOWORD (wParam)) { case ID_FILE_PRINT : OnDrucken(); return 0; } } ... } </pre>	<pre> BEGIN_MESSAGE_MAP(CDemoView, CView) ON_WM_LBUTTONDOWN() ON_WM_SIZE() ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint) ... END_MESSAGE_MAP() </pre>

Es gibt aber auch Unterschiede zwischen Fensterfunktionen und Antworttabellen.

Namen von Behandlungsmethoden

In der Fensterfunktion steht es Ihnen frei, die Nachrichten in den `case`-Blöcken direkt zu bearbeiten oder den gesamten Code in einer Funktion zu kapseln. Letzteres käme dann der Antworttabelle schon sehr nahe, denn in der Antworttabelle kann man keinen Code verwenden, sondern nur eine Windows-Nachricht mit einer Behandlungsmethode verbinden.

In der Fensterfunktion steht es dem Programmierer aber gänzlich frei, wie er seine Behandlungsmethode nennen will. Nicht so in den Antworttabellen. Hier müssen Sie sich spezieller Makros bedienen. Wichtig ist, dabei zu be-

achten, daß für jede Nachricht der Name des Makros und der zugehörigen Behandlungsmethode schon festgelegt sind.

Daß die Namen der Behandlungsmethoden vorgegeben sind, liegt daran, daß die Behandlungsmethoden selbst schon in den Basisklassen definiert und von uns lediglich überschrieben werden. Um nicht alle Behandlungsmethoden als `virtual` deklarieren zu müssen, implementieren die Antworttabellen eine eigene Suchfunktion, die sicherstellt, daß für abgeleitete Objekte immer die überschriebene Version der Behandlungsmethode aufgerufen wird (siehe Online-Hilfe Technischer Hinweis TN006).



Um aus dem Namen einer Nachricht (beispielsweise `WM_LBUTTONDOWN`) den Namen des Makros für die Behandlungsmethode abzuleiten, stellt man dem Namen der Nachricht einfach das Präfix `ON_` voran (`ON_WM_LBUTTONDOWN()`). Der Name der Behandlungsmethode ergibt sich dann aus dem Makronamen, indem man sämtliche Unterstriche und das Präfix `WM` herausschneidet und nur noch die Anfangsbuchstaben der Silben groß schreibt (`OnLButtonDown()`). Es fehlt Ihnen dann aber noch die Signatur der Funktion. Diese können Sie der Online-Hilfe entnehmen, indem Sie einfach im Index nach dem On-Makro der Nachricht suchen.

Wenn Sie zur Einrichtung Ihrer Behandlungsmethoden den Klassen-Assistenten nutzen, brauchen Sie sich um die Namensgebung überhaupt nicht zu kümmern (siehe Kapitel 7).



Empfang von Nachrichten

Wie die Nachrichten an die Fensterfunktionen gesendet werden, haben Sie im ersten Abschnitt dieses Kapitels gelesen. Wie steht es aber mit den Antworttabellen?

Zuerst einmal gilt, daß auch in MFC-Programmen Nachrichten von Windows an Fensterfunktionen geschickt werden. In MFC-Programmen geht die Reise von hier aus aber noch weiter.

- ✗ Handelt es sich um eine WM-Nachricht, werden spezielle Methoden der MFC aufgerufen, die nachschauen, ob in dem Fenster, an das die Nachricht gerichtet ist, eine Antworttabelle mit einem passenden Eintrag definiert ist. Wenn ja, wird die zugehörige Behandlungsmethode aufgerufen. Dieser Mechanismus trägt auch der Tatsache Rechnung, daß für ein

Fenster mehrere Antworttabellen (aufgeteilt auf die Klasse des Fensterobjekts und deren Basisklassen) vorliegen können.

- ✗ Für `COMMAND`-Nachrichten (Menübefehle, Tastaturkürzel) kann die Nachricht sogar innerhalb der Klassen des Anwendungsgerüsts weitergereicht werden – um letztendlich beispielsweise auch von dem Dokumentobjekt bearbeitet werden zu können.

14.2.5 Gerätekontexte

Vielleicht ist Ihnen aufgefallen, daß wir in dem API-Programm aus dem ersten Abschnitt den Gerätekontext, den uns die API-Funktion `GetDC()` lieferte, durch einen Aufruf von `ReleaseDC()` selbst gelöscht haben. Dies liegt daran, daß der Programmierer grundsätzlich dafür Sorge tragen muß, daß Gerätekontexte, die er erzeugt, auch wieder gelöscht werden.

Wenn Sie sich aber an unsere MFC-Programme zurückerinnern, werden Sie vielleicht mit Schrecken feststellen, daß wir dies bisher offensichtlich nie getan haben. Nun, das stimmt nicht ganz. Wo wir uns eigene Gerätekontexte besorgen mußten, beispielsweise in den Behandlungsmethoden zu den Mausclicks, haben wir statt der API-Funktion `GetDC()` stets eine der MFC-Gerätekontextklassen instantiiert:

```
void CKreiseView::OnLButtonDown(UINT nFlags, CPoint point)
{
    ...
    CClientDC dc(this);
    int b = (int) (100.0 * rand() / RAND_MAX);
    dc.Ellipse(point.x-b, point.y-b, point.x+b, point.y+b);
    CView::OnLButtonDown(nFlags, point);
}
```

Bei dieser Vorgehensweise ist der Gerätekontext in einem lokalen Objekt gekapselt. Wird die `OnLButtonDown()`-Methode beendet, wird das lokale Objekt (hier `dc`) aufgelöst, und der Destruktor der Klasse sorgt dafür, daß der gekapselte Gerätekontext korrekt gelöscht wird.

14.3 Abschlußbemerkung

Ich denke, wenn Sie den obigen Ausführungen gefolgt sind und alles soweit verstanden haben, dürften Sie mittlerweile weit mehr von der MFC-Programmierung und Windows verstehen, als dies der Durchschnitt der Windows-Programmierer von sich behaupten kann. Viele Bücher für Einsteiger verschweigen die Grundlagen von Windows und der Windows-API-Pro-

grammierung. Das ist keineswegs ein Manko, denn Aufgabe eines solchen Buches ist es zunächst, den Leser vorsichtig und mit Spaß an der Programmierung an das Thema heranzuführen; und nicht, ihn durch schwer zu verstehende theoretische Ausführungen, deren praktischer Nutzen gar noch zweifelhaft ist, abzuschrecken. Daß ich in diesem Buch so viel Wert auf die Vermittlung von Hintergrundwissen lege, liegt daran, daß es meines Erachtens viel zu wenig Bücher gibt, die dieses Wissen anschaulich vermitteln. Die Einsteigertitel halten das Thema für zu schwierig, die Fortgeschrittenenbücher setzen das Hintergrundwissen voraus – auf der Strecke bleibt jedes Mal der Leser.

14.4 Zusammenfassung

Windows-API-Programme

- ✗ beginnen mit der Eintrittsfunktion `WinMain()`
- ✗ registrieren für das Hauptfenster eine eigene Fensterklasse, auf deren Grundlage das Hauptfenster dann erzeugt wird
- ✗ implementieren eine Message Loop zum Eintritt in die Nachrichtenverarbeitung
- ✗ behandeln Nachrichten in den Fensterfunktionen der Fenster der Anwendung

Windows-MFC-Programme

- ✗ definieren statt der `WinMain()`-Funktion ein Anwendungsobjekt
- ✗ erzeugen ihre Hauptfenster in der Methode `CWinApp::InitInstance` als Objekte ihrer Rahmenfensterklasse (meist auf der Grundlage der von `CWnd` vorgegebenen Fensterklasse)
- ✗ haben ihre Message Loop in der `CWinApp`-Methode `Run()` implementiert, die vom internen Startcode der MFC aufgerufen wird
- ✗ richten die Methoden zur Nachrichtenbehandlung mit Hilfe von Antworttabellen ein

14.5 Fragen

1. Was ist die Message Loop?
2. Werden alle Nachrichten über die Message Loop geleitet?
3. Wo ist festgehalten, welche Fensterfunktion zu einem Fenster gehört?

4. Wer ruft die Fensterfunktion auf?
5. Wie werden Windows-Anwendungen beendet?
6. Wie heißen das Antworttabellenmakro und die Behandlungsmethode zu der Windows-Nachricht WM_SIZE?
7. Kann man in MFC-Anwendungen Gerätekontexte durch Aufruf der API-Funktion GetDC() erzeugen?

14.6 Aufgaben

1. Informieren Sie sich in der Online-Hilfe über die Nachricht WM_SIZE. Wann wird sie ausgelöst?
2. Schreiben Sie ein API-Programm mit einem Schalter als Hauptfenster.

14.7 Lösung zu Aufgabe 2

```
// Schalter.cpp
#include "stdafx.h"
int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow )
{
    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))
    {
        if (msg.message == WM_LBUTTONDOWN)
        {
            DestroyWindow(hwnd);
            PostQuitMessage(0);
        }
        DispatchMessage(&msg);
    }
    return msg.wParam;
}
```

Das Doc/View- Modell

Wie man mit der Dokument/Ansicht-Architektur programmiert, haben Sie bereits in den Kapiteln des zweiten Teils dieses Buches gesehen. In diesem Kapitel möchte ich zur besseren Referenz einige wichtige Punkte zum Doc/View-Modell zusammenfassen.

Dokument, Ansicht und Dokumentvorlage

Beim Doc/View-Modell handelt es sich nicht um ein konkretes Element einer Windows-Anwendung, sondern lediglich um einen bestimmten Programmierstil (der allerdings in der MFC-Programmierung eine große Rolle spielt). Die Idee dahinter ist recht einfach:

Sie geht davon aus, daß die Trennung von Datenverwaltung und Datenanzeige konzeptionelle Vorteile bietet und Programme übersichtlicher und leichter wartbar macht. Auf der einen Seite haben wir also die eigentlichen Rohdaten, repräsentiert durch die Dokumentklasse, die stets von der Basisklasse `CDocument` abgeleitet ist. Auf der anderen Seite haben wir die Ansichtsklasse, die für die Anzeige der Daten verantwortlich ist und auf `CView` basiert. Der große Vorteil besteht nun darin, daß Sie für die gleichen Daten (also ein `CDocument`-Objekt) mehrere Ansichten einrichten können, in denen die Daten auf jeweils unterschiedliche Weise angezeigt werden können.

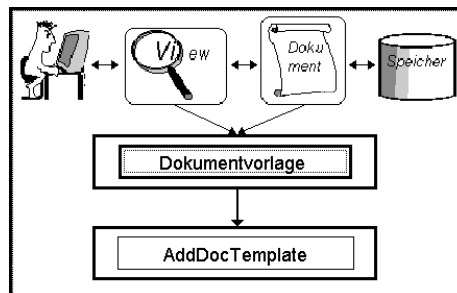




Nehmen Sie beispielsweise die Daten einer Tabellenkalkulation: Während eine Ansicht dafür zuständig ist, dem Anwender die Daten in Form einer Tabelle anzuzeigen, erzeugt eine andere Ansicht ein Balkendiagramm, eine dritte ein Tortendiagramm und eine vierte Ansicht eine statistische Auswertung der Daten.

Das übersichtliche Konzept vereinfacht auch die nachträgliche Implementierung weiterer Ansichten. Der Nachteil, oder besser gesagt die Schwierigkeit, die sich dem Programmierer bei der Implementierung stellt, liegt vor allem in der korrekten Kommunikation zwischen Dokumentklasse und Ansichtsklassen.

Bild 15.1:
Daten in der
Dokument/
Ansicht-
Architektur



Ansichten und Dokumente ergänzen sich insofern, als die Ansicht die Schnittstelle zum Benutzer und das Dokument die Schnittstelle zum Speichermedium bilden. Dokument und Ansicht werden daher immer paarweise zugeordnet. Haben Sie also eine Dokumentklasse und eine Ansichtsklasse, so verbinden Sie diese in einer Dokumentvorlage, damit beide zusammenarbeiten können.

Dokumentvorlagen

Zur Einrichtung Ihrer Doc/View-Architektur erzeugen Sie also zuerst für jeden Dokumenttyp eine Dokumentvorlage.

- ✗ Um die Daten eines Dokumenttyps (beispielsweise `.cpp`-Dateien) anzuzeigen, benötigen Sie genau eine Dokumentvorlage (mit einem Dokument und einer Ansicht).
- ✗ Um die Daten eines Dokumenttyps (beispielsweise `.cpp`-Dateien) auf verschiedene Weise anzuzeigen (beispielsweise als ASCII-Text und in Hex-Format), benötigen Sie ebenfalls nur eine Dokumentvorlage (mit einem Dokument und mehreren Ansichten zu dem Dokument).

- ✗ Um die Daten mehrerer Dokumenttypen (beispielsweise *.cpp*- und *.rc*-Dateien) anzuzeigen, benötigen Sie für jeden Dokumenttyp eine eigene Dokumentvorlage (nur in MDI-Anwendungen möglich).
- ✗ Um mehrere Ansichten gleichzeitig in einer Anwendung anzuzeigen (unabhängig davon, ob es die Ansichten eines Dokuments oder mehrerer Dokumente unterschiedlichen Typs sind), benötigen Sie eine MDI-Anwendung statt einer SDI-Anwendung.

Mit den Dokumentvorlagen verfahren Sie auf zweierlei Art:

1. Sie erzeugen sie, und
2. Sie registrieren sie.

Für SDI-Anwendungen erzeugen Sie die Dokumentvorlagen als Instanzen der Klasse `CSingleDocTemplate`, für MDI-Anwendungen als Instanzen von `CMultiDocTemplate`. (Beide Klassen gehen auf die abstrakte Basisklasse `CDocTemplate` zurück.) Dem Konstruktor der `CDocTemplate`-Klasse übergeben Sie:

- ✗ Eine Ressourcen-ID (für die Ressourcen der Dokumentvorlage, üblicherweise Menü, Symbol, Tastaturkürzel und Stringtabelle, die die Extension und Beschreibung des Dokumenttyps enthält)
- ✗ Die Dokumentklasse
- ✗ Die Rahmenklasse
- ✗ Die Ansichtsklasse

Zur Registrierung übergeben Sie die `CDocTemplate`-Instanzen an die `CWinApp`-Methode `AddDocTemplate()`.

Dokumentklasse

Die Dokumentklasse dient zur Verwaltung und Speicherung der Daten.

- ✗ Basisklasse für Ihre Dokumentklasse ist `CDocument`.
- ✗ Ein Dokument kann über mehrere Ansichten verfügen.
- ✗ Deklarieren Sie in Ihrer Dokumentklasse Elementvariablen für die Verwaltung der Daten des Dokuments (beispielsweise den anzuzeigenden Text).
- ✗ Implementieren Sie nach Bedarf Methoden, die die Ansichten zum Bearbeiten der Daten des Dokuments aufrufen können.

- ✗ Die Ansichten rufen die CView-Methode `GetDocument()`, wenn Sie auf die Daten und Methoden des Dokuments zugreifen wollen. `GetDocument()` liefert einen Zeiger auf das zur Ansicht gehörende Dokument.
- ✗ Zum Laden und Speichern überschreiben Sie die CObject-Methode `Serialize()`.
- ✗ Üblicherweise werden auch die Methoden `OnNewDocument()` und `OnOpenDocument()` zur Initialisierung der Daten eines Dokuments und die Methode `DeleteContents()` zum Löschen der Daten überschrieben.
- ✗ Sobald sich die Dokumentdaten ändern, sollten Sie die Methode `SetModifiedFlag()` mit dem Argument `true` aufrufen. Der consequente Einsatz dieser Methode gewährleistet, daß der Anwendungsrahmen den Anwender benachrichtigt, bevor ein verändertes, nicht gespeichertes Dokument zerstört wird. Der Überarbeitungsstatus des Dokuments kann mit der Methode `IsModified()` ermittelt werden.

Die Kommunikation mit den Ansichten und dem Anwendungsgestüt wird durch mehrere Konzepte unterstützt:

- ✗ Jedes Dokument verwaltet eine Liste seiner Ansichten. Sie können diese Liste über die Methoden `AddView()` und `RemoveView()` manipulieren und mit den Methoden `GetFirstViewPosition()` und `GetNextView()` durchlaufen.
- ✗ Um die Ansichten über Änderungen in den Daten zu informieren (wenn zum Beispiel die Daten aus der Datei wiederhergestellt wurden oder Änderungen über eine Ansicht eingegeben wurden und diese an die anderen geöffneten Ansichten weitergegeben werden sollen), ruft man die Methode `UpdateAllView()` auf (diese Methode wird beispielsweise von einer Ansicht aufgerufen, wenn über diese Ansicht die Daten des Dokuments geändert wurden).
- ✗ Die Methode `GetDocTemplate()` liefert einen Zeiger auf die zugehörige Dokumentvorlage.

Ansichtsfensterklasse

Ansichten dienen zur Anzeige der Daten und als Schnittstelle zum Anwender, über die er Eingaben vornehmen kann.

- ✗ Basisklasse ist CView oder eine der abgeleiteten Klassen (CCtrlView, CDaoRecordView, CEditView, CFormView, CListView, CRecordView, CRichEditView, CScrollView oder CTreeView, die teils auch direkt instantiiert werden können).

- ✗ Eine Ansicht ist ein untergeordnetes Fenster eines Rahmenfensters (CFrameWnd oder CMDIChildWnd).
- ✗ Eine Ansicht ist immer nur mit einem Dokument verbunden.
- ✗ Überschreiben Sie die Methode OnDraw(), um den Fensterinhalt aufzubauen (für Ausgabe auf Bildschirm oder Drucker).
- ✗ Ansichten rufen die Methode GetDocument() auf, wenn Sie auf die Daten und Methoden des Dokuments zugreifen wollen. GetDocument() liefert einen Zeiger auf das zur Ansicht gehörende Dokument.
- ✗ Um das Dokument und alle anderen Ansichten des Dokuments über Änderungen in der aktuellen Ansicht zu unterrichten, ruft die aktuelle Ansicht die CDocument()-Methode UpdateAllViews() auf.

Die Kommunikation mit dem Dokument und dem Anwendungsgestüst wird durch mehrere Konzepte unterstützt:

- ✗ Um das Dokument und die anderen Ansichten über Änderungen in den Daten zu informieren (wenn zum Beispiel die Daten aus der Datei wiederhergestellt wurden oder Änderungen über eine Ansicht eingegeben wurden und diese an die anderen geöffneten Ansichten weitergegeben werden sollen), ruft eine Ansicht die CDocument()-Methode UpdateAllViews() auf.
- ✗ Die Methode GetDocument() liefert einen Zeiger auf das zugehörige Dokument (beispielsweise um die Eingaben durch den Anwender im Dokument abzuspeichern).
- ✗ Die Methode GetParentFrame() liefert einen Zeiger auf das Rahmenfenster der Ansicht.

Rahmenfenster und Anwendung

- ✗ Die Anwendung speichert einen Zeiger auf das Rahmenfenster der Anwendung in der Elementvariablen m_pMainWnd.
- ✗ Das Rahmenfenster kann sich mit Hilfe der Methode GetActiveView() die aktuelle Ansicht zurückliefern lassen.
- ✗ MDI-Rahmenfenster rufen MDIGetActive() auf, um das aktive MDI-Kindfenster zu ermitteln.
- ✗ Das Rahmenfenster kann sich mit Hilfe der Methode GetActiveDocument() das Dokument zur aktuellen Ansicht zurückliefern lassen.

Das Doc/View-Gerüst anpassen

Die saubere Trennung von Datenverwaltung und Datenanzeige im Doc/View-Modell dient nicht nur der besseren Übersichtlichkeit, sie hat auch handfeste Vorteile für die Programmierung – beispielsweise, wenn man ein und dieselben Daten auf unterschiedliche Weise anzeigen möchte.

Sie lernen in diesem Kapitel:

- ✕ Wie man eine SDI-Anwendung mit zwei Ansichten implementiert

16.1 Ein Dokument – zwei Ansichten

Ein Dokument – mehrere Ansichten, das kennen Sie wahrscheinlich von Word (Ansichten Normal, Gliederung und Layout) oder von Ihrem HTML-Editor, der Ihnen Ihre Web-Seite als reinen ASCII-Text oder als formatiertes Web-Dokument anzeigt. Ganz so hoch hinaus wollen wir noch nicht. Wir werden uns mit ganz einfachen Daten und wenig spektakulären Ansichten zufrieden geben. Doch den Mechanismus, wie man bei Ausführung des Programms zwischen zwei Ansichten hin- und herwechseln kann, den wollen wir uns schon noch genauer ansehen.

Bevor es losgeht, erstellen Sie bitte ein ganz einfaches SDI-Programm, das in seinem Dokument die Daten für die linke obere Ecke und die rechte untere Ecke eines Rechtecks verwahrt und in seiner `OnDraw()`-Methode dieses Rechteck auf den Bildschirm malt.



Übung 16-1: Das Ausgangsprojekt

1. Legen Sie eine neue SDI-Anwendung mit Doc/View, aber ohne große Dekorationen an. Nennen Sie das Projekt beispielsweise »Ansichten«.

Drücken Sie nicht zu schnell auf den Schalter FERTIGSTELLEN. Auf der letzten Seite des Anwendungs-Assistenten im Schritt 6 markieren Sie bitte die Ansichtsklasse CANSICHTENVIEW und benennen diese im Feld KLASSENNAME in CAnsichtenView1 um.

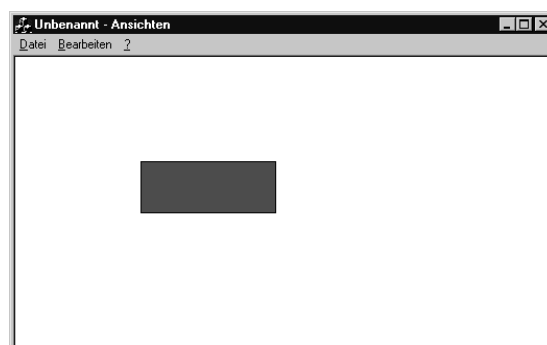
Alles weitere kennen Sie aus den Übungen aus Kapitel 5.

2. Legen Sie in der Dokumentklasse Elementvariablen an, in denen Sie die Koordinaten für die linke obere und die rechte untere Ecke des Rechtecks speichern können.
3. Initialisieren Sie die Variablen im Konstruktor der Dokumentklasse oder in der OnNewDocument()-Methode des Dokuments.
4. Zeichnen Sie das Rechteck in der OnDraw()-Methode des Ansichtsfensters CAnsichtenView1.

```
void CAnsichtenView1::OnDraw(CDC* pDC)
{
    CAnsichtenDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CBrush roterPinzel(RGB(255, 0, 0));
    CBrush *p_alterPinzel = pDC->SelectObject(&roterPinzel);
    pDC->Rectangle(pDoc->x1, pDoc->y1, pDoc->x2, pDoc->y2);
    pDC->SelectObject(p_alterPinzel);
}
```

5. Testen Sie das Programm (**Strg** + **F5**).

Bild 16.1:
Die erste
Ansicht des
Programms



Erstellen und Einbinden einer zweiten Ansicht

Der erste Schritt zur Erstellung einer zweiten Ansicht besteht natürlich darin, eine zweite Ansichtsklasse zu definieren. Eine zweite Ansicht als Objekt unserer bereits bestehenden Ansichtsklasse `CAnsichtenView1` zu instantiieren, reicht natürlich nicht, da dann ja beide Ansichten über die gleiche `OnDraw()`-Methode verfügen und die Daten aus dem Dokument in gleicher Weise anzeigen würden.

Also muß eine zweite Ansichtsklasse namens `CAnsichtenView2` her. Dazu kann man entweder die Klassendeklaration und die Definitionen der ersten Ansichtsklasse kopieren (nicht vergessen, in der Kopie alle Vorkommen von `CAnsichtenView1` in `CAnsichtenView2` zu ändern), oder man legt die neue Klasse mit dem Klassen-Assistenten an.

Doch damit ist es noch lange nicht getan. Um dem Programm eine zweite Ansicht hinzuzufügen, müssen wir

1. ein Objekt der neuen Ansichtsklasse instantiieren,
2. die `Create()`-Methode des Objekts aufrufen,
3. die Ansicht mit dem Dokument verbinden.

Zu 1:

Um selbst ein Objekt der Klasse instantiieren zu können, benötigen wir einen `public`-Konstruktor. Der Konstruktor in den von den Assistenten eingerichteten Klassen ist als `protected` deklariert und kann daher von uns nicht direkt zur Erzeugung eines Ansichtsklassenobjekts aufgerufen werden. (Der `protected`-Konstruktor wird intern bei der Einrichtung der Dokumentvorlage von der `CreateObject()`-Methode der `CRuntimeClass` aufgerufen.)

Zu 2:

Nach der Instanzbildung muß die `Create()`-Methode aufgerufen werden. Dabei stellt sich die Frage, wo man die neue Ansicht erzeugen soll. Eine Möglichkeit wäre, die Ansicht erst dann zu erzeugen, wenn sie benötigt wird, eine andere Möglichkeit wäre, die zweite Ansicht direkt mit der ersten erzeugen zu lassen. Dazu überschreibt man mit Hilfe des Klassen-Assistenten die `CMainFrame`-Methode `OnCreateClient()`.

Zu 3:

Die Methode `OnCreateClient()` ist auch deshalb ein geeigneter Ort für die Erzeugung der zweiten Ansicht, weil sie uns das `CCreateContext`-Objekt für die Einrichtung der ersten Ansicht liefert:

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                               CCreateContext* pContext)
{
```

In dem `CCreateContext`-Objekt ist nämlich neben anderen Informationen auch festgehalten, mit welchem Dokument die Ansicht zu verbinden ist. Indem wir beim Aufruf der `Create()`-Methode unserer zweiten Methode das `pContext`-Objekt übergeben, stellen wir also gleich die Verbindung zum Dokument her (ansonsten würde man einen eigenen `CCreateContext`-Kontext einrichten oder die `CDocument`-Methode `AddView()` aufrufen müssen).

```
pView = (CView*) new CAnsichtenView2;
pView->Create(NULL, NULL, WS_BORDER,
              rectDefault, this, VIEW2, pContext);
```

Identifizieren der Ansichten

Um die beiden Ansichten, die untergeordnete Fenster des Rahmenfensters sind, identifizieren zu können, definiert man zwei IDs (beispielsweise `VIEW1` und `VIEW2`) und weist diese den Ansichtsfensterobjekten zu: entweder direkt über die `Create()`-Methode oder mit Hilfe der `CWnd`-Methode `GetDlgItem()`:

```
CWnd* GetDlgItem( int nID ) const;
```



Stören Sie sich nicht an dem Namen »`GetDlgItem`«. Der Name scheint anzudeuten, daß die Methode nur für Steuerelemente in einem Dialog geeignet ist, doch sie wird ebenso für Steuerelemente in Fenstern wie auch für die untergeordneten Fenster eines Rahmenfensters verwendet.

Wechsel zwischen den Ansichten

Will man zwischen zwei Ansichten wechseln, muß man erst einmal wissen, welche Ansicht aktiviert werden soll. Wenn man wie im nachfolgenden Beispiel für den Aufruf der einzelnen Ansichten jeweils eigene Menübefehle einrichtet, weiß man natürlich als Programmierer, daß in der Behandlungsmethode zum Befehl `ANSICHTEN/ANSICHT1` die Ansicht der Klasse `CAnsichtenView1` zu aktivieren ist.

Jetzt muß man diese Ansicht aber noch finden. Für solche Aufgaben verwahrt das Dokument eine Liste aller Ansichten zu dem Dokument. Mit Hilfe der Methode `GetFirstViewPosition()` lassen wir uns einen Zeiger auf den Anfang dieser Liste (vor die erste Ansicht) zurückliefern. Danach kann man die Liste mit Hilfe der Methode `GetNextView()` durchgehen und nachprüfen, ob man auf eine Ansicht der Klasse `CAnsichtenView1` trifft.

```
POSITION pos = pDoc->GetFirstViewPosition();
while (pos != NULL)
{
    pView = pDoc->GetNextView(pos);
    if (pView->IsKindOf(RUNTIME_CLASS(CAnsichtenView1)))
    {
        // Ansicht aktivieren
    }
}
```

Wurde die Ansicht gefunden, muß sie aktiviert und angezeigt werden. Dazu muß man:

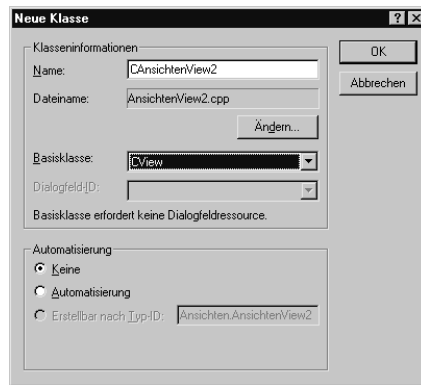
- ✗ `SetActiveView()` aufrufen, um die Ansicht zur aktiven Ansicht zu machen,
- ✗ die neue Ansicht anzeigen und die alte Ansicht verbergen,
- ✗ der neuen Ansicht die ID `AFX_IDW_PANE_FIRST` zuweisen, damit die nachfolgend aufzurufende Methode `RecalcLayout()` der Ansicht den Client-Bereich des Rahmenfensters zuweist.

```
SetActiveView(pView);
pView->ShowWindow(SW_SHOWNORMAL);
pAktView->ShowWindow(SW_HIDE);
pView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
pAktView->SetDlgCtrlID(VIEW2);
```

Übung 16-2: Eine zweite Ansicht einrichten

In dieser Übung werden wir für das in Übung 16-1 erstellte Projekt eine zweite Ansicht einrichten, das die Rechteckskoordinaten aus dem Dokument als umschließendes Rechteck interpretiert und in diesem Rechteck eine Ellipse einzeichnet.

Bild 16.2:
In einen
Gerätekontext
für ein Fenster
zeichnen



1. Rufen Sie den Klassen-Assistenten auf, und drücken Sie auf den Schalter KLASSE HINZUFÜGEN/NEU, um eine zweite Ansichtsklasse anzulegen.
2. Implementieren Sie die `OnDraw()`-Methode für die neue Ansichtsklasse.

Beachten Sie, daß der Aufruf von `GetDocument()` in der neu angelegten Ansichtsklasse lediglich einen Zeiger auf die Basisklasse `CDocument` zurückliefert. Vergessen Sie nicht, diesen in einen Zeiger auf Ihre Dokumentklasse umzuwandeln.

```
void CAnsichtenView2::OnDraw(CDC* pDC)
{
    CAnsichtenDoc* pDoc = (CAnsichtenDoc*) GetDocument();
    // ZU ERLEDIGEN: Code zum Zeichnen hier einfügen
    CBrush roterPinzel(RGB(255, 0, 0));
    CBrush *p_alterPinzel = pDC->SelectObject(&roterPinzel);
    pDC->Ellipse(pDoc->x1, pDoc->y1, pDoc->x2, pDoc->y2);
    pDC->SelectObject(p_alterPinzel);
}
```

3. Da wir in der `OnDraw()`-Methode der zweiten Ansichtsklasse auf ein Objekt der Dokumentklasse zugreifen, müssen wir die Header-Datei für die Dokumentklasse in die Quelltextdatei der Ansichtsklasse aufnehmen.

```
// AnsichtenView2.cpp: Implementierungsdatei
//
#include "stdafx.h"
#include "Ansichten.h"
#include "AnsichtenDoc.h"
#include "AnsichtenView2.h"
```

4. Zu guter Letzt deklarieren wir die Konstruktoren der Ansichtsklassen als public.

```
class CAnsichtenView2 : public CView
{
public:
    CAnsichtenView2();
    DECLARE_DYNCREATE(CAnsichtenView2)
```

Jetzt erzeugen wir die zweite Ansicht.

5. Definieren Sie in der Datei *resource.h* zwei IDs zur Unterscheidung der beiden Ansichten.

```
#define VIEW1      201
#define VIEW2      202
```

6. Rufen Sie den Klassen-Assistenten auf, und überschreiben Sie die CMainFrame-Methode OnCreateClient().

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
CCreateContext* pContext)
{
    CView *pView;
    pView = (CView*) GetDlgItem(VIEW2);
    pView = (CView*) new CAnsichtenView2;
    pView->Create(NULL, NULL, WS_BORDER,
                  rectDefault, this, VIEW2, pContext);
    return CFrameWnd::OnCreateClient(lpcs, pContext);
}
```

7. Um in der Quelltextdatei der Rahmenfensterklasse ein Ansichtsobjekt erzeugen zu können, muß noch die zugehörige Header-Datei eingebunden werden.

Nehmen Sie auch gleich die Header-Datei der zweiten Ansichtsklasse und der Dokumentklasse auf. (Achten Sie auf die Reihenfolge der #include-Dateien!)

```
// MainFrm.cpp : Implementierung der Klasse CMainFrame
//
#include "stdafx.h"
#include "Ansichten.h"
#include "AnsichtenDoc.h"
#include "AnsichtenView.h"
#include "AnsichtenView2.h"
```

So langsam kommen wir zum eigentlichen Wechseln zwischen den Ansichten.

8. Laden Sie die Menü-Ressource in den Menü-Editor, und legen Sie ein weiteres Menü ANSICHTEN mit den Menübefehlen ANSICHT1 und ANSICHT2 an.

Vergessen Sie nicht, den Menübefehlen Ressourcen-IDs zuzuweisen.

9. Richten Sie mit Hilfe des Klassen-Assistenten in der CMainFrame-Klasse Behandlungsmethoden für die neuen Menübefehle ein.
10. In der Behandlungsmethode zu dem Menübefehl ANSICHT1 aktivieren Sie die Ansicht der Ansichts-klasse CANSICHTENVIEW1.

```
void CMainFrame::OnAnsichten1()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    CView *pView, *pAktView;
    CDocument *pDoc;
    pAktView = GetActiveView();
    pDoc = pAktView->GetDocument();
    POSITION pos = pDoc->GetFirstViewPosition();
    while (pos != NULL)
    {
        pView = pDoc->GetNextView(pos);
        if (pView->IsKindOf(RUNTIME_CLASS(CAnsichtenView1)))
        {
            SetActiveView(pView);
            pView->ShowWindow(SW_SHOWNORMAL);
            pView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
            pAktView->ShowWindow(SW_HIDE);
            pAktView->SetDlgCtrlID(VIEW2);
        }
    }
    RecalcLayout();
}
```

11. Der Code für die Behandlungsmethode des Menübefehls ANSICHT2 wird analog aufgebaut (aber vergessen Sie nicht, jede 1 in eine 2 und jede 2 in eine 1 umzuwandeln).

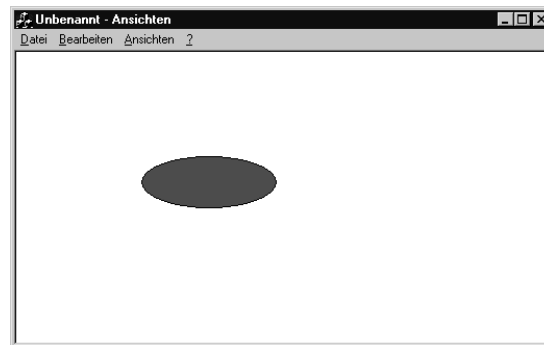


Bild 16.3:
Die zweite
Ansicht des
Programms

16.2 Zusammenfassung

Wenn man weitere Ansichten für Dokumente erzeugt, muß man darauf achten, daß die Ansichten auch mit dem Dokument verbunden werden. Dies kann man durch Übergabe eines `CCreateContext`-Objekts an die `Create()`-Methode des Ansichtsfensters erreichen oder indem man nach Erzeugung der Ansicht diese mittels der `CDocument`-Methode `AddView()` mit dem Dokument verbindet.

Um in einer SDI-Anwendung zwischen zwei Ansichten zu wechseln, müssen Sie

- ✗ `SetActiveView()` aufrufen, um die Ansicht zur aktiven Ansicht zu machen,
- ✗ die neue Ansicht anzeigen (`ShowWindow(SW_HIDE)`) und die alte Ansicht verbergen (`ShowWindow(SW_HIDE)`),
- ✗ der neuen Ansicht die ID `AFX_IDW_PANE_FIRST` zuweisen, damit die nachfolgend aufzurufende Methode `RecalcLayout()` der Ansicht den Client-Bereich des Rahmenfensters zuweist.

16.3 Fragen

1. Wie ist die `CCreateContext`-Struktur aufgebaut? Wo ist die Information über das Dokument abgelegt? (Hinweis: Schauen Sie in der Online-Hilfe nach.)
2. Wer ruft die Methode `OnCreateClient()` auf?

16.4 Aufgaben

1. Die alternative Ausgabe eines Rechtecks und einer Ellipse ist für Übungszwecke zwar ausreichend, ansonsten aber ziemlich langweilig. Implementieren Sie einen SDI-Editor mit zwei Ansichten, der in seinem Dokument eine Reihe von Koordinatenpaaren verwahrt.

Die erste Ansicht zeichnet vor einem blauen Hintergrund an jeder Koordinate einen farbigen Fleck.

Die zweite Ansicht gibt an den Koordinaten statt Scheiben Nummern aus, die die Reihenfolge der festgelegten Koordinaten anzeigen (ähnlich der Tabulatorreihenfolge der Steuerelemente im Dialogeditor).

2. Wenn Sie Lust haben, erweitern Sie die SDI-Anwendung aus Aufgabe 1 noch um eine dritte Ansicht, in der Sie die Koordinaten als Zahlenkolonne ausgeben.

Programme ohne Doc/View

Die Programmierung nach dem Doc/View-Modell ist unter MFC-Programmierern recht weit verbreitet (was wohl vor allem daran liegt, daß der Anwendungs-Assistent früher nur Programme mit Doc/View erzeugen konnte), aber sie ist kein unbedingtes Muß. Wir haben uns daher nebenbei in den Aufgaben zu den ersten Kapiteln des zweiten Teils dieses Buches immer auch mit dem Aufbau eigener Anwendungsgerüste ohne Doc/View beschäftigt. Dies sollte Sie durchaus in die Lage versetzen, in nahezu allen Belangen auch ohne Doc/View zu programmieren – zumal viele Aufgaben, wie Textverarbeitung, Grafikausgaben, Anzeige von Dialogfeldern, weitgehend bis vollständig unabhängig davon sind, ob man mit Doc/View arbeitet oder nicht.

Ab der Version 6.0 des Visual C++-Compilers kann der MFC-Anwendungs-Assistent auch Anwendungsgerüste ohne Doc/View einrichten. Hierzu ein paar kurze Anmerkungen.

Grundsätzlich nichts Neues

Grundsätzlich gilt, daß Sie – wenn Sie den Aufbau des Doc/View-Anwendungsgerüsts verstanden haben – sich auch ohne große Probleme in dem Anwendungsgerüst zurechtfinden werden, das der Anwendungs-Assistent für Programme ohne Doc/View erstellt.

Unterstützung für Befehlsweiterleitung u.a.

Im Unterschied zu unseren selbst erstellten Anwendungsgerüsten, unterstützt das vom Anwendungs-Assistenten eingerichtete Anwendungsgerüst



die Weiterleitung von Befehlsnachrichten (Menübefehle, Tastaturkürzel) innerhalb der Klassen des Anwendungsgerüsts (Rahmenfenster, Client-Fenster, Anwendung).

Der Anwendungs-Assistent implementiert daher für das Rahmenfenster die Methode `OnCmdMsg()`, die ihrerseits wieder die Methode `OnCmdMsg()` des Client-Fensters aufruft.

Des weiteren ruft der Assistent die Methode `LoadFrame()` zur Erzeugung des Rahmenfensters auf.

Diese Vorgehensweise kann aber zu Problemen führen.

Probleme mit dem Anwendungsgerüst

Die im vorangehenden Punkt beschriebenen Erweiterungen des Anwendungsgerüsts setzen voraus, daß ein Client-Fenster vorhanden ist. Wenn Sie aber ein Rahmenfenster ohne Dekorationen erstellen, legt der Anwendungs-Assistent keine `OnCreate()`-Methode an und versäumt es auch, ein Client-Fenster zu erzeugen (jedenfalls war dies bei dem mir vorliegenden ersten Release von Visual C++ 6.0 der Fall). Da der restliche Code aber auf ein Client-Fenster angewiesen ist, führt dies spätestens dann zum Programmabsturz, wenn der Code irgendwo versucht, auf das nicht vorhandene Client-Fenster zuzugreifen.

In diesem Fall müssen Sie entweder das Anwendungsgerüst so umprogrammieren, daß es ohne Client-Fenster auskommt (dann hätten Sie aber besser gleich mit einem selbst aufgebauten Anwendungsgerüst angefangen, siehe Aufgaben aus Kapitel 5), oder Sie sorgen selbst dafür, daß das Client-Fenster erzeugt wird (siehe Übung 17-1).



Wenn Sie mit dem Anwendungs-Assistenten ein Anwendungsgerüst ohne Doc/View-Unterstützung erstellen, sollten Sie nachprüfen, ob der Assistent Code zur Erstellung des Client-Fensters erzeugt hat.

Übung 17-1: Anwendungsgerüst ohne Doc/View und ohne Dekorationen anlegen

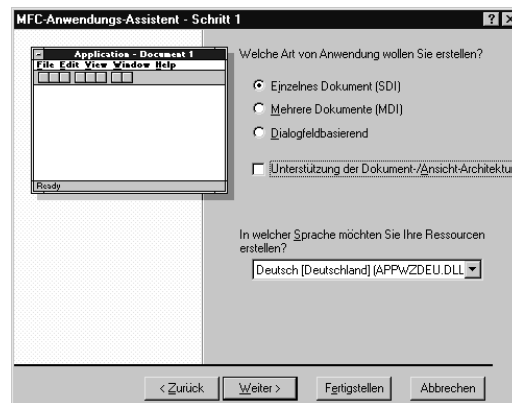


Bild 17.1:
SDI-Anwen-
dung ohne
Doc/View

1. Legen Sie eine neue SDI-Anwendung ohne Doc/View und ohne große Dekorationen an. Nennen Sie das Projekt beispielsweise »Clientfenster«.
2. Richten Sie im Klassen-Assistenten in der Klasse CMainFrame eine Behandlungsmethode für die Windows-Nachricht WM_CREATE ein.

In dieser erzeugen Sie das Client-Fenster. Übergeben Sie der Methode Create() als fünftes Argument den this-Zeiger auf das Rahmenfenster.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Speziellen Erstellungscode hier einfügen
    m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
        CRect(0, 0, 0, 0), this,
        AFX_IDW_PANE_FIRST, NULL);

    return 0;
}
```

3. Zeichnen Sie zur Kontrolle etwas in das Client-Fenster, beispielsweise:

```
void CChildView::OnPaint()
{
    CPaintDC dc(this); // Gerätekontext zum Zeichnen
```



```
// ZU ERLEDIGEN: Fügen Sie hier Ihren Code für die
// Nachrichtenbehandlung hinzu
CBrush roterPinzel(RGB(255, 0, 0));
CBrush *p_alterPinzel = dc.SelectObject(&roterPinzel);
dc.Rectangle(100, 120, 250, 200);
dc.SelectObject(p_alterPinzel);
}
```



Eine andere Möglichkeit ist, das Anwendungsgerüst vom Anwendungs-Assistenten mit einer Symbolleiste als Dekoration anlegen zu lassen und dann den Code für die Symbolleiste zu löschen.

Rückbesinnung auf die API

Von der API, der Sammlung von C-Funktionen zur direkten Windows-Programmierung, war in diesem Buch schon öfter die Rede – allerdings immer nur dann, wenn es darum ging, Hintergrundwissen zur Funktionsweise von Windows aufzuarbeiten oder zu untersuchen, welche API-Funktion hinter einer MFC-Methode steht.

Darüber hinaus kann man aus den API-Funktionen aber auch praktischen Nutzen ziehen und sie in MFC-Programme einbauen. Wie dies geht, werden wir uns in diesem Kapitel anschauen.

Sie lernen in diesem Kapitel:

- ✗ Wie man API-Funktionen in MFC-Programmen aufruft
- ✗ Wie man sich über Systemressourcen informiert

18.1 Aufruf von API-Funktionen

Warum sollten Sie überhaupt eine API-Funktion in einem MFC-Programm aufrufen? Nun, Sie haben ganz recht. Für Standardaufgaben braucht man nicht auf die API zurückzugreifen. Die MFC kapselt die am häufigsten benötigten API-Funktionen in ihren Methoden und Klassen – doch eben nur die am häufigsten benötigten Funktionen, und nicht alle. Beispielsweise gibt es für viele API-Funktionen, die auf Systeminformationen zugreifen oder der Shell-Programmierung dienen, keine Entsprechung in der MFC.



- ✗ Wie soll man also herausfinden, welche Schriftarten auf einem Rechner installiert sind? Nur indem man sich der API-Funktion `EnumFonts()` bedient.
- ✗ Wie soll man den Zustand des Arbeitsspeichers überwachen? Nur indem man auf die API-Funktion `GlobalMemoryStatus()` zurückgreift.
- ✗ Wie soll man die Systemregistrierdatenbank auf eigene Faust durchforsten, wenn nicht mit den Registry-Funktionen der API?



Darüber hinaus gibt es zusätzlich zur Windows-API noch eine Reihe weiterer APIs für spezielle Gebiete der Windows-Programmierung: die TAPI für Telefonie-Anwendungen, die MAPI für Nachrichtenanwendungen, die DDI zum Schreiben von Gerätetreibern.

Wenn Sie auf diese API-Funktionen zurückgreifen wollen, müssen Sie folgende Punkte beachten:

Gültigkeitsbereichsoperator

Wenn Sie eine API-Funktion aus einer Klassenmethode heraus aufrufen wollen, müssen Sie eventuell der API-Funktion den Gültigkeitsbereichsoperator `::` voranstellen, damit der Compiler weiß, daß es sich um eine globale Funktion und nicht eine Klassenmethode handelt. (Für API-Funktionen, für die es in der betreffenden Klasse keine gleichnamigen Klassenmethoden gibt, ist dies nicht notwendig.)

Handles

Viele Funktionen der API beziehen sich auf Fenster, Gerätekontexte, Dateien – also auf individuelle Objekte. All diese Funktionen verlangen als erstes Argument den Handle des Objekts, auf das sie sich beziehen. Bei der MFC-Programmierung kommen Sie praktisch nie mit diesen Handles in Kontakt, da die Methoden sich immer auf das Objekt beziehen, für das sie aufgerufen werden. Angenommen, Sie wollen eine Ellipse in einen Gerätekontext zeichnen.

In der MFC schreiben Sie einfach

```
CClientDC dc(this);
dc.Ellipse(100, 100, 200, 200);
```

Die Methode `Ellipse()` weiß, daß sie in den Gerätekontext des `dc`-Objekts zeichnen soll, für das sie aufgerufen wurde. Handles werden hier also nicht

benötigt. Dennoch sind die Handles vorhanden. Denken Sie daran, daß Windows-Objekte in MFC-Programmen immer in zwei Schritten erzeugt werden:

1. Zuerst wird eine Instanz der entsprechenden MFC-Klasse gebildet.
2. Dann wird das eigentliche Windows-Objekt erzeugt und mit der Klasse verbunden.

In letzterem Schritt bekommt das Windows-Objekt einen Handle zugewiesen, und die Instanz der MFC-Klasse speichert diesen Handle in einer Elementvariablen (beispielsweise `m_hWnd` für Fenster, `m_hDC` für Gerätekontexte).

Mit Hilfe dieser Elementvariablen kann man API-Funktionen, die Handles benötigen, aus MFC-Methoden heraus aufrufen:

```
HDC hdc = ::GetDC(this->m_hWnd);
::Ellipse(hdc, 100, 100, 200, 200);
::ReleaseDC(this->m_hWnd, hdc);
```

API-Funktionen verwenden keine Vorgabeargumente

In C++ kann man Funktionen und Methoden mit Vorgabeargumenten für wenig genutzte Parameter ausstatten:

```
int MessageBox( LPCTSTR lpszText, LPCTSTR lpszCaption = NULL,
               UINT nType = MB_OK );
```

Windows-API-Funktionen kennen keine Vorgabeargumente und müssen immer mit Argumenten für alle Parameter aufgerufen werden.

```
int MessageBox( HWND hWnd, LPCTSTR lpText,
               LPCTSTR lpCaption, UINT uType);
```

API-Funktionen sind nicht überladen

Etliche MFC-Methoden sind überladen, um mit verschiedenen Argumenten aufgerufen werden zu können:

```
virtual BOOL TextOut( int x, int y, LPCTSTR lpszString,
                    int nCount );
BOOL TextOut( int x, int y, const CString& str );
```

Da die Windows-API in C programmiert ist, gibt es hier keine Überladung.

```
BOOL TextOut( HDC hdc, int nXStart, int nYStart,
             LPCTSTR lpString, int cbString);
```



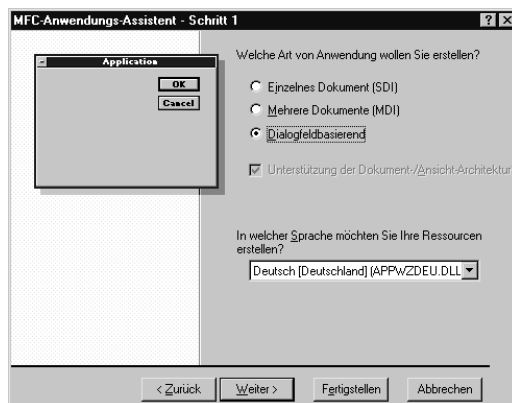
Aus diesem Grund gibt es mittlerweile auch etliche API-Funktionen mit der Extension EX. Diese enthalten (manchmal marginale) Erweiterungen zu gleichnamigen älteren API-Funktionen.

18.2 Systeminformationen abfragen

Zur Übung werden wir eine kleine dialogfeldbasierte Anwendung erstellen, die den aktuellen Zustand des Speichers abfragt.

Übung 18-1: Speicherzustand abfragen

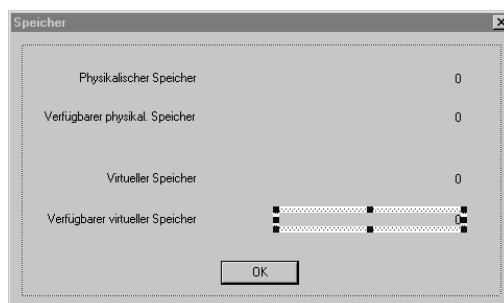
Bild 18.1:
Anwendung
mit Dialogfeld
als Haupt-
fenster



1. Erstellen Sie mit Hilfe des Anwendungs-Assistenten eine dialogfeldbasierte Anwendung.

Deaktivieren Sie in Schritt 2 des Assistenten die Option DIALOGFELD "INFO".

Bild 18.2:
Der Dialog der
Anwendung



2. Richten Sie das Dialogfeld der Anwendung ein.

Nehmen Sie dazu acht statische Textfelder auf, und arrangieren Sie diese wie in Abbildung 18.2. Für die rechts abgelegten Textfelder numerieren Sie die IDs durch (beispielsweise ID_STATIC5 bis ID_STATIC8) und aktivieren auf der Seite ERWEITERTE FORMATE des EIGENSCHAFTEN-Dialogs die Option TEXT RECHTSBÜNDIG.

3. Richten Sie mit Hilfe des Klassen-Assistenten Elementvariablen für die Textfelder der rechten Seite ein (Seite MEMBER-VARIABLEN).

Über diese Steuerelemente werden wir die Werte ausgeben, die uns die API-Funktion GlobalMemoryStatus() zurückliefert.

4. Im Konstruktor der Dialogklasse rufen Sie die API-Funktion GlobalMemoryStatus() auf.

Zuvor müssen Sie aber noch eine Variable vom Typ der Struktur MEMORYSTATUS deklarieren und dem Feld dwLength der Struktur die Größe der Struktur zuweisen.

Dann übergeben Sie die Adresse der Strukturvariable an die API-Funktion GlobalMemoryStatus(), die den restlichen Feldern der Struktur Werte zuweist.

Die uns interessierenden Werte lesen Sie dann aus und kopieren sie in die Elementvariablen für die Textfelder.

```
CSpeicherDlg::CSpeicherDlg(CWnd* pParent /*=NULL*/)
: CDialog(CSpeicherDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CSpeicherDlg)
    m_text5 = _T("");
    m_text6 = _T("");
    m_text7 = _T("");
    m_text8 = _T("");
   //}}AFX_DATA_INIT
    // Beachten Sie, dass LoadIcon unter Win32 keinen
    // nachfolgenden DestroyIcon-Aufruf benötigt
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    MEMORYSTATUS ms;
    ms.dwLength = sizeof(ms);
    GlobalMemoryStatus(&ms);
    char str[100];
    sprintf(str, "%ld", ms.dwTotalPhys);
    m_text5 = str;
```

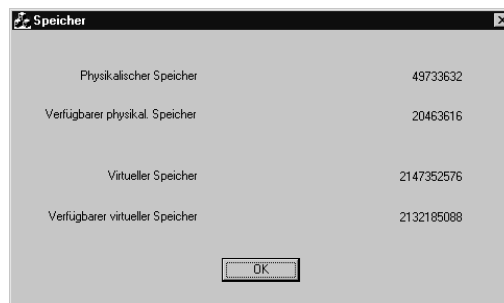
```

        sprintf(str, "%ld", ms.dwAvailPhys);
        m_text6 = str;
        sprintf(str, "%ld", ms.dwTotalVirtual);
        m_text7 = str;
        sprintf(str, "%ld", ms.dwAvailVirtual);
        m_text8 = str;
    }

```

5. Führen Sie das Programm aus (Strg + F5).

Bild 18.3:
Abfragen des
Speicher-
zustands



18.3 Zusammenfassung

Wo man mit den Möglichkeiten der MFC nicht mehr auskommt oder nach effizienteren Lösungen sucht, kann man jederzeit auf die Funktionen der Windows-API zurückgreifen.

Ruft man API-Funktionen aus MFC-Methoden heraus auf, ist zu beachten, daß man den API-Funktionen eventuell den Gültigkeitsbereichoperator :: voranstellen muß, um Verwechslungen mit MFC-Methoden zu verhindern.

Viele API-Funktionen erwarten als erstes Argument einen Handle auf das Windows-Objekt, das sie bearbeiten sollen. Für Windows-Objekte, die in MFC-Objekten gekapselt sind, sind diese Handles in speziellen public-Elementvariablen abgelegt, die man abfragen und an die API-Funktionen übergeben kann (beispielsweise `m_hWnd` für Fenster, `m_hDC` für Gerätekontexte).

18.4 Fragen

1. Was ist ein Handle?
2. Kann man die API-Funktion `TextOut()` mit einem `CString`-Argument als auszugebenden Text aufrufen?
3. Wo kann man sich darüber informieren, welche API-Funktionen es gibt und wofür man diese Funktionen einsetzt?

18.5 Aufgaben

1. Informieren Sie sich in der Online-Hilfe darüber, welche weiteren Informationen in der Struktur `MEMORYSTATUS` abgelegt werden.
2. Schreiben Sie eine dialogfeldbasierte Anwendung, die mit Hilfe der API-Funktion `GetSystemInfo()` Informationen über das aktuelle System abfragt. Gehen Sie dazu analog zur Übung 18-1 vor.



TEIL 4

**MFC-Program-
mierung für
Fortgeschrittene**



Multimedia

In den Kapiteln dieses vierten und letzten Teils des Buches werden wir uns mit ein paar ausgesuchten, fortgeschrittenen Themen beschäftigen. Zwangsweise werden dabei etliche Fragen offen bleiben, denn all diese Themen auf so engem Raum in angemessener Ausführlichkeit zu beschreiben, ist natürlich ein unmögliches Unterfangen. Die einzelnen Kapitel sind daher eher als Starthilfe für Leser gedacht, die sich ein wenig in das eine oder andere Themengebiet einarbeiten möchten. Und das erste Themengebiet, an das wir uns heranwagen wollen, lautet: Multimedia.

Sie lernen in diesem Kapitel:

- ✕ Wie man über den Computer-Lautsprecher einen Piepton ausgibt
- ✕ Wie man Klangdateien (.wav) abspielt
- ✕ Wie man Videos abspielt
- ✕ Wie man die Größe des Rahmenfensters an ein Client-Fenster anpaßt

19.1 Allgemeines

Die MFC bietet derzeit keine nennenswerte Unterstützung für Multimedia. Wer daher Multimedia-Dateien in seinen Anwendungen abspielen möchte, der muß auf die API-Funktionen zurückgreifen.



19.2 Sound

Zum Erzeugen von Tönen und Abspielen von Klangdateien stehen Ihnen im wesentlichen drei Möglichkeiten zur Verfügung.

MessageBeep()

Rufen Sie die API-Funktion `MessageBeep(UINT)` auf, um einfache Töne oder Klänge abzuspielen.

Welcher Klang abzuspielen ist, wird durch das übergebene Argument spezifiziert (`MB_OK`, `MB_ICONASTERISK`, `MB_ICONEXCLAMATION`, `MB_ICONHAND`, `MB_ICONQUESTION`). Die einzelnen Klänge müssen dazu unter den jeweiligen Konstanten registriert sein (eventuell hören Sie für alle Konstanten den gleichen Sound).

Wenn Sie `-1` übergeben, ertönt ein einfacher System-Beep.

```
MessageBeep(-1);
```

PlaySound()

Rufen Sie die API-Funktion `PlaySound()` auf:

```
BOOL PlaySound(LPCSTR pszSound, HMODULE hmod,  
               DWORD fdwSound)
```

Mit dieser Funktion können Sie – vorausgesetzt Ihr Computer verfügt über eine Soundkarte – Klangdateien (.wav) abspielen.

- ✘ `pszSound`. Als Klang übergeben Sie den Namen der abzuspielenden Klangdatei oder den Namen einer Ressource.
- ✘ `hmod`. Wenn Sie eine Klangdatei abspielen, übergeben Sie als zweiten Parameter `NULL`, ansonsten den Instanz-Handle des Moduls, dem die Sound-Ressource angehört.
- ✘ `fdwSound`. Zuletzt übergeben Sie eine Kombination verschiedener `SND`-Konstanten (siehe Online-Hilfe zur API-Funktion). Um beispielsweise eine Klangdatei fortwährend abzuspielen, setzen Sie die Flags `SND_ASYNC` und `SND_LOOP`.

```
PlaySound("C:\\Windows\\media\\Der Microsoft-Sound.wav",  
          SND_FILENAME | SND_ASYNC | SND_LOOP);
```

Um die Funktion `PlaySound()` aufrufen zu können, müssen Sie die Liste der Header-Dateien in *stdafx.h* um den Eintrag `#include <mmsystem.h>` erweitern und auf der Seite **LINKER** der Projekteinstellungen im Feld **OBJEKT-/BIBLIOTHEK-MODULE** die Bibliothek *winmm.lib* einfügen.



MCIWnd

Um mehr Einflußmöglichkeiten zu haben, nutzen Sie das **MCIWnd**-Befehlsset, das im folgenden Abschnitt zum Abspielen von Video-Dateien (.avi) genutzt wird.

19.3 Video

Zum Abspielen von Videos (und anderen Multimedia-Dateien) stehen eine Reihe von **MCIWnd**-Funktionen und -Makros zur Verfügung (**MCI** steht für **Media Control Interface**), deren Gebrauch an die Erzeugung eines **MCIWnd**-Fensters gekoppelt ist.

Übung 19-1: Videos abspielen

1. Erstellen Sie mit Hilfe des **MFC-ANWENDUNGS-ASSISTENTEN (EXE)** eine **SDI-Anwendung** ohne weitere Fensterdekorationen.
2. Richten Sie die Anwendung für den Gebrauch des »Video for Windows«-SDK ein.

Erweitern Sie die Liste der Header-Dateien in *stdafx.h* um den Eintrag:

```
#include <vfw.h>
```

Bearbeiten Sie die Projekteinstellungen (Befehl **PROJEKT/EINSTELLUNGEN**), und wechseln Sie zur Seite **LINKER**. Geben Sie dort in das Feld **OBJEKT-/BIBLIOTHEK-MODULE** den Namen »VFW32.LIB« ein.

3. Deklarieren Sie eine Instanz für das Ausgabefenster der Videoanzeige.

Deklarieren Sie dafür in der Ansichtsklasse eine **HWND-Elementvariable**.

```
class CVideoView : public CView
{
...
public:
    CVideoDoc* GetDocument();
    HWND m_hAVI;
```

4. Initialisieren Sie das Ausgabefenster der Videoanzeige.

Überschreiben Sie mit dem Klassen-Assistenten die Methode `OnInitialUpdate()` der Ansichtsklasse. (Wählen Sie die Methode im Feld **NACHRICHTEN** aus, und klicken Sie auf den Schalter **FUNKTION HINZUFÜGEN**.)

Rufen Sie die Funktion `MCIWndCreate()` auf, und übergeben Sie den Handle des übergeordneten Fensters (das Video-Fenster verschmilzt mit dem übergeordneten Fenster), den Instanz-Handle, etwaige Stil-Parameter (siehe Online-Hilfe zur Funktion) sowie die zu öffnende Datei (hier `NULL`, die Datei wird erst später geöffnet).

Setzen Sie als Stil `MCIWNDF_NOTIFYSIZE`, damit das Video-Fenster beim Laden eines Videos eine `MCIWNDF_NOTIFYSIZE`-Nachricht abschickt, um anzuzeigen, daß sich die Größe des Ausgabebereichs geändert hat.

Setzen Sie als Stil `MCIWNDF_SHOWALL`, um die Bedienungsleiste des Video-Fensters einblenden zu lassen.

```
void CVideoView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    m_hAVI = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
                          MCIWNDF_NOTIFYSIZE |
                          MCIWNDF_SHOWALL, NULL);
}
```

5. Spielen Sie eine Video-Datei (.avi) ab.

Laden Sie die Datei mit Hilfe der Methode `MCIWndOpen()`. Übergeben Sie den Namen und Pfad der abzuspielenden AVI-Datei (passende Videos finden Sie beispielsweise im *Windows/Help*-Verzeichnis). (Die 0 als letztes Argument gibt an, daß eine bestehende Datei zu laden und nicht eine neue Datei anzulegen ist.)

Rufen Sie die Methode `MCIWndPlay()` zum Abspielen auf.

```
void CVideoView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    m_hAVI = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
                          MCIWNDF_NOTIFYSIZE |
                          MCIWNDF_SHOWALL, NULL);

    if(m_hAVI)
```

```

    {
        MCIWndOpen(m_hAVI, ".\\res\\Clock.avi", 0);
        MCIWndPlay(m_hAVI);
    }
}

```

Wenn Sie wollen, können Sie die Anwendung jetzt schon einmal abspielen.

6. Passen Sie die Größe des Rahmenfensters an das abzuspielende Fenster an. Fangen Sie dazu die MCIWDM_NOTIFYSIZE-Nachricht des Video-Fensters ab.

Deklarieren Sie in der Ansichtsklasse eine passende Antwortmethode.

```

protected:
    afx_msg LONG OnNotifySize(UINT wParam, LONG lParam);

```

Nehmen Sie einen entsprechenden Eintrag in die Antworttabelle der Ansichtsklasse vor.

```

BEGIN_MESSAGE_MAP(CVideoView, CView)
    ON_MESSAGE(MCIWDM_NOTIFYSIZE, OnNotifySize)
END_MESSAGE_MAP()

```

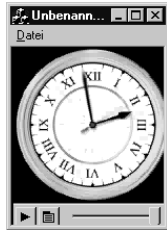
Implementieren Sie die Behandlungsmethode. Die API-Funktion `GetClientRect()` liefert Ihnen die Maße des Video-Fensters. Aufgrund dieses Wertes können Sie dann mit Hilfe der API-Funktion `AdjustWindowRect()` die Größe für ein passendes Rahmenfenster (Fensterstil `WS_OVERLAPPEDWINDOW` mit Menü (dritter Parameter `true`) bestimmen. Zuletzt übergeben Sie die neu berechneten Abmessungen an die Methode `SetWindowPos()`, um die Größe des Rahmenfensters anzupassen.

```

LONG CVideoView::OnNotifySize(UINT wParam, LONG lParam)
{
    CRect rect;
    if(m_hAVI) {
        ::GetClientRect(m_hAVI, rect);
        AdjustWindowRect(rect, WS_OVERLAPPEDWINDOW, true);
        GetParentFrame()->SetWindowPos(NULL, 0, 0,
            rect.Width(), rect.Height(),
            SWP_NOZORDER | SWP_NOACTIVATE | SWP_NOMOVE);
    }
    return 1;
}

```


Bild 19.1:
Abspielen
eines AVI-
Videos



19.4 Zusammenfassung

Zum Ausgeben einfacher Signaltöne ruft man die API-Funktion `MessageBeep()` auf, zum Abspielen von Klangdateien die Funktion `PlaySound()`.

Die weitreichendste Multimedia-Unterstützung bieten die `MCIWnd`-Funktionen, mit denen man beispielsweise auch AVI-Videos abspielen kann.

Dynamische Linkbibliotheken (DLLs)

Dynamische Linkbibliotheken sind eine speicherschonende Abart der üblichen statischen Linkbibliotheken. Während der Code statischer Linkbibliotheken (LIBs) in die EXE-Dateien der Programme eingebunden wird, die Funktionen, Klassen oder Ressourcen aus der Bibliothek verwenden, werden für DLLs nur Verweise auf die Elemente in der DLL in die EXE-Datei aufgenommen. Werden zwei, drei oder mehrere Anwendungen gestartet, die alle die gleiche statische Linkbibliothek verwenden, wird diese mehrfach zusammen mit dem Code der Anwendungen in den Arbeitsspeicher geladen. Anders bei den DLLs – hier wird der Code nur einmal in den Arbeitsspeicher geladen, und alle laufenden Anwendungen, die Elemente der DLL verwenden, greifen gemeinsam auf den Code der DLL zu.

Sie lernen in diesem Kapitel:

- ✗ Wie man mit Hilfe des MFC-Anwendungs-Assistenten dynamische Linkbibliotheken (DLLs) erstellt
- ✗ Wie man mehrere Projekte in einem Arbeitsbereich verwaltet
- ✗ Wie man Funktionen aus DLLs aufruft
- ✗ Wie man DLLs in der Visual C++-IDE ausführt



20.1 Allgemeines

Bei der Implementierung eigener DLLs gilt es, folgende Punkte zu beachten:

Die Eintrittsfunktion DllMain()

Eine DLL besitzt eine spezielle Eintrittsfunktion: `DllMain()`, die im übrigen auch als Austrittsfunktion fungiert.

In der Ein-/Austrittsfunktion 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 dwReason`). 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 `CDLLApp`-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 der zu exportierenden Funktion (Klasse) einfach das Schlüsselwort `__declspec(dllexport)` voran.



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.

DLLs laden

Eine Anwendung, die Elemente 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 mit erzeugt 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.

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.

Funktionen/Klassen importieren

Die aufrufende Anwendung muß die DLL-Funktionen (DLL-Klassen), die sie aufrufen will, zuerst importieren. Dazu stellen Sie der Definition der zu importierenden Funktion einfach das Schlüsselwort `__declspec(dllimport)` voran.

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 funcName)` um sich einen Zeiger auf die gewünschte Funktion zurückliefern zu lassen.

20.2 Erstellung der DLL

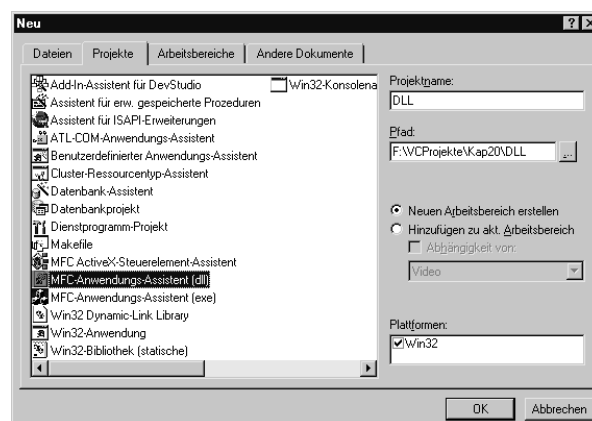


Bild 20.1:
Anlegen eines
DLL-Projekts

1. Erstellen Sie mit Hilfe des MFC-ANWENDUNGS-ASSISTENTEN (DLL) ein DLL-Projekt.

Die vordefinierten Einstellungen im Schritt 1 des Assistenten können Sie unverändert beibehalten.

2. Definieren Sie die Funktionen der DLL in der CPP-Datei. Funktionen oder Klassen, die exportiert werden sollen, werden als

```
extern "C" __declspec(dllexport)
```

deklariert.

```
extern "C" __declspec(dllexport)
void respond_to_rechterMaus(CWnd* ptr, CPoint& point)
{
    CClientDC dc(ptr);
    char s[30];
    wsprintf(s, "DLL: rechte Maustaste");
    dc.TextOut(point.x, point.y, s, strlen(s));
}
extern "C" __declspec(dllexport)
void respond_to_linkerMaus(CWnd* ptr, CPoint& point)
{
    CClientDC dc(ptr);
    char s[30];
    wsprintf(s, "DLL: linke Maustaste");
    dc.TextOut(point.x, point.y, s, strlen(s));
}
```

3. Erstellen Sie die DLL, ohne sie auszuführen (F7).

Der Linker bindet nicht nur die DLL, sondern erstellt zu Ihrer Bequemlichkeit auch gleich eine passende Importbibliothek für die DLL.

20.3 Erstellung der EXE-Anwendung

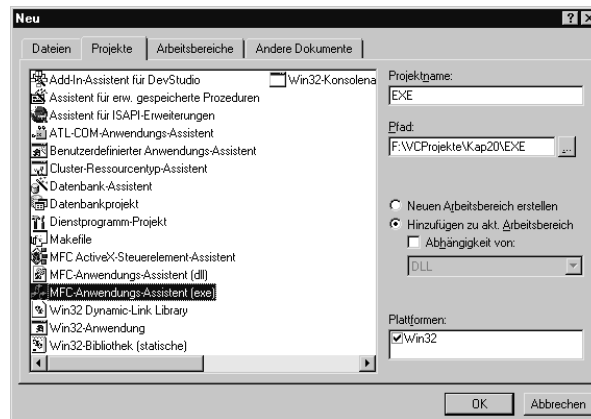


Bild 20.2:
Anlegen einer
Testanwen-
dung

1. Erstellen Sie das Projekt der Testanwendung, die die Funktionen der DLL verwenden soll, im gleichen Arbeitsbereich wie die DLL.

Lassen Sie das DLL-Projekt geladen, und rufen Sie den Befehl DATEI/NEU auf. Auf der Seite PROJEKTE wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN (EXE) zum Anlegen einer SDI-Anwendung aus.

Achten Sie darauf, daß Sie das EXE-Projekt im gleichen Arbeitsbereich wie die DLL anlegen (Option HINZUFÜGEN ZU AKT. ARBEITSBEREICH aktivieren). Dies ist zwar nicht unbedingt erforderlich, ist aber übersichtlicher.

Das neu erstellte Projekt wird im Arbeitsbereichsfenster angezeigt. An der Hervorhebung in Fettschrift können Sie ablesen, daß das neue Projekt gleich zum aktiven Projekt des Arbeitsbereichs gemacht wurde. Über den Befehl PROJEKT/AKTIVES PROJEKT FESTLEGEN können Sie jederzeit selbst festlegen, auf welches Projekt sich die Menübefehle der Visual C++-IDE beziehen sollen.

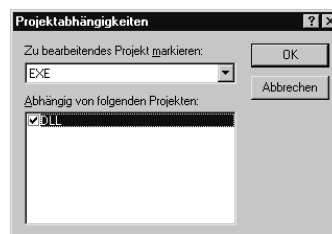


Bild 20.3:
Importbibliothek der DLL
in Projekt aufnehmen

2. Binden Sie die Importbibliothek der DLL ein. Rufen Sie den Befehl PROJEKT/ABHÄNGIGKEITEN auf.

Im Feld ABHÄNGIG VON FOLGENDEN PROJEKTEN aktivieren Sie das Kontrollkästchen des DLL-Projekts. Damit wird die Importbibliothek der DLL in das Projekt der Anwendung eingebunden. Alternativ können Sie die Importbibliothek (*.lib*) auch direkt über den Befehl PROJEKT/DEM PROJEKT HINZUFÜGEN aufnehmen.

Sie müssen noch dafür sorgen, daß die DLL auch zu finden ist. Kopieren Sie die *.dll*-Datei aus dem Ausgabeverzeichnis des DLL-Projekts in das Ausgabeverzeichnis des EXE-Projekts. (Die DLL sucht die Datei zuerst im Verzeichnis der EXE-Datei, dann im Windows-Systemverzeichnis zuletzt im Pfad.)

3. Deklarieren Sie die zu verwendenden DLL-Funktionen. Dies geschieht, um die Funktionsnamen dem Compiler bekannt zu machen. Deklarieren Sie die DLL-Funktionen in der Header-Datei der Anwendung (*EXE.h*) mit den Schlüsselwörtern `extern "C" __declspec(dllimport)`:

```
//Deklaration der importierten DLL_Funktionen
extern "C" {
    __declspec(dllimport)
    void respond_to_rechterMaus(CWnd* ptr, CPoint& point);
    __declspec(dllimport)
    void respond_to_linkerMaus(CWnd* ptr, CPoint& point);
}
```

4. Rufen Sie die DLL-Funktionen ganz normal auf. Fangen Sie die Windows-Nachrichten `WM_LBUTTONDOWN` und `WM_RBUTTONDOWN` in der Ansichtsklasse ab, und rufen Sie in den Behandlungsmethoden zu diesen Nachrichten die DLL-Funktionen auf.

```
void CEXEView::OnLButtonDown(UINT nFlags, CPoint point)
{
    respond_to_linkerMaus(this, point);

    CView::OnLButtonDown(nFlags, point);
}
void CEXEView::OnRButtonDown(UINT nFlags, CPoint point)
{
    respond_to_rechterMaus(this, point);
    CView::OnRButtonDown(nFlags, point);
}
```

5. Führen Sie die Anwendung aus.

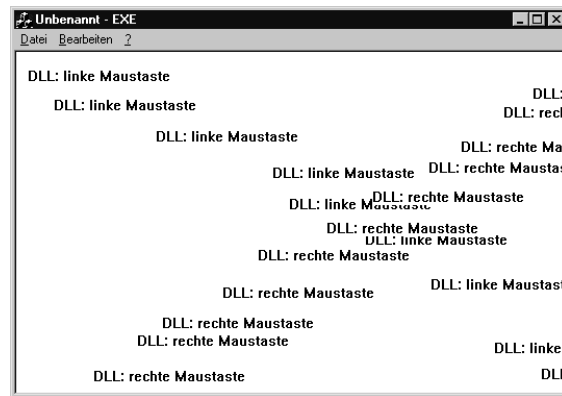


Bild 20.4:
Aufruf der
DLL-Funktionen in einer
Anwendung

20.4 DLL debuggen und ausführen

Nachdem die Testanwendung erstellt ist, können Sie die DLL-Anwendung auch direkt ausführen.

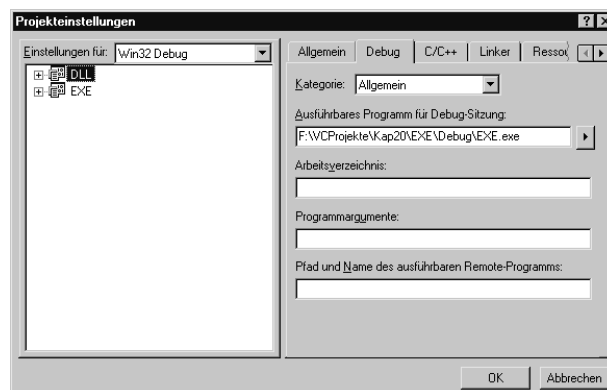


Bild 20.5:
Testanwendung zum
Debuggen
festlegen

1. Aktivieren Sie das DLL-Projekt (Menübefehl PROJEKT/AKTIVES PROJEKT FESTLEGEN).
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.



Wenn Sie Änderungen am Code der DLL vornehmen und diese danach im Debugger überprüfen wollen, dürfen Sie nicht vergessen, die DLL zuvor neu zu erstellen und die *.dll*-Datei in das Ausgabeverzeichnis der Testanwendung zu kopieren.

20.5 Zusammenfassung

Dynamische Linkbibliotheken haben den Vorteil, daß sie stets nur einmal in den Arbeitsspeicher geladen werden – auch wenn mehrere Anwendungen auf den Code der DLL zugreifen.

Projekte für dynamische Linkbibliotheken legt man mit dem *MFC-Anwendungs-Assistenten (dll)* an.

Funktionen und Klassen der DLL, die von anderen Anwendungen aufgerufen werden sollen, werden in der DLL als `__declspec(dllexport)` und in der Anwendung als `__declspec(dllimport)` deklariert.

Damit eine Anwendung auf eine DLL zugreifen kann, müssen Sie eine Importbibliothek für die DLL erstellen (geschieht automatisch bei der Erstellung des DLL-Projekts) und in das Projekt der Anwendung einbinden. Zudem muß die DLL selbst verfügbar sein, das heißt, sie muß im Verzeichnis der Anwendung oder in einem Systemverzeichnis zu finden sein.

MDI-Anwendungen

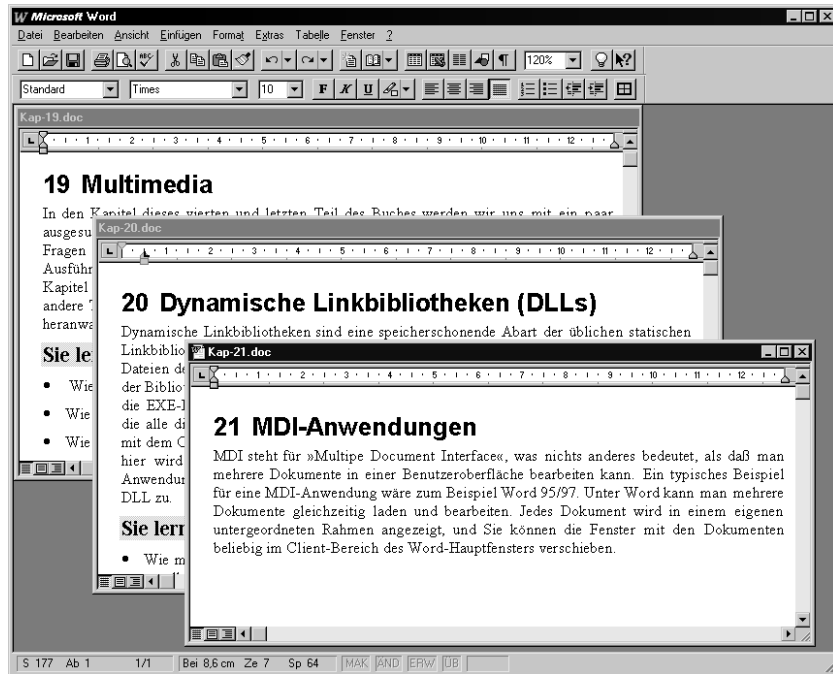
MDI steht für »Multiple Document Interface«, was nichts anderes bedeutet, als daß man mehrere Dokumente in einer Benutzeroberfläche bearbeiten kann. Ein typisches Beispiel für eine MDI-Anwendung wäre zum Beispiel Word 95/97. Unter Word kann man mehrere Dokumente gleichzeitig laden und bearbeiten. Jedes Dokument wird in einem eigenen untergeordneten Dokumentfenster angezeigt, und Sie können die Fenster mit den Dokumenten beliebig im Client-Bereich des Word-Hauptfensters verschieben.

Die Frage ist allerdings, welche Zukunftschancen MDI hat. Einer der großen Vorteile von MDI ist, oder zumindest war, daß man zur Bearbeitung mehrerer Dokumente nicht mehrfach die gleiche Anwendung aufrufen muß. Dies schont den Arbeitsspeicher und nimmt weniger Platz auf dem Bildschirm weg, weil das Menü der Anwendung nur einmal angezeigt wird. Doch wer arbeitet heute noch mit Auflösungen 800 x 600 oder weniger? Wer hat noch Arbeitsspeicher, die mit »mageren« 8 Mbyte bestückt sind? Einiges spricht dafür, daß die traditionellen MDI-Anwendungen langsam von unseren Bildschirmflächen verschwinden werden. Der Trend geht zu SDI-Anwendungen (siehe zum Beispiel die Internet-Browser) und Sammelmappen (siehe Office).



KAPITEL 21 MDI-Anwendungen

Bild 21.1:
Ein MDI-Editor



Trotzdem wollen wir noch einen kurzen Blick auf die Erstellung von MDI-Anwendungen werfen: zum einem, weil der Anwendungs-Assistent uns MDI als Option offeriert, zum anderen weil man in MDI-Anwendungsgerüsten mehrere Dokumentvorlagen einrichten kann.

Sie lernen in diesem Kapitel:

- ✗ Wie man mit Hilfe des MFC-Anwendungs-Assistenten MDI-Anwendungen erstellt
- ✗ Wie die Aufgabenverteilung zwischen den Fensterklassen des MDI-Anwendungsgerüsts aussieht
- ✗ Ein wenig über die Unterstützung verschiedener Dokumenttypen in einer Anwendung

21.1 Erstellung eines MDI-Editors

Die Erstellung eines MDI-Editors ist dank des MFC-Anwendungs-Assistenten genauso einfach wie die Erstellung eines SDI-Editors (siehe Kapitel 11). Auch die nötigen Arbeitsschritte sind die gleichen.

Übung 11-4: Grundgerüst für einen Texteditor

1. Legen Sie ein neues Projekt an (Befehl DATEI/NEU, Seite PROJEKTE).

Geben Sie als Namen »Texteditor« ein. Wählen Sie ein passendes übergeordnetes Verzeichnis für das Projekt, und lassen Sie einen neuen Arbeitsbereich für das Projekt anlegen.

Links wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN aus.

2. Im ersten Schritt des Assistenten entscheiden Sie sich für eine MDI-Anwendung mit Dokument/Ansicht-Architektur.

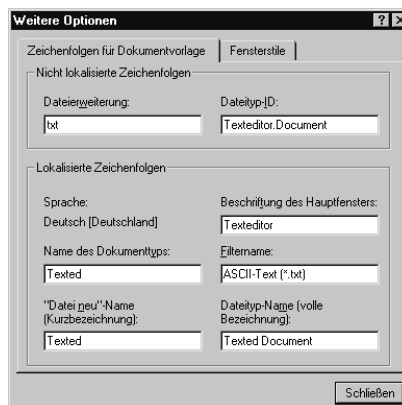


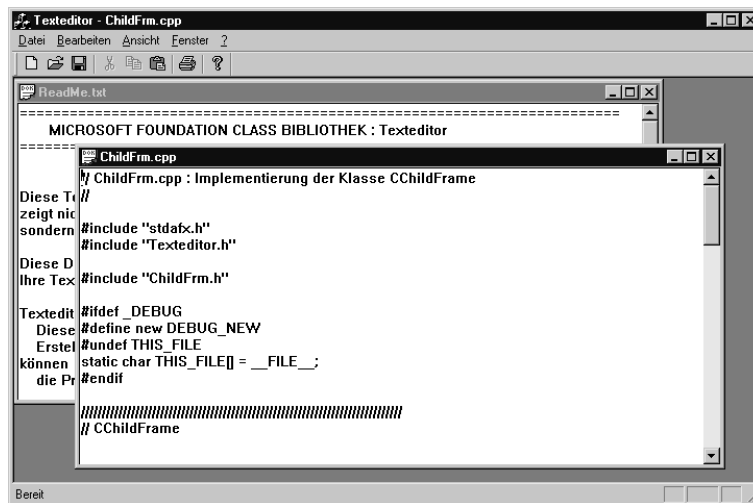
Bild 21.2:
Einstellungen
für die Datei-
Dialoge

3. Im vierten Schritt behalten Sie die Voreinstellungen bei und klicken auf den Schalter WEITERE. Auf der Seite ZEICHENFOLGEN FÜR DOKUMENTVORLAGE

- ✗ geben Sie als Dateierweiterung »txt« an (diese Extension wird automatisch beim Speichern angehängt, wenn der Anwender keine Extension für einen Dateinamen angibt),
- ✗ geben Sie im Feld FILTERNAME die Extensionen an, nach denen die Dialoge zum Öffnen und Speichern die Verzeichnisse durchsuchen sollen.

4. Jetzt kommt die wichtigste Einstellung überhaupt! Im sechsten Schritt klicken Sie oben auf die Klasse für das Ansichtsfenster und wählen als Basisklasse nicht mehr CVIEW, sondern CEDITVIEW aus.
5. Lassen Sie das Projekt jetzt fertigstellen und ausführen (**Strg** + **F5**).

Bild 21.3:
Der fertige
MDI-Editor



21.2 Die Klassen des MDI-Anwendungsgerüsts

Das Auffälligste am MDI-Anwendungsgerüst ist, daß wir es mit zwei Typen von Rahmenfenstern zu tun haben (siehe Abbildung 21.3).

- ✗ Zum einen natürlich das Hauptfenster unserer Anwendung. Die Klasse unseres Hauptfensters heißt nach wie vor CMainFrame und ist als ordentliches Rahmenfenster von CFrameWnd abgeleitet – allerdings nicht auf direktem Wege, sondern über die Klasse CMDIFrameWnd, die ihrerseits von CFrameWnd abgeleitet ist, darüber hinaus unserem Hauptfenster aber noch zusätzliche Funktionalität zur Unterstützung des MDI-Konzepts zur Verfügung stellt (beispielsweise Methoden zur Verwaltung der Dokumentfenster, vergleiche Menü FENSTER).
- ✗ Zum anderen aber auch die Rahmenfenster der Klasse CChildFrame. Diese Klasse ist von CMDIChildWnd abgeleitet, und CMDIChildWnd selbst ist von CFrameWnd abgeleitet.

Die Arbeitsteilung zwischen diesen Fenstern sieht so aus, daß

- ✗ das Hauptfenster Menü, Symbolleiste und Statusleiste anzeigt und die Spielwiese für die CChildFrame-Fenster stellt, und daß
- ✗ für jedes Dokument, das geladen wird, ein CChildFrame-Rahmenfenster erzeugt wird. Der Client-Bereich eines CChildFrame-Rahmenfensters wird von einem Ansichtsfenster ausgefüllt, in dem der Inhalt der Datei angezeigt wird. Die CChildFrame-Rahmenfenster sind dem Hauptfenster untergeordnet und können daher nicht aus dem Client-Bereich des Hauptfensters heraus bewegt werden. Die CChildFrame-Rahmenfenster verfügen über eigene Menüs (siehe Ressourcen-Ansicht), die jedoch nicht in den untergeordneten Rahmenfenstern angezeigt werden, sondern mit dem Menü des Hauptfensters verschmolzen werden.

Beim Start der Anwendung werden in der InitInstance()-Methode wie gewohnt Hauptfenster und Dokumentvorlage erzeugt. Neu ist, daß die Dokumentvorlage vom Typ der Klasse CMultiDocTemplate ist und nicht auf der Basis des Hauptfensters, sondern der untergeordneten Rahmenfenster erstellt wird:

```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_TEXTEDTYPE,
    RUNTIME_CLASS(CTexteditorDoc),
    RUNTIME_CLASS(CChildFrame),
    RUNTIME_CLASS(CTexteditorView));
AddDocTemplate(pDocTemplate);
```

Das Hauptfenster muß aus diesem Grund explizit erzeugt werden:

```
CMainFrame* pMainFrame = new CMainFrame;
if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
    return FALSE;
m_pMainWnd = pMainFrame;
```

21.3 MDI mit mehreren Dokumentvorlagen

MDI bietet auch die Möglichkeit, mehrere Dokumenttypen zu unterstützen – beispielsweise Textdateien und Tabellenblätter.

Dazu muß man aber für jeden Dokumenttyp eine eigene `CMultiDocTemplate`-Dokumentvorlage erstellen und registrieren (Aufruf von `AddDocTemplate()`). Und für jede Dokumentvorlage braucht man:

- ✗ eine eigene Dokumentklasse, die die Daten der Dokumente aufnehmen kann
- ✗ eine eigene Ansichtsklasse, die die Daten der Dokumente anzeigen kann
- ✗ eine eigene Rahmenfensterklasse, die das Menü für die Bearbeitung der Daten stellt
- ✗ eine eigene Stringressource, die die Angaben für die Öffnen- und Speicher-Dialoge enthält (mit Angaben zu Dokumentextension, Filter etc. – vergleiche Dialogfeld WEITERE OPTIONEN des Anwendungs-Assistenten).

Sind mehrere Dokumentvorlagen eingerichtet, kann das Anwendungsgerüst beim Öffnen neuer Dateien automatisch an der Extension der Dateien erkennen, welche Dokumentvorlage für das Dokument zu aktivieren ist. Mit Hilfe der `CWinApp`-Methoden `GetFirstDocTemplatePosition()` und `GetNextDocTemplate()` kann man aber auch selbst Dokumentvorlagen aktivieren.

Wenn Sie sich selbst in der Implementierung mehrerer Dokumentvorlagen versuchen wollen, informieren Sie sich am besten in der Online-Hilfe. Expandieren Sie im Inhaltsverzeichnis die Knoten `VISUAL C++-DOKUMENTATION/ARBEITEN MIT VISUAL C++/VISUAL C++-PROGRAMMIERHANDBUCH/HINZUFÜGEN VON FUNKTIONEN DER BENUTZEROBERFLÄCHE/VORGEHENSWEISE/AUFGABEN BEI DOKUMENT/ANSICHT`, und lassen Sie die gleichnamige Seite anzeigen. Dort finden Sie etliche interessante Links, beispielsweise `ERSTELLEN MEHRERER DOKUMENTTYPEN`, die Sie weiterbringen.

21.4 Zusammenfassung

MDI-Anwendungen sind Anwendungen, in denen in einem Hauptfenster mehrere Dokumente in jeweils eigenen untergeordneten Rahmenfenstern angezeigt und verwaltet werden können.

Zur Erstellung von MDI-Anwendungen bedient man sich am einfachsten des MFC-Anwendungs-Assistenten.

MDI-Anwendungen haben nicht nur den Vorteil, daß der Anwender mehrere Dokumente gleichzeitig öffnen kann, MDI-Anwendungen erlauben auch das Öffnen unterschiedlicher Dokumenttypen – wozu allerdings für jeden Dokumenttyp eine eigene Dokumentvorlage eingerichtet werden muß.

COM

COM, das sogenannte Component Object Model, ist eine technische Spezifikation, die festlegt, nach welchen Regeln Windows-Anwendungen miteinander kommunizieren können. Wenn Sie in eines Ihrer Word-Dokumente über den Befehl EINFÜGEN/OBJEKT eine Excel-Tabelle aufnehmen, steht dahinter COM. Daß Sie zur Bearbeitung der Tabelle diese nur im Word-Dokument anzuklicken brauchen, und schon wird Excel aufgerufen, wird erst durch COM möglich. COM kann aber noch mehr. Dank COM können Anwendungen Teile ihres Codes anderen Anwendungen zur Verfügung stellen. Auf diese Weise können Programme in ihrem Code Funktionen, Methoden und ganze Klassen verwenden, die in anderen Programmen definiert sind. Und auf COM basiert auch die gesamte ActiveX-Technologie, die es uns beispielsweise erlaubt, Programmkomponenten aus dem Internet herunterzuladen und auf unserem Computer ausführen zu lassen.

Bevor Sie jetzt aber in Euphorie ausbrechen, muß ich Ihnen einen kleinen Dämpfer versetzen:

- ✗ Die COM-Spezifikation ist recht kompliziert und nicht von einem Tag auf den anderen zu erlernen.
- ✗ COM ist nicht die Lösung aller Probleme. Nicht immer ist eine COM-Lösung schneller und besser als eine konventionelle Lösung.
- ✗ COM ist derzeit auf die Windows-Plattform begrenzt. COM-Implementierungen sind also von vornherein plattformabhängig.



- ✗ Programme, die auf dem Weg über COM die Funktionalität anderer Anwendungen nutzen, können nur ausgeführt werden, wenn diese Anwendungen unter Windows registriert sind. Hier entstehen also Abhängigkeiten zwischen Anwendungen, die zumindest dem Anwender nicht immer zum Nutzen gereichen. Wenn Sie beispielsweise in einer Ihrer Anwendungen auf die Rechtschreibhilfe von Word zurückgreifen, können Anwender, die mit WordPerfect arbeiten, Ihr Programm nicht in vollem Umfang nutzen. Wenn Ihre Anwendung die Funktionalität eines bestimmten ActiveX-Steuerelements nutzt, dieses aber vom Anwender aus Versehen gelöscht oder sein Eintrag in der Systemregistrierung korumpiert wird, ist sofort auch Ihre Anwendung betroffen.¹
- ✗ Die Ausführung von Programmkomponenten, die vom Internet heruntergeladen werden (ermöglicht durch die ActiveX-Spezifikation), ist mit erheblichen Sicherheitsrisiken verbunden, da diese ActiveX-Komponenten vollen Zugriff auf das lokale System haben (im Gegensatz etwa zu Java-Applets, die mit Web-Seiten heruntergeladen werden). Niemand kann diese ActiveX-Komponenten daran hindern, Daten auf Ihrer Festplatte auszuspionieren oder Ihre Festplatte zu formatieren.
- ✗ COM wird demnächst durch eine erweiterte und überarbeitete Version namens COM+ ersetzt werden.

Sie lernen in diesem Kapitel:

- ✗ Was man unter dem Einbetten und Verknüpfen von Objekten versteht
- ✗ Was man unter Automatisierung versteht
- ✗ Wie man einen Automatisierungs-Server implementiert
- ✗ Wie man einen Automatisierungs-Client implementiert

¹ Beim letzten großen Erdbeben in Kalifornien stürzte eine große Highway-Brücke fast vollständig ein. Der Grund für diese Katastrophe in der Katastrophe war, daß man die Brücke besonders erdbebensicher gebaut hatte. Die Ingenieure hatten die einzelnen Pfeiler der Brücke mit Stahltrossen verbunden, damit sich die Pfeiler im Falle eines Bebens gegenseitig stabilisieren. Als das Beben kam, brach ein einzelner Pfeiler ein, sackte zusammen und zog über die Stahltrossen seine Nachbarn zu beiden Seiten mit sich.

22.1 OLE (Object Linking and Embedding)

Eine der ältesten und gebräuchlichsten Möglichkeiten von COM ist das Einbetten (Embedding) und Verknüpfen (Linking) von Objekten in Verbunddokumenten.

Ein **Verbunddokument** ist ein Dokument, in dem Daten unterschiedlicher Natur verwaltet und angezeigt werden. Um beispielsweise eine ansprechende Präsentationsmappe zu erstellen, müssen Sie in Ihren Text auch Tabellen und Grafiken einbauen. Text, Tabelle und Grafik liegen aber unterschiedliche Datenformate zugrunde. Ein Dokument, in dem solche unterschiedlichen Datenformate gemeinsam untergebracht werden können, nennt man Verbunddokument.

Damit eine Anwendung das oben beschriebene Verbunddokument bearbeiten kann, müßten Sie über die Funktionalität einer Textverarbeitung, einer Tabellenkalkulation und eines Grafikprogramms verfügen. Für eine Multimedia-Präsentation möchten Sie in das Dokument aber vielleicht noch eine Klangdatei und eine AVI-Animation aufnehmen. Jetzt müßte die Anwendung also auch noch in der Lage sein, diese Dateien abzuspielen. Es dürfte klar sein, daß man auf diesem Weg nicht weitergehen kann.

Andererseits stehen für alle oben genannten Objekte bereits Programme zur Verfügung, die sie bearbeiten können. Das Problem wäre also gelöst, wenn man eine Anwendung hätte, die Verbunddokumente mit beliebigen integrierten Objekten verwalten kann, während zur eigentlichen Bearbeitung der Objekte die speziellen Programme aufgerufen werden. Dies bringt uns zum Client/Server-Modell und zum Einbetten und Verknüpfen von Objekten.

Der Client (Container)

Ein Client oder Container ist eine Anwendung, die Objekte aufnehmen kann. Der Client übernimmt die Anzeige und Abspeicherung der Objekte in seinem Dokument, wobei er – für den Benutzer unsichtbar – von verschiedenen dynamischen Linkbibliotheken unterstützt wird. Die Bearbeitung der Objekte bleibt alleinige Aufgabe des Servers, der die Objekte erzeugt hat. Der Austausch der Objekte wird stets vom Client angestoßen, der entweder ein Objekt zum Aufnehmen in sein Dokument auswählt oder einen Server zum Bearbeiten eines bereits integrierten Objekts aufruft. Da das Einfügen der Objekte vom Client ausgeht, sollte er in seiner Menüstruktur Befehle zum Einbetten und Verknüpfen von Objekten haben. Die Auswahl eines bereits integrierten Objekts erfolgt durch Doppelklick mit der Maus.

Der Server

Server sind Anwendungen, die ihre Daten in Form von Objekten zur Einbindung oder Verknüpfung den OLE-Clients zur Verfügung stellen. Der Server bleibt stets für die Bearbeitung seiner Objekte verantwortlich, auch wenn diese in einem Clienten abgelegt sind. Dies hat zur Folge, daß der Server aufgerufen wird, wenn Sie in einem Dokument ein Objekt doppelt anklicken. Der Server sollte unterscheiden können, ob er von einer Client-Anwendung oder vom Benutzer aufgerufen wurde. Wird er von einer Client-Anwendung aufgerufen, zeigt er meist eine andere Menüstruktur an. Insbesondere das Datei-Menü wird üblicherweise angepaßt.

Wird ein Objekt in einem Dokument durch Doppelklick zur Bearbeitung aufgerufen, erscheint der Server. Das Erscheinungsbild des Servers hängt von der Implementierung des Clients und des Servers ab. Im einfachsten Fall erscheint der Server als Anwendung in seinem Hauptfenster mit entsprechend angepaßter Menüstruktur. Es ist aber auch möglich, daß sich der Server in das Fenster der Client-Anwendung eingliedert. Durch Menüverschmelzung erscheint im Rahmenfenster der Client-Anwendung dann das Menü des Servers. Dieses Konzept, das man auch als Vor-Ort-Bearbeitung bezeichnet, gibt es erst seit OLE 2, aber es ist heute schon Standard für anspruchsvolle OLE-Anwendungen.



Damit eine Container-Anwendung Objekte eines Servers überhaupt einbetten kann, muß sich der Server unter Windows registrieren und angeben, welche Art von Objekten er unterstützt.

Einbettung und Verknüpfung

- ✗ Die *Einbettung* (Embedding) ist die erste Form der Aufnahme von Objekten in Dokumente. Einbettung bedeutet, daß das Objekt zu einem physikalischen Bestandteil des Dokuments wird. Der Container übernimmt die Daten des Objekts – auch wenn er selbst mit den Daten nichts anfangen kann – und speichert sie mit dem Dokument ab.
- ✗ Die *Verknüpfung* (Linking) ist das Gegenstück zur Einbettung. Bei der Verknüpfung enthält das Dokument lediglich einen Verweis auf die Originaldaten des Objekts und den zugehörigen Server. Ansonsten wird das Objekt wie üblich vom Container angezeigt und vom Server bearbeitet. Wenn Sie beispielsweise in einem Word-Dokument eine Verknüpfung zu einer Excel-Tabelle herstellen, wird der Inhalt der Excel-Datei in Ihrem Word-Dokument angezeigt. Bearbeiten Sie später die Tabelle in Excel und speichern die Änderungen in der Datei ab, wird auch das Word-

Dokument automatisch aktualisiert, da es ja auf die Daten in der Datei zugreift. Wenn Sie die Excel-Datei aber verschieben oder löschen, kann das Word-Dokument nicht mehr auf die Daten zugreifen, und die Tabelle kann nicht mehr angezeigt werden.

- ✗ Objekt-Pakete. Soll das Objekt im Dokument des Containers nur als Symbol angezeigt werden, wird es als Objekt-Paket übergeben. Die Darstellung als Symbol legt der Server fest.

Die Programmierung von OLE-Anwendungen ist verhältnismäßig aufwendig und kann nicht in einigen wenigen Schritten erläutert werden. Wenn Sie OLE-Anwendungen programmieren möchten, informieren Sie sich in der Online-Hilfe (Indexeintrag OLE) über das Prozedere, und verwenden Sie vor allem den MFC-Anwendungs-Assistenten als Ausgangspunkt für Ihre Projekte.

22.2 Automatisierung

Automatisierung bedeutet, daß ein Programm einen Teil seiner Funktionalität auf dem Weg über das Betriebssystem anderen Programmen zur Verfügung stellt. Es gibt also einen Server (das Programm, das seine Funktionalität in Form von Methoden und Eigenschaften zur Verfügung stellt) und einen Client, der auf diese Funktionalität zugreifen möchte.

22.2.1 Der Server

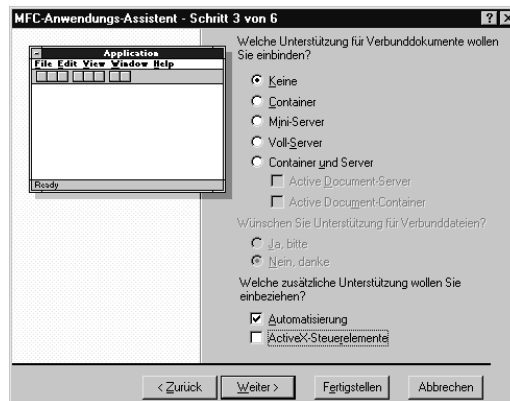
Mittler bei der Automatisierung ist das Windows-Betriebssystem. Bei diesem meldet sich der Server als erstes an. Damit das Betriebssystem die verschiedenen Server auseinanderhalten kann, müssen diese über eindeutige IDs verfügen – die sogenannten GUIDs (GUID = Globally Unique Identifier). Wenn Sie Ihre Server mit einem der passenden Assistenten erstellen, wird diese GUID automatisch für Sie erzeugt und dem Server zugewiesen; ansonsten können Sie sich diese IDs auch von dem GUID-Generator (*Guidgen.exe*) erzeugen und in die Zwischenablage kopieren lassen.

Die Automatisierung selbst erfolgt auf dem Weg über spezielle von COM vorgegebenen Schnittstellen (Schnittstellen sind grob gesagt Klassen mit ausschließlich abstrakten Methoden vergleichbar). Wenn Sie den Anwendungs- und Klassen-Assistenten zur Automatisierung verwenden, wird Ihnen die Arbeit mit diesen Schnittstellen (einschließlich der Erstellung einer passenden Typbibliothek) weitgehend abgenommen.

Übung 22-1: Erstellung eines Servers

Die folgende Ausführung beschreibt die Erstellung eines einfachen Servers, der eine Methode einer speziellen Klasse automatisiert.

Bild 22.1:
Unterstützung
für Automa-
tisierung ein-
schalten



1. Erstellen Sie das Programmgerüst. Rufen Sie den Befehl DATEI/NEU auf, und wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN (EXE) zum Anlegen einer SDI-Anwendung aus.

Aktivieren Sie im Schritt 3 die Option AUTOMATISIERUNG. Der Anwendungs-Assistent richtet Ihre Anwendung dann als Automatisierungs-Server ein und bereitet zusätzlich die Dokumentklasse für die Automatisierung vor.

2. Legen Sie eine neue automatisierungsfähige Klasse an. Statt der Dokumentklasse wollen wir eine eigene Klasse automatisieren.

Rufen Sie dazu den Klassen-Assistenten auf, und klicken Sie auf den Schalter KLASSE HINZUFÜGEN/NEU.

Geben Sie einen Namen für die Klasse ein (beispielsweise *Meldung*), und wählen Sie CCMDTARGET als Basisklasse aus (beachten Sie, daß nicht alle angebotenen Basisklassen automatisierungsfähig sind).

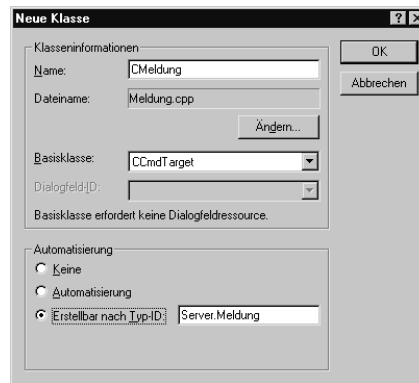


Bild 22.2:
Anlegen einer
automatisie-
rungsfähigen
Klasse

Aktivieren Sie die Option ERSTELLBAR NACH TYP-ID, und notieren Sie sich die dort von Ihnen spezifizierte ID für den späteren Aufruf aus der Client-Anwendung.

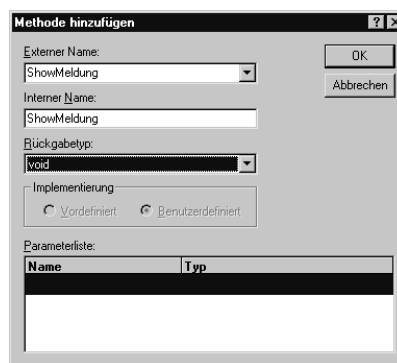


Bild 22.3:
Automatisie-
ren einer
Methode

3. Zur Automatisierung einer Methode wechseln Sie im Klassen-Assistenten zur Seite AUTOMATISIERUNG.

Wählen Sie im Feld KLASSENNAME die gerade angelegte Klasse aus, und drücken Sie den Schalter METHODE HINZUFÜGEN. (Übrigens: Die Automatisierung von Datenelementen (Eigenschaften genannt) läuft im wesentlichen ähnlich ab.)

Geben Sie einen – frei wählbaren – externen Namen für die Methode an.

Wählen Sie VOID als Rückgabetyt aus.

Schicken Sie das Dialogfeld ab.

4. Implementieren Sie die automatisierte Methode. In unserem Beispiel soll die Methode nur einfach eine Meldung ausgeben.

```
void Meldung::ShowMeldung()
{
    MessageBox(0, "Server-Methode wurde aufgerufen!",
               "Meldung", MB_OK);
}
```

5. Rufen Sie die automatisierte Methode aus dem Server heraus auf. Wird der Server direkt aufgerufen (und nicht von einer Client-Anwendung), soll er bei einem Klick mit der linken Maustaste die automatisierte Methode ShowMeldung() aufrufen. Dafür ist natürlich nicht der Umweg über das Betriebssystem erforderlich, aber es ist ein guter Test, um zu sehen, ob die Methode überhaupt das macht, was sie soll.

Sorgen Sie dafür, daß Konstruktor und Destruktor der automatisierten Klasse sowie die Deklaration der automatisierten Methode in public-Abschnitten liegen.

Nehmen Sie die Header-Datei der automatisierten Klasse in die Liste der von der Quelltextdatei der Ansichtsklasse eingebundenen Header auf.

```
// ServerView.cpp : Implementierung der Klasse CServerView
//
#include "stdafx.h"
#include "Server.h"
#include "Meldung.h"
```

Richten Sie in der Ansichtsklasse eine Behandlungsmethode für die Windows-Nachricht WM_LBUTTONDOWN ein, und rufen Sie in der Behandlungsmethode die Meldung()-Funktion auf.

```
void CServerView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CMeldung myMeldung;
    myMeldung.ShowMeldung();

    CView::OnLButtonDown(nFlags, point);
}
```

- Führen Sie den Server einmal aus, damit er sich selbst unter Windows registriert.

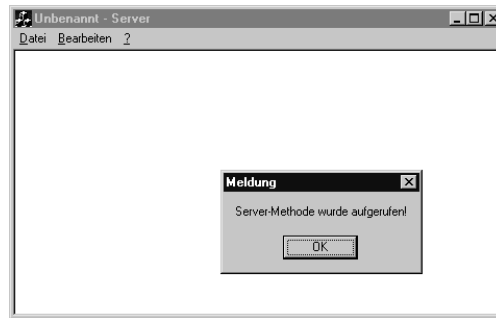


Bild 22.4:
Aufruf der
Methode ohne
Automatisie-
rung

Die Registrierung des Servers können Sie mit Hilfe des Windows-Registrierungseditors (*regedit.exe*) kontrollieren. Suchen Sie einfach nach dem Wert »Server.Meldung« (bzw. nach Ihrer Eingabe im Feld ERSTELLBAR NACH TYP-ID, siehe Schritt 2).



22.2.2 Der Client

Damit der Client auf die automatisierten Methoden eines Servers zugreifen kann, muß er sich zuerst einen Zeiger auf die Dispatch-Schnittstelle des Servers besorgen. Dies erledigt die `COleDispatchDriver`-Methode `CreateDispatch()`. Das dazugehörige `COleDispatchDriver`-Objekt erzeugt man wiederum am einfachsten mit Hilfe des Klassen-Assistenten und der Typbibliothek(en) des Servers.

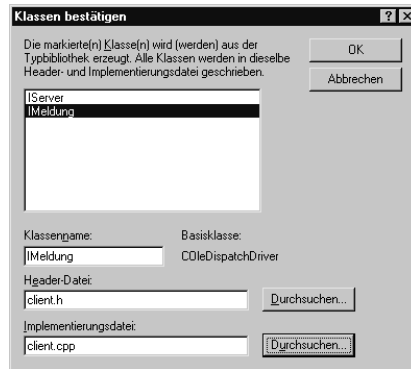
Übung 22-2: Erstellung eines Clients

Die folgende Ausführung beschreibt die Erstellung eines einfachen Clients, der die automatisierte Methode des Servers aus Übung 22-1 aufruft.

- Erstellen Sie das Programmgerüst. Rufen Sie den Befehl DATEI/NEU auf, und wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN (EXE) zum Anlegen einer SDI-Anwendung aus.

Aktivieren Sie die Option AUTOMATISIERUNG im Schritt 3.

Bild 22.5:
Anlegen einer
Proxy-Klasse



2. Legen Sie eine *Proxy-Klasse* für die automatisierte Klasse des Servers an. Dies geschieht mit Hilfe des Klassen-Assistenten.

Klicken Sie im Klassen-Assistenten auf den Schalter **KLASSE HINZUFÜGEN/AUS TYPBIBLIOTHEK**.

Wählen Sie im erscheinenden Dialogfeld die Typbibliothek (*.tlb*) des Servers aus.

Wählen Sie im Dialogfeld **KLASSE BESTÄTIGEN**, den Eintrag **IMELDUNG** aus, und spezifizieren Sie als Quelldateien für die anzulegende Klasse beispielsweise die Quelldateien des Anwendungsobjekts der Client-Anwendung.

3. Rufen Sie die automatisierte Methode auf. Dies geschieht wiederum mit Hilfe des Klassen-Assistenten.

Richten Sie in der Ansichtsklasse eine Behandlungsmethode für die Windows-Nachricht **WM_LBUTTONDOWN** ein.

In der Methode wird zuerst ein **IDispatch**-Zeiger für den Zugriff auf die automatisierte Methode erzeugt. Dazu wird die **COleDispatchDriver**-Methode **CreateDispatch()** mit dem Namen des Automatisierungs-Objekts als Argument aufgerufen.

Danach kann die Methode aufgerufen werden.

Zum Schluß wird der Schnittstellenzeiger durch Aufruf der Methode **ReleaseDispatch()** freigegeben.

```

void CClientView::OnLButtonDown(UINT nFlags, CPoint point)
{
    IMeldung myMeldung;
    if(!myMeldung.CreateDispatch("Server.Meldung")) {
        AfxMessageBox("Server nicht gefunden");
    }
    myMeldung.ShowMeldung();
    myMeldung.ReleaseDispatch();
    CView::OnLButtonDown(nFlags, point);
}

```

4. Führen Sie das Programm aus.

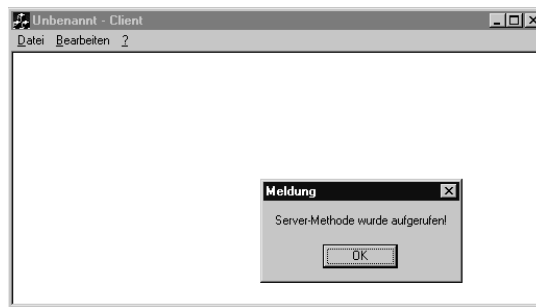


Bild 22.6:
Aufruf der
Server-
Methode in
der Client-
Anwendung

22.3 Zusammenfassung

COM ist ein binärer Standard, der festlegt, auf welche Weise Windows-Anwendungen miteinander kommunizieren können. COM definiert keine eigenen Klassen, sondern nur Regeln. Die MFC dagegen definiert etliche Klassen, die COM unterstützen.

Auf COM basieren

- ✗ OLE (das Einbetten und Verknüpfen von Objekten)
- ✗ Automatisierung (der Zugriff auf binären Code anderer Anwendungen)
- ✗ ActiveX-Steuerelemente (in sich abgeschlossene binäre Programmkomponenten, die in andere Anwendungen eingebaut werden können)

Multithreading

Unter Windows 3.x konnte wie unter DOS immer nur ein Programm gleichzeitig ausgeführt werden. Die Windows-Oberfläche mit den vielen Fenstern und den verschiedenen aufgerufenen Anwendungen erweckte zwar den Eindruck, die Anwendungen könnten parallel ausgeführt werden, doch dem war nicht so. Wurde eine Anwendung ausgeführt, druckte man beispielsweise gerade einen längeren Text aus, waren alle anderen Anwendungen zum Stillstand verdammt, bis der Druckjob beendet war.

Unter den 32-Bit-Betriebssystemen Win 95 und Win NT können Anwendungen nun tatsächlich gleichzeitig ausgeführt werden. (Das Betriebssystem regelt dies, indem es den einzelnen Anwendungen kurze Zeitscheiben zuteilt und den Anwendungen dann der Reihe nach jeweils für die Dauer ihrer Zeitscheiben die Kontrolle über die CPU gibt.) Man bezeichnet diese gleichzeitige Ausführung von Anwendungen auch als Multitasking. Ein echtes 32-Bit-Betriebssystem kann aber noch mehr.

Unter Win 95 und Win NT kann ein einziges Programm mehrere Handlungsfäden erzeugen, die dann parallel zueinander ausgeführt werden. Muß ein Programm einen zeitintensiven Job erledigen (eine aufwendige Berechnung oder eine Druckausgabe), kann es für diesen Job einen eigenen Handlungsfaden erzeugen, der diesen Job erledigt, während der Anwender schon wieder im Programm weiterarbeiten kann. Diese Handlungsfäden nennt man im Englischen *Threads*, und die Unterstützung mehrerer Threads bezeichnet man als *Multithreading*.

Wie man ein Programm dazu bringt, weitere Threads zu erzeugen und auszuführen, wollen wir uns in diesem Kapitel kurz anschauen.



Sie lernen in diesem Kapitel:

- ✗ Was Threads sind
- ✗ Wie Windows das Multithreading unterstützt
- ✗ Wie man eigene Threads erzeugt
- ✗ Wie man Threads synchronisiert

23.1 Allgemeines

Vorab noch ein Wort zur Terminologie.

- ✗ 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. Ein Programm ist ein Programm. Ein Programm kann mehrfach aufgerufen werden, wobei jeder Aufruf des Programms dazu führt, daß der Code des Programms in den Arbeitsspeicher geladen und ausgeführt wird. Jede Ausführung des Programmcodes entspricht einem Task.
- ✗ In Win32 spricht man dagegen von *Prozessen* und *Threads*. Jede Ausführung eines Programms entspricht nun einem Prozeß, und jeder Prozeß verfügt automatisch über einen Thread, der den eigentlichen auszuführenden Handlungsfaden bezeichnet.



Unter Win32 werden Nachrichten an Threads gesendet, und Threads sind es, die sich die Kontrolle über die CPU teilen.

Der wesentliche Unterschied zwischen Threads und Tasks ist, daß Threads selbst neue Threads erzeugen können (wobei erzeugter und erzeugender Thread dem gleichen Prozeß angehören). Da alle erzeugten Threads am Multitasking teilnehmen, hat eine Anwendung damit die Möglichkeit, zeit-aufwendige Routinen (beispielsweise das Ausdrucken eines Textes) als Thread abzuspalten, so daß die Anwendung, genauer gesagt ihr Hauptthread, während des Druckens weiter ausgeführt werden kann.

Die Verwaltung der Threads unter Win32 sieht dabei so aus, daß der Windows-Manager jedem Thread eine Zeitscheibe zuteilt, die bestimmt, wie lange der Thread über die CPU verfügen kann, wenn er die Kontrolle über letztere vom Windows-Manager zugesprochen bekommt. Sorgt der Prozeßmanager dann noch dafür, daß alle laufenden Threads der Reihe nach

die Kontrolle über die CPU erhalten, entsteht der Eindruck der parallelen Verarbeitung mehrerer Programme.

Dies ist allerdings nur ein vereinfachtes Modell, das so umgesetzt die Performance des Systems stark herabsetzen würde. (Meist haben Windows-Anwender mehrere Anwendungen geöffnet, arbeiten aber nur mit einem Programm. Nach obigem Modell würden dann alle geöffneten Anwendungen gleichlang über die CPU verfügen und die einzig arbeitende Anwendung würde quälend langsam ablaufen.) Windows paßt das Modell daher in mehreren Punkten an:

- ✕ Threads bekommen Prioritäten zugewiesen.
- ✕ Das Betriebssystem kann Prioritäten dynamisch anpassen.
- ✕ Untätige Threads werden in den Hintergrund gedrängt.

Prozesse höherer Priorität haben Vorrang vor Prozessen niedrigerer Priorität. Dies bedeutet allerdings nicht, daß sie zwangsläufig mehr Prozessorzeit zugewiesen bekommen. Ein Prozeß hoher Priorität, der keine Arbeit zu verrichten hat, wird in den Hintergrund gedrängt. Empfängt dieser Prozeß dann aber eine zu verarbeitende Nachricht, entfernt Windows sofort den aktuell laufenden Prozeß (niedrigerer Priorität) aus der CPU und übergibt diese dem Prozeß höherer Priorität.

Des weiteren unterliegen alle Prozesse der Prioritätsklasse »Normal« einer Sonderbehandlung (»Normal« ist die Standardpriorität für alle unter Windows erzeugten Prozesse). Wenn mehrere Prozesse normaler Priorität gleichzeitig laufen und arbeiten (beispielsweise, wenn der Anwender mit einem Grafikprogramm arbeitet, während gleichzeitig ein Text ausgedruckt und eine zeitaufwendige Datenbankrecherche durchgeführt wird), wird der Prozeß, mit dem der Anwender gerade arbeitet, als Vordergrundprozeß bezeichnet, und die Prioritätsstufen seiner Threads werden heraufgesetzt. Dies hat den Vorteil, daß dieser Prozeß schneller auf Benutzereingaben reagieren kann.

23.2 Threads erzeugen

Die einfachste Vorstellung von einem Thread ist eine Funktion, die nach ihrem Aufruf parallel zum Hauptprogramm abläuft. Wenn Sie also ein Programm schreiben, das zwei Unterfunktionen aufruft, eine zum Laden eines großen Bitmaps und eine zweite zum Zeichnen einer Freihandlinie mit der Maus, so müßten Sie zuerst warten, bis das Bitmap geladen ist, ehe Sie mit der Maus eine Linie einzeichnen können. Wenn Sie die Ladefunktion als Thread definieren, können Sie mit dem Zeichnen schon während des Lade-

vorgangs beginnen. Das Beste daran ist, daß die Ausführung des Threads den Ablauf der Zeichenfunktion nicht merklich verzögert (was daran liegt, daß die einzelnen Threads unter Win32 sehr kurze Zeitscheiben zugeteilt bekommen und sich schnell abwechseln).

Das folgende Beispiel hält am Konzept der »Thread-Funktion« fest, d.h., es definiert einen Thread als Funktion – allerdings mit besonderen Charakteristika:

- ✗ Der Rückgabewert der Funktion muß `UINT` sein.
- ✗ Die Funktion spezifiziert genau einen Parameter vom Typ `LPCVOID`.
- ✗ Die Funktion muß global definiert werden.
- ✗ Der Thread wird durch Aufruf der Funktion `AfxBeginThread()` gestartet.
- ✗ Der Thread beendet sich selbst, wenn die Thread-Funktion zurückkehrt (`return`-Anweisung) oder `AfxEndThread()` aufruft.

Übung 23-1: Fraktale in Threads berechnen

Das folgende Programm erzeugt beim Klick mit der linken Maus einen neuen Thread, der eine Julia-Menge berechnet und in das Fenster der Anwendung ausgibt. Im Gegensatz zu dem Fraktale-Programm aus Kapitel 12 kann man während der Erzeugung des Fraktals weiter mit dem Programm arbeiten – beispielsweise das Fenster der Anwendung verkleinern, das Zeichnen per Klick mit der Maus neu anstoßen, das Programm beenden.

1. Erstellen Sie das Programmgerüst. Rufen Sie den Befehl `DATEI/NEU` auf, und wählen Sie den `MFC-ANWENDUNGS-ASSISTENTEN (EXE)` zum Anlegen einer SDI-Anwendung aus.
2. Nehmen Sie die Header-Datei `<afxmt.h>` in die Header-Datei `stdafx.h` auf.
3. Legen Sie die mit Hilfe des Befehls `PROJEKT/DEM PROJEKT HINZUFÜGEN/NEU` eine neue Quelltextdatei und eine Header-Datei für die Thread-Funktion an.
4. In der Header-Datei des Threads deklarieren Sie die Thread-Funktion und ein `CEvent`-Ereignis, das später zur Synchronisierung von Thread und Anwendung (Hauptthread) benötigt wird.

Beachten Sie, daß das Ereignis als extern deklariert wird.

```
// ThreadFkt.h   Header-Datei für den Thread
extern CEvent g_Exit;
UINT JuliaMenge(LPVOID hWnd);
```

5. Thread-Funktionen definieren. Den Algorithmus zur Berechnung der Julia-Menge kennen Sie bereits aus Kapitel 12.

Nehmen Sie die benötigten Header-Dateien in die Quelltextdatei der Threadfunktion auf, und deklarieren Sie das CEvent-Objekt, das in der Header-Datei als extern deklariert wurde.

```
// ThreadFkt.cpp   Quelltextdatei für den Thread
#include "stdafx.h"
#include <complex>
#include "ThreadFkt.h"
using namespace std;
CEvent g_Exit;
```

Als Parameter erwartet die Thread-Funktion einen void-Zeiger. Beim Aufruf wird diesem Parameter ein Zeiger auf das aufrufende Ansichtsfenster übergeben. Über diesen Zeiger beschafft sich der Thread einen Gerätekontext für das Fenster und die aktuellen Fenstermaße.

Danach wird die Julia-Menge pixelweise berechnet. Die einzelnen Pixel werden mit Hilfe der CDC-Methode SetPixel() eingefärbt.

```
UINT JuliaMenge(LPVOID pWnd)
{
    CClientDC dc((CWnd*) pWnd);
    RECT rect;
    ((CWnd *) pWnd)->GetClientRect(&rect);
    complex<double> c(-0.012, 0.74);
    for(int i = rect.left; i < rect.right; i++)
        for(int j = rect.top; j < rect.bottom; j++)
        {
            complex<double> x(0.0001 * i, 0.0001 * j);
            for(int n = 0; n < 100; n++)
            {
                if(abs(x) > 100)
                    break;
                x = pow(x, 2) + c;
            }
        }
}
```

```

        if(abs(x) < 1)
            dc.SetPixel(i, j, RGB(0, 0, 255));
        else
            dc.SetPixel(i, j, RGB(2 * abs(x), 255, 255));
    }
    AfxEndThread(0); // noetig wegen m_bAutoDelete = false
    return 0;
}

```

6. Thread aufrufen. Richten Sie eine Behandlungsmethode für die Windows-Nachricht WM_LBUTTONDOWN ein.

Nehmen Sie zuerst in die Quelltextdatei der Ansichtsklasse die Header-Datei des Threads auf.

```

// ThreadsView.cpp : Implementierung der Klasse CThreadsView
//
#include "stdafx.h"
#include "Threads.h"
#include "ThreadFkt.h"

```

Deklarieren Sie in Ihrer Ansichtsklasse einen CWinThread-Zeiger für den Thread.

```

class CThreadsView : public CView
{
...
// Attribute
public:
    CThreadsDoc* GetDocument();
    CWinThread* m_myThread;

```

Initialisieren Sie den Thread-Zeiger im Konstruktor der View zu NULL.

```

CThreadsView::CThreadsView()
{
    // ZU ERLEDIGEN: Hier Code zur Konstruktion einfügen,
    m_myThread = NULL;
}

```

Starten Sie in der WM_LBUTTONDOWN-Behandlungsmethode den Thread. Übergeben Sie dazu der AfxBeginThread()-Methode die Adresse der Thread-Funktion, ein Argument für den Parameter der Thread-Funktion (in diesem Beispiel der this-Zeiger der Ansichtsklasse) und eine Priorität (hier sollten Sie üblicherweise THREAD_PRIORITY_NORMAL verwenden, damit Windows die Priorität selbst nach Bedarf anpassen kann).

```

void CThreadsView::OnLButtonDown(UINT nFlags, CPoint point)
{
    m_myThread = AfxBeginThread(JuliaMenge, this,
                                THREAD_PRIORITY_NORMAL);
    CView::OnLButtonDown(nFlags, point);
}

```

7. Ereignis für die Synchronisierung definieren. Um zu verhindern, daß mehrere Threads gleichzeitig ablaufen, wenn der Anwender mehrfach mit der linken Maus in das Fenster der Anwendung klickt, ist es erforderlich, den laufenden Thread zu stoppen, bevor ein neuer Thread gestartet wird. Dies geschieht mit Hilfe eines Ereignisses, das von der Ansichtsklasse in den Zustand Signalled versetzt und vom Thread überwacht wird.

Deklarieren Sie das Ereignis als globale CEvent-Instanz (ist bereits in den Schritten 4 und 5 geschehen).

Beobachten Sie das Ereignis in der Thread-Funktion. Rufen Sie dazu die API-Funktion WaitForSingleObject() auf. Übergeben Sie dieser als ersten Parameter das zu überwachende CEvent-Objekt, als zweiten Parameter einen Wert, der angibt, wie lange die Funktion auf den Eintritt des Ereignisses warten soll (hier 0, damit die Funktion direkt zurückkehrt; dafür wird die Funktion in die innere Schleife des Threads gesetzt, um fortlaufend den Zustand des Ereignisses abzufragen). Wurde das Ereignis auf Signalled gesetzt, liefert die Funktion WaitForSingleObject() den Wert WAIT_OBJECT_0 zurück, und wir beenden die Thread-Funktion.

```

UINT JuliaMenge(LPVOID pWnd) {
    ...
    for(int i = rect.left; i < rect.right; i++)
        for(int j = rect.top; j < rect.bottom; j++)
        {
            if(WaitForSingleObject(g_Exit, 0) == WAIT_OBJECT_0)
                return 1;
        }
    ...
}

```

Ereignis auslösen. Wurde die linke Maustaste gedrückt, während noch ein Thread ausgeführt wird, wird das Ereignis gesetzt (Methode SetEvent()) und darauf gewartet, daß der Thread sich auch tatsächlich beendet hat (Funktion GetExitCodeThread()). Danach wird das Fenster neu gezeichnet und das Thread-Objekt gelöscht.

Hat sich der Thread durch Aufruf der Funktion `AfxEndThread(0)` beendet, ist der Rückgabewert 0. In diesem Fall wird das Ereignis zurückgesetzt (Methode `ResetEvent()`).

Standardmäßig werden Threads automatisch gelöscht. Um dies zu verhindern, setzen Sie nach Erzeugen des neuen Threads dessen Datenelement `m_bAutoDelete` auf `FALSE`. Sie müssen dann aber auch das Thread-Objekt korrekt löschen.

```
void CThreadsView::OnLButtonDown(UINT nFlags, CPoint point)
{
    DWORD exitcode;
    if(m_myThread) {
        g_Exit.SetEvent();
        do GetExitCodeThread(m_myThread->m_hThread, &exitcode);
            while (exitcode == STILL_ACTIVE);
        if(exitcode == 0)
            g_Exit.ResetEvent();
        Invalidate();
        UpdateWindow();
        delete(m_myThread);
        m_myThread = NULL;
    }

    m_myThread =
        AfxBeginThread(JuliaMenge, this, THREAD_PRIORITY_NORMAL);
    m_myThread->m_bAutoDelete = FALSE;
    CView::OnLButtonDown(nFlags, point);
}
```

8. Löschen Sie das Thread-Objekt. Dies ist nur notwendig, wenn Sie die Eigenschaft `m_bAutoDelete` des Threads auf `FALSE` gesetzt haben.

```
CThreadsView::~CThreadsView()
{
    if(m_myThread)
        delete(m_myThread);
}
```

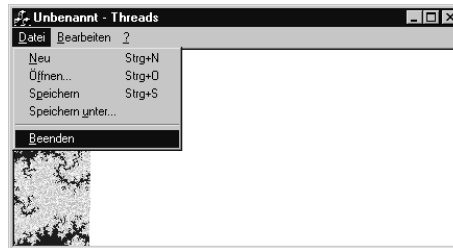


Bild 23.1:
Ausführung
des Threads

Die Ausgabe des Threads wird nicht an die Größe des Fensters angepaßt, wenn diese durch den Anwender verändert wird. Um dem abzuhelpen, können Sie die WM_SIZE-Nachricht abfangen und als Antwort darauf den Thread neu starten.



23.3 Zusammenfassung

Unter Windows 95/NT können nicht nur mehrere Anwendungen gleichzeitig ausgeführt werden, auch eine einzige Anwendung kann mehrere Handlungsfäden gleichzeitig ausführen lassen. Der einfachste Weg, einen untergeordneten Thread in einer Anwendung abzuspalten, ist, den Code des Threads in eine Funktion der Signatur `UINT FuncName(LPVOID)` zu kapseln und die Funktion dann durch Aufruf der Methode `AfxBeginThread()` als eigenständigen Thread zu starten.

Weitere Unterstützung bieten spezielle MFC-Klassen wie `CWinThread`, mit deren Methoden man sich über den aktuellen Thread-Status informieren und die Thread-Ausführung steuern kann (vorausgesetzt, der Thread wurde mit einem `CWinThread`-Objekt verbunden), oder die MFC-Klassen zur Thread-Synchronisierung (`CEvent`, `CMutex`, `CCriticalSection`, `CSemaphore`, `CSingleLock` und `CMultiLock`).

Internet- Programmierung

Was bedeutet eigentlich Internet-Programmierung? Das Internet besteht aus Tausenden von vernetzten Computern und ist damit das größte existierende Wide Area Network (WAN), ja es ist das Netzwerk schlechthin. Jeder einzelne am Netz beteiligte Computer kann dabei durchaus wie ein eigenständiger Personalcomputer behandelt werden (umgekehrt kann jeder ans Netz angeschlossene PC mit entsprechender Software als Server eingerichtet werden). So betrachtet macht es keinen Unterschied, ob Sie ein Programm auf einem PC oder einem Internet-Server ablaufen lassen.

Das Besondere am Netz und an der Internet-Programmierung ist erst der Austausch von Daten zwischen den vernetzten Computern (oder auch zwischen Servern und über Modem angeschlossenen PCs). Um diesen Datenverkehr zu regeln, bedarf es einer Reihe von Protokollen.

Dem Anwender bleiben diese Protokolle (TCP/IP, HTTP, FTP) allerdings weitgehend verborgen, da er entweder mit den verschiedenen im Internet verfügbaren Diensten (E-Mail, FTP etc.) arbeitet, die auf diesen Protokollen aufbauen, oder auf einer noch höheren Ebene mit Programmen (Webbrowser, Gopher) arbeitet, die wiederum die bequeme Arbeit mit diesen Diensten erlauben.

Inwieweit Sie als Programmierer mit diesen Protokollen bekannt sein müssen, hängt im wesentlichen davon ab, auf welcher Ebene der Internet-Programmierung Sie einsteigen wollen. In diesem Kapitel werden wir auf zwei Ebenen einsteigen. Beginnen werden wir auf einer mittleren Ebene mit den



MFC-Internet-Klassen `CInternetConnection`, `CHttpConnection` etc. (Die darunterliegenden Ebenen der Winsock-API und der Winsock-Klassen lassen wir aus.) Schließen werden wir mit der höchstmöglichen Ebene, mit der Ansichtsklasse `CHtmlView`, und der Implementierung eines eigenen Browsers.

Sie lernen in diesem Kapitel:

- ✗ Die Internet-Protokolle FTP, Gopher und HTTP kennen
- ✗ Die WinInet-Klassen der MFC kennen
- ✗ Wie man eine Internet-Verbindung mit Hilfe der WinInet-Klassen herstellt
- ✗ Wie man mit Hilfe von `CHtmlView` einen eigenen Browser schreiben kann
- ✗ Was Dialogleisten sind

24.1 Die Internet-Protokolle

Wir werden uns hier nicht mit der ganzen Bandbreite der Internet-Protokolle auseinandersetzen, auch nicht mit den wichtigsten (denn hierzu gehörte mit Sicherheit TCP/IP), sondern den Protokollen, mit denen wir uns bei der Programmierung befassen müssen und die von der MFC unterstützt wären, als da wären:

- ✗ FTP
- ✗ Gopher
- ✗ HTTP

24.1.1 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:

```

C:\>ftp vtt1
Connected to vtt1.
220 vtt1 FTP server (Version wu-2.4(1) Sat Feb 18 13:40:36 CST 1995) ready.
User (vtt1:(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@vtt1
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.)

Die beiden wichtigsten Befehle sind *get* und *put*, mit denen man Dateien vom Server zum Client oder vom Client zum Server kopieren kann. Das Protokoll bietet aber auch einige Befehle zum Manipulieren des Dateisystems, die zum Löschen von Dateien sowie zum Erzeugen, Entfernen und Wechseln von Verzeichnissen verwendet werden. Besitzer einer Homepage wird dies wahrscheinlich alles von der Übertragung ihrer Web-Dateien auf den Webserver bekannt sein.

24.1.2 Das Gopher-Protokoll

Gopher ist ein Internet-Protokoll, das 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 Auswählen eines Menüpunkts verbindet den User mit anderen Gopher-Servern, wo er entweder Dokumente herunterladen oder in weiteren Menüs navigieren kann.

In der Vergangenheit war Gopher-Client-Software typischerweise textbasiert. Heutzutage können die meisten Webclients wie Netscape oder Internet Explorer mit Gopher-Servern Verbindungen aufstellen. Sie machen somit andere Gopher-Programme überflüssig, ja in gewisser Weise sogar das Gopher-System, denn dieses hat durch die Ausbreitung des Webs im Internet stark an Bedeutung und Popularität verloren.

24.1.3 Das Hypertext Transfer Protokoll HTTP

Das bei weitem populärste Protokoll im Internet ist heutzutage HTTP. Dieses Protokoll wird im gesamten World Wide Web verwendet, um Multimedia-Informationen als Hypertext Markup Language (HTML)-Dokumente, kurz Web-Seiten, zu versenden.

Zum Web und auch zum HTTP-Protokoll braucht man heute wohl nicht mehr viel zu erklären. Zudem reicht es für unsere Ansprüche vollkommen, wenn man weiß, daß ein URL (Uniform Resource Locator) so etwas wie die Adresse einer Datei im Web ist und aus

- ✗ der Angabe des Protokolls: *http://*
- ✗ dem Namen des Servers – beispielsweise *localhost* für die lokale Maschine
- ✗ und dem Pfad zur gewünschten HTML-Datei – beispielsweise *mein-Web/index.html*

besteht.

Das tatsächliche Protokoll ist natürlich viel komplexer und transportiert weit mehr Informationen als nur die Adressen von angeforderten Web-Seiten. Doch dies ist mehr das Aufgabenfeld der CGI-Programmierer und soll uns hier nicht belasten.

Schauen wir lieber, wie man eine HTTP-Verbindung zu einem Webserver aufbauen kann.

24.2 Aufbau von Internet-Verbindungen

Wie schon angedeutet, werden wir nicht bis auf die Winsock-API zurückgehen, um zu sehen, wie man Internet-Verbindungen aufbaut. Uns soll es reichen, mit den WinInet-Klassen der MFC zu arbeiten. Diese Klassen teilen sich in vier Aufgabengebiete auf:

Klasse	Aufgabe
CInternetSession	Jede Internet-Sitzung beginnt mit der Einrichtung eines Objekts dieser Klasse (die Klasse kann direkt instantiiert oder als Basisklasse verwendet werden).
CURLConnection CFTPConnection CGopherConnection	Diese Klassen dienen der Herstellung einer Internet-Verbindung auf der Grundlage des jeweiligen Protokolls. Die Klassen werden dabei nicht direkt instantiiert, vielmehr lässt man sich mit Hilfe der zugehörigen CInternetSession-Methoden (GetURLConnection(), GetFTPConnection(), GetGopherConnection()) Objekte der Klassen zurückliefern.
CURLConnection CFTPFile CGopherFile	Diese Klassen dienen der Übertragung der Daten über die Internet-Verbindung, wozu sie mit speziellen Methoden ausgestattet sind.
CInternetException	Zum Abfangen eventuell auftretender Ausnahmen und Fehler, die man auch nutzen kann, um den Anwender über bestimmte Probleme bei Herstellung der Verbindung, beispielsweise nicht gefundene Web-Seiten, zu informieren.

Tabelle 24.1:
Die wichtigsten WinInet-Klassen

Umgesetzt in einer Konsolenanwendung, die Sie mit dem Kommandozeilenaufruf

```
cl /mt /gx httpget.cpp
```

kompilieren können, stellt sich die gesamte Internet-Sitzung wie folgt dar:

```
#include <afxinet.h>
#include <iostream.h>
void main(void)
{
    CInternetSession is("HTTPGET");
    CURLConnection *pHC = NULL;
    CURLConnection *pHF = NULL;
    try
    {
        pHC = is.GetURLConnection(_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;
}

```

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 das Auswerten und Formatieren von HTML-Dokumenten. Keine Funktion bietet die Möglichkeit, die in einem Dokument angezeigten URLs zu bearbeiten. Trotzdem muß man nicht gleich selbst einen Parser für HTML schreiben, wenn man Web-Seiten in ansprechender Formatierung statt als HTML-Code anzeigen möchte. Abhilfe schafft hier die Ansichtsklasse `CHtmlView`.

24.3 Erstellung eines Webbrowsers

Bisher haben ich Ihnen gezeigt, wie man mit Hilfe des MFC-Anwendungs-Assistenten einen SDI-Texteditor oder einen MDI-Texteditor aufsetzen kann. Beides war nicht allzu schwierig, und so dürfen wir hoffen, daß wir ähnlich wenig Mühe mit einem Web-Seiten-Browser haben werden.

Übung 24-1: Erstellung eines Webbrowsers

1. Legen Sie ein neues Projekt an (Befehl DATEI/NEU, Seite PROJEKTE).

Geben Sie als Namen »Browser« ein. Wählen Sie ein passendes übergeordnetes Verzeichnis für das Projekt, und lassen Sie einen neuen Arbeitsbereich für das Projekt anlegen.

Links wählen Sie den MFC-ANWENDUNGS-ASSISTENTEN aus.

2. Im ersten Schritt des Assistenten entscheiden Sie sich für eine SDI-Anwendung mit Dokument/Ansicht-Architektur.
3. Jetzt kommt die wichtigste Einstellung überhaupt! Im sechsten Schritt klicken Sie oben auf die Klasse für das Ansichtsfenster und wählen als Basisklasse nicht mehr CVIEW, sondern CHtmlView aus.

Läßt man das Programm jetzt ausführen, kann es nicht viel mehr, als die Microsoft-Website <http://www.microsoft.com/visualc/> anzusteuern – vorausgesetzt, man hat zuvor eine Internet-Verbindung via Netzwerk oder DFÜ-Netzwerk hergestellt. Damit wollen wir uns aber nicht zufrieden geben. Zuerst wollen wir das Laden der Startseite unterbinden, und dann wollen wir das Laden von HTML-Dateien von der Festplatte erlauben.

4. Das Unterbinden der Startseite ist schnell erledigt. Springen Sie mit Hilfe der KLASSEN-ANSICHT des Arbeitsbereichsfensters in die Methode OnInitialUpdate() der Ansichtsklasse, und löschen Sie den Aufruf

```
Navigate2(_T("http://www.microsoft.com/visualc/"),
        NULL, NULL);
```

oder setzen Sie eine andere URL ein – vielleicht *www.mut.de*?

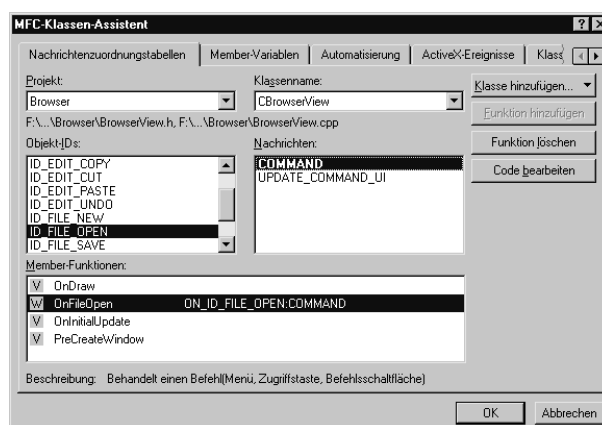


Bild 24.1:
Datei/Öffnen-
Befehl bearbeiten

5. Rufen Sie jetzt den Klassen-Assistenten auf, und richten Sie eine eigene Behandlungsmethode zu dem DATEI/ÖFFNEN-Befehl (Ressourcenbezeichner ID_FILE_OPEN) ein.

In der Methode erzeugen Sie einen eigenen Öffnen-Dialog, wobei wir zur Bequemlichkeit des Anwenders Dateifilter für HTML-Dateien und beliebige Dateien anbieten. (Achten Sie darauf, daß Sie den Filterstring mit zwei geraden Strichen || abschließen.)

Dann wird der Dialog ausgeführt. Schließt der Anwender ihn über den OK-Schalter, steht in der Methode GetPathName() der vollständige Pfad der zu öffnenden Datei, und diesen Pfad brauchen wir dann nur noch der CHtmlView-Methode Navigate2() zu übergeben.

```
void CBrowserView::OnFileOpen()
{
    // TODO: Code für Befehlsbehandlungsroutine hier einfügen
    CFileDialog oeffnenDlg(true, NULL, NULL,
        OFN_HIDEREADONLY,
        "HTML-Dateien|*.htm; *.html|Alle Dateien|*.*||");
    if(oeffnenDlg.DoModal() == IDOK)
        Navigate2(oeffnenDlg.GetPathName(), 0, NULL);
}
```

Jetzt soll der Anwender aber auch noch echte URLs zum Laden von HTML-Seiten aus dem Web eingeben können. Zu diesem Zweck richten wir eine spezielle Symbolleiste, eine sogenannte Dialogleiste, die nicht nur Schaltflächen, sondern beliebige Steuerelemente enthalten kann, ein.

Bild 24.2:
Dialogleiste



6. Legen Sie eine neue Dialog-Ressource an (Befehl EINFÜGEN/RESOURCE). Klicken Sie aber nicht gleich auf NEU, sondern expandieren Sie den Dialog-Knoten, und doppelklicken Sie auf die Dialogvorlage IDD_DIALOGBAR.
7. Passen Sie die Eigenschaften der Werkzeugleiste an. Klicken Sie hierzu mit der rechten Maustaste in den Hintergrund der Dialogleiste, und rufen Sie den Befehl EIGENSCHAFTEN auf.

Wählen Sie auf der Seite FORMATE im Feld STIL den Eintrag UNTERGEORDNET und im Feld RAND die Option KEINE. Deaktivieren Sie alle weiteren Optionen. (Das sollte so ziemlich den Voreinstellungen entsprechen.)

8. Nehmen Sie die Steuerelemente in die Dialogleiste auf (siehe Abbildung 24.2).

Setzen Sie für den Schalter die Eigenschaft `STANDARDSCHALTFLÄCHE`, damit der Schalter automatisch aktiviert wird, wenn der Anwender nach der Eingabe einer URL in das Eingabefeld die Eingabetaste drückt.

9. Deklarieren Sie in der Klasse des Hauptfensters eine Elementvariable für die Dialogleiste.

```
class CMainFrame : public CFrameWnd
{
...
public:
    CDialogBar m_wndDialogBar;
```

10. Erzeugen Sie die Dialogleiste im Zuge der Erzeugung des Hauptfensters. Rufen Sie dazu den Klassen-Assistenten auf, und richten Sie in der Rahmenfensterklasse eine Behandlungsmethode für die Windows-Nachricht `WM_CREATE` ein.

Geben Sie zur Erzeugung der Dialogleiste den nachfolgenden Code ein, wobei der Parameter `CBRS_TOP` dafür sorgt, daß die Dialogleiste in den oberen Rahmen des Hauptfensters integriert wird. Mit Hilfe der Methode `SetDlgItemText()` kann man auf das Eingabefeld in der Dialogleiste zugreifen und den anfänglich anzuzeigenden Text einstellen.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO: Speziellen Erstellungscode hier einfügen
    m_wndDialogBar.Create(this, IDD_DIALOGBAR, CBRS_TOP, 1);
    m_wndDialogBar.SetDlgItemText(IDC_EDIT1, "http://");

    return 0;
}
```

Drückt der Anwender den Schalter der Dialogleiste, soll die entsprechende Datei angefordert werden. Dazu muß die `WM_COMMAND`-Nachricht für den Schalter abgefangen werden. Statt den Klassen-Assistenten zu bemühen, erledigen wir die Einrichtung der Behandlungsmethode ausnahmsweise mal selbst.

11. Deklarieren Sie die Behandlungsmethode in der Deklaration der Ansichtsklasse.

```
class CBrowserView : public CHtmlView
{
...
protected:
    afx_msg void OnRequest();
};
```

12. Erweitern Sie die Antworttabelle der Ansichtsklasse (steht in der Quelltextdatei).

```
BEGIN_MESSAGE_MAP(CBrowserView, CHtmlView)
    ON_COMMAND(IDC_BUTTON1, OnRequest)
    //{AFX_MSG_MAP(CBrowserView)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
```

13. Implementieren Sie die Behandlungsmethode in der Quelltextdatei der Ansichtsklasse. Benutzen Sie die `GetDlgItemText()`-Methode, um den Text aus dem Eingabefeld abzufragen, und rufen Sie dann die `CHtmlView`-Klasse `Navigate2()` auf, um die gewünschte Web-Seite vom Server anzufordern und anzuzeigen.

```
void CBrowserView::OnRequest()
{
    CBrowserDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CString str;
    CWnd& rBar = ((CMainFrame*)
        AfxGetApp()->m_pMainWnd)->m_wndDialogBar;
    rBar.GetDlgItemText(IDC_EDIT1, str);
    Navigate2(str, 0, NULL);
}
```

14. Die Methode `OnRequest()` verwendet den Klassenbezeichner `CMainFrame` in der expliziten Typumwandlung. Damit der Compiler nicht über diesen Bezeichner stolpert, müssen Sie noch die Header-Datei für die Rahmenfensterklasse einbinden.

```
// BrowserView.cpp : Implementierung der Klasse
CBrowserView
```

```
#include "stdafx.h"
#include "Browser.h"
#include "MainFrm.h"
```

15. Lassen Sie das Projekt jetzt fertigstellen und ausführen (**Strg** + **F5**).



Bild 24.3:
Der fertige
Browser

24.4 Zusammenfassung

In Visual C++ können Sie Internet-Programmierung auf verschiedenen Ebenen betreiben: angefangen bei der Winsock-API zur Einrichtung eigener Netzknoten über die verschiedenen MFC-Klassen (beispielsweise die WinInet-Klassen) bis zur Ansichtsklasse `CHtmlView`, mit der man im Handumdrehen einen eigenen Webbrowser implementieren kann.

Die WinInet-Klassen unterstützen die drei Internet-Protokolle FTP, Gopher und HTTP und erlauben es Ihnen, Client-Programme zu schreiben, sich über diese Protokolle mit passenden Servern kurzzuschließen sowie Daten auszutauschen.

Die Klasse `CHtmlView` kapselt dagegen bereits die vollständige Funktionalität eines Webbrowsers. Alles, was Sie tun müssen, ist, eine Benutzeroberfläche um die Klasse herum aufzubauen und die Methode `Navigate2()` zum Anfordern von Web-Seiten aufzurufen.

Datenbank- Programmierung

Als Anwender kennt man Datenbanken üblicherweise als eine Art dynamischer Tabelle, deren Aufbau durch ihre Felder (Spalten) definiert wird und deren Inhalt in Datensätzen (Zeilen) festgehalten wird. Als Datenbank-Programmierer ändert sich daran erst mal nichts, sofern Sie gewillt sind, sich das Leben leicht zu machen, und zur Realisierung Ihrer Datenbankanwendungen auf bestehende Datenbanktreiber zurückgreifen.

Natürlich braucht man nicht gleich an Datenbanken zu denken, wenn man vor dem Problem steht, größere Datenmengen in einer Anwendung zu verwalten. Je nach Umfang der Daten und Anforderung an die Verwaltung der Daten gibt es verschiedene grundlegende Datenstrukturen:

- ✕ Arrays (statisch, aber effiziente Suche, wenn das Array sortiert ist)
- ✕ Listen (dynamisch, aber ineffiziente Suchverfahren)
- ✕ Bäume (dynamisch, mit effizienten Suchverfahren in sortierten Bäumen)

Steigen aber die Anforderungen an Ihre Datenverwaltung, macht es meist keinen Sinn mehr, das Rad zum zweiten Mal zu erfinden. Statt dessen überläßt man die eigentliche Verwaltung der Daten einer bestehenden Datenbankanwendung und konzentriert sich auf die Kommunikation mit dieser. Zu diesem Zweck stellt Ihnen Visual C++ die *ODBC* (Open Database Connectivity) und passende MFC-Klassen zur Verfügung.



Sie lernen in diesem Kapitel:

- ✗ Ein wenig über Datenbanken, Tabellen, Indizes und SQL
- ✗ Etwas mehr über ODBC und die Einrichtung von Datenbankverbindung über die ODBC
- ✗ Eine ganze Menge über Datenbank-Programmierung und die Anzeige von Daten aus Datenbanken

25.1 Grundlagen der Datenbanken

Daten, Datenbanken und Tabellen

Der Begriff der Datenbank (database) hat im allgemeinen Sprachgebrauch zwei Bedeutungen. Zum einem bezeichnet er eine Sammlung von Daten, einen Datenbestand. Zum anderen bezeichnet er die Software, sprich das Programm, mit dem ein Datenbestand verwaltet und bearbeitet werden kann. In diesem Kapitel und bei der Programmierung unter Visual C++ ist unter einer Datenbank vornehmlich der Datenbestand zu verstehen.

Daten in Datenbanken werden üblicherweise in Tabellenform dargestellt. Für einige Datenbankprogramme gilt eine einfache 1:1-Entsprechung von Datenbank und Tabelle, d.h., jede Datenbank besteht aus einer Tabelle. Andere Datenbanken – und dazu gehören vor allem die relationalen Datenbanken (siehe unten) – sind allerdings aus mehreren Tabellen zusammengesetzt. Als Datenbank wird daher meist ein Verzeichnis (oder ein Alias) angegeben, in dem die Tabellen der Datenbank zu finden sind.

Aufbau von Tabellen

Bild 25.1:
Aufbau einer
Tabelle



ID	Vorname	Name	Straße	PLZ	Stadt
1	Ida	Schmitt	Villastr	93055	Regensburg
2	Hung	Vu	Egerstr	93057	Regensburg
3	Griselda	Schmitt	Saarstr	66111	Saarbrücken
4	Franz	Munz	Hollerstr	80995	München

In einer Datenbanktabelle repräsentieren die Spalten der Tabelle die Felder der Datenbank, während die Zeilen die einzelnen Datensätze enthalten. Betrachtet man als typisches Beispiel den Aufbau einer Datenbank zur Adressverwaltung, so würde man damit beginnen, durch die Definition der Fel-

der (Name, Vorname, Straße, PLZ, Stadt) die Struktur der Datenbank festzulegen.

Danach werden die Datensätze (sprich Adressen) eingegeben, wobei ein vollständiger Datensatz zu jeder Spalte der Datenstruktur einen Eintrag enthält. Ob zu einer bestimmten Spalte in jedem Datensatz ein Wert eingegeben werden muß, kann beim Aufbau der Tabellenstruktur festgelegt werden.

Indizes

Die einzelnen Datensätze sind in der Tabelle in der Reihenfolge abgespeichert, in der sie eingegeben wurden. Dieses ungeordnete Ablagesystem erschwert und verlangsamt die Suche nach bestimmten Datensätzen – einer häufigen Operation auf Datenbanken. Um dem abzuweichen, werden Tabellen indiziert. Dies bedeutet prinzipiell nichts anderes, als daß für eine bestimmte Spalte der Tabelle eine geordnete Kopie angelegt und eine Zuordnung zwischen den Einträgen der geordneten und der ungeordneten Spalte hergestellt wird (diese Informationen werden je nach Datenbanksystem in speziellen Index-Dateien oder zusammen mit den Tabellen in einer gemeinsamen Datei (.mdb-Dateien von MSAccess) gespeichert).

Im Falle der Adressenverwaltung wird man meist für einen gegebenen Namen die zugehörige Adresse suchen. Statt nun die Datensätze einzeln durchzugehen, erstellt man für das Feld Name einen Index und läßt dann direkt nach dem gewünschten Eintrag suchen. Die Datenbankanwendung sucht in der sortierten Kopie der Spalte nach dem vorgegebenen Namen. Findet sie ihn, hat sie damit auch den Verweis auf den entsprechenden Eintrag in der eigentlichen Tabelle und kann den gesuchten Datensatz mit der gewünschten Adresse anzeigen.

Ein Index muß aber nicht unbedingt aus einem einzigen Feld bestehen. Er kann auch zwei oder mehrere Felder umfassen, wobei die Felder in der Reihenfolge, in der sie angegeben wurden, zur Sortierung herangezogen werden. Auch mehrere voneinander unabhängige Indizes sind möglich, um nach verschiedenen Kriterien (beispielsweise nach Name oder nach Telefonnummer) in einer Datenbank suchen zu können. Viele Datenbankanwendungen setzen den Primärindex derart um, daß sie die Datensätze direkt in der entsprechenden Sortierung abspeichern.

Relationale Datenbanken

Das besondere Kennzeichen relationaler Datenbanken ist, daß sie die Daten mehrerer Tabellen zueinander in Beziehung setzen beziehungsweise – um es von der Seite des Anwenders zu betrachten – es ermöglichen, die zu verwaltende Information auf mehrere Tabellen aufzugliedern. Die Verbindung zwi-

schen den Tabellen wird dabei durch sogenannte Schlüsselfelder hergestellt, die indiziert werden müssen und aus mehreren Feldern zusammengesetzt sein können.

Ein Vorteil relationaler Datenbanken zeigt sich, wenn Sie viele Datensätze haben, die gleichlautende speicherintensive Einträge zu einem Feld der Datenbank besitzen. Ein Beispiel wäre eine Datenbank, in der alle Dateien auf Ihrem Computer gespeichert sind. Zu jeder Datei speichern Sie das Erstellungsdatum, den Datentyp und natürlich den Verzeichnispfad. Dabei haben alle Dateien eines Verzeichnisses den gleichen Verzeichnispfad, und dieser kann naturgemäß sehr lang sein, weswegen diese Information recht speicherintensiv ist. Hier kann man Speicher sparen, indem man in der Tabelle mit den Dateien statt des Verzeichnispfades ein Kürzel oder einen simplen Zahlenindex aufführt. Die gleichen Zahlenindizes werden dann in einer zweiten Tabelle mit den Verzeichnispfaden verbunden, wobei in dieser zweiten Tabelle jeder Verzeichnispfad nur einmal abgespeichert werden muß. Die Verknüpfung der beiden Tabellen erfolgt hier also über das Feld mit den Zahlenindizes.

Ein weiterer Vorteil der relationalen Datenbanken ist ihr modulares Konzept, das sie ohne große Mühe beliebig erweiterbar macht und dabei sparsam mit Speicherressourcen umgeht.

SQL

SQL steht für Structured Query Language und bezeichnet eine von IBM entwickelte Sprache zur Kommunikation mit relationalen Datenbanken. Anders als die Übersetzung des Akronyms andeutet, ist SQL jedoch nicht strukturiert, sondern besteht aus einer einfachen Sammlung von Datenbankbefehlen, mit denen Daten aus Datenbanken abgefragt und angezeigt, aber auch Datenbanken neu angelegt werden können. Neben der Mächtigkeit der Sprache liegt ihr Wert natürlich auch darin, daß sie einen Quasi-Standard definiert, der von den meisten Datenbank Anwendungen unterstützt wird.

25.2 ODBC – die Verbindung zur Datenbank

Bevor Sie auf die Daten einer externen Datenbank zugreifen können, müssen Sie eine Verbindung zwischen Ihrem Programm und der Datenbank herstellen. Visual C++ läßt Ihnen hierbei die Wahl zwischen ODBC und DAO.

ODCB (Open Database Connectivity)

ODBC definiert eine standardisierte Schnittstelle, über die C/C++-Programme mittels eines Standardsets von SQL-Anweisungen auf verschiedenste Datenbanken zugreifen können. Voraussetzung ist, daß ein passender ODBC-Treiber für die anzusprechende Datenbank registriert ist (siehe Übung 25-2).

Der Vorteil des ODBC-Systems liegt vor allem darin, daß Sie eine einheitliche Schnittstelle für alle möglichen Datenbanken verwenden. Ihr Programm kommuniziert in immer gleicher Weise mit der ODBC-Schnittstelle, und diese reicht die Befehle unter Zuhilfenahme der zugehörigen ODBC-Treiber dann in Form von SQL-Anweisungen an die Zieldatenbank weiter. Um also eine Datenbankanwendung, die mit einer Access-Datenbank arbeitet, zu Paradox zu portieren, brauchen Sie nur die ODBC-Treiber zu wechseln – eine Überarbeitung des Programmcodes ist üblicherweise nicht erforderlich.

Der Nachteil dieser standardisierten Schnittstelle ist, daß über ODBC lediglich ein Teil des jeweiligen Leistungsspektrums der eingebundenen Datenbanksysteme verwendet werden kann (quasi der kleinste gemeinsame Nenner), wobei spezielle Optionen und Möglichkeiten einzelner Datenbank-Systeme verloren gehen (beispielsweise bietet ODBC wenig Unterstützung für die Arbeit mit Indizes).



Zugriffen wird auf die ODBC-Schnittstelle über die Funktionen der ODBC-API oder der ODBC-Datenbankklassen der MFC (vornehmlich CDatabase und CRecordSet).

DAO (Data Access Objects) und OLE DB

DAO und OLE DB basieren im Gegensatz zu ODBC auf COM-Schnittstellen, die in entsprechenden DLLs implementiert sind. Diese Schnittstellen sind wiederum weitgehend in den zu DAO und OLE DB angebotenen MFC-Klassen gekapselt.

Datenbanken wie Microsoft Access, für die es native Treiber zur Microsoft Jet Engine gibt, sollten Sie stets DAO den Vorzug geben. Da ODBC aber universeller verwendbar ist, werden wir uns in diesem Kapitel trotz der Verwendung von Access-Datenbanken ausschließlich mit ODBC beschäftigen. Der Wechsel zu DAO dürfte Ihnen aber wegen der Ähnlichkeit der DAO- und der ODBC-Klassen nicht schwer fallen (siehe Online-Hilfe, Indexeintrag ODBC und DAO).

OLE DB ist der neueste Microsoft-Standard zur Datenbankunterstützung, und es wird von ihm erwartet, daß er sich in der professionellen Anwendungserstellung durchsetzen wird. Da die OLE DB-Unterstützung aber noch in der Entwicklung ist und es hier noch gelegentlich zu Änderungen kommt, wollen wir OLE DB noch ein bißchen Zeit geben und uns auf ODBC konzentrieren.

25.3 Datenbanken anlegen

Zum Anlegen einer Datenbank benutzen Sie entweder das zugehörige Datenbankprogramm (beispielsweise Access oder Paradox) oder die ODBC-Verwaltung aus der Windows-Systemsteuerung (siehe nächsten Abschnitt).

Übung 25-1: MS Access-Tabelle anlegen

Im folgenden werden wir eine kleine Adressen-Datenbank einrichten. Ich verwende MS Access als Datenbankprogramm in der Annahme, daß die meisten Leser, die mit Datenbanken arbeiten wollen, ebenfalls über Access verfügen. Wenn Sie statt dessen Paradox oder dBase oder irgendeine andere Datenbank verwenden wollen, schauen Sie in der ODBC-Verwaltung nach, ob ODBC-Treiber für Ihre Datenbank eingerichtet sind (siehe nächste Übung).

Bild 25.2:
Felderstruktur
einer Access-
Tabelle

Feldname	Felddatentyp	Beschreibung
Vorname	Text	
Name	Text	
Straße	Text	
PLZ	Zahl	
Stadt	Text	

Felderigenschaften	
Allgemein	Nachschlagen
Feldgröße	20
Format	
Eingabeformat	
Beschriftung	
Standardwert	
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Leere Zeichenfolge	Nein
Indiziert	Ja (Duplikate möglich)

1. Rufen Sie Microsoft Access auf.
2. Legen Sie eine neue Datenbank an.
3. Erstellen Sie in der Entwurfsansicht eine neue Tabelle.

4. Legen Sie die Felderstruktur der Tabelle fest. Für eine Adressenverwaltung könnten Sie beispielsweise die Felder VORNAME, NAME, STRASSE, PLZ und STADT anlegen. Zu jedem Feld geben Sie
 - ✗ den Feldnamen (Spaltenüberschrift),
 - ✗ den Felddatentyp, der bestimmt, welche Eingaben in dem Feld erlaubt sind (eine Auswahl der erlaubten Typbezeichnungen wird über die Pfeilschalter der Eingabefelder angeboten),
 - ✗ eine optionale Beschreibung und
 - ✗ die Feldgröße an.
5. Speichern Sie die Tabellenstruktur ab, und schließen Sie die Entwurfsansicht. Wenn Sie es nicht selbst schon getan haben, legt Access jetzt noch einen Primärschlüssel für die Tabelle an.

ID	Vorname	Name	Straße	PLZ	Stadt
1	Ida	Schmitt	Villastr	93055	Regensburg
2	Hung	Vu	Egerstr	93057	Regensburg
3	Griselda	Schmitt	Saarstr	66111	Saarbrücken
4	Franz	Munz	Hollerstr	80995	München

Bild 25.3:
Die Datensätze

6. Öffnen Sie die Tabelle zum Eingeben Ihrer Datensätze (geben Sie zumindest ein paar Datensätze zum Austesten Ihrer Datenbankanwendungen ein).

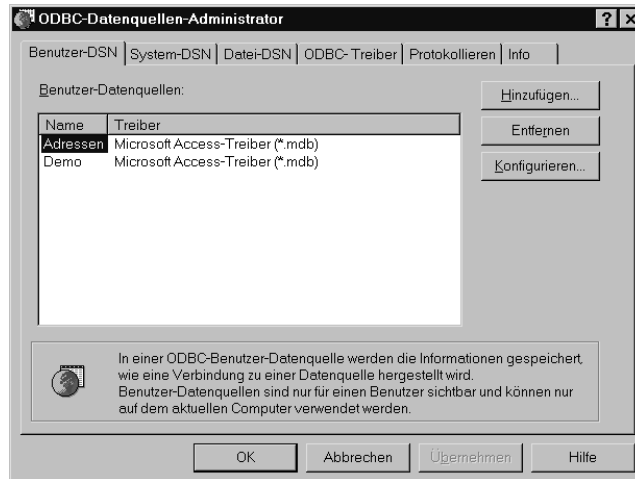
25.4 Datenbanktreiber und Treiberverbindungen

Damit Sie über ODBC auf eine Datenbank (beispielsweise Paradox oder Access) zugreifen können, benötigen Sie einen passenden Treiber für diese Datenbank.

Übung 25-2: Neuen ODBC-Treiber für Access installieren

1. **Zuerst muß der Treiber unter Windows registriert werden.** Üblicherweise geschieht dies bei der Installation der zugehörigen Datenbank-anwendung bzw. bei der Installation von Visual C++. Sie können aber auch entsprechende Treiber nachkaufen und mit den (hoffentlich) vorhandenen Setup-Routinen installieren.

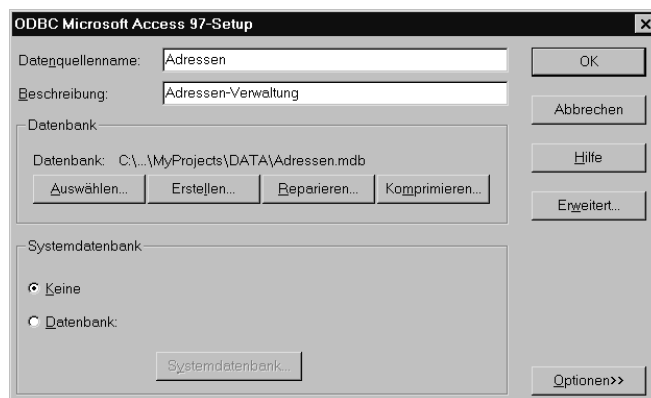
Bild 25.4:
Die ODBC-
Verwaltung



2. **Definieren Sie die ODBC-Benutzer-Datenquelle.** Auf diese Weise wird die Verbindung des Treibers mit einer Datenquelle (eine existierende Datenbank oder ein Verzeichnis, in dem später noch eine Datenbank angelegt werden soll) eingerichtet. Gehen Sie dazu folgendermaßen vor:

Rufen Sie das Windows-Fenster SYSTEMSTEUERUNG auf, und doppelklicken Sie auf den ODBC-Eintrag. (Hinweis: Auf der Seite für die TREIBER können Sie sich noch einmal davon überzeugen, daß Ihr ODBC-Treiber korrekt registriert ist).

Bild 25.5:
Einrichten
einer Daten-
bankverbin-
dung



Klicken Sie auf der Seite BENUTZER-DSN auf den Schalter HINZUFÜGEN, wählen Sie in dem erscheinenden Dialogfenster Ihren Access-Treiber aus, und schicken Sie das Dialogfenster ab (Schalter FERTIGSTELLEN). (Die Seite BENUTZER-DSN ist nur für lokale Datenbanken.)

In dem zweiten Dialogfenster wird die Verbindung konfiguriert. Geben Sie einen beliebigen Namen und eine Beschreibung für Ihre Datenquelle an. Mit den Schaltern im Feld DATENBANK können Sie sodann eine existierende Datenbank auswählen oder eine neue Datenbank erstellen (siehe Übung 25-2). Über den Schalter ERWEITERT können Sie den Zugriff an ein spezielles Paßwort binden.

Wenn Sie das Fenster abschicken, wird die neue Datenbankverbindung auf der Seite BENUTZER-DSN angezeigt. Springen Sie zuvor aber zur Übung 25-2, um eine neue Datenbank für unsere spätere Datenbank-Anwendung anzulegen.

25.5 Datenbank-Programmierung

Jetzt wollen wir versuchen, aus einem Programm heraus auf die Daten in der Datenbank zuzugreifen. Unsere wichtigste Hilfe sind dabei die ODBC-Datenbankklassen.

25.5.1 Die ODBC-Datenbankklassen

Die Funktionalität der ODBC-API ist in der MFC in den Klassen `CDatabase` und `CRecordset` gekapselt.

CDatabase

`CDatabase` dient der Erstellung und Verwaltung von Datenbankverbindungen.

Um mit Hilfe von `CDatabase` eine Datenbankverbindung herzustellen, geht man grundsätzlich wie folgt vor:

1. Erzeugen Sie ein `CDatabase`-Objekt.
2. Rufen Sie die `Open()` oder `OpenEx()`-Methode des `CDatabase`-Objekts auf.
3. Wenn Sie später ein `CRecordset`-Objekt zum Zugriff auf die Daten der Datenbank erzeugen, übergeben Sie dem `CRecordset`-Konstruktor einen Zeiger auf das `CDatabase`-Objekt.

4. Zum Schließen der Datenbankverbindung schließen Sie alle offenen CRecordset-Objekte, rufen die CDatabase-Methode Close() auf und löschen das CDatabase-Objekt.



Wenn Sie die Methoden GetDefaultConnection() und GetDefaultSQL() des CRecordset-Objekts überschreiben (geschieht bei Verwendung des Anwendungs- oder Klassen-Assistenten automatisch), brauchen Sie selbst kein CDatabase-Objekt einzurichten, sondern können dem Konstruktor des CRecordset-Objekts den Wert NULL übergeben, worauf der Konstruktor automatisch für Sie ein passendes CDatabase-Objekt für Ihre Datenbankverbindung einrichtet.

CRecordset

Ein CRecordset-Objekt repräsentiert eine Gruppe von Datensätzen einer Datenquelle (für SQL-Profis: ein CRecordset-Objekt repräsentiert das Ergebnis einer SELECT-Abfrage).

Um eine Gruppe von Datensätzen aus einer Datenbank abzufragen, gehen Sie wie folgt vor:

1. Leiten Sie eine eigene Datensatz-Klasse von CRecordSet ab.
2. Erzeugen Sie ein Objekt Ihrer CRecordset-Klasse. Rufen Sie dazu den Konstruktor Ihrer Klasse auf, und übergeben Sie diesem
 - ✗ einen Zeiger auf das CDatabase-Objekt, das die offene Datenbankverbindung repräsentiert, oder
 - ✗ NULL, wenn eine CDatabase-Datenbankverbindung automatisch auf Grund der Informationen aus den überschriebenen Methoden GetDefaultConnection() und GetDefaultSQL() eingerichtet werden soll (die Methoden werden bei Verwendung des Anwendungs- oder Klassen-Assistenten automatisch überschrieben).
3. Rufen Sie die Open()-Methode Ihres CRecordset-Objekts auf, um die Datensätze auszulesen, die das CRecordset-Objekt repräsentieren soll. Dabei können Sie über die Parameter der Methode festlegen, ob die zurückgelieferte Datensatzgruppe dynamisch aktualisiert werden soll.

Datenbanken können von mehreren Anwendern gleichzeitig genutzt werden. Es kann daher passieren, daß die Datensätze, die ein CRecordset-Objekt repräsentiert, plötzlich nicht mehr aktuell sind, weil ein zwei-

ter Anwender die Datensätze der Datenbank überarbeitet hat. In solchen Fällen stellt sich die Frage, ob das `CRecordset`-Objekt danach die aktualisierten Datensätze oder die alten, zuvor abgerufenen Datensätze repräsentiert:

- ✗ `nOpenType = CRecordset::dynaset`. Das `CRecordset`-Objekt wird aktualisiert und repräsentiert die geänderten Daten.
- ✗ `nOpenType = CRecordset::snapshot`. Das `CRecordset`-Objekt wird nicht aktualisiert. Um es auf den neuesten Stand zu bringen, muß es geschlossen und wieder geöffnet werden.

Und welche Datensätze überhaupt zurückgeliefert werden sollen:

- ✗ `lpszSQL = NULL`. Wählt Daten nach Rückgabewert der Methode `GetDefaultSQL()` aus. (Diese Methode wird vom Anwendungs-Assistenten automatisch überschrieben.)
- ✗ `lpszSQL = Tabellename`. Wählt die Daten einer Tabelle der Datenbank aus (die Felder werden der überschriebenen Methode `DoFieldExchange()` entnommen).
- ✗ `lpszSQL = SQL-SELECT-Befehl` zur Auswahl bestimmter Datensätze.
- ✗ `lpszSQL = CALL-Befehl` zur Auswahl einer gespeicherten Datenbankprozedur (stored procedure).

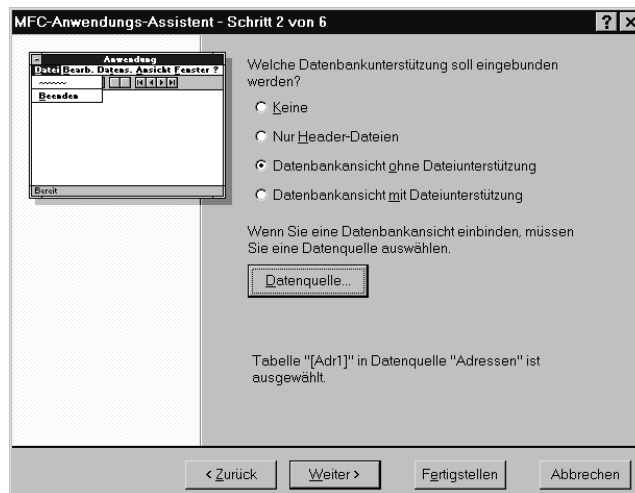
25.5.2 Datensätze einlesen

Um Sie von den Formalitäten der Einrichtung einer Datenbankverbindung zu einer bestehenden Datenquelle zu befreien, unterstützt der MFC-Anwendungs-Assistent auch Datenbanken. Mit seiner Hilfe kann man bequem Anwendungen erstellen, die auf Daten einer Datenbank zugreifen und diese anzeigen.

Übung 25-3: Datenbankanwendung anlegen

Die folgende Ausführung beschreibt die Erstellung einer Datenbankanwendung, die eine `CRecordView` zum Anzeigen von Adressen aus einer Adressen-Datenbank (siehe vorangehende Übungen) benutzt.

Bild 25.6:
Datenbank-
unterstützung
auswählen



1. Starten Sie den MFC-ANWENDUNGS-ASSISTENTEN (.EXE) zum Anlegen einer neuen SDI-Anwendung.

Bild 25.7:
Datenquelle
auswählen



2. Wählen Sie die Datenquelle aus. Aktivieren Sie auf der zweiten Seite des Anwendungs-Assistenten die Option DATENBANKANSICHT OHNE DATEI-UNTERSTÜTZUNG, und klicken Sie auf den Schalter DATENQUELLE, um die zu verwendende Datenbank (und Datenbanktabelle) auszuwählen.
3. Lassen Sie den Assistenten das Projekt fertigstellen.

Der Assistent erstellt daraufhin eine lauffähige, aber noch nicht funktionsfähige Anwendung. Bevor in der nächsten Ausführung aufgezeigt wird, wie man die Daten aus der Datenbank in der Ansichtsklasse anzeigt, verschaffen wir uns einen Überblick darüber, welche Arbeiten uns der Assistent abgenommen hat.

Die Vorarbeiten des Anwendungs-Assistenten

- ✗ Die ODBC-Headerdateien wurden per `#include` aufgenommen.

`<afxdb.h>` in `stdafx.h`

`#include "l.deu\afxdb.rc"` in der Ressourcenskriptdatei

- ✗ Zur Repräsentation der Daten in Ihrer Anwendung wurde eine Datensatzklasse von `CRecordset` abgeleitet.
 - In dieser Datensatzklasse wurde für jedes Feld der im Assistenten ausgewählten Datenbanktabelle ein Datenelement deklariert.

```
class CAdressenSet : public CRecordset
{
public:
    CAdressenSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(CAdressenSet)
// Feld-/Parameterdaten
    //{AFX_FIELD(CAdressenSet, CRecordset)
    long    m_ID;
    CString m_Vorname;
    CString m_Name;
    CString m_Strasse;
    long    m_PLZ;
    CString m_Stadt;
    //}}AFX_FIELD
    ...
};
```

- Für die Verbindung zur Datenbank wurden die beiden Methoden `GetDefaultConnect()` und `GetDefaultSQL()` überschrieben. Die von diesen Methoden zurückgelieferten Information werden zur internen Herstellung der Datenbankverbindung genutzt (falls der Programmierer kein eigenes `CDatabase`-Objekt erzeugt).

```
CString CAdressenSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=Adressen");
}
CString CAdressenSet::GetDefaultSQL()
{
    return _T("[Adr1]");
}
```


Für die Verbindung der Datenelemente der Datensatz-Klasse mit den Feldern der Datenbanktabelle sorgt die überschriebene Version der DoFieldExchange()-Methode.

```
void CAdressenSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(CAdressenSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Long(pFX, _T("[ID]"), m_ID);
    RFX_Text(pFX, _T("[Vorname]"), m_Vorname);
    RFX_Text(pFX, _T("[Name]"), m_Name);
    RFX_Text(pFX, _T("[Straße]"), m_Stra_e);
    RFX_Long(pFX, _T("[PLZ]"), m_PLZ);
    RFX_Text(pFX, _T("[Stadt]"), m_Stadt);
    //}AFX_FIELD_MAP
}
```

- ✗ In der Dokumentklasse wurde eine Variable vom Typ der abgeleiteten CRecordset-Klasse deklariert, um die Verbindung zwischen den Datenbankdaten und der Dokumentklasse herzustellen.

```
class CAdressenDoc : public CDocument
{
    ...
    // Attribute
public:
    CAdressenSet m_adressenSet;
```

- ✗ Die Ansichtsklasse wurde von CRecordView abgeleitet. CRecordViews basieren auf Dialog-Ressourcen und verwenden Steuerelemente zum Anzeigen und Editieren von Daten aus einer Datenquelle. Die Dialog-Ressource für die Ansichtsklasse ist bereits in der Ressourcenskriptdatei angelegt und kann im Dialog-Editor bearbeitet werden. Für die Verbindung zu den Datenbankdaten wurde ein Zeiger auf ein Objekt vom Typ der abgeleiteten CRecordset-Klasse deklariert.

```
class CAdressenView : public CRecordView
{
    ...
public:
    //{AFX_DATA(CAdressenView)
    enum{ IDD = IDD_ADRESSEN_FORM };
    CAdressenSet* m_pSet;
    ...
}
```

- ✗ Die CRecordView-Methode OnInitialUpdate() wurde überschrieben. Die Überschreibung dieser Methode ist obligatorisch. Sie initialisiert den Zeiger auf das CRecordset-Datenelement und ruft die Basisklassenversion auf, um die Datensatzgruppe (und implizit die Datenbankverbindung) zu öffnen.

```
void CAdressenView::OnInitialUpdate()
{
    m_pSet = &GetDocument()->m_adressenSet;
    CRecordView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();
}
```

- ✗ Die CRecordView-Methode OnGetRecordset() wurde überschrieben. Die Überschreibung dieser Methode ist obligatorisch. Die Methode liefert einfach einen Zeiger auf das CRecordset-Objekt zurück.

```
CRecordset* CAdressenView::OnGetRecordset()
{
    return m_pSet;
}
```

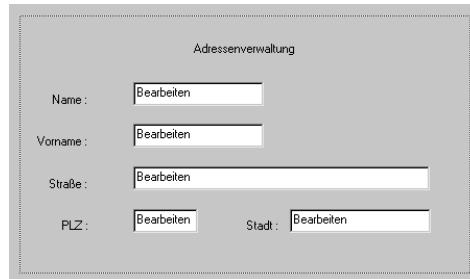
Im Zuge des Programmstarts werden (falls keine anderweitigen Kommandozeilenargumente übergeben wurden) die Behandlungsmethode OnFileNew() zum Anlegen eines neuen Dokuments und die OnInitialUpdate()-Methode der Ansichtsklasse aufgerufen. Dabei wird auch gleich die Datensatzgruppe geöffnet (Methode CRecordset::Open()). Diese ruft ihrerseits wieder die CDatabase-Methode Open() auf, um die Datenbankverbindung herzustellen. Zu welcher Datenbank eine Verbindung hergestellt werden soll und welche Datensätze von dem CRecordset-Objekt repräsentiert werden sollen, wird dabei von den überschriebenen CRecordset-Methoden GetDefaultConnect() und GetDefaultSQL() abgefragt.



25.5.3 Datensätze anzeigen

Der nächste Schritt besteht darin, die Dialog-Ressource der Ansichtsklasse einzurichten und die Steuerelemente des Dialogs mit den Feldern der Datenbank zu verbinden.

Bild 25.8:
Das Dialogfeld
für die An-
sichtsklasse



Übung 25-4: Daten anzeigen

1. Öffnen Sie die Ressourcenskriptdatei, und laden Sie die Dialog-Ressource für die Ansichtsklasse in den Dialog-Editor. Richten Sie den Dialog für die Anzeige Ihrer Daten ein. Üblicherweise nehmen Sie dazu für jedes Datenbankfeld, dessen Inhalt angezeigt werden soll, ein passendes Steuerelement (in unserem Falle Eingabefelder) und ein statisches Beschriftungsfeld in den Dialog auf.

Beachten Sie, daß Sie die Eigenschaften `STIL = UNTERGEORDNET` und `RAND = KEINE` für das Dialogfeld nicht ändern.

2. Speichern Sie die Ressource.
3. Verbinden Sie die Steuerelemente des Dialogs mit den `CRecordSet`-Datenelementen für die Felder der Datenbanktabelle. Nutzen Sie dazu die bereits für Sie angelegte `DoDataExchange()`-Methode und die `DDX_FieldText`-Makros:

```
void CAdressenView::DoDataExchange(CDataExchange* pDX)
{
    CRecordView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAdressenView)
    // HINWEIS: Der Klassenassistent fügt an dieser
    // Stelle DDX- und DDV-Aufrufe ein
    DDX_FieldText(pDX, IDC_EDIT1, m_pSet->m_Name, m_pSet);
    DDX_FieldText(pDX, IDC_EDIT2, m_pSet->m_Vorname, m_pSet);
    DDX_FieldText(pDX, IDC_EDIT3, m_pSet->m_Stra_e, m_pSet);
    DDX_FieldText(pDX, IDC_EDIT4, m_pSet->m_PLZ, m_pSet);
    DDV_MinMaxLong(pDX, m_pSet->m_PLZ, 0, 100000);
    DDX_FieldText(pDX, IDC_EDIT5, m_pSet->m_Stadt, m_pSet);
    //}}AFX_DATA_MAP
}
```

4. Da wir in dem Datenbankprogramm keine Dateien unterstützen, braucht auch kein Dateiname im Titel des Rahmenfensters angezeigt zu werden. Rufen Sie daher die Methode `SetWindowText()` auf, um dem Rahmenfenster einen sinnvollen Titel zuzuweisen:

```
BOOL CAdressenApp::InitInstance()
{
    ...
    // Das einzige Fenster ist initialisiert und kann jetzt
    // angezeigt und aktualisiert werden.
    m_pMainWnd->SetWindowText("Datenbank-Betrachter");
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    ...
}
```

5. Führen Sie die Anwendung aus.



Bild 25.9:
Das fertige
Programm

25.5.4 In der Datenbank navigieren

Für das obige Beispiel hat der Anwendungs-Assistent bereits für ein eigenes Menü mit den Befehlen zum Bewegen in der Datenbank (genauer gesagt, zum Springen zwischen den Datensätzen, die von dem `CRecordSet`-Objekt repräsentiert werden) gesorgt.

Die gesamte Einrichtung der Nachrichtenbehandlung für diese Menübefehle (von der Menübefehls-ID über die Antworttabelle bis zu den Behandlungsmethoden und der Befehlsaktivierung) ist bereits in der Implementierung der MFC-Klasse `CRecordView` vordefiniert.

Das eigentliche Wechseln der Datensätze übernehmen aber die entsprechenden Methoden der Klasse `CRecordSet`:

Tabelle 9.4:
Datensatz-
zeiger ver-
schieben

Klassenelement	Beschreibung
<code>MoveFirst</code>	Wählt als aktuellen Datensatz den ersten Datensatz (in der ersten Datensatzgruppe) aus.
<code>MoveLast</code>	Wählt als aktuellen Datensatz den letzten Datensatz aus (oder den ersten Datensatz in der letzten Datensatzgruppe – für Satzgruppen > 1).
<code>MoveNext</code>	Wählt als aktuellen Datensatz den nächsten Datensatz aus (oder den ersten Datensatz in der nächsten Datensatzgruppe – für Satzgruppen > 1).
<code>MovePrev</code>	Wählt als aktuellen Datensatz den vorigen Datensatz aus (oder den ersten Datensatz in der vorangehenden Datensatzgruppe – für Satzgruppen > 1).

Beim Verschieben des Datensatzzeigers ist allerdings darauf zu achten, daß der Datensatzzeiger nicht über den ersten oder letzten Datensatz hinaus bewegt wird. Zu diesem Zweck gibt es die `CRecordSet`-Methoden `IsBOF()` und `IsEOF()` sowie die `CRecordView`-Methoden `IsOnFirstRecord()` und `IsOnLastRecord()`.

25.5.5 Datensätze editieren

Auch das Hinzufügen, Löschen oder Bearbeiten von Datensätzen ist dank der entsprechenden Methoden der Klasse `CRecordSet` kein großes Problem. Erkundigen Sie sich dazu in der Online-Hilfe nach den Methoden

- ✕ `Edit()`
- ✕ `AddNew()`
- ✕ `Delete()`
- ✕ `UpdateData()`

25.5.6 Nach Daten suchen

Ein häufiges Problem bei der Programmierung von Datenbanken ist die Suche nach speziellen Daten. Wenn Sie über ODBC auf eine Datenbank zugreifen, vereinfacht sich die Suche jedoch insofern, als Sie lediglich Ihre Suchkriterien in Form einer passenden SQL-Abfrage formulieren und diese

dann an die Datenbank schicken müssen. Die eigentliche Suche und das Zurückliefern der gefundenen Datensätze läuft dann automatisch ab.

SELECT

Der SQL-Befehl, über den Abfragen an Datenbanken geschickt werden, lautet `SELECT`. Der SQL-Befehl liefert eine Untermenge einer Datenbank, die sich aus den im Befehl spezifizierten Feldern zusammensetzt. Ist eine `WHERE`-Klausel definiert, werden nur die Datensätze zurückgeliefert, die der Suchbedingung entsprechen. Die resultierende »Tabelle« wird danach durch das `CRecordSet`-Objekt repräsentiert, das den SQL-Befehl losgeschickt hat.

```
SELECT Spaltenliste FROM Tabellenliste
      WHERE Suchbedingung ORDER BY Sortierung.
```

Parameter	Beschreibung
Spaltenliste	Hier geben Sie an, welche Felder der nachfolgenden Tabellen zurückgeliefert werden sollen – beispielsweise Vorname, Name, Straße, PLZ, Stadt.
Tabellenliste	Liste der Tabellen, in denen die zuvor aufgeführten Felder zu finden sind – in unserem Falle <code>Adr1</code> .
Suchbedingung	Optional; wenn spezifiziert, werden nur die Datensätze zurückgeliefert, die die Bedingung erfüllen – beispielsweise <code>Stadt = 'München'</code> .
Sortierung	Optional; wenn spezifiziert, werden die zurückgelieferten Datensätze in der Reihenfolge der aufgeführten Felder aufsteigend (ASC) oder absteigend (DESC) sortiert – beispielsweise <code>Name ASC, Vorname DESC</code> .

Wenn Sie SQL-Befehle oder Teile von SQL-Befehlen als Strings erstellen, müssen Sie folgende Schreibweisen beachten:

- ✗ Namen von Spalten oder Tabellen werden so geschrieben, wie sie in der Datenbank definiert sind.
- ✗ Spalten- oder Tabellennamen, die Leerzeichen enthalten, werden in eckige Klammern gefaßt: `[Kosten in DM]`.
- ✗ Feldinhalte werden in Hochkommata gesetzt: `'München'`.

Übung 25-5: Datensätze filtern

Springen Sie im Adressen-Programm in die Methode `GetDefaultSQL()`, und ändern Sie den zurückgelieferten SQL-Befehl, so daß beim Öffnen der Datenbank nur die Adressen aus Regensburg eingelesen werden.

```
CString CAdressenSet::GetDefaultSQL()
{
    return _T("SELECT * FROM [Adr1] \
        WHERE Stadt = 'Regensburg'");
}
```

25.5.7 Daten grafisch darstellen

In vielen Fällen ist die Anzeige der Daten in Datensätzen unübersichtlich und nur schwer zu interpretieren. Gerade wenn es darum geht, Zahlen oder numerische Datenreihen (wie zum Beispiel Meßwerte) darzustellen, spielt die grafische Aufbereitung der Daten eine wichtige Rolle.

Das folgende Programm verwendet keine dialogbasierte `CRecordView`, sondern stellt die aus der Datenbank eingelesenen DAX-Werte in Form einer Kurve dar.

Wichtig ist dabei nur, daß Sie in der `DoFieldExchange()`-Methode der abgeleiteten `CRecordset`-Klasse für die Datenübertragung von den Feldern der Datenbank zu den Datenelementen der `CRecordset`-Klasse sorgen, damit Sie über letztere auf die Daten zugreifen können (wenn Sie Ihre `CRecordset`-Klasse vom Klassen-Assistenten einrichten lassen, können Sie auch die Verbindungen zu den Datenbankfeldern von diesem generieren lassen).

Übung 25-5: Daten grafisch anzeigen

Bild 25.10:
Die Datenbanktabelle



ID	Monat	DAX
1	Apr	4300
2	Mai	4500
3	Jun	4650
4	Jul	4900
5	Aug	5300
6	Sep	4850
7	Okt	5100
(AutoWert)		0

Datensatz: 1 von 8

1. Legen Sie eine Datenbank mit den beiden Feldern MONAT und DAX an, und geben Sie einige Werte ein (siehe Abbildung).
2. Richten Sie eine ODBC-Treiber Verbindung zu der Datenbank ein (siehe Übung 25-2).

3. Legen Sie mit dem Anwendungs-Assistenten eine Datenbankanwendung an. Wählen Sie auf der zweiten Seite die Option NUR HEADER-DATEIEN aus. (Die Verbindung mit der Datenquelle wird im nächsten Schritt vorgenommen.)

Jetzt müssen wir den Zugriff auf die Daten einrichten.

4. Richten Sie mit dem Klassen-Assistenten eine abgeleitete CRecordset-Klasse ein (Schalter KLASSE HINZUFÜGEN/NEU). Nennen Sie die Klasse beispielsweise CDaten, verbinden Sie sie im Dialogfeld DATENBANK-OPTIONEN mit der Datenbank, und lassen Sie ALLE SPALTEN VERBINDEN (wofür Sie im nachfolgenden Dialogfeld eine Tabelle auswählen müssen). Wenn Sie die Spalten verbinden, setzt der Klassen-Assistent in der überladenen Methode DoFieldExchange() den Code für die Verbindung der Datenelemente Ihrer CRecordset-Klasse zu den Feldern der Datenbank auf.
5. Sorgen Sie dafür, daß die Datenbankverbindung geöffnet wird.

Deklarieren Sie in der Dokumentklasse eine Variable vom Typ Ihrer CRecordset-Klasse. (Nehmen Sie dazu auch eine #include-Anweisung zur Einbindung der Header-Datei der CRecordset-Klasse auf.)

```
#include "Daten.h"
class CDAXDoc : public CDocument
{
...
public:
    CDaten m_pRecords;
```

Das CRecordset-Objekt m_pRecords wird dann zusammen mit dem Dokument instantiiert. Es muß allerdings noch geöffnet werden.

```
BOOL CDAXDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    m_pRecords.Open();
    return TRUE;
}
```

6. Damit die Methode OnNewDocument() nicht für eine bereits geöffnete Datenbank ausgeführt wird, müssen wir den Befehl DATEI/NEU löschen. Löschen Sie gleich alle Menübefehle bis auf DATEI/BEENDEN. Vergessen Sie auch nicht die zugehörigen Tastaturkürzel.

Jetzt können wir die eingelesenen Daten im Fenster der Anwendung grafisch visualisieren.

7. Geben Sie die Datenbankdaten in der `OnDraw()`-Methode der Ansichtsklasse in Form einer Grafik aus.

Zuerst benötigt man einen Zeiger auf das `CRecordset`-Objekt des Dokuments. Über diesen kann man dann auf die Datenelemente des `CRecordset`-Objekt zugreifen, die wiederum über die Methode `DoFieldExchange()` (die vom Klassen-Assistenten eingerichtet wurde) mit den Feldern der Datenbank verbunden sind.

```
CDaten* pDaten = &(pDoc->m_pRecords);
```

Danach müssen für die formatfüllende Skalierung der Kurve der größte und der kleinste anzuzeigende *y*-Wert ermittelt werden. Auch die Anzahl der anzuzeigenden Daten muß auf diese Weise ermittelt werden, da die `CRecordset`-Methode `GetRecordCount()` nur den höchsten Index eines bereits mit `MoveNext()` besuchten Datensatzes anzeigt (Sprünge mit `MoveLast()` werden ohne Zählung durchgeführt).

```
int count = 1;
long max = 1;
long min = 1000000;

pDaten->MoveFirst();
while (!pDaten->IsEOF()) {
    if(pDaten->m_DAX > max)
        max = pDaten->m_DAX;
    if(pDaten->m_DAX < min)
        min = pDaten->m_DAX;
    count++;
    pDaten->MoveNext();
}
```

Danach wird das Koordinatensystem zur übersichtlicheren Grafikausgabe umgesetzt (und zwar so, daß der Ursprung links unten liegt und positive *y*-Werte nach oben abgetragen werden). Beachten Sie, daß die Einstellungen im `SetViewportExt()`-Aufruf nur wirksam werden, wenn der Abbildungsmodus zuvor auf `MM_ISPTROPIC` oder `MM_ANISOTROPIC` umgesetzt und `SetWindowExt()` aufgerufen wurde.

```
RECT rect;
GetClientRect(&rect);
int Hoehe = rect.bottom - rect.top;
int Breite = rect.right - rect.left;
pDC->SetMapMode(MM_ISOTROPIC);
pDC->SetViewportOrg(0, Hoehe);
pDC->SetWindowExt(Breite, Hoehe);
pDC->SetViewportExt(Breite, -Hoehe);
```

Zum Schluß wird die Datensatzgruppe noch einmal durchlaufen, und die DAX-Daten werden als Kreise angezeigt. Die gesamte OnDraw()-Methode sieht wie folgt aus:

```
void CDAXView::OnDraw(CDC* pDC)
{
    CDAXDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // Zugriff auf Daten
    CDaten* pDaten = &(pDoc->m_pRecords);

    if(pDaten->IsBOF())
        return;

    // Anzahl der Datensätze sowie größten und
    // kleinsten anzuzeigenden y-Wert (DAX) bestimmen
    int count = 1;
    long max = 1;
    long min = 1000000;

    pDaten->MoveFirst();
    while (!pDaten->IsEOF())
    {
        if(pDaten->m_DAX > max)
            max = pDaten->m_DAX;
        if(pDaten->m_DAX < min)
            min = pDaten->m_DAX;
        count++;
        pDaten->MoveNext();
    }

    // Grafik formatfüllend in Fenster zeichnen
    RECT rect;
    GetClientRect(&rect);
```

```

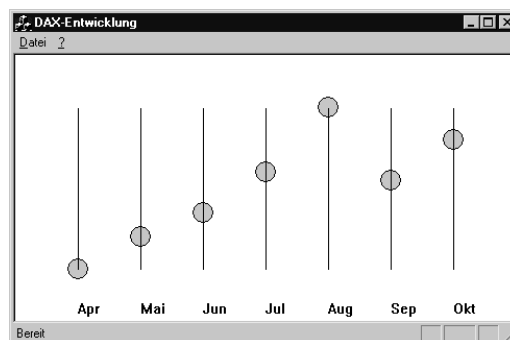
int Hoehe = rect.bottom - rect.top;
int Breite = rect.right - rect.left;
pDC->SetMapMode(MM_ISOTROPIC);
pDC->SetViewportOrg(0, Hoehe);
pDC->SetWindowExt(Breite, Hoehe);
pDC->SetViewportExt(Breite, -Hoehe);

int xStep = (int) (Breite / (count));
double yFaktor = (double)(Hoehe - 100) /
                 (double)(max - min);

// Daten ausgeben
pDC->SelectStockObject(LTGRAY_BRUSH);
pDaten->MoveFirst();
for(int i = 1; i < count; i++)
{
    pDC->Ellipse(i * xStep - 10,
                40 + (int)((pDaten->m_DAX - min) * yFaktor),
                i * xStep + 10,
                60 + (int)((pDaten->m_DAX - min) * yFaktor));
    pDC->MoveTo(i * xStep, 50);
    pDC->LineTo(i * xStep, Hoehe - 50);
    pDC->TextOut(i * xStep, 20, pDaten->m_Monat);
    pDaten->MoveNext();
}
}

```

Bild 25.11:
Die fertige
Datenbank-
Anwendung



25.6 Zusammenfassung

Anwendungen, die größere oder ständig wechselnde Datenmengen zu verarbeiten haben, arbeiten oft mit externen Datenbanken zusammen, in denen die Daten verwaltet und gespeichert werden.

Der Zugriff auf die Datenbank erfolgt über eine Treiberschnittstelle – üblicherweise ODBC. Ist ODBC erst einmal für den Zugriff auf eine Datenbank eingerichtet, kann man die Daten in der Datenbank ohne große Mühe mit der Unterstützung der MFC-ODBC-Klassen `CDatabase` und `CRecordset` auslesen.

Die ODBC-Klassen enthalten aber nicht nur Methoden zum Herstellen einer Datenbankverbindung und zum Einlesen von Daten. Weitere Methoden erlauben das Navigieren in der Datenbank, das Editieren von Datensätzen, das Suchen nach bestimmten Daten etc.

Nutzt man den MFC-Anwendungs-Assistenten zur Einrichtung des Anwendungsgerüsts und zur Verbindung mit der Datenbank, werden viele dieser Aufgaben bereits funktionsfähig implementiert.

Windows-Nachrichten

Windows-Botschaft	Bedeutung der Botschaft
WM_ACTIVATE	Fenster aktiviert oder deaktiviert
WM_ACTIVATEAPP	Das zu aktivierende Fenster gehört zu einer anderen Anwendung als das derzeit aktive Fenster
WM_ASKCBFORMATNAME	Fragt nach Datenformat in Zwischenablage
WM_CANCELMODE	Systemmodus wurde annulliert
WM_CHANGECHAIN	Ein Fenster wurde aus der Liste der Zwischenablagebetrachter entfernt
WM_CHAR	Übermittelt den ASCII-Code einer gedrückten Taste
WM_CHARTOITEM	Wird von LBS_WANTKEYBOARDINPUT-Listenfeldern an Elternfenster gesendet
WM_CHILDACTIVATE	Wird an das Elternfenster gesendet, wenn ein Kindfenster verschoben oder aktiviert wurde
WM_CHILDINVALID	Wird an das Elternfenster gesendet, wenn ein Kindfenster ungültig ist
WM_CHOOSEFONT_GETLOGFONT	Ermittelt vom Schriftart-Dialog den aktuell ausgewählten Fonts
WM_CHOOSEFONT_SETFLAGS	Setzt die Anzeigeoptionen für das Schriftart-Dialogfeld



Windows-Botschaft	Bedeutung der Botschaft
WM_CHOOSEFONT_SETLOGFONT	Wird an Schriftart-Dialog gesendet, um die aktuellen logischen LOGFONT-Informationen zu setzen
WM_CLEAR	Löschvorgang im Eingabefeld ohne Kopieren in die Zwischenablage
WM_CLOSE	Fenster wird geschlossen
WM_COMMAND	Anwender hat Menü-Eintrag, Steuerelemente oder Tastaturkürzel aktiviert
WM_COMPACTING	Nachricht an Hauptfenster, daß Speicher knapp wird
WM_COMPAREITEM	Neuer Eintrag in besitzergezeichnetem LBS_SORT- oder CBS_SORT- Listen- oder Kombinationsfeld
WM_CONTEXTMENU	Informiert ein Fenster, daß der Anwender die rechte Maustaste im Fenster gedrückt hat
WM_COPY	Kopiert ausgewählten Text aus Eingabefeld in die Zwischenablage
WM_COPYDATA	Anwendung übergibt Daten an eine andere Anwendung
WM_CREATE	Fenster wird erzeugt
WM_CTLCOLORBTN	Wird an das Elternfenster gesendet, wenn ein Schalter gezeichnet wird
WM_CTLCOLORDLG	Wird an das Dialogfeld gesendet, bevor es gezeichnet wird
WM_CTLCOLOREDIT	Wird an das Elternfenster gesendet, wenn ein Eingabefeld gezeichnet wird
WM_CTLCOLORLISTBOX	Wird an das Elternfenster gesendet, wenn ein Listefeld gezeichnet wird
WM_CTLCOLORMSGBOX	Wird an das Elternfenster gesendet, wenn ein Nachrichtenfenster gezeichnet wird
WM_CTLCOLORSCROLLBAR	Wird an das Elternfenster gesendet, wenn eine Bildlaufleiste gezeichnet wird
WM_CTLCOLORSTATIC	Wird an das Elternfenster gesendet, wenn ein statisches Textfeld gezeichnet wird
WM_CUT	Kopiert Auswahl in Eingabefeld in die Zwischenablage und löscht Text in Eingabefeld

Windows-Botschaft	Bedeutung der Botschaft
WM_DDE_...	Nachrichten für DDE-Anwendungen
WM_DEADCHAR	Anwender hat eine unbelegte Taste gedrückt, die den Wert der nachfolgenden Taste beeinflusst
WM_DELETEITEM	Eintrag in besitzergezeichnetem Listen- oder Kombinationsfeld wurde gelöscht
WM_DESTROY	Fenster wird entfernt, nachdem es vom Bildschirm gelöscht wurde
WM_DESTROYCLIPBOARD	Informiert Besitzer der Zwischenablage, daß diese gelöscht wurde
WM_DEVICECHANGE	Informiert eine Anwendung oder einen Gerätetreiber über eine Änderung der Hardware-Konfiguration eines Geräts oder Computers
WM_DEVMODECHANGE	Wird an alle Hauptfenster gesendet, wenn Anwender die Geräteeinstellungen geändert hat
WM_DISPLAYCHANGE	Informiert alle Fenster über Änderungen der Bildschirmauflösung
WM_DRAWCLIPBOARD	Informiert erste Anwendung in Zwischenablagebetrachter-Liste, daß der Inhalt der Zwischenablage sich geändert hat
WM_DRAWITEM	Eintrag in besitzergezeichnetem Listen- oder Kombinationsfeld hat sich geändert
WM_DROPFILES	Datei wurde über Client-Anwendung losgelassen
WM_ENABLE	Fenster wurde aktiviert oder deaktiviert, betrifft vor allem dessen Schalter
WM_ENDSESSION	Benachrichtigung, daß die Arbeitssitzung endgültig beendet wird
WM_ENTERIDLE	Informiert Fenster über ein modales Dialogfenster oder ein Meldungsfenster, das auf Eingabe wartet
WM_ENTERSIZEMOVE	Informiert Fenster, daß der Modus zur Größenänderung oder zum Verschieben aktiviert wurde
WM_ERASEBKGD	Mitteilung, daß der Hintergrund des Fensters neu gezeichnet werden muß

Windows-Botschaft	Bedeutung der Botschaft
WM_EXITSZEMOVE	Informiert Fenster, daß der Modus zur Größenänderung oder zum Verschieben beendet wurde
WM_FONTCHANGE	Der Bestand an Fonts einer Anwendung hat sich geändert
WM_GETDLGCODE	Wird an die Eingabeprozedur eines Steuerelements gesendet
WM_GETFONT	Liefert den aktuellen Font eines Kind-Steuerelements
WM_GETHOTKEY	Ruft das mit einem Fenster verbundene Tastaturkürzel auf
WM_GETICON	Ermittelt das Handle des mit einem Fenster verbundenen großen oder kleinen Symbols
WM_GETMINMAXINFO	Windows checkt die Größe des Fensters, wenn maximiert oder minimiert
WM_GETTEXT	Kopiert den Text des Fensters (meist Titel)
WM_GETTEXTLENGTH	Ermittelt die Länge des zum Text gehörenden Textes
WM_HELP	Zeigt an, daß der Anwender die F1 -Taste gedrückt hat
WM_HOTKEY	Ein Tastenkürzel wurde gedrückt
WM_HSCROLL	Die waagerechte Bildlaufleiste wurde angeklickt
WM_HSCROLLCLIPBOARD	Die waagerechte Bildlaufleiste des Zwischenablage-Besitzers wurde angeklickt
WM_ICONERASEBKGND	Hintergrund des Icons muß neu gezeichnet werden
WM_GETFONT	Liefert den aktuellen Font eines Kind-Steuerelements
WM_IME_...	Der IME erhält ein Zeichen des Konvertierungsergebnisses
WM_INITDIALOG	Wird unmittelbar vor der Darstellung eines Dialogfeldes gesendet
WM_INITMENU	Anwender hat einen Hauptmenü-Eintrag ausgewählt
WM_INITMENUPOPUP	Wird unmittelbar vor der Darstellung eines Popup-Menüs gesendet

Windows-Botschaft	Bedeutung der Botschaft
WM_INPUTLANGCHANGE	Wird an das oberste übergeordnete Fenster geschickt, nachdem das Gebietsschema eines Tasks geändert wurde
WM_INPUTLANGCHANGEREQUEST	Wird an das oberste Anwendungsfenster gesendet, wenn der Anwender eine Eingabesprache gewählt hat
WM_KEYDOWN	Eine Taste wurde gedrückt, [Alt] wurde nicht gleichzeitig gedrückt
WM_KEYUP	Eine Taste wurde losgelassen, [Alt] wurde nicht gedrückt
WM_KILLFOCUS	Benachrichtigung, daß das Fenster dabei ist, den Fokus zu verlieren
WM_LBUTTONDOWNBLCLK	Linke Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_LBUTTONDOWN	Linke Maustaste wurde gedrückt
WM_LBUTTONUP	Linke Maustaste wurde losgelassen
WM_MBUTTONDOWNBLCLK	Mittlere Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_MBUTTONDOWN	Mittlere Maustaste wurde gedrückt
WM_MBUTTONUP	Mittlere Maustaste wurde losgelassen
WM_MDIIVATE	Aktiviert und deaktiviert MDI-Kindfenster
WM_MDICASCADE	Ordnet alle MDI-Kindfester überlappend an
WM_MDICREATE	Erzeugt ein MDI-Kindfenster
WM_MDIDESTROY	Schließt ein MDI-Kindfenster
WM_MDIGETACTIVE	Liefert das aktive MDI-Kindfenster
WM_MDIICONARRANGE	Ordnet diejenigen MDI-Kindfenster, die minimiert sind
WM_MDIMAXIMIZE	Stellt MDI-Kindfenster als Vollbild dar
WM_MDIINEXT	Aktiviert das nächste MDI-Kindfenster
WM_MDIREFRESHMENU	Das Fenster-Menü des MDI-Rahmenfensters wird neu gezeichnet
WM_MDIRESTORE	Stellt Normalgröße eines MDI-Kindfensters wieder her
WM_MDISETMENU	Verbindet ein Menü mit dem MDI-Rahmenfenster

Windows-Botschaft	Bedeutung der Botschaft
WM_MDITILE	Ordnet alle MDI-Kindfenster nebeneinander an
WM_MEASUREITEM	Fordert die Maße eines besitzergezeichneten Listen- oder Kombinationsfelds oder Menüeintrags an
WM_MENUCHAR	Gedrückte Tastenkombination entspricht keinem Menüeintrag
WM_MENUSELECT	Anwender hat Menüpunkt ausgewählt
WM_MOUSEACTIVATE	Maustaste wurde in inaktivem Fenster gedrückt
WM_MOUSEMOVE	Maus wurde bewegt
WM_MOUSEWHEEL	Das Mausrad wurde rotiert
WM_MOVE	Fenster wurde verschoben
WM_MOVING	Fenster wird gerade verschoben
WM_NCACTIVATE	Der Nicht-Clientbereich eines Fensters muß neu gezeichnet werden, um seinen aktuellen Aktivierungszustand anzuzeigen
WM_NCCALCSIZE	Die Gesamtgröße des Fensters muß neu berechnet werden
WM_NCCREATE	Nicht-Clientbereich wird erzeugt. Diese Botschaft wird vor WM_CREATE gesendet
WM_NCDESTROY	Nicht-Clientbereich wird gelöscht, wird nach WM_DESTROY gesendet
WM_NCHITTEST	Wird bei Mausbewegung an Fenster gesendet, das den Maus-Capture besitzt
WM_NCLBUTTONDBLCLK	Linke Maustaste wurde doppelt betätigt, während Maus in Nicht-Clientbereich war
WM_NCLBUTTONDOWN	Linke Maustaste wurde gedrückt, während Maus in Nicht-Clientbereich war
WM_NCLBUTTONUP	Linke Maustaste wurde losgelassen, während Maus in Nicht-Clientbereich war
WM_NCMBUTTONDBLCLK	Mittlere Maustaste wurde doppelt betätigt, während Maus in Nicht-Clientbereich war
WM_NCMBUTTONDOWN	Mittlere Maustaste wurde gedrückt, während Maus in Nicht-Clientbereich war
WM_NCMBUTTONUP	Mittlere Maustaste wurde losgelassen, während Maus in Nicht-Clientbereich war

Windows-Botschaft	Bedeutung der Botschaft
WM_NCMOUSEMOVE	Maus wurde bewegt, während Maus in Nicht-Clientbereich war
WM_NCPAINT	Nicht-Clientbereich muß neu gezeichnet werden
WM_NCRBUTTONDBLCLK	Rechte Maustaste wurde doppelt betätigt, während Maus in Nicht-Clientbereich war
WM_NCRBUTTONDOWN	Rechte Maustaste wurde gedrückt, während Maus in Nicht-Clientbereich war
WM_NCRBUTTONUP	Rechte Maustaste wurde losgelassen, während Maus in Nicht-Clientbereich war
WM_NEXTDLGCTL	Setzt den Fokus auf das nächste Steuerelement innerhalb eines Dialogfelds
WM_NOTIFY	In einem Steuerelement ist ein Ereignis aufgetreten, oder das Steuerelement benötigt Informationen
WM_NOTIFYFORMAT	Legt fest, ob das Steuerelement ANSI- oder Unicode-Strukturen für die Kommunikation mit dem Eltern-Fenster verwendet
WM_PAINT	Teil des Client-Bereichs des Fensters muß neu gezeichnet werden
WM_PAINTCLIPBOARD	Client-Bereich des Zwischenablagebetrachters muß neu gezeichnet werden
WM_PAINTICON	Icon muß neu gezeichnet werden
WM_PALETTECHANGED	Die System-Farbpalette hat sich geändert
WM_PALETTEISCHANGING	System-Farbpalette wird gerade geändert
WM_PARENTNOTIFY	Informiert Elternfenster, daß Kindfenster erzeugt, gelöscht oder angeklickt wurde
WM_PASTE	Kopiert denText aus der Zwischenablage in ein Eingabefeld
WM_POWER	Informiert über Moduswechsel des APM (Advanced Power Management)
WM_POWERBROADCAST	Zeigt Power-Management-Ereignisse an
WM_PRINT	Fenster soll sich in angegebenen Gerätekontext, meistens Drucker, zeichnen
WM_PRINTCLIENT	Clientbereich eines Fensters soll sich in angegebenen Gerätekontext, meistens Drucker, zeichnen

Windows-Botschaft	Bedeutung der Botschaft
WM_QUERYDRAGICON	Symbol des Fensters wird verschoben.
WM_QUERYENDSESSION	Anwender ist dabei, Windows-Sitzung zu beenden
WM_QUERYNEWPALETTE	Teilt Fenster mit, daß es den Fokus erhalten wird
WM_QUERYOPEN	Wird an Symbol gesendet, wenn es zu Fenster expandiert werden soll
WM_QUEUESYNC	Wird von CBT-Anwendungen (Computer-Based Training) gesendet
WM_QUIT	Letzte Meldung einer Anwendung
WM_RBUTTONDOWNCLK	Rechte Maustaste wurde doppelt betätigt, Fenster hat Stilattribut CS_DBLCLKS
WM_RBUTTONDOWN	Rechte Maustaste wurde gedrückt
WM_RBUTTONUP	Rechte Maustaste wurde losgelassen
WM_RENDERALLFORMATS	Informiert den Zwischenablagebesitzer, bevor seine Anwendung geschlossen wird
WM_RENDERFORMAT	Besitzer der Zwischenablage soll Daten in angegebenem Format in die Zwischenablage kopieren
WM_SETCURSOR	Maus wird über Fenster bewegt, das nicht den Maus-Capture hat
WM_SETFOCUS	Benachrichtigung, daß Fenster den Fokus erhalten hat
WM_SETFONT	Ändert den Font von Dialogfenstern
WM_SETHOTKEY	Ein Tastaturkürzel wird mit einem Fenster verbunden
WM_SETICON	Ein kleines oder großes Icon wird mit einem Fenster verbunden
WM_SETREDRAW	Wird an Listen- oder Kombinationsfelder gesendet und beschleunigt Kopieren
WM_SETTEXT	Setzt den Text eines Fensters, üblicherweise den Titel
WM_SETTINGCHANGE	Systemparameter haben sich geändert
WM_SHOWWINDOW	Benachrichtigung, daß Fenster verborgen oder angezeigt werden soll
WM_SIZE	Größe des Fensters hat sich geändert

Windows-Botschaft	Bedeutung der Botschaft
WM_SIZECLIPBOARD	Zwischenablagebesitzer hat seine Größe geändert
WM_SIZING	Größe des Fensters wird geändert
WM_SPOOLERSTATUS	Informiert Druck-Manager, daß sich die Druckliste geändert hat
WM_STYLECHANGED	Ein oder mehrere Fensterstile wurden geändert
WM_STYLECHANGING	Ein oder mehrere Fensterstile werden gerade geändert
WM_SYSCHAR	Entsteht bei Übersetzung von WM_SYSKEYDOWN oder WM_SYSKEYUP
WM_SYSCOLORCHANGE	Systemfarben-Einstellung wurde geändert
WM_SYSCOMMAND	Ein Befehl aus dem Systemmenü wurde ausgewählt
WM_SYSDEADCHAR	Entsteht bei Drücken einer toten Taste, die den Wert der nachfolgenden Taste verändert, wenn gleichzeitig Alt gedrückt wurde
WM_SYSKEYDOWN	Wird gesendet, wenn Taste und Alt gedrückt wurden
WM_SYSKEYUP	Wird gesendet, wenn Alt gedrückt und eine andere Taste losgelassen wurde
WM_TIMECHANGE	Systemzeit wurde neu eingestellt
WM_TIMER	Ein mittels SetTimer() gesetzter Zeitgeber hat sein Intervall überschritten
WM_UNDO	Kopiert Text von der Zwischenablage zurück ins Eingabefeld
WM_USER	Der Wert von WM_USER kann als Offset für die Vergabe einer Nachrichten-ID verwendet werden
WM_USERCHANGED	Beim Ein- und Ausloggen aktualisiert das System die benutzerdefinierten Einstellungen
WM_VKEYTOITEM	Wird von LBS_WANTKEYBOARDINPUT-Listenfeldern gesendet, wenn Taste gedrückt wurde

Windows-Botschaft	Bedeutung der Botschaft
WM_VSCROLL	Die senkrechte Bildlaufleiste wurde angeklickt
WM_VSCROLLCLIPBOARD	Die senkrechte Bildlaufleiste des Zwischenablagebesitzers wurde angeklickt
WM_WINDOWPOSCHANGED	Informiert darüber, daß Größe oder Position eines Fensters von einer Windows-verwaltungsfunktion geändert wurden
WM_WINDOWPOSCHANGING	Informiert darüber, daß Größe oder Position eines Fensters von einer Windows-verwaltungsfunktion geändert werden wird

Objektorientierte Programmierung in C++

Objektorientierte Programmierung kann man nicht auf dem Weg über Referenzen und schon gar nicht auf wenigen Seiten lehren. Dieser Anhang ist daher auch gar nicht dazu gedacht, Sie in der objektorientierten Programmierung zu unterweisen. Er soll lediglich denjenigen Lesern, die mit der objektorientierten Programmierung noch nicht so vertraut sind, als kleine Hilfe dienen, um den Ausführungen in diesem Buch besser folgen und die MFC besser verstehen zu können.

Grundgedanke der objektorientierten Programmierung

Das Konzept, welches hinter den Klassen und allgemein der objektorientierten Programmierung steht, beruht auf der alltäglichen Erfahrung, daß wir die Objekte der realen Welt nach zwei Maßstäben beurteilen: einmal nach Eigenschaften wie Form und Farbe, zum anderen nach bestimmten »Verhaltensweisen«, die ein Objekt aufweist und die beispielsweise festlegen, wie man mit ihm umzugehen hat. In der objektorientierten Terminologie entsprechen die Eigenschaften den Elementvariablen und die Verhaltensweisen den Methoden (auch Elementfunktionen genannt).

Objekte, die gleiche Merkmale und Verhaltensweisen aufweisen, können in einer Klasse zusammengefaßt werden. So könnte eine Klasse `videorecor-` der folgendermaßen aussehen:




```
class videorecorder {
    char *hersteller;           /* Eigenschaften */
    int anzahl_videokoepfe;
    bool zeitlupe, longplay;
public:
    void abspielen();           /* Verhaltensweisen */
    void aufnahme();
    void in_zeitlupe_abspielen();
};
```

Bei der Bildung einer Variablen der Klasse – im folgenden als Instanzbildung bezeichnet – werden den Eigenschaften Werte zugewiesen. Auf diese Weise entsteht durch Spezialisierung ein Objekt (man sagt auch eine Instanz), das nur noch einen speziellen Videorecorder repräsentiert.

Klassen

Klassen dienen dazu, Datenelemente und Funktionen in einem Datentyp zu kapseln. Klassen gestatten es, verschiedene Zugriffsberechtigungen (`public`, `protected` und `private`) für die einzelnen Elemente zu vergeben, und können durch Vererbung Klassenhierarchien aufbauen.

Deklaration

```
class Klassenname
{
public:
    // Deklaration der Elementvariablen und Methoden;
protected:
    // Deklaration der Elementvariablen und Methoden;
private:
    // Deklaration der Elementvariablen und Methoden;
};
```

Die Zugriffsspezifizierer `public`, `private` und `protected`

Die Zugriffsspezifizierer `public`, `private` und `protected` regeln, wie von innerhalb und außerhalb der Klasse auf deren Elemente zugegriffen werden kann. Sie sind daher ein wichtiges Instrument der Kapselung. Erst durch diese Spezifizierer kann eine Klasse vor Mißbrauch geschützt werden.

Spezifizierer	Bedeutung
public	Das Element kann von jeder Methode der eigenen und abgeleiteten Klassen benutzt werden. Der Zugriff von außen erfolgt über den Namen der Instanz, beispielsweise <code>objekt.element</code> .
private	Das Element kann nur von Methoden der Klasse, in der es deklariert wurde, benutzt werden.
protected	Es gilt das gleiche wie für <code>private</code> . Zusätzlich ist jedoch die Benutzung durch Methoden von Klassen, die von der deklarierten Klasse abgeleitet sind, möglich. Gibt es keine abgeleiteten Klassen, sind <code>private</code> und <code>protected</code> identisch.

- ✗ Ein Zugriffsspezifizierer gilt für alle folgenden Klassenelemente bis zum Auftreten eines neuen Zugriffsspezifizierers.
- ✗ Zugriffsspezifizierer können nicht kombiniert werden.
- ✗ Zugriffsspezifizierer dürfen in der Klassendeklaration mehrfach auftauchen.
- ✗ Klassenelemente am Anfang der Deklaration, für die kein Zugriffsspezifizierer angegeben wurde, gelten automatisch als `private`.

Elementvariablen

Klassen bringen ihre eigenen Daten mit – die Elementvariablen. Wird eine Instanz von der Klasse gebildet, wird der Instanz Speicher für die Elementvariablen zur Verfügung gestellt. Jede Instanz hat also ihre eigene Kopie der Elementvariablen, die man daher auch Instanzvariablen nennt. Gleichzeitig wird die Instanzbildung dazu genutzt, die Datenelemente zu initialisieren. Diese Aufgabe übernimmt der Konstruktor (siehe unten).

```
class CVektor
{
public:
    int x;
    int y;
};
```

Methoden

Klassen können nicht nur Elementvariablen, sondern auch Methoden enthalten. Meist dienen diese Methoden dazu, die Daten in den Elementvariablen einzulesen, zu bearbeiten oder auszugeben. Alle Instanzen, die von einer Klasse gebildet werden, verwenden die gleichen Methoden. Durch einen

Trick (siehe `this`-Zeiger weiter unten) wird aber erreicht, daß die Methode immer auf die Elementvariablen zugreift, die zu der Instanz gehören, für die die Methode aufgerufen wurde.

```
class CVektor
{
public:
    int x;
    int y;
    Add(CVektor v);
};
```

Methoden können sowohl innerhalb als auch außerhalb der Klassendefinition definiert werden.

Definition innerhalb der Klassendefinition

```
class CVektor
{
public:
    int x;
    int y;
    void Add(CVektor vektor)
    {
        x = x + vektor.x;
        y = y + vektor.y;
    }
};
```

Definition außerhalb der Klassendefinition

Bei der Definition außerhalb der Klassendefinition muß der Klassenname mit dem Bereichsoperator vorangestellt werden.

```
class CVektor
{
public:
    int x;
    int y;
    void Add(CVektor vektor);
};
void CVektor::Add(CVektor vektor)
{
    x = x + vektor.x;
    y = y + vektor.y;
}
```

Zugriff auf Klassenelemente

Es gibt eine Vielzahl von Szenarien mit ganz unterschiedlichen Zugriffsmöglichkeiten. Die drei häufigsten sind:

Zugriff innerhalb der Klasse

Zugriff innerhalb einer Klasse bedeutet, daß innerhalb des Funktionskörpers von Methoden auf andere Elemente (Elementvariablen oder Methoden) der eigenen Klasse zugegriffen wird. Dies ist der innerste Bereich, der praktisch keinen Beschränkungen unterliegt. Der Zugang zu den Elementen erfolgt direkt über deren Namen.

```
class demo
{
private:
    int wert1, wert2;
    int func1() {return wert1 * wert2;} // direkter Zugriff
public:
    void func2(int i)
    {
        wert1 = i + 1;           // direkter Zugriff
        wert2 = func1();         // direkter Zugriff
    }
};
```

Zugriff von außerhalb der Klasse

Von außerhalb kann auf Elemente der Klasse T über eine Instanz der Klasse T in Verbindung mit einem der Operatoren `., ->, .*, ->*` zugegriffen werden.

Der Zugriff unterliegt den Einstellungen durch die Modifizierer `public`, `protected` und `private`, d.h., Elemente, die in der Klasse als `protected` oder `private` deklariert sind, können auf diese Weise nicht angesprochen werden.

```
class demo
{
    int wert1;           // standardmäßig private
    int func1();
public:
    int wert2;
    int func2();
};

int main()
```

```
{
    class demo d, *ptr_d;
    int dummy;
    dummy = d.wert1;           // Fehler, da private
    dummy = ptr_d->func1();     // Fehler, da private
    dummy = d.wert2;
    dummy = ptr_d->func2();
}
```

Zugriff auf Basisklassenelemente

Für abgeleitete Klassen muß man zwischen den eigenen und den geerbten Elementen unterscheiden.

Die Methoden, die in der abgeleiteten Klasse deklariert sind, können sowohl auf die eigenen als auch auf die geerbten Elemente direkt über den Elementnamen zugreifen.

Während der Zugriff auf die eigenen Elemente jedoch keinerlei Beschränkungen unterliegt, wird der Zugriff auf die geerbten Elemente durch die Zugriffsspezifizierer aus der Basisklassendeklaration geregelt. Eine Methode der abgeleiteten Klasse hat Zugriff auf public- und protected-Elemente, nicht aber auf private-Elemente der Basisklasse.

Geerbte private-Elemente können in den Methoden der abgeleiteten Klasse nur auf dem Umweg über public- oder protected-Methoden der Basisklasse angesprochen werden.

```
class Basis
{
    int basis_priv;
protected:
    int basis_prot;
public:
    int basis_pub;
};
class Abgeleitet : private Basis
{
public:
    void abg_func(int n)
    {
        basis_pub  = n;
        basis_prot = n;
        basis_priv = n;    // Fehler, kein Zugriff
    }
};
```

Der Konstruktor

Jede Instanzbildung eines Klassenobjekts ist mit dem Aufruf eines Konstruktors verbunden, der vom Programmierer zur Initialisierung der Datenelemente und für Eingangsarbeiten (beispielsweise der Reservierung von dynamischem Speicher) genutzt werden kann.

```
class CVektor
{
public:
    int x;
    int y;
    CVektor(int w1, int w2);    // Konstruktor
    void Add(CVektor vektor);
};
CVektor::CVektor(int w1, int w2)
{
    x = w1;
    y = w2;
}
```

Durch Definition eines oder mehrerer eigener Konstruktoren kann die Instanzbildung gesteuert werden. Dabei ist unter anderem zu beachten:

- ✗ Ein Konstruktor hat immer den gleichen Namen wie seine Klasse.
- ✗ Ein Konstruktor hat keinen Rückgabewert (auch nicht `void`).
- ✗ Konstruktoren können überladen werden. (Die Überladung wird meist dazu genutzt, dem Konstruktor Argumente für die Initialisierung der Datenelemente der Klasse zu übergeben.)
- ✗ Konstruktoren werden nicht vererbt.
- ✗ Es können keine Zeiger auf Konstruktoren definiert werden.
- ✗ Konstruktoren können nach der Instanzbildung nicht mehr aufgerufen werden.

Um sicherzustellen, daß jede Klasse über einen Konstruktor verfügt, weist der Compiler Klassen ohne explizit deklarierten Konstruktor einen Standardkonstruktor zu (Konstruktor, der ohne Argument aufgerufen werden kann).



Standardkonstruktor

Ein Standardkonstruktor ist ein Konstruktor, der ohne Argument aufgerufen werden kann.

Eine Klasse ohne Konstruktor bekommt vom Compiler einen Standardkonstruktor zugewiesen.

Wird ein Konstruktor explizit definiert, steht der Standardkonstruktor des Compilers nicht mehr zur Verfügung; der Programmierer kann aber einen eigenen Standardkonstruktor definieren.

Der Standardkonstruktor vereinfacht die Einrichtung eingebetteter und geerbter sowie von Arrays von Klassenobjekten, die ansonsten nur durch Angabe von Konstruktorlisten bzw. Initialisierungslisten zu bewerkstelligen sind.

Der Destruktor

Destrukturen werden automatisch aufgerufen, wenn Klasseninstanzen ihre Gültigkeit verlieren (beispielsweise bei Verlassen des Gültigkeitsbereichs oder bei delete-Aufrufen für Zeiger auf Klasseninstanzen). Sie lösen die Instanz auf und geben den reservierten Speicher frei.

```
~klassenname() {}
```

Durch die Definition von Destrukturen kann die Instanzauflösung gesteuert werden (was im Grunde nur dann erforderlich ist, wenn bestimmte Ressourcen wie dynamisch reservierter Speicher oder Datei-Handles freizugeben sind). Dabei ist unter anderem zu beachten:

- ✗ Destrukturen tragen immer den Namen ihrer Klasse, jedoch mit vorangehender Tilde.
- ✗ Ein Destruktor hat keinen Rückgabewert (auch nicht `void`) und übernimmt keine Argumente
- ✗ Eine Klasse ohne explizit deklarierten Destruktor bekommt einen Standarddestruktor zugewiesen.
- ✗ Destrukturen werden nicht vererbt.
- ✗ Es können keine Zeiger auf Destrukturen definiert werden.

Der this-Zeiger

Werden mehrere Instanzen einer Klasse gebildet, bekommt jede Instanz eine eigene Kopie der Elementvariablen der Klasse. Die Methoden der Klasse stehen aber nur einmal im Arbeitsspeicher und werden von allen In-

stanzen einer Klasse gemeinsam benutzt. Da die Methoden üblicherweise auf den Elementdaten der Klasse operieren und diese instanzspezifisch sein sollen, muß sichergestellt werden, daß die jeweilige Methode nur auf die Daten zugreift, die zur aufrufenden Instanz gehören.

Der Compiler weist aus diesem Grunde jeder Instanz einen Zeiger auf sich selbst zu, den `this`-Zeiger. Bei jedem Aufruf einer Methode wird dieser Zeiger automatisch an die Methode übergeben. Somit ist der `this`-Zeiger in jeder Methode verfügbar (mit Ausnahme statischer Methoden).

Instanzbildung

Als Instanzbildung bezeichnet man die Einrichtung von Klassenobjekten (Variablen eines Klassentyps). Die Instanzbildung ist stets mit dem Aufruf eines Konstruktors verbunden, der für die Einrichtung des Speichers und die Initialisierung der Datenelemente verantwortlich ist.

```
class demo
{
    int privat;
public:
    demo() {};           // Standardkonstruktor
    demo(int i)         {privat = i;}
};
int main()
{
    demo obj1();
    demo obj2(14);
}
```

Vererbung und Polymorphismus

Vererbung ist ein weiteres, ganz wesentliches Konzept der objektorientierten Programmierung, welches es ermöglicht, neue Klassen auf der Basis bereits vorhandener Klassen zu definieren.

Die einfachste Form der Vererbung besteht aus einer Basisklasse und einer abgeleiteten Klasse. Durch die Vererbung wird festgelegt, daß die abgeleitete Klasse – zusätzlich zu den in ihrer Definition aufgeführten Eigenschaften (Datenelemente und Methoden) – über sämtliche Elemente der Basisklasse verfügt.

```
class Klassenname : BASISKLASSENLISTE
{
```



```
public:
    // Deklaration eigener Elementvariablen und Methoden;
    ...
};
```

In der Basisklassenliste werden, durch Kommata voneinander getrennt, die Basisklassen aufgeführt, deren Eigenschaften geerbt werden sollen. Ein Eintrag in dieser Liste hat folgende Syntax:

```
[virtual] [public, protected, private] Basisklassenname
```

Die Zugriffsspezifizierer `public`, `protected` und `private` regeln bei der Vererbung nicht den Zugriff aus den Methoden der abgeleiteten Klassen auf die geerbten Elemente (dies wird schon durch die Zugriffsspezifizierer in der Deklaration der Basisklasse geregelt), sondern welche Zugriffsrechte die abgeleitete Klasse für die geerbten Elemente nach außen weitergibt (beispielsweise, wenn über eine Instanz der abgeleiteten Klasse auf diese Elemente zugegriffen wird oder wenn die abgeleitete Klasse selbst als Basisklasse benutzt wird). Wird kein Zugriffsspezifizierer angegeben, wird die Klasse standardmäßig `private` vererbt.

Das Schlüsselwort `virtual` ist nur für die Mehrfachvererbung von Bedeutung.

Vererbung und Konstruktor

Konstrukoren und Destruktoren werden nicht vererbt. Daraus folgt, daß bei der Instanzbildung einer abgeleiteten Klasse die ererbten Datenelemente vom Konstruktor der jeweiligen Basisklasse eingerichtet werden müssen. Tatsächlich bilden die von einer Basisklasse geerbten Elemente in der abgeleiteten Klasse eine Art Unterobjekt mit eigenen Konstruktoren, Destruktoren, Zugriffsrechten etc. Dieses Konzept ist die Grundlage für die in der objektorientierten Programmierung sehr gebräuchliche Reduzierung eines abgeleiteten Objekts in ein Objekt einer seiner Basisklassen.

Nutzung des Standardkonstruktors der Basisklasse

Ist in der Basisklasse ein Standardkonstruktor (siehe oben) vorhanden, braucht sich die abgeleitete Klasse überhaupt nicht um die Einrichtung und die Initialisierung der geerbten Elemente zu kümmern, sondern kann dies dem Compiler überlassen, der automatisch im Zuge der Instanzbildung für Objekte der abgeleiteten Klasse den Standardkonstruktor der Basisklasse aufruft.

Aufruf eines speziellen Konstruktors der Basisklasse

Um einen Konstruktor mit Argumenten für die Einrichtung der Basisklassenobjekte aufzurufen, muß man eine Konstruktorliste anlegen.

```
Abgeleitet(int b1, double b2) : Basis1(b1), Basis(b2) { }
```

Die Konstruktorliste folgt mit einem Doppelpunkt auf den Namen des abgeleiteten Konstruktors.

Um Argumente aus der Instanzbildung an einen Basisklassenkonstruktor weiterzugeben, muß für das jeweilige Argument im Konstruktor der abgeleiteten Klasse ein Parameter für das Argument definiert werden.

```
class Basis {
public:
    int b_wert;
    Basis(int par)      { b_wert = par;}
};
class Abgeleitet : public Basis {
public:
    int abg_wert;
    Abgeleitet(int i)                { abg_wert = i;}
    Abgeleitet(int i, int n) : Basis(n) { abg_wert = i;}
};
int main()
{
    class Abgeleitet obj(1, 2);
    ...
}
```

Polymorphismus

Der Begriff des Polymorphismus ist eng an das Konzept der Vererbung gebunden. Er bezeichnet das Phänomen, daß eine Methode in verschiedenen abgeleiteten Klassen einer Hierarchie unterschiedlich implementiert sein kann.

Auf diese Weise können einander ähnliche Klassen über eine Reihe gleichnamiger und gleichartig zu benutzender Methoden verfügen, die es dem Programmierer erlauben, Instanzen dieser Klassen gleich zu behandeln, ohne auf klassenspezifische Details, die in der Implementierung der Methoden versteckt sind, zu achten.

Es liegt nahe, Methoden, die von mehreren Klassen benötigt werden, über eine gemeinsame Basisklasse zu vererben. In den abgeleiteten Klassen werden die Methoden überschrieben, um eine klassenspezifische Behandlung zu

implementieren. Methoden, die überschrieben werden sollen, werden üblicherweise in der Basisklasse mit dem Schlüsselwort `virtual` deklariert.

```
class ZeichenObjekt {
protected:
    struct Punkt referenzpunkt;
    virtual int zeichne(struct Punkt p) {
        // zeichne Referenzpunkt an Koordinate p
        ...
    }
};

class Rechteck : public ZeichenObjekt {
protected:
    struct Punkt ecken[4];
    virtual int zeichne(struct Punkt p) {
        // zeichne Rechteck mit linker unterer Ecke in
        // Koordinate p
        ...
    };
};

class Kreis : public ZeichenObjekt {
protected:
    float radius;
    virtual int zeichne(struct Punkt p) {
        // zeichne Kreis mit Mittelpunkt in Koordinate p
        ...
    }
};
```

Überschreibung und virtuelle Methoden

Überschreibung bedeutet, daß eine Methode, die bereits in einer Basisklasse definiert ist, in einer abgeleiteten Klasse neu definiert wird. Um zu erreichen, daß für Objekte der abgeleiteten Klasse stets die Implementierung aus der abgeleiteten Klasse und nicht die Basisklassenversion aufgerufen wird, muß man die Methode in der Basisklasse als `virtual` deklarieren. (Grundsätzlich sollte man die Methode auch in der abgeleiteten Klasse als `virtual` deklarieren.)

Basisklassenzeiger

Da in einer abgeleiteten Klasse die Elemente der Basisklassen als Unterobjekte enthalten sind, kann man ein abgeleitetes Objekt wie eines seiner

Basisklassenobjekte behandeln, es quasi auf sein Basisklassenunterobjekt reduzieren.

Insbesondere in Verbindung mit Basisklassenzeigern und virtuellen Methoden ist dies ein nützliches Konzept, das in vielen Fällen äußerst effektive und elegante generische Lösungen erlaubt.

Ein Basisklassenzeiger ist wie ein generischer Zeiger, dem man die Adressen abgeleiteter Objekte zuweisen kann (unter der Voraussetzung, daß das Basisklassenunterobjekt in dem abgeleiteten Objekt eindeutig identifizierbar und zugänglich ist). Mit Hilfe dieser generischen Zeiger kann man

- ✕ Objekte verschiedener abgeleiteter Klassen mit gemeinsamer Basisklasse in einem Array (oder dynamischen Feld) verwalten.

```
class ZeichenObjekt {
    ... // Implementierung wie oben
};
class Rechteck : public ZeichenObjekt {
    ...
};
class Kreis : public ZeichenObjekt {
    ...
};
int main()
{
    class ZeichenObjekt *geomFig[10];
    geomFig[0] = new Kreis;
    geomFig[1] = new Rechteck;
    ...
}
```

- ✕ generische Funktionen/Methoden implementieren, denen man Objekte verschiedener abgeleiteter Klassen mit gemeinsamer Basisklasse als Argumente übergeben kann.

```
void markieren(const ZeichenObjekt& objekt)
{
    ...
}
```

Um sinnvoll mit Basisklassenzeiger auf abgeleitete Objekte arbeiten zu können, muß es aber auch eine Möglichkeit geben, wieder auf die volle Funktionalität des abgeleiteten Objekts zuzugreifen. Dafür gibt es verschiedene Möglichkeiten.

- ✗ Rückverwandlung von Basisklassenzeigern
- ✗ `dynamic_cast`-Operator
- ✗ Nutzung von Laufzeitinformationen

Überladung und Vorgabeargumente

Die beiden nachfolgenden Konzepte sind nicht an Klassen gebunden. Sie gelten für Methoden wie für normale Funktionen.

Vorgabeargumente

In C++ können in der Funktionsdeklaration Vorgabeargumente vergeben werden. Parametern, für die Vorgabeargumente deklariert wurden, brauchen beim Funktionsaufruf keine Argumente mehr übergeben zu werden. Auf diese Weise kann der Aufruf für häufig übergebene Argumente vereinfacht werden.

```
int func (int n, int m = 3);
```

Auf einen Parameter mit Vorgabeargument dürfen keine Parameter ohne Vorgabeargumente mehr folgen.

Überladung von Funktionen

C++ erlaubt die Definition mehrerer Funktionen (oder Operatoren) gleichen Namens, sofern die Funktionen sich in Anzahl oder Typ der Parameter unterscheiden. Prinzipiell gilt für C++ ebenso wie für C, daß Objekte über einen eindeutigen Namen adressiert werden können. C++ lockert dieses Dogma für den Programmierer etwas auf, indem Bezeichner automatisch um codierte Informationen zu den Parametern respektive Operanden erweitert werden. Auf diese Weise kann der Programmierer identische Bezeichner vergeben, während der Compiler mit eindeutigen Namen arbeitet.

Die Überladung von Funktionen wird eingesetzt, um das Verhalten einer Routine an die ihr übergebenen Parametertypen anzupassen. Der Vorteil für den Programmierer besteht darin, daß er nicht mehrere Funktionsnamen für eine Routine zu vergeben braucht, und der Compiler die Aufgabe übernimmt, die Parameter zu checken und die richtige Version der Routine aufzurufen.

```
int max(int a, int b) {  
    return ( a > b ? a : b);  
}
```

```
const char* max(const char* s1, const char* s2) {  
    return (strcmp(s1, s2) ? s1 : s2);  
}  
int max(int a, char* s) {  
    return ( a > strlen(s) ? a : strlen(s));  
}
```

Überladene Funktionen müssen in einem gemeinsamen Gültigkeitsbereich liegen, sonst findet keine Überladung, sondern die übliche Verdeckung statt.

Da die Auswahl der richtigen Funktion anhand der übergebenen Argumente erfolgt, können Parametertypen, die gleiche Argumenttypen akzeptieren, nicht unterschieden werden.

Ererbte Methoden können in abgeleiteten Klassen nicht überladen werden.

Antworten zu den Fragen

In diesem Anhang finden Sie die Antworten zu den Fragen der einzelnen Kapitel.



Kapitel 1

Konsolenanwendungen sind Anwendungen ohne grafische Windows-Oberfläche, die im MD-DOS-Eingabeaufforderungsfenster ausgeführt werden. *Antwort 1*

GUI-Anwendungen sind Anwendungen mit grafischer Windows-Oberfläche.

IDE steht für eine integrierte Entwicklungsumgebung, von der aus alle für die Programmerstellung wichtigen Werkzeuge und Informationen aufgerufen werden können. *Antwort 2*

Nein, MFC-Programme müssen nicht notwendigerweise in der IDE von Visual C++ erstellt werden. MFC-Programme können auch über die Kommandozeile kompiliert werden. Allerdings ist dies für umfangreichere Programme meist viel zu aufwendig. *Antwort 3*

Der Visual C++-Compiler heißt *cl.exe*. *Antwort 4*

Um in Visual C++ ein neues Programm zu erstellen, legt man zuerst ein passendes Projekt an (DATEI/NEU). *Antwort 5*

Die Dateien *StdAfx.cpp* und *StdAfx.h* dienen der Erstellung und Nutzung eines vorkompilierten Headers, der nachfolgende Kompilationen beschleunigt (siehe Kapitel 2). *Antwort 6*

Kapitel 2

Antwort 1 Die wichtigsten Elemente der Projektverwaltung sind:

- ✗ Das Dialogfeld NEU zum Anlegen neuer Projekte.
- ✗ Das Arbeitsbereichsfenster als Schaltzentrale der Projektverwaltung.
- ✗ Die Projekteinstellungen zur Konfiguration des Compilers und des Linkers.

Antwort 2 Die für uns am interessantesten Projekttypen und Assistenten sind:

- ✗ der MFC-Anwendungs-Assistent (exe) für MFC-Programme
- ✗ Win32-Konsolenanwendung für Konsolenanwendungen
- ✗ Win32-Anwendung für Windows-API-Programme

Antwort 3 Bestehende Projekte lädt man über den Befehl DATEI/ARBEITSBEREICH ÖFFNEN. In dem erscheinenden Öffnen-Dialog wählt man die *dsw*-Datei des Arbeitsbereichs aus, in dem das Projekt abgelegt ist.

Antwort 4 Wechseln Sie im Arbeitsbereichsfenster zur Seite DATEIEN, und doppelklicken Sie auf den Knoten der Datei.

Antwort 5 Wechseln Sie im Arbeitsbereichsfenster zur Seite KLASSEN, und doppelklicken Sie auf den Knoten der Methode.

Antwort 6 Dialoge, Menüs, Tastaturkürzel, Stringtabellen, Bitmaps, Anwendungssymbole, Versionsinformationen etc.

Antwort 7 Die Datei *Stdafx.cpp* wird als erstes kompiliert, weil auf ihrer Grundlage der vorkompilierte Header für die anderen *cpp*-Quelltextdateien erstellt wird.

Kapitel 3

Enthält keine Fragen.

Kapitel 4

Antwort 1 Ein Bug ist ein Fehler, der erst bei Ausführung des Programms auftritt (nicht notwendigerweise bei jeder Ausführung des Programms) und entweder auf falschen Anweisungen (ungültige Indizes, Dereferenzierung nicht-initialisierter Zeiger) oder einem falschen Programmablauf (logische Fehler) basiert. Der Begriff »Bug« (englisch für »Wanze«, »Insekt«) geht auf einen Vorfall an der Harvard University zurück, wo eine in die Schaltungen eingedrungene Motte den Computer lahmlegte.

Ein Debugger ist ein Hilfsprogramm, das andere Programme schrittweise ausführt und kontrolliert. *Antwort 2*

Aufgabe des Debuggers ist es, andere Programme zeilenweise auszuführen und dabei den Zustand des Programms zu überwachen und gegebenenfalls anzuzeigen. Zu diesem Zweck benötigt der Debugger den Maschinencode des Programms, den Quelltext und spezielle Informationen, die es ihm ermöglichen, eine Verbindung zwischen den Quelltextzeilen und dem Maschinencode herzustellen sowie eine Tabelle mit den Symbolen des Programms (Namen von Variablen, Funktionen, Klassen etc.) *Antwort 3*

Die wichtigsten Techniken beim Debuggen sind das Anhalten und schrittweise Ausführen des Programms und das Begutachten der aktuellen Variablenwerte. *Antwort 4*

Mit dem ASSERT-Makro können boolesche Bedingungen überprüft werden. Mit dem TRACE-Makro können Diagnoseausgaben in ein Programm aufgenommen werden, die bei Ausführung des Programms im Debugger im Ausgabefenster von Visual C++ angezeigt werden. *Antwort 5*

Um die Debug-Makros ASSERT und TRACE zu deaktivieren, braucht man das Programm nur in der Release-Konfiguration zu erstellen. *Antwort 6*

Kapitel 5

Die vier wichtigsten Klassen des MFC-Anwendungsgerüsts sind *Antwort 1*

- ✗ die Anwendungsklasse, die die Anwendung selbst repräsentiert
- ✗ die Rahmenfensterklasse, die das Hauptfenster der Anwendung repräsentiert
- ✗ die Ansichtsfensterklasse, die den Ausgabebereich der Anwendung repräsentiert
- ✗ die Dokumentklasse, die die Daten der Anwendung verwaltet

Für alle diese Klassen werden im Anwendungsgerüst eigene Klassen von passenden MFC-Basisklassen abgeleitet.

Die Dokument/Ansicht-Architektur beruht auf der Idee der Trennung von Datenverwaltung und Datenanzeige. *Antwort 2*

Um in einem Doc/View-Anwendungsgerüst Daten auszugeben, lädt man die Daten in das Dokumentobjekt. In der OnDraw()-Methode der Ansichtsfensterklasse greift man auf das Dokumentobjekt zu, liest die Daten ein, bereitet sie eventuell noch auf und gibt sie dann im Ansichtsfenster aus. *Antwort 3*

Kapitel 6

- Antwort 1* Wenn Sie mit dem MFC-Anwendungs-Assistenten ein Doc/View-Anwendungsgerüst erstellen lassen, werden die Fenster im Zuge der Erstellung der Dokumentvorlage erzeugt, so daß von den `Create()`-Aufrufen zur Erzeugung der Fenster im Quelltext nichts zu sehen ist. Anders sieht es freilich aus, wenn Sie ein Anwendungsgerüst ohne Doc/View erstellen wollen.
- Antwort 2* Die Methode `PreCreateWindow()` gibt dem Programmierer die Möglichkeit an die Hand, vor der Erzeugung eines Fensters das Aussehen und Verhalten des Fensters festzulegen. In Doc/View-Anwendungsgerüsten entschädigt die `PreCreateWindow()`-Methode daher für den vor dem Programmierer verborgenen `Create()`-Aufruf.
- Antwort 3* Die Anfangsgröße des Hauptfensters kann in der `PreCreateWindow()`-Methode festgelegt werden. Man muß dazu nur den `cx`- und `cy`-Elementen des `cs`-Parameters die gewünschten Werte für Breite und Höhe zuweisen.
- Antwort 4* Eine Beschreibung der verschiedenen Fensterstile finden Sie in Tabelle 6.1 sowie der Online-Hilfe zu Visual C++.
- Antwort 5* Wenn Sie nicht möchten, daß die Größe Ihres Hauptfensters vom Anwender verändert werden kann, müssen Sie in der `PreCreateWindow()`-Methode den Stil `WS_THICKFRAME` für den veränderbaren Rahmen aus dem Fensterstil löschen.

Kapitel 7

- Antwort 1* Die »Message Loop« ist eine `while`-Schleife, in der eine Windows-Anwendung die Nachrichten aus ihrer Nachrichtenwarteschlange einliest und an die Fenster und Behandlungsmethoden verteilt. In MFC-Anwendungen ist die Implementierung dieser Schleife in den MFC-Basisklassen versteckt.
- Antwort 2* Um mit Hilfe des Klassen-Assistenten eine Behandlungsmethode zu einer Nachricht einzurichten:
- ✗ Wechseln Sie zur Seite NACHRICHTENZUORDNUNGSTABELLEN.
 - ✗ Wählen Sie in den Feldern KLASSENNAME und OBJEKT-IDS die Klasse, in der die Behandlungsmethode deklariert und definiert werden soll.
 - ✗ Im Feld NACHRICHTEN scrollen Sie bis zu der zu behandelnden Nachricht.
 - ✗ Drücken Sie den Schalter FUNKTION HINZUFÜGEN.
 - ✗ Drücken Sie den Schalter CODE BEARBEITEN.

WM_LBUTTONDOWNBLCLK

Antwort 3

WM_LBUTTONDOWN

WM_LBUTTONUP

Die Nachricht WM_PAINT wird von Windows an ein Fenster gesendet, wenn ein Neuzeichnen des Fensters nötig wird (Fenster wurde aus dem Hintergrund in den Vordergrund gehoben, Fensterrahmen wurde aufgezo- gen). Mit Hilfe der Methode UpdateWindow() können Anwendungen selbst WM_PAINT-Nachrichten verschicken und dadurch ein Neuzeichnen ihres Fensters erzwingen. Antwort 4

In MFC-Anwendungen wird die Nachricht WM_PAINT von der internen Methode OnPaint() abgefangen. Diese ruft die Methode OnDraw() auf. In dieser Methode setzen Sie den Code zum Zeichnen des Fensterinhalts auf. Antwort 5

Um bestimmte Arbeiten in regelmäßigen Abständen ausführen zu lassen, kann man einen Zeitgeber einrichten und sich von Windows WM_TIMER-Nachrichten schicken lassen. Antwort 6

Kapitel 8

[Alt]-Tastenkombinationen für Menübefehle richtet man im Eigenschaften-Dialog des Menübefehls ein, indem man im Titel des Menübefehls dem Buchstaben für die [Alt]-Tastenkombination ein Kaufmännisches Und (&) voranstellt. Antwort 1

Tastaturkürzel für Menübefehle richtet man als Tastaturkürzel-Ressource (Accelerator) ein. Wichtig ist, daß das Tastaturkürzel mit der Ressourcen-ID des Menübefehls verbunden wird. Will man das Tastaturkürzel im Titel des Menübefehls anzeigen, fügt man zwischen dem eigentlichen Titel und dem Tastaturkürzel die Zeichenfolge \t ein. Antwort 2

Quickinfos für Menübefehle richtet man im EIGENSCHAFTEN-Dialog des Menübefehls ein, indem man im Feld STATUSZEILENTEXT an den Hilfetext für die Statusleiste die Zeichenfolge \n und den Hilfetext für das Quickinfo anhängt. Antwort 3

Die Klasse CMenu benötigt man zur Implementierung von Kontextmenüs oder zur dynamischen Anpassung des Menüaufbaus zur Laufzeit. Antwort 4

Kapitel 9

- Antwort 1* Die MFC-Klasse für Schalter heißt `CButton`.
- Antwort 2* Die MFC-Klasse für Optionsfelder heißt ebenfalls `CButton`. Um aus einem Schalter ein Optionsfeld zu machen, übergibt man bei der Erzeugung des Schalters der `Create()`-Methode den Fensterstil `BS_AUTORADIOBUTTON`.
- Antwort 3* Ein Kombinationsfeld ist eine Kombination aus Eingabefeld und Listenfeld.
- Antwort 4* Der Text statischer Textfelder kann zur Laufzeit von dem Programm, nicht aber dem Anwender verändert werden.
- Antwort 5* Für Paßwortabfragen verwendet man Eingabefelder, die man mit dem Fensterstil `ES_PASSWORD` erzeugt, so daß statt der eingetippten Zeichen im Eingabefeld nur Sternchen zu sehen sind.

Kapitel 10

- Antwort 1* Steuerelemente richtet man am geschicktesten mit Hilfe des Menübefehls `LAYOUT/AUSRICHTEN` aus (wozu die auszurichtenden Steuerelemente zuvor markiert und das Referenzelement korrekt bestimmt werden müssen).
- Antwort 2* Um Steuerelemente pixelweise im Dialog zu verschieben, markiert man das Steuerelement und drückt dann die Pfeiltasten auf der Tastatur.
- Antwort 3* ✗ Zur visuellen Gruppierung von Optionsfeldern verwendet man das Steuerelement Gruppenfeld.
- ✗ Die interne Gruppierung wird bestimmt durch die Definition der jeweils ersten Elemente einer Gruppe und die Tabulator-Reihenfolge der Elemente. Um ein Element als erstes Element einer Gruppe festzulegen, rufen Sie das Kontextmenü des Steuerelements auf, wählen den Befehl `EIGENSCHAFTEN` aus und aktivieren die Option `GRUPPE`. Alle Steuerelemente, die in der Tabulator-Reihenfolge auf dieses Element folgen, bilden dann eine Gruppe, bis zum nächsten ersten Element einer Gruppe.
- Antwort 4* Dialoge werden von der MFC-Basisklasse `CDialog` abgeleitet.
- Antwort 5* Dialogklassen richtet man am bequemsten mit Hilfe des Klassen-Assistenten ein. Man erstellt vorab die Dialog-Ressource und ruft dann den Klassen-Assistenten auf, der für alles weitere Sorge trägt.
- Antwort 6* Die Methode `DoDataExchange()` dient dem Datenaustausch zwischen den Steuerelementen eines Dialogfeldes und den zugehörigen Elementvariablen der Dialogklasse. Die Methode regelt den Datenaustausch in beiden Richtungen.

Ein typischer nicht-modaler Dialog, der sich in vielen Windows-Programmen findet, ist der Suchen-Dialog. *Antwort 7*

Kapitel 11

Am schnellsten gibt man Text mit Hilfe eines Meldungsfensters aus (Methode `AfxMessageBox()`). *Antwort 1*

Welche Schriftart für eine Textausgabe mit `TextOut()` oder `DrawText()` verwendet wird, hängt von dem `CFont`-GDI-Objekt ab, das in den Gerätekontext für die Ausgabe geladen wurde. In Kapitel 12 werden wir uns eingehender mit Gerätekontexten und GDI-Objekten beschäftigen. Speziell die Programmierung mit `CFont`-Objekten wird im Rahmen dieses Buches allerdings nicht behandelt. Schauen Sie in der weiterführenden Literatur oder der Online-Hilfe nach (Indexeinträge `CFont` und `EnumFonts()`). *Antwort 2*

Selbstverständlich können Sie in Ihren MFC-Anwendungen die C++-Klasse `string` verwenden. *Antwort 3*

Für Dateioperationen stellt die MFC die Klasse `CFile` zur Verfügung. *Antwort 4*

Selbstverständlich können Sie in Ihren MFC-Anwendungen neben `CFile` auch die C/C++-Funktionen und -Klassen zur Dateibehandlung verwenden. *Antwort 5*

Klassen, deren Objekte serialisiert werden sollen, müssen von `CObject` abgeleitet sein, die `CObject`-Methode `Serialize()` überschreiben und mit dem `DECLARE_SERIAL`-Makro deklariert und mit `IMPLEMENT_SERIAL` implementiert werden. *Antwort 6*

Kapitel 12

Um in ein Fenster zu zeichnen, braucht man *Antwort 1*

- ✗ eine Leinwand – den Gerätekontext
- ✗ Zeichenwerkzeuge – die GDI-Objekte
- ✗ Zeichenoperationen – die CDC-Methoden

Gerätekontexte verfügen von Haus aus über einen Satz vordefinierter Zeichenwerkzeuge. *Antwort 2*

Sofern Sie nicht in der Methode `OnDraw()`, sondern einer anderen Methode Ihrer Ansichtsklasse zeichnen wollen, müssen Sie sich einen eigenen Gerätekontext für die Klasse beschaffen. Dazu deklariert man eine Instanz der Klasse `CClientDC` und übergibt dem Konstruktor den `this`-Zeiger auf das Ansichtsfensterobjekt. *Antwort 3*

- Antwort 4* Ein GDI-Objekt kann man nur dadurch aus einem Gerätekontext löschen, daß man ein anderes GDI-Objekt der gleichen Kategorie in den Gerätekontext lädt.
- Antwort 5* Um Linien auszugeben, setzt man mit Hilfe der Methode `MoveTo()` die aktuelle Zeichenposition auf den Anfangspunkt der Linie. Dann zeichnet man mit Hilfe der Methode `LineTo()` eine Linie von dort bis zum Endpunkt der Linie.
- Antwort 6* Einige CDC-Zeichenmethoden arbeiten mit dem Konzept des umschließenden Rechtecks. Das umschließende Rechteck ist nichts anderes als das kleinste Rechteck, das man um die gewünschte Form herum einzeichnen könnte. Statt beispielsweise einen Kreis durch Mittelpunkt und Radius zu definieren, gibt man das umschließende Rechteck an, und die Zeichenmethode paßt beispielsweise einen Kreis in dieses Rechteck ein.
- Antwort 7* Farben werden als RGB-Werte an MFC-Methoden übergeben. Dazu kann man die Farben entweder als hexadezimale 32-Bit-Werte oder als Rückgabewert der Funktion `RGB()` übergeben.

Kapitel 13

- Antwort 1* Es gibt keine Standardbitmaps, über die automatisch jeder Gerätekontext verfügt.
- Antwort 2* Um ein Bitmap in einen vorgegebenen Rahmen einzupassen, verwendet man die Methode `StretchBlt()`.
- Antwort 3* Natürlich kann man Bitmaps auch aus Dateien laden, doch muß man dies selbst implementieren (jedenfalls ist mir keine MFC-Methode zum Laden von Bitmaps aus Dateien, statt aus Ressourcen, bekannt). Dies ist wegen des speziellen Dateiformats für Bitmaps nicht ganz einfach. Ich muß Sie daher diesbezüglich an die Fachliteratur verweisen.

Kapitel 14

- Antwort 1* Die »Message Loop« ist eine `while`-Schleife, in der eine Windows-Anwendung die Nachrichten aus ihrer Nachrichtenwarteschlange einliest und an die Fenster und Behandlungsmethoden verteilt. In MFC-Anwendungen ist die Implementierung dieser Schleife in den MFC-Basisklassen versteckt.
- Antwort 2* Nicht alle Nachrichten werden über die Message Loop geleitet. Viele interne Windows-Nachrichten werden direkt an Fensterfunktionen gesendet. Für Nachrichten zu Benutzerereignissen ist die chronologische Abarbeitung, die durch die Message Loop sichergestellt wird, aber unabdingbar.

Welche Fensterfunktion zu einem Fenster gehört, ist in der Fensterklasse des Fensters festgehalten. *Antwort 3*

Die Fensterfunktionen werden von Windows aufgerufen. *Antwort 4*

Schließt der Anwender das Hauptfenster einer Anwendung, um diese zu beenden, löst er eine WM_DESTROY-Nachricht aus. Diese muß in der Fensterfunktion abgefangen und mit einer WM_QUIT-Nachricht beantwortet werden. Letztere sorgt dann dafür, daß die Message Loop verlassen und die Anwendung beendet wird. *Antwort 5*

Das Antworttabellenmakro zu der Windows-Nachricht WM_SIZE lautet ON_WM_SIZE, die zugehörige Behandlungsmethode heißt OnSize(). *Antwort 6*

Selbstverständlich kann man in MFC-Anwendungen Gerätekontexte auch durch Aufruf der API-Funktion GetDC() erzeugen. Dann muß man allerdings auch ReleaseDC() zum Löschen des Gerätekontextes selbst aufrufen. *Antwort 7*

Kapitel 15

Enthält keine Fragen.

Kapitel 16

Das m_pCurrentDoc-Element der CCreateContext-Struktur enthält einen Zeiger auf das Dokument, das mit der Ansicht verbunden wird. *Antwort 1*

Die Methode OnCreateClient() wird vom Anwendungsgerüst aus der Methode OnCreate() heraus aufgerufen. (Zur Erinnerung: OnCreate() wird zum Abschluß der Create()-Methode zur Erzeugung eines Fensters aufgerufen – also nachdem das Windows-Fenster bereits existiert.) Rufen Sie diese Methode niemals selbst auf. *Antwort 2*

Kapitel 17

Enthält keine Fragen.

Kapitel 18

Ein Handle ist ein 32-Bit-Wert, der von Windows vergeben wird und der eindeutigen Identifizierung von Windows-Objekten (Fenster, Gerätekontexte, Dateien) dient. *Antwort 1*

Die API-Funktion TextOut() kann im Gegensatz zur MFC-Methode nur mit ASCII-Strings und nicht mit CString-Argumenten aufgerufen werden. *Antwort 2*

◀ **ANHANG** **C** Antworten zu den Fragen

Antwort 3 Um sich einen Überblick über die Funktionen der Windows-API zu verschaffen, rufen Sie das Inhaltsverzeichnis der Online-Hilfe auf und expandieren dort die Knoten PLATFORM-SDK und REFERENCE. Unter dem Knoten REFERENCE finden Sie Untereinträge, in denen die API-Funktionen alphabetisch oder nach Themen geordnet aufgeführt werden.

Buch-CD und Autoren- Edition

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. Es handelt sich dabei um die deutschsprachige Introductory-Version, die weitestgehend der im Handel erhältlichen Standard-Version entspricht, aber einige Einschränkungen aufweist.

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 fragt Sie gegen Ende der Installation, ob es für Sie die Batch-Datei *vcvars32.bat* anlegen und im BIN-Verzeichnis des Visual C++-Compilers speichern soll. Stimmen Sie dem auf jeden Fall zu: In dieser Datei sind unter anderem die Pfade zu den Kommandozeilenversionen des Compilers und des Linkers enthalten.
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.)

Ausführung der Beispielprogramme

Um die Beispielprogramme auf der CD auszuprobieren, kopieren Sie die entsprechenden Projektverzeichnisse einfach auf Ihre Festplatte, heben Sie eventuell den Schreibschutz auf, und laden Sie die Arbeitsbereichsdatei des Programms (Extension *.dsu*) in Visual C++.

Bevor Sie die Internet-Programme aus Kapitel 24 ausprobieren, müssen Sie eine Netzwerkverbindung herstellen, über die die Programme ins Internet kommen (üblicherweise eine DFÜ-Netzwerkverbindung).

Bevor Sie die Datenbankprogramme aus Kapitel 25 ausprobieren, müssen Sie die entsprechenden ODBC-Verbindungen zu den Datenbanken herstellen (wie, wird im Kapitel erklärt).

Probleme mit der Autorenedition

Probleme mit der Autorenedition könnte es bei der Kompilation von MFC-Programmen von der Kommandozeile aus geben. Die Autorenedition verfügt lediglich über eine dynamische Version der MFC-Bibliothek. Wenn Sie mit der Autorenedition MFC-Programme von der Kommandozeile aus kompilieren, müssen Sie statt */MT* (wie im Buch angegeben) die Optionen */MDd /D "_AFXDLL"* verwenden. Dies betrifft nur wenige Programme, beispielsweise die Archive-Programme in Kapitel 11 oder das Verbindungsprogramm in Kapitel 24.

Sollten Sie wider Erwarten bei Erstellung eines Programms mit der Autorenedition vom Linker einen Hinweis auf eine fehlende *afx...lib*-Datei erhalten, liegt dies ebenfalls daran, daß der Linker nach einer nicht vorhandenen

Version der MFC sucht. In der Kommandozeile sollten Sie dann wie oben beschrieben die Optionen /MDd /D "_AFXDLL" verwenden, in der IDE prüfen Sie die Projekteinstellungen, und wählen Sie auf der Seite ALLGEMEIN zuerst die Option für die Erstellung ohne MFC und dann die Option zur Erstellung mit der gemeinsam genutzten dynamischen Version.

Stichwortverzeichnis

A

- ActiveX 437
- AddDocTemplate (CWinApp) 387
- AddView (CDocument) 388
- AfxWinMain() 378
- Ansichtsfensterklasse 388
- Antworttabellen 192, 379
- Anwendung, beenden 230, 239
- Anwendungsassistent 99
- Anwendungserstellung 41
 - , Anwendungen mit Dialog als Hauptfenster 52
 - , API-Programmierung 364
 - , Debuggen 117
 - , GUI-Anwendungen 29
 - , Klassen-Assistent 103
 - , Kompilierung 83
 - , Konsolenanwendungen 20, 23
 - , MDI-Anwendungen 429
 - , MFC-Anwendungsassistent 99
 - , Projekte anlegen 45
 - , Quelltext aufsetzen 59
 - , Ressourcen 61
 - , SDI-Anwendungen 142
- Anwendungsgerüst 28, 139
 - , Ansichtsfenster 147
 - , Anwendungsobjekt 145
 - , Doc/View 142
 - , Dokumentklasse 148
 - , Dokumentvorlage 146, 217
 - , erweitern 157
 - , Hauptfenster 145
 - , MDI-Anwendungen 432
 - , Probleme mit Nicht-Doc/View-Programmen 402
 - , Serialisierung 342
 - , Startcode 149
 - , Ungarische Notation 154
 - , vordefinierte Nachrichtenbehandlung 230
- API 27
 - , API-Programm 364
 - , Fenster erzeugen 370
 - , Fensterfunktionen 369, 372
 - , Fensterklassen 367
 - , Funktionen aufrufen 405
 - , Handles 365, 406
 - , Hauptfenster 367
 - , hInstance 365
 - , Kommandozeilenargumente 365
 - , Message Loop 371
 - , Nachrichtenverarbeitung 370
 - , WinMain() 364
 - , WNDCLASS 367
 - Arbeitsbereiche 48
 - , konfigurieren 49
 - , leeren Arbeitsbereich anlegen 298
 - , mit einem Projekt 48
 - , mit mehreren Projekten 49
 - , öffnen 55
 - Arbeitsbereichfenster 53
 - , Dateien-Ansicht 55
 - , Klassen-Ansicht 54
 - , Ressourcen-Ansicht 54, 67
 - arithmetische Folgen 44
 - ASSERT-Makro 134
 - Automatisierung 441
 - , Automatisierungs-Client 445
 - , Automatisierungs-Server 441
 - Autoren-Edition 533

B

- Behandlungsmethoden
 - , Deklaration 192
 - , Klassen-Assistent 192
 - , Namen 380
 - , Parameter 196
 - , zu Menübefehlen 227
- BitBlt() (CDC) 352
- BITMAP 350
- Bitmaps 347
 - , Abmaße bestimmen 350
 - , als Fensterhintergrund 352
 - , aus Dateien laden 358
 - , aus Ressourcen laden 350
 - , in Fenster zeichnen 350
 - , in Speicherkontext laden 350
 - , manipulieren 354
 - , Quellen 348
 - , Ressourcen 75
 - , selbst erstellen 359
 - , Speicherkontexte 349
- Browser 90

C

CArchive 303
 CBitmap 330, 350
 CBrush 330, 332
 CButton 250
 CClientDC 314
 CComboBox 256
 CCommandLineInfo 178
 CCreateObject 394
 CD, zum Buch 533
 CDC 314
 CDocTemplate 387
 CDocument 387
 CEdit 248, 293
 CEditView 293, 308, 432
 CFile 294, 295
 CFont 330
 cl (Compiler) 18
 CListBox 256
 clock() 259
 CMDIChildWnd 432
 CMDIFrameWnd 432
 CMenu 235
 CMultiDocTemplate 433
 CObArray 326
 CObject 303
 COLORREF 335
 COM (Component Object Model) 437
 COMMAND 231
 Compiler 18, 83
 –, Browser-Informationen aufnehmen 90
 –, Debug-Informationen aufnehmen 120
 –, Konfiguration 83
 –, vorkompilierte Header 85
 Container-Klassen 326
 CPaintDC 314
 CPalette 330
 CPen 330
 CPoint 196
 Create() (CWnd) 167, 379
 CreateCompatibleDC() (CDC) 350
 CREATESTRUCT 170
 CreateWindow() 367
 CRgn 330
 CRichEdit 293
 CRichEditView 294
 CSingleDocTemplate 148
 CStatic 245, 292

CStatusBar 219
 CString 294
 CStudioFile 301
 CToolBar 218
 Cursor 77
 CWindowDC 314

D

DAO 475
 Dateien 294
 –, Fehlerbehandlung 297
 –, lesen 296, 302
 –, öffnen 295
 –, schließen 296
 –, schreiben 297, 299
 Datenbank-Programmierung 471
 Debugger 117
 –, Anzeigefenster 124
 –, Aufrufliste 125
 –, Disassemblierung 127
 –, Register 126
 –, Schnellüberwachung 128
 –, Speicher 125
 –, Überwachung 124
 –, Variablen 125
 –, Bearbeiten und Fortfahren 128
 –, Dateninfo 127
 –, Debug-Informationen 120
 –, Diagnosemakros
 –, ASSERT 134
 –, TRACE 134
 –, Dynamische Linkbibliotheken 427
 –, Haltepunkte 122
 –, Programm anhalten 121
 –, Programm laden 121
 –, Programm schrittweise ausführen 123
 –, Windows-Anwendungen 130
 Debug-Konfiguration 89
 DefWindowProc() 372
 DeleteContents() (CDocument) 320
 Dialogfelder 265
 –, als Hauptfenster 52
 –, aufrufen 280
 –, Datenaustausch 278
 –, Dialogklasse anlegen 272
 –, Eingaben auswerten 282
 –, initialisieren 279
 –, Klassen-Assistent 272
 –, Member-Variablen 275

- , Control-Typ 82
- , Wert-Typ 276
- , modale 283
- , nicht-modale 283
- , Ressourcen 68
- , Steuerelemente
- , ausrichten 267
- , einfügen 266
- , konfigurieren 268
- , löschen 267
- , markieren 267
- , Optionsfelder gruppieren 269
- , Schalter zum Verlassen 269
- , Tabulatorreihenfolge 268
- , verschieben 267
- , Zugriff auf Steuerelemente 275
- DispatchMessage() 371
- DllMain() 422
- DLLs Siehe Dynamische Linkbibliotheken
- Doc/View 142, 385
- , Ansichten identifizieren 394
- , Ansichten wechseln 394
- , Ansichtsfensterklasse 147, 388
- , Anwendungsgerüst 144
- , CDocTemplate 387
- , Dokumentklasse 148, 387
- , Dokumentvorlage 148, 386, 434
- , mehrere Ansichten 391
- , Rahmenfenster 389
- DoDataExchange() (CDialog) 278
- Dokumentklasse 387
- DoModal() (CDialog) 280
- DOSKEY 20
- DrawText() (CDC) 291
- Dynamische Linkbibliotheken 421
- E**
- Editor 59
- , Anweisungsvervollständigung 59
- , Debug-Unterstützung 127
- , Konfiguration 60
- , Quelltext laden 59
- Eingabefelder 248
- Eintrittsfunktion 364, 378
- Ellipse() (CDC) 316
- Enable() (CCommandUI) 234
- EnumFonts() 406
- F**
- Farben 335
- Fenster 166
- , Client-Bereich 145
- , Dialogfelder 52
- , erzeugen (API) 370
- , erzeugen (MFC) 379
- , Fensterfunktionen 369, 372
- , Fensterklassen 367
- , Fensterstile 169, 172
- , Hauptfenster 145, 165, 367
- , MFC
- , Create()-Methode 167
- , Fensterklassen 167
- , neu zeichnen lassen 319
- , Rahmenfenster 145
- , schließen 239
- , unter Windows 166
- , Fensterklasse 167
- , Handle 167
- Fensterfunktionen 369, 372
- Fensterklassen 367
- FillSolidRect() (CDC) 345
- Folgen
- , arithmetische 44
- , geometrische 45
- FTP 460
- G**
- geometrische Folgen 45
- Gerätekontexte 314, 382
- , Bitmaps laden 350
- , Klassen 314
- , OnDraw() 316
- , selbst erzeugen 315
- , Speicherkontexte 349
- , Zeichenmethoden 321
- , Zeichenwerkzeuge 330
- , einrichten 331
- , Klassen 330
- , löschen 331
- , vordefinierte 330
- GetActiveDocument (CFrameWnd) 390
- GetActiveView (CFrameWnd) 389
- GetClientRect() (CWnd) 232
- GetDC() 382
- GetDlgItem (CWnd) 394
- GetDocTemplate (CDocument) 388
- GetDocument (CView) 388
- GetFirstViewPosition (CDocument) 388, 395

GetNextView (CDocument) 388, 395
 GetParentFrame (CView) 389
 GetPixel() (CDC) 355
 GetSize() (CObArray) 326
 GetSubMenu() (CMenu) 236
 GlobalMemoryStatus() 409
 Gopher 462
 Grafik 311
 –, Bitmaps 347
 –, als Fensterhintergrund 352
 –, aus Dateien laden 358
 –, aus Ressourcen laden 350
 –, in Fenster zeichnen 350
 –, in Speicherkontext laden 350
 –, manipulieren 354
 –, Maße bestimmen 350
 –, selbst erstellen 359
 –, Farben 335
 –, Gerätekontexte 314
 –, Klassen 314
 –, OnDraw() 316
 –, selbst erzeugen 315
 –, RGB-Modell 335
 –, Speicherkontexte 349
 –, Zeichenmethoden 321
 –, Zeichenwerkzeuge 330
 –, einrichten 331
 –, Klassen 330
 –, löschen 331
 –, vordefinierte 330
 GUI-Anwendungen 26, 29
 Gültigkeitsbereichoperator 406

H

Haltepunkte 122
 Handles 365, 406
 Hauptfenster 165
 –, Anfangsposition festlegen 171
 –, API-Programmierung 367
 –, CMainFrame 145
 –, Fensterstile 169, 172
 –, Größe an Client anpassen 419
 –, Größe festlegen 171
 –, mit Statusleiste ausstatten 169, 174
 –, mit Symbolleiste ausstatten 169, 174
 –, mit unveränderlicher Größe 261, 526
 –, schließen 230

hInstance 365
 HTTP 462

I

IDE 23 Siehe auch Integrierte Entwicklungsumgebung
 Importbibliotheken 423
 InitInstance() (CWinApp) 146
 Integrierte Entwicklungsumgebung 23
 –, Arbeitsbereiche 48
 –, Arbeitsbereichfenster 53
 –, Compiler 83
 –, Debugger 117
 –, Editor 59
 –, Klassen-Assistent 103
 –, Linker 83
 –, MFC-Anwendungsassistent 99
 –, Projektverwaltung 45
 –, Ressourcen-Editoren 61
 Internet-Programmierung 459
 Invalidate() (CWnd) 319

K

KillTimer() (CWnd) 203
 Klassen-Assistent 103
 –, ActiveX-Ereignisse 112
 –, auf aktuellen Stand bringen 114
 –, Automatisierung 110
 –, Behandlungsmethoden einrichten 106
 –, CLW-Datei 113
 –, in Methodendefinition springen 108
 –, Member-Variablen (für Dialogklassen) 109
 –, Control-Typ 82
 –, Wert-Typ 276
 –, Methoden löschen 108
 –, neue Klassen anlegen 105
 –, virtuelle Methoden überschreiben 108
 Kombinationsfelder 256
 Kommandozeilenargumente 178, 365
 Kommandozeilentools
 –, Compiler 18
 –, DOSKEY 20
 –, Linker 18, 34
 –, NMAKE 35
 –, Ressourcen-Compiler 34

- , vcvars32.bat 19
- Kompilierung 83
- , Browser-Informationen aufnehmen 90
- , Debug-Informationen aufnehmen 120
- komplexe Zahlen 336
- Konfiguration
 - , Arbeitsbereiche 49
 - , Compiler 83
 - , Editor 60
- Konsolenanwendungen 20, 21, 23
- Kontextmenüs 235
- Kontrastdetektor 355
- Kontrollkästchen 252

L

- Laufzeitinformationen 156
- LineTo() (CDC) 345
- Linker 18, 34, 83
- , spezielle Bibliotheken verwenden 417
- Listenfelder 256
- LoadBitmap() (CBitmap) 350
- LoadFrame() (CFrameWnd) 217
- Lokalisierung 63
- Lösungen zu den Fragen 523

M

- m_nCmdShow 379
- Make-Dateien 35
- Makros
 - , Diagnosemakros 134
 - , für Laufzeitinformationen 156
 - , für Nachrichtentabellen 156
 - , für Serialisierung 304
- Maus
 - , Nachrichtenverarbeitung 194
 - , Doppelklick 194
 - , Klick im Client-Bereich 195
 - , Klick im Nicht-Clientbereich 194
 - , Mauskoordinaten 196
 - , mittlere Maustaste 194
- Mauszeiger 77
- MCIWnd 417
- MDI-Anwendungen 429
- Meldungsfenster 196, 287
- Menüs 214
 - , ALT-Tastenkombination 222

- , Behandlungsmethoden zu Menübefehlen 227
- , Hilfetext 222
- , Kontextmenüs 235
- , Menübefehle inaktivieren 233
- , mit Rahmenfenster verbinden 217
- , Quickinfos 222
- , Ressourcen 72, 220
- , Schalter 222
- , Tastaturkürzel 222
- , Trennlinien 222
- , Untermenüs 222
- Message Loop 189, 371
- MessageBeep() 416
- MessageBox() (CWnd) 288
- Methoden
 - , AddDocTemplate (CWinApp) 387
 - , AddView (CDocument) 388
 - , AfxMessageBox() 289
 - , automatisieren 443
 - , BitBlt() (CDC) 352
 - , Create() (CWnd) 167, 379
 - , CreateCompatibleDC() (CDC) 350
 - , DeleteContents() (CDocument) 320
 - , DoDataExchange() (CDialog) 278
 - , DoModal() (CDialog) 280
 - , DrawText() (CDC) 291
 - , Ellipse() (CDC) 316
 - , Enable() (CCommandUI) 234
 - , FillSolidRect() (CDC) 345
 - , GetActiveDocument (CFrameWnd) 390
 - , GetActiveView (CFrameWnd) 389
 - , GetClientRect() (CWnd) 232
 - , GetDlgItem (CWnd) 394
 - , GetDocTemplate (CDocument) 388
 - , GetDocument (CView) 388
 - , GetFirstViewPosition (CDocument) 388, 395
 - , GetNextView (CDocument) 388, 395
 - , GetParentFrame (CView) 389
 - , GetPixel() (CDC) 355
 - , GetSize() (COleArray) 326
 - , GetSubMenu() (CMenu) 236
 - , InitInstance() (CWinApp) 146

- , Invalidate() (CWnd) 319
- , KillTimer() (CWnd) 203
- , LineTo() (CDC) 345
- , LoadBitmap() (CBitmap) 350
- , LoadFrame() (CFrameWnd) 217
- , MessageBox() (CWnd) 288
- , MoveTo() (CDC) 345
- , OnAppExit() (CWinApp) 230
- , OnCreate() (CFrameWnd) 218, 219, 245
- , OnCreateClient (CFrameWnd) 394
- , OnDraw() (CView) 200, 202, 316
- , OnPaint() (CWnd) 200
- , PreCreateWindow() (CWnd) 170
- , ReadString (CStudioFile) 301
- , RecalcLayout (CFrameWnd) 395
- , Run() (CWinApp) 379
- , SelectObject() (CDC) 331
- , SendMessage() (CWnd) 239
- , Serialize() (CDocument) 343
- , SetActiveView (CDocument) 395
- , SetIndicators() (CStatusBar) 219
- , SetModifiedFlag (CDocument) 388
- , SetPixel() (CDC) 355
- , SetTimer() (CWnd) 203
- , SetWindowPos() (CFrameWnd) 419
- , ShowWindow() (CWnd) 147
- , StretchBlt() (CDC) 353
- , TextOut() (CDC) 290
- , TrackPopupMenu() (CMenu) 236
- , UpdateAllView (CDocument) 388
- , UpdateAllViews() (CDocument) 341
- , UpdateWindow() (CWnd) 319
- , WriteString (CStudioFile) 301
- MFC 27
 - , Antworttabellen 192, 379
 - , API-Funktionen aufrufen 405
 - , Container-Klassen 326
 - , Eintrittsfunktion 378
 - , Fenster erzeugen 379
 - , Gerätekontexte 382
 - , Laufzeitinformationen 156
 - , Makros 156
 - , Nachrichtenverarbeitung 191
 - , Run() 379
 - , Ungarische Notation 154
- MFC-Anwendungs-Assistent 99
- MoveTo() (CDC) 345
- MS Access 476
- Multimedia 415
 - , Sound 416
 - , Video 417
- Multithreading 449
- N**
 - Nachrichten 189
 - , chronologische Verarbeitung 376
 - , COMMAND 231
 - , PostMessage() 377
 - , selbst senden 376
 - , SendMessage() 377
 - , UPDATE_COMMAND_UI 233
 - , Windows-Nachrichten 497
 - , WM_CLOSE 239
 - , WM_COMMAND 227
 - , WM_CONTEXTMENU 235
 - , WM_CREATE 244
 - , WM_DESTROY 372
 - , WM_KEYDOWN 198
 - , WM_LBUTTONDOWN 195
 - , WM_PAINT 199
 - , WM_QUIT 373
 - , WM_RBUTTONDOWN 205
 - , WM_SIZE 384
 - , WM_TIMER 203
 - Nachrichtenverarbeitung 370
 - , Antworttabellen 192, 379
 - , Anwendungsgertüst 229
 - , Behandlungsmethoden 192, 227
 - , Namen 380
 - , chronologische Verarbeitung 376
 - , in der MFC 191
 - , Klassen-Assistent 192, 228
 - , Mausereignisse 194
 - , Doppelklick 194
 - , Klick im Client-Bereich 195
 - , Klick im Nicht-Clientbereich 194
 - , Mauskoordinaten 196
 - , mittlere Maustaste 194
 - , Message Loop 189, 371
 - , Nachrichten selbst versenden 376
 - , ohne Message Loop 376
 - , Parameter der
 - , Behandlungsmethoden 196
 - , PostMessage() 377
 - , SendMessage() 377

- , Tastaturereignisse 198
- , Tastaturkürzel 223
- , Taste gedrückt 198
- , unter Windows 189
- , WM_PAINT 199
- , Zeitgeber 203
- NMAKE 35

O

- Objektorientierte Programmierung 507
- ODBC 474
- OLE (Object Linking and Embedding) 439
- OLE DB 476
- OnAppExit() (CWinApp) 230
- OnCreate() (CFrameWnd) 218, 219, 245
- OnCreate() (CWnd) 174
- OnCreateClient (CFrameWnd) 394
- OnDraw() (CView) 200, 202, 316
- OnPaint() (CWnd) 200
- OOP Siehe Objektorientierte Programmierung
- Optionsfelder 253
- , gruppieren 269

P

- Paßwortabfragen 293
- Paßwörter 528
- PlaySound() 416
- PostMessage() 377
- PostQuitMessage() 373
- PreCreateWindow (CWnd) 170
- Programmierbeispiele
 - , Einarmiger Bandit 259
 - , Fraktale 335, 452
 - , Liniengrafik 355
 - , Mäuse-Editor 324
 - , Reaktionstestprogramm 203
 - , Texteditor 307
 - , Webbrowser 464
 - , Zinsberechnung 42
- Projekte 45
 - , Debug-Informationen 120
 - , exportieren 36
 - , GUI-Anwendungen 29
 - , konfigurieren 83
 - , Konsolenanwendungen 23
 - , MDI-Anwendungen 429
 - , neu anlegen 49

- , öffnen 55
- , Projekttypen 51, 52
- , Win32-API-Projekt 366
- Projektverwaltung 36, 45
 - , Arbeitsbereiche 48
 - , Debug-Konfiguration 84, 89
 - , Dialog Neu 49
 - , DSP-Datei 48
 - , Konfiguration 83
 - , Make-Datei erstellen 36
 - , Projekte neu anlegen 49
 - , Projekte öffnen 55
 - , Projektkonfigurationen 84, 89
 - , Projekttypen 51, 52
 - , Projektverzeichnis 49
 - , Quelltextdateien hinzufügen 56
 - , Release-Konfiguration 84, 89
 - , spezielle Bibliotheken verwenden 417
 - , überflüssige Dateien löschen 88
 - , Übersicht 46
 - , Vorteile 48
 - , Zwischendateien 88

Q

- Quickinfos 222, 226

R

- Rahmenfenster 145, 389
 - , in MDI-Anwendungen 432
- rand() 259
- ReadString (CStudioFile) 301
- RecalcLayout (CFrameWnd) 395
- RegisterClass() 370
- ReleaseDC() 382
- Release-Konfiguration 89
- Ressourcen 61
 - , anlegen 67
 - , Bitmaps 75
 - , Dialogfelder 68
 - , laden 66
 - , löschen 68
 - , Mauszeiger 77
 - , Menüs 72, 220
 - , RES-Datei 65
 - , Ressourcen-Ansicht 67
 - , Ressourcen-Compiler 65
 - , Ressourcen-IDs 65
 - , Ressourcenskriptdatei 64
 - , Stringtabellen 80, 157
 - , Symbole 77

- , Symbolleisten 78
- , Tastaturkürzel 73
- , Versionsinformation 80
- , Vorteile 62
- Ressourcen-Compiler 34
- RGB-Modell 335
- Run() (CWinApp) 379

S

- Schaltflächen 250
- SelectObject() (CDC) 331
- SendMessage() 377
- SendMessage() (CWnd) 239
- Serialisierung 303, 342
- Serialize() (CDocument) 343
- SetActiveView (CDocument) 395
- SetIndicators() (CStatusBar) 219
- SetModifiedFlag (CDocument) 388
- SetPixel() (CDC) 355
- SetTimer() (CWnd) 203
- SetWindowPos() (CFrameWnd) 419
- ShowWindow() (CWnd) 147
- Sound 416
- Speicherkontexte 349
- SQL 474
- Statusleisten 215
- , CStatusBar 219
- , erzeugen 219
- , Hinfetext für Menübefehle 222
- , in Hauptfenster aufnehmen 169, 174, 178
- StdAfx.cpp (vorkompilierte Header) 86
- stdin 21
- stdout 21
- Steuerelemente 243
- , Eingabefelder 248
- , im MFC-Anwendungsgerüst 260
- , in Fenster integrieren 244
- , Kombinationsfelder 256
- , Kontrollkästchen 252
- , Listenfelder 256
- , Optionsfelder 253
- , Schaltflächen 250
- , Textfelder 245
- StretchBlt() (CDC) 353
- Strings 294
- Stringtabellen
- , anpassen 226
- , Ressourcen 80
- , verwenden 157

Symbole

- , mit Anwendung verbinden 175
- , Ressourcen 77
- Symbolleisten 215
- , anpassen 224
- , CToolBar 218
- , erzeugen 218
- , Hilfetexte 226
- , in Hauptfenster aufnehmen 169, 174
- , Quickinfos 226
- , Ressourcen 78
- Systeminformationen 408

T

- Tabulatorreihenfolge 268
- Tastatur
- , Nachrichtenverarbeitung 198
- , Tastaturkürzel 223
- , Taste gedrückt 198
- Tastaturkürzel
- , einrichten 223
- , in Menü anzeigen 222
- , Ressourcen 73
- Text 287
- , Ansichtsklassen 293
- , Eingabefelder 293
- , Formatierung 293
- , MDI-Editor 431
- , Meldungsfenster 287
- , statische Textfelder 292
- , Texteditor 307
- , zeichnen 290
- Textfelder 245
- TextOut() (CDC) 290
- Threads 449
- TRACE-Makro 134
- TrackPopupMenu() (CMenu) 236

U

- Ungarische Notation 154
- UPDATE_COMMAND_UI 233
- UpdateAllView (CDocument) 388
- UpdateAllViews() (CDocument) 341
- UpdateWindow() (CWnd) 319
- URL 462

V

- vcvars32.bat 19
- Versionsinformation 80
- Video 417

- Visual C++
 - , Assistenten 99
 - , beiliegende Compiler-Version 533
 - , Compiler 18, 83
 - , Debugger 117
 - , Editor 59
 - , Klassen-Assistent 103
 - , Linker 34, 83
 - , Projektverwaltung 36, 45
 - , Ressourcen 61
 - , Ressourcen-Compiler 34
 - Vorgabeargumente 407
 - Vorkompilierte Header 85
- W**
- Webbrowser 464
- Windows
 - , API-Programmierung 364
 - , Fenster 166
 - , Fensterfunktionen 369, 372
 - , Fensterklassen 367
 - , Handles 365
 - , Hauptfenster 367
 - , Kommandozeilenargumente 365
 - , Message Loop 189, 371
 - , Multithreading 449
 - , Nachrichtenverarbeitung 189, 370
 - , WinMain() 149, 364
 - , WM-Nachrichten 497
 - WinMain() 149, 364
 - WM_CLOSE 239
 - WM_COMMAND 227
 - WM_CONTEXTMENU 235
 - WM_CREATE 244
 - WM_DESTROY 372
 - WM_KEYDOWN 198
 - WM_LBUTTONDOWN 195
 - WM_PAINT 199
 - WM_QUIT 373
 - WM_RBUTTONDOWN 205
 - WM_SIZE 384
 - WM_TIMER 203
 - WNDCLASS 367
 - WriteString (CStudioFile) 301
- Z**
- Zeichenmethoden 321
- Zeichenwerkzeuge 330
 - , einrichten 331
 - , Klassen 330
 - , löschen 331
 - , vordefinierte 330
- Zeichnen 311 Siehe auch Grafik
- Zeitgeber 203
- Zinsberechnung 42
- Zufallszahlen 259
- Zwischendateien 88