



Visual C++ 6 in 21 Tagen

<http://kickme.to/tiger/>

Inhaltsverzeichnis

[Einführung](#)

[Woche 1 im Überblick](#)

[Tag 1 Erste Schritte mit Visual C++](#)

[Tag 2 Steuerelemente](#)

[Tag 3 Maus und Tastatur](#)

[Tag 4 Timer](#)

[Tag 5 Dialogfelder](#)

[Tag 6 Menüs](#)

[Tag 7 Text und Schriften](#)

[Woche 1 im Rückblick](#)

[Woche 2 im Überblick](#)

[Tag 8 Bilder, Zeichnungen und Bitmaps](#)

[Tag 9 ActiveX-Steuerelemente](#)

[Tag 10 SDI-Anwendungen](#)

[Tag 11 MDI-Anwendungen](#)

[Tag 12 Symbolleisten und Statusleisten](#)

[Tag 13 Dateizugriff](#)

[Tag 14 Daten aus ODBC-Datenbanken abrufen](#)

[Woche 2 im Rückblick](#)

[Woche 3 im Überblick](#)

[Tag 15 Datenbanken per ADO bearbeiten](#)

[Tag 16 Eigene Klassen und Module](#)

[Tag 17 DLLs](#)

[Tag 18 Multitasking](#)

[Tag 19 Eigene ActiveX-Steuerelemente](#)

[Tag 20 Internet und Netzwerke](#)

[Tag 21 Funktionen für Webbrowser](#)

[Woche 3 im Rückblick](#)

[Anhang A C++ im Überblick](#)

[Anhang B Antworten](#)

[Anhang C Drucken und Druckvorschau](#)

[Anhang D Ausnahmebehandlung](#)

[Anhang E Debugger und Profiler](#)

[Anhang F MFC-Hilfsklassen](#)

[Anhang G Die CD zum Buch](#)

[Stichwortverzeichnis](#)

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Stichwortverzeichnis

- (Subtraktion)

- [C++ im Überblick](#)

Symbols

! (logisches NOT)

- [C++ im Überblick](#)

!= (Nicht gleich)

- [C++ im Überblick](#)

#import-Direktive

- [Datenbanken per ADO bearbeiten](#)

#include-Anweisungen

- [Dialogfelder](#)

% (Modulo-Division)

- [C++ im Überblick](#)

& (Adreßoperator)

- [C++ im Überblick](#)

& (binäres AND)

- [Maus und Tastatur](#)

& (Referenzoperator)

- [C++ im Überblick](#)

& (Zugriffstaste)

- [Steuerelemente](#)

&& (logisches AND)

- [Maus und Tastatur](#)
- [C++ im Überblick](#)

(Indexoperator)

- [MFC-Hilfsklassen](#)

* (Indirektionsoperator)

- [C++ im Überblick](#)

* (Multiplikation)

- [C++ im Überblick](#)

+ (Addition)

- [C++ im Überblick](#)

. (Strukturelementoperator)

- [C++ im Überblick](#)

. (Zugriff über Objekt selbst)

- [SDI-Anwendungen](#)

/ (Division)

- [C++ im Überblick](#)

// (Kommentare)

- [C++ im Überblick](#)

: (Basisklasse)

- [C++ im Überblick](#)

:: (Zugriffsoperator)

- [C++ im Überblick](#)

< (Kleiner als)

- [C++ im Überblick](#)

<< (Ausgabeoperator)

- [C++ im Überblick](#)

<= (Kleiner als oder gleich)

- [C++ im Überblick](#)

= (Zuweisung)

- [Steuerelemente](#)
- [C++ im Überblick](#)

== (Vergleich)

- [Steuerelemente](#)
- [C++ im Überblick](#)

> (Größer als)

- [C++ im Überblick](#)

-> (Zugriff über Zeiger)

- [SDI-Anwendungen](#)

>= (Größer als oder gleich)

- [C++ im Überblick](#)

>> (Eingabeoperator)

- [C++ im Überblick](#)

__uuidof

- [Datenbanken per ADO bearbeiten](#)

_ConnectionPtr

- [Datenbanken per ADO bearbeiten](#)

| (binäres OR)

- [Maus und Tastatur](#)

|| (logisches OR)

- [Maus und Tastatur](#)
- [C++ im Überblick](#)

~ (Destruktor)

- [C++ im Überblick](#)

Numerics

24-Stunden-Format

- [MFC-Hilfsklassen](#)

A

Abbildungsmodi

- [Bilder, Zeichnungen und Bitmaps](#)
- [Multitasking](#)

Drucken

- [Drucken und Druckvorschau](#)

MM_XXXX-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

abbrechen

Drucken

- [Drucken und Druckvorschau](#)

Ablaufverfolgung

- [Debugger und Profiler](#)

Ableitung

- [C++ im Überblick](#)

accelerator keys (Schnelltasten)

- [Menüs](#)

Accelerator-Tabelle

- [Funktionen für Webbrowser](#)

Accept

- [Internet und Netzwerke](#)

Access 97

Datenbank-Assistent

- [Daten aus ODBC- Datenbanken abrufen](#)

ActiveX Data Objects Siehe ADO

- [Datenbanken per ADO bearbeiten](#)

ActiveX-Container

- [ActiveX-Steuerelemente](#)

ActiveX-Server

- [ActiveX-Steuerelemente](#)

ActiveX-Steuerelemente

- [ActiveX-Steuerelemente](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

ältere

- [ActiveX-Steuerelemente](#)

ambient

- [Eigene ActiveX- Steuerelemente](#)

aus anderen Anwendungen

- [ActiveX-Steuerelemente](#)

Automatisierung

- [ActiveX-Steuerelemente](#)

benutzerdefinierte Ereignisse

- [Eigene ActiveX- Steuerelemente](#)

benutzerdefinierte Methoden

- [Eigene ActiveX- Steuerelemente](#)

COM-Technologie

- [ActiveX-Steuerelemente](#)

eigene

- [Eigene ActiveX- Steuerelemente](#)

Eigenschaften

- [ActiveX-Steuerelemente](#)

Eigenschaftsseiten

- [Eigene ActiveX- Steuerelemente](#)

Ereignisse

- [ActiveX-Steuerelemente](#)

erstellen

- [Eigene ActiveX- Steuerelemente](#)

FlexGrid

- [ActiveX-Steuerelemente](#)

IDispatch

- [ActiveX-Steuerelemente](#)

installieren

- [ActiveX-Steuerelemente](#)

Internet Explorer

- [Funktionen für Webbrowser](#)

Klassen

- [ActiveX-Steuerelemente](#)

Laufzeitlizenzen

- [Eigene ActiveX- Steuerelemente](#)

Lizenzen

- [ActiveX-Steuerelemente](#)

Marshaling

- [Eigene ActiveX- Steuerelemente](#)

Methoden

- [ActiveX-Steuerelemente](#)

Methoden ermitteln

- [ActiveX-Steuerelemente](#)

OLE 2.0

- [ActiveX-Steuerelemente](#)

registrieren

- [ActiveX-Steuerelemente](#)

Steuerelementpalette

- [ActiveX-Steuerelemente](#)

Testcontainer

- [Eigene ActiveX- Steuerelemente](#)

Variablen zuordnen

- [ActiveX-Steuerelemente](#)

ActiveX-Steuerelemente Siehe auch Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Add

- [MFC-Hilfsklassen](#)

AddHead

- [MFC-Hilfsklassen](#)

Addition

- [C++ im Überblick](#)

AddLine (CTag10Doc)

- [SDI-Anwendungen](#)
- [SDI-Anwendungen](#)
- [Antworten](#)

AddNew

- [Daten aus ODBC- Datenbanken abrufen](#)

AddNew (CDbAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

AddNewRecord (CSerialDoc)

- [Dateizugriff](#)

AddTail

- [MFC-Hilfsklassen](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

Befehle ausführen

- [Datenbanken per ADO bearbeiten](#)

Command

- [Datenbanken per ADO bearbeiten](#)

Connection

- [Datenbanken per ADO bearbeiten](#)

Cursortypen

- [Datenbanken per ADO bearbeiten](#)

Daten abrufen

- [Datenbanken per ADO bearbeiten](#)

Datensatzfelder

- [Datenbanken per ADO bearbeiten](#)

DLL importieren

- [Datenbanken per ADO bearbeiten](#)

Error

- [Datenbanken per ADO bearbeiten](#)

Field

- [Datenbanken per ADO bearbeiten](#)

Makros

- [Datenbanken per ADO bearbeiten](#)

Parameter

- [Datenbanken per ADO bearbeiten](#)

Recordset

- [Datenbanken per ADO bearbeiten](#)

SQL-Befehle

- [Datenbanken per ADO bearbeiten](#)

Transaktionen

- [Datenbanken per ADO bearbeiten](#)

ADO_FIXED_LENGTH_ENTRY

- [Datenbanken per ADO bearbeiten](#)

ADO_NUMERIC_ENTRY

- [Datenbanken per ADO bearbeiten](#)

ADO_VARIABLE_LENGTH_ENTRY

- [Datenbanken per ADO bearbeiten](#)

Adressen

- [Internet und Netzwerke](#)

aus Eingabefeld übernehmen

- [Funktionen für Webbrowser](#)

loopback

- [Internet und Netzwerke](#)

Adreßoperator

- [C++ im Überblick](#)

AFX_DAO_ERROR_DFX_BIND

- [Ausnahmebehandlung](#)

AFX_DAO_ERROR_ENGINE_INITIALIZATION

- [Ausnahmebehandlung](#)

AFX_DAO_ERROR_OBJECT_NOT_OPEN

- [Ausnahmebehandlung](#)

AFX_EXT_CLASS

- [DLLs](#)

AFX_MANAGE_STATE

- [DLLs](#)

AFX_MODULE_STATE

- [DLLs](#)

AfxBeginThread

- [Multitasking](#)

AfxThowMemoryException

- [Ausnahmebehandlung](#)

AfxThrowArchiveException

- [Ausnahmebehandlung](#)

AfxThrowFileException

- [Ausnahmebehandlung](#)

AfxThrowNotSupportedException

- [Ausnahmebehandlung](#)

AfxThrowOleDispatchException

- [Ausnahmebehandlung](#)

AfxThrowOleException

- [Ausnahmebehandlung](#)

AfxThrowResourceException

- [Ausnahmebehandlung](#)

AfxThrowUserException

- [Ausnahmebehandlung](#)

aktivieren

Steuerelemente

- [Steuerelemente](#)

aktualisieren

Steuerelemente

- [Steuerelemente](#)

AM/PM

- [MFC-Hilfsklassen](#)

ambient

- [Eigene ActiveX- Steuerelemente](#)

AND

- [Maus und Tastatur](#)
- [C++ im Überblick](#)

andocken

Symbolleisten

- [Symbolleisten und Statusleisten](#)

anhalten

OnIdle-Tasks

- [Multitasking](#)

Threads

- [Multitasking](#)

Timer

- [Timer](#)

Anschlüsse

- [Internet und Netzwerke](#)

ANSI_CHARSET

- [Text und Schriften](#)

Ansichten

Dateien

- [Erste Schritte mit Visual C++](#)

erste ermitteln

- [Symbolleisten und Statusleisten](#)

Klassen

- [Erste Schritte mit Visual C++](#)

Ressourcen

- [Erste Schritte mit Visual C++](#)

Zeiger auf

- [Symbolleisten und Statusleisten](#)

Ansichtsklassen

Navigation

- [Dateizugriff](#)

Ansichtsobjekte

selbst neu zeichnen

- [SDI-Anwendungen](#)

Anweisungen

- [C++ im Überblick](#)

Assertion

- [Debugger und Profiler](#)

if

- [C++ im Überblick](#)

Include

- [C++ im Überblick](#)

Prototypen

- [C++ im Überblick](#)

return

- [C++ im Überblick](#)

Trace

- [Debugger und Profiler](#)

Anwendungen

16-Bit-

- [Multitasking](#)

ActiveX-Server

- [ActiveX-Steuerelemente](#)

Bibliotheksmodule testen

- [Eigene Klassen und Module](#)

Code erstellen

- [Erste Schritte mit Visual C++](#)

Container

- [Eigene ActiveX- Steuerelemente](#)

Datenbank mit ODBC

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenbanken

- [Dateizugriff](#)
- [Datenbanken per ADO bearbeiten](#)

DbAdo

- [Datenbanken per ADO bearbeiten](#)

dialogfeldbasierende

- [Erste Schritte mit Visual C++](#)

disassemblieren

- [Eigene Klassen und Module](#)

DLLs testen

- [DLLs](#)

Drucken

- [Drucken und Druckvorschau](#)

Fächer

- [Multitasking](#)

Gerüst

- [Erste Schritte mit Visual C++](#)

herunterfahren

- [Datenbanken per ADO bearbeiten](#)
- [Multitasking](#)

in andere einbetten

- [ActiveX-Steuerelemente](#)

kompilieren

- [Erste Schritte mit Visual C++](#)

Konsolen-

- [C++ im Überblick](#)

loopback

- [Internet und Netzwerke](#)

MDI

- [MDI-Anwendungen](#)

MDI-Zeichenprogramm

- [MDI-Anwendungen](#)

MessageBox

- [Dialogfelder](#)

Multitasking

- [Multitasking](#)

Netzwerk

- [Internet und Netzwerke](#)

Projekt-Arbeitsbereich

- [Erste Schritte mit Visual C++](#)

schließen

- [Steuerelemente](#)

Schriften

- [Text und Schriften](#)

SDI

- [SDI-Anwendungen](#)

serialisierte

- [Dateizugriff](#)

Singlethreading

- [Multitasking](#)

starten

- [Steuerelemente](#)

Steuerelemente testen

- [Eigene ActiveX- Steuerelemente](#)

Symbol

- [Erste Schritte mit Visual C++](#)

Uhr

- [Timer](#)

verbinden

- [Internet und Netzwerke](#)

Webbrowser

- [Funktionen für Webbrowser](#)

Zeichnen

- [Eigene Klassen und Module](#)

Anwendungs-Assistent

- [Erste Schritte mit Visual C++](#)

CArchive

- [Dateizugriff](#)

Drucken und Druckvorschau

- [Drucken und Druckvorschau](#)

Internet Explorer-Infoleisten

- [Funktionen für Webbrowser](#)

Kommentare für Quellcode

- [Erste Schritte mit Visual C++](#)

ODBC-Datenbankunterstützung

- [Daten aus ODBC- Datenbanken abrufen](#)

Serialisierung bei dialogfeldbasierenden Anwendungen

- [Dateizugriff](#)

Windows Sockets

- [Internet und Netzwerke](#)

Anwendungsgerüst

- [Erste Schritte mit Visual C++](#)

Drucken

- [Drucken und Druckvorschau](#)

Drucken ohne

- [Drucken und Druckvorschau](#)

Anwendungsrückruf-Routinen Siehe Callback-Funktionen

- [Text und Schriften](#)

anzeigen

Steuerelemente

- [Steuerelemente](#)

API-Funktionen

- [Steuerelemente](#)

Arbeitsbereich

- [Erste Schritte mit Visual C++](#)

anlegen

- [Erste Schritte mit Visual C++](#)
- [C++ im Überblick](#)

Archive

Ausnahmen

- [Ausnahmebehandlung](#)

Arrayklassen

- [SDI-Anwendungen](#)

- [MFC-Hilfsklassen](#)

Add

- [MFC-Hilfsklassen](#)

benutzerdefinierte

- [MFC-Hilfsklassen](#)

CByteArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CDWordArray

- [MFC-Hilfsklassen](#)

CObArray

- [MFC-Hilfsklassen](#)

CPtrArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CStringArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CUIIntArray

- [MFC-Hilfsklassen](#)

CWordArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

Elemente

- [MFC-Hilfsklassen](#)

GetAt

- [MFC-Hilfsklassen](#)

SetAt

- [MFC-Hilfsklassen](#)

Arrays

größter Index

- [MFC-Hilfsklassen](#)

Arrays Siehe auch Felder

- [MFC-Hilfsklassen](#)

ASSERT

- [Debugger und Profiler](#)
- [Debugger und Profiler](#)

ASSERT_KINDOF

- [Debugger und Profiler](#)

ASSERT_VALID

- [Debugger und Profiler](#)

Assertion

SetAt

- [MFC-Hilfsklassen](#)

Assistenten

ADO

- [Datenbanken per ADO bearbeiten](#)

Anwendungs-

- [Erste Schritte mit Visual C++](#)

Klassen-

- [Erste Schritte mit Visual C++](#)

MFC ActiveX-Steuerelement-Assistent

- [Eigene ActiveX- Steuerelemente](#)

MFC-Anwendungs-Assistent (dll)

- [DLLs](#)

Assistentenleiste

- [Erste Schritte mit Visual C++](#)

Attach

- [Bilder, Zeichnungen und Bitmaps](#)

Attribute

Flags

- [Maus und Tastatur](#)

Auflistungsklassen

- [MFC-Hilfsklassen](#)

benutzerdefinierte

- [MFC-Hilfsklassen](#)

Typsicherheit

- [MFC-Hilfsklassen](#)

Vorlagen

- [MFC-Hilfsklassen](#)

Auflösung

Datum/Zeit

- [MFC-Hilfsklassen](#)

Aufrufliste

- [Debugger und Profiler](#)

Aufzählungen

- [C++ im Überblick](#)

Aufzählungsfunktionen

- [Text und Schriften](#)

ausblenden

Steuerelemente

- [Steuerelemente](#)

ausführen

Anwendungen

- [Steuerelemente](#)

Ausgabebereich

- [Erste Schritte mit Visual C++](#)

Ausgabefenster

- [C++ im Überblick](#)

Ausgabeoperator

- [C++ im Überblick](#)

Ausgabequalität

- [Text und Schriften](#)

auslösen

Ausnahmen

- [Ausnahmebehandlung](#)

Ausnahmebehandlung

catch-Blöcke

- [SDI-Anwendungen](#)

Serialisierung

- [Dateizugriff](#)

try-Blöcke

- [SDI-Anwendungen](#)

Ausnahmen

- [Ausnahmebehandlung](#)

auslösen

- [Ausnahmebehandlung](#)

Benutzer-

- [Ausnahmebehandlung](#)

benutzerdefinierte

- [Ausnahmebehandlung](#)

Betriebssystem

- [Ausnahmebehandlung](#)

catch

- [Ausnahmebehandlung](#)

CDAOException

- [Ausnahmebehandlung](#)

CFileException

- [Ausnahmebehandlung](#)

CMemoryException

- [Ausnahmebehandlung](#)

COleDispatchException

- [Ausnahmebehandlung](#)

CResourceException

- [Ausnahmebehandlung](#)

Datenbanken

- [Ausnahmebehandlung](#)

Datensatzfeldaustausch (DFX)

- [Ausnahmebehandlung](#)

DECLARE_DYNAMIC

- [Ausnahmebehandlung](#)

Fehlercodes

- [Ausnahmebehandlung](#)

IMPLEMENT_DYNAMIC

- [Ausnahmebehandlung](#)

löschen

- [Ausnahmebehandlung](#)

new

- [Ausnahmebehandlung](#)

nicht unterstützte

- [Ausnahmebehandlung](#)

OLE

- [Ausnahmebehandlung](#)

OLE-SCODE-Wert

- [Ausnahmebehandlung](#)

Ressourcen freigeben

- [Ausnahmebehandlung](#)

Speicher

- [Ausnahmebehandlung](#)

ThrowOsError

- [Ausnahmebehandlung](#)

try

- [Ausnahmebehandlung](#)

Ursachencodes

- [Ausnahmebehandlung](#)

verschachteln

- [Ausnahmebehandlung](#)

Zugriffsverletzungen

- [Ausnahmebehandlung](#)

Ausrufezeichen-Symbol

- [Dialogfelder](#)

Ausschnitte

in Statusleisten

- [Symbolleisten und Statusleisten](#)

Automatisierung

- [ActiveX-Steuerelemente](#)

Eigenschaften

- [Eigene ActiveX- Steuerelemente](#)

IDispatch

- [ActiveX-Steuerelemente](#)

AutoWert-Feld

- [Daten aus ODBC- Datenbanken abrufen](#)

B

Basisklassen

- [C++ im Überblick](#)

Datenbankanwendungen

- [Dateizugriff](#)

Quellcode-Browser

- [Debugger und Profiler](#)

Bearbeiten und Fortsetzen

- [Debugger und Profiler](#)

beenden

Programme

- [Steuerelemente](#)

BEGIN_ADO_BINDING

- [Datenbanken per ADO bearbeiten](#)

BeginWaitCursor

- [Maus und Tastatur](#)

Begriffe

Destruktor

- [C++ im Überblick](#)

Dokument

- [SDI-Anwendungen](#)

Funktion

- [C++ im Überblick](#)

Konstruktor

- [C++ im Überblick](#)

Methode

- [C++ im Überblick](#)

Zeiger

- [SDI-Anwendungen](#)

Behandlungsroutinen

mehrfach verwenden

- [Menüs](#)

Nachrichten

- [Erste Schritte mit Visual C++](#)

Benutzerausnahmen

- [Ausnahmebehandlung](#)

Benutzereingaben

mit Mausereignissen abfangen

- [SDI-Anwendungen](#)

Benutzeroberflächen

grafische

- [Menüs](#)

Menüs

- [Symbolleisten und Statusleisten](#)

Statusleisten

- [Symbolleisten und Statusleisten](#)

Symbolleisten

- [Symbolleisten und Statusleisten](#)

Threads

- [Multitasking](#)

Bereichs-Auflösungsoperator

- [C++ im Überblick](#)

Betriebssystem

Ausnahmen

- [Ausnahmebehandlung](#)

Steuerelemente registrieren

- [ActiveX-Steuerelemente](#)

Bibliotheksmodule

- [Eigene Klassen und Module](#)

in MFC-DLL umwandeln

- [DLLs](#)

Klassen definieren

- [Eigene Klassen und Module](#)

Testanwendung

- [Eigene Klassen und Module](#)

und DLLs

- [DLLs](#)

Zeichenanwendung

- [Eigene Klassen und Module](#)

Bilder

- [Bilder, Zeichnungen und Bitmaps](#)

bItalic

- [Text und Schriften](#)

BitBlt

- [Bilder, Zeichnungen und Bitmaps](#)

BITMAP

- [Bilder, Zeichnungen und Bitmaps](#)

Bitmaps

- [Bilder, Zeichnungen und Bitmaps](#)

anpassen

- [Bilder, Zeichnungen und Bitmaps](#)

anzeigen

- [Bilder, Zeichnungen und Bitmaps](#)

Attach

- [Bilder, Zeichnungen und Bitmaps](#)

BitBlt

- [Bilder, Zeichnungen und Bitmaps](#)

CBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

kopieren

- [Bilder, Zeichnungen und Bitmaps](#)

laden

- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)

Ressourcen einfügen

- [Antworten](#)

StretchBlt

- [Bilder, Zeichnungen und Bitmaps](#)

verbinden mit CBitmap-Objekt

- [Bilder, Zeichnungen und Bitmaps](#)

BN_CLICKED

- [Steuerelemente](#)

BOF

- [Datenbanken per ADO bearbeiten](#)

BottomRight

- [MFC-Hilfsklassen](#)

Browse-Infos

- [Debugger und Profiler](#)

bUnderline

- [Text und Schriften](#)

C

C++

Anweisungen

- [C++ im Überblick](#)

Groß-/Kleinschreibung

- [C++ im Überblick](#)

Grundlagen

- [C++ im Überblick](#)

CADORecordBinding

- [Datenbanken per ADO bearbeiten](#)

CalcPoint (CTaskingDoc)

- [Multitasking](#)

Callback-Funktionen

- [Timer](#)
- [Text und Schriften](#)
 - EnumFontFamProc
 - [Text und Schriften](#)

CArchive

- [Dateizugriff](#)
 - Ausnahmen
 - [Ausnahmebehandlung](#)
 - Dateitypen
 - [Dateizugriff](#)
 - IsLoading
 - [Dateizugriff](#)
 - IsStoring
 - [Dateizugriff](#)
 - load
 - [Eigene ActiveX- Steuerelemente](#)
 - store
 - [Eigene ActiveX- Steuerelemente](#)

CArray

- [MFC-Hilfsklassen](#)

CAsyncSocket

- [Internet und Netzwerke](#)

catch-Blöcke

- [SDI-Anwendungen](#)
- [Ausnahmebehandlung](#)

CBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

CBRS_XXXX-Konstanten

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

CBS_XXXX-Konstanten

- [Symbolleisten und Statusleisten](#)

CByteArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CChildFrame

- [MDI-Anwendungen](#)

CClientDC

- [Maus und Tastatur](#)
- [Text und Schriften](#)

CCmdTarget

- [Maus und Tastatur](#)

CColorDialog

- [Dialogfelder](#)

CDaoErrorInfo

- [Ausnahmebehandlung](#)

CDAOException

- [Ausnahmebehandlung](#)

CDatabase

- [Daten aus ODBC- Datenbanken abrufen](#)

CDialog

- [Steuerelemente](#)

OnCancel

- [Steuerelemente](#)

OnOK

- [Steuerelemente](#)

CDocument

- [SDI-Anwendungen](#)

CDWordArray

- [MFC-Hilfsklassen](#)

CEditView

- [SDI-Anwendungen](#)

CEvent

- [Multitasking](#)

CException

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

CFile

- [Dateizugriff](#)

modeRead

- [Eigene ActiveX- Steuerelemente](#)

modeWrite

- [Eigene ActiveX- Steuerelemente](#)

CFileDialog

- [Dialogfelder](#)

OPENFILENAME

- [Dialogfelder](#)

CFileException

- [Ausnahmebehandlung](#)

CFindReplaceDialog

- [Dialogfelder](#)

CFont

- [Text und Schriften](#)

CFontDialog

- [Dialogfelder](#)

CFormView

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

CFrameView

- [SDI-Anwendungen](#)

CHtmlView

- [SDI-Anwendungen](#)
- [Funktionen für Webbrowser](#)

class

- [C++ im Überblick](#)

ClearDrawing (CModArt)

- [Eigene Klassen und Module](#)

Client

- [ActiveX-Steuerelemente](#)

Sockets

- [Internet und Netzwerke](#)

Client-Bereich

Mittelpunkt

- [MFC-Hilfsklassen](#)

CLine

- [SDI-Anwendungen](#)

Farben aufnehmen

- [SDI-Anwendungen](#)

Konstruktor

- [SDI-Anwendungen](#)

Typumwandlung

- [SDI-Anwendungen](#)

zeichnen

- [SDI-Anwendungen](#)

CLIP_XXXX-Konstanten

- [Text und Schriften](#)

CList

- [MFC-Hilfsklassen](#)

CListView

- [SDI-Anwendungen](#)

Close

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Internet und Netzwerke](#)

CMainFrame

- [MDI-Anwendungen](#)

CMap

- [MFC-Hilfsklassen](#)

CMapPtrToPtr

- [MFC-Hilfsklassen](#)

CMapPtrToWord

- [MFC-Hilfsklassen](#)

CMapStringToOb

- [MFC-Hilfsklassen](#)

CMapStringToPtr

- [MFC-Hilfsklassen](#)

CMapStringToString

- [MFC-Hilfsklassen](#)

CMapWordToOb

- [MFC-Hilfsklassen](#)

CMapWordToPtr

- [MFC-Hilfsklassen](#)

CMDIChildWnd

- [MDI-Anwendungen](#)

CMDIFrameWnd

- [MDI-Anwendungen](#)

CMemoryException

- [Ausnahmebehandlung](#)

CMutex

- [Multitasking](#)

CNotSupportedException

- [Ausnahmebehandlung](#)

CObArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

GetSize

- [SDI-Anwendungen](#)

Größe ermitteln

- [SDI-Anwendungen](#)

Serialisierung

- [SDI-Anwendungen](#)

CObList

- [MFC-Hilfsklassen](#)

Code

bearbeiten

- [Erste Schritte mit Visual C++](#)

Eintrittspunkt

- [C++ im Überblick](#)

in Anwendung aufnehmen

- [Erste Schritte mit Visual C++](#)

mnemonischer (Zugriffstasten)

- [Steuerelemente](#)

wiederverwendbarer

- [Eigene Klassen und Module](#)

Code-Änderungen zuweisen

- [Debugger und Profiler](#)

CoInitialize

- [Datenbanken per ADO bearbeiten](#)

COleDateTime

- [Daten aus ODBC- Datenbanken abrufen](#)
- [MFC-Hilfsklassen](#)

Differenzen

- [MFC-Hilfsklassen](#)

Status

- [MFC-Hilfsklassen](#)

COleDateTimeSpan

Status

- [MFC-Hilfsklassen](#)

COleDispatchException

- [Ausnahmebehandlung](#)

COMMAND

- [Menüs](#)

Command

ADO

- [Datenbanken per ADO bearbeiten](#)

CommandToIndex

- [Symbolleisten und Statusleisten](#)

Compare

- [MFC-Hilfsklassen](#)

CompareNoCase

- [MFC-Hilfsklassen](#)

Compiler

Optionen

- [Debugger und Profiler](#)

Compilerdirektiven

- [Dialogfelder](#)

Computernamen

loopback

- [Internet und Netzwerke](#)

COM-Technologie

- [ActiveX-Steuerelemente](#)

ActiveX-Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Umgebung initialisieren

- [Datenbanken per ADO bearbeiten](#)

Connect

Sockets

- [Internet und Netzwerke](#)

Connection

ADO

- [Datenbanken per ADO bearbeiten](#)

schließen

- [Datenbanken per ADO bearbeiten](#)

Container

ActiveX

- [ActiveX-Steuererelemente](#)

ActiveX-Steuererelemente

- [Eigene ActiveX- Steuererelemente](#)

Methoden

- [Eigene ActiveX- Steuererelemente](#)

CoUninitialize

- [Datenbanken per ADO bearbeiten](#)

CPageSetupDialog

- [Dialogfelder](#)

CPen

- [Bilder, Zeichnungen und Bitmaps](#)

CPerson

- [Dateizugriff](#)

CPoint

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CPrintDialog

- [Dialogfelder](#)

CPrintInfo

- [Drucken und Druckvorschau](#)

CPtrArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CPtrList

- [MFC-Hilfsklassen](#)

Create

Sockets

- [Internet und Netzwerke](#)

CREATE_SUSPENDED

- [Multitasking](#)

CreateBlankRecord (CdbAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

CreateColorBar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

CreateEx

- [Symbolleisten und Statusleisten](#)

CreateFont

- [Text und Schriften](#)

CreateFontIndirect

- [Text und Schriften](#)

CreateInstance

- [Datenbanken per ADO bearbeiten](#)

CreateProcess

- [Steuerelemente](#)

CRecordset

- [Daten aus ODBC- Datenbanken abrufen](#)

CRect

- [MFC-Hilfsklassen](#)

CResourceException

- [Ausnahmebehandlung](#)

CRichEditView

- [SDI-Anwendungen](#)

CScrollView

- [SDI-Anwendungen](#)

CSemaphore

- [Multitasking](#)

CSize

- [MFC-Hilfsklassen](#)

cStrikeOut

- [Text und Schriften](#)

CString

- [Steuerelemente](#)
- [MFC-Hilfsklassen](#)

CStringArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

CStringList

- [MFC-Hilfsklassen](#)

CTime

- [Timer](#)
- [Daten aus ODBC- Datenbanken abrufen](#)

- [MFC-Hilfsklassen](#)

Datenbanken

- [MFC-Hilfsklassen](#)

CTimeSpan

- [MFC-Hilfsklassen](#)

CToolBar

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

CTreeView

- [SDI-Anwendungen](#)

CUIIntArray

- [MFC-Hilfsklassen](#)

Cursor Siehe auch Mauszeiger

- [Maus und Tastatur](#)

Cursortypen

- [Datenbanken per ADO bearbeiten](#)

CUserException

- [Ausnahmebehandlung](#)

CView

- [SDI-Anwendungen](#)

CWaitCursor

- [Maus und Tastatur](#)

CWinApp

- [SDI-Anwendungen](#)

CWinThread

- [Multitasking](#)

CWnd

- [Steuerelemente](#)

CWordArray

- [SDI-Anwendungen](#)
- [MFC-Hilfsklassen](#)

D

DataGrid

- [Datenbanken per ADO bearbeiten](#)

DATE

- [MFC-Hilfsklassen](#)

Dateiansicht

- [Erste Schritte mit Visual C++](#)

Dateiausnahmen

Ursachencodes

- [Ausnahmebehandlung](#)

Dateiauswahl

- [Dialogfelder](#)

Dateien

afxtemp1.h

- [MFC-Hilfsklassen](#)

Ausnahmebehandlung

- [Ausnahmebehandlung](#)

Ausnahmen

- [Ausnahmebehandlung](#)

DLLs

- [DLLs](#)

Erweiterung für Dokumente festlegen

- [SDI-Anwendungen](#)

Header-

- [C++ im Überblick](#)

iostream.h

- [C++ im Überblick](#)

LIB

- [DLLs](#)

load

- [Eigene ActiveX- Steuerelemente](#)

modeRead

- [Eigene ActiveX- Steuerelemente](#)

modewrite

- [Antworten](#)

Moduldefinitions-

- [DLLs](#)

Msado15.tlh

- [Datenbanken per ADO bearbeiten](#)

Name

- [Antworten](#)

Pfad

- [Antworten](#)

Quellcode-

- [C++ im Überblick](#)

Remote-Debugger

- [Debugger und Profiler](#)

Ressourcen (.rc)

- [Menüs](#)

store

- [Eigene ActiveX- Steuerelemente](#)

Vorlagen

- [MFC-Hilfsklassen](#)

zum Lesen öffnen

- [Eigene ActiveX- Steuerelemente](#)

Dateigliederung

- [Debugger und Profiler](#)

Dateinamen

Dialogfeld Öffnen

- [Dialogfelder](#)

Dateitypen

CArchive

- [Dateizugriff](#)

Dateizugriff

- [Dateizugriff](#)

Daten

abrufen (ADO)

- [Datenbanken per ADO bearbeiten](#)

Kombinationsfelder

- [Steuerelemente](#)

zufällige erzeugen

- [ActiveX-Steuerelemente](#)

Datenbankanwendungen

Basisklassen

- [Dateizugriff](#)

Navigation

- [Dateizugriff](#)

Datenbank-Assistent

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenbankausnahmen

Fehlercodes

- [Ausnahmebehandlung](#)

Datenbanken

- [Daten aus ODBC- Datenbanken abrufen](#)

Access

- [Daten aus ODBC- Datenbanken abrufen](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

Ausnahmen

- [Ausnahmebehandlung](#)

AutoWert-Feld

- [Daten aus ODBC- Datenbanken abrufen](#)

CADORecordBinding

- [Datenbanken per ADO bearbeiten](#)

CDatabase

- [Daten aus ODBC- Datenbanken abrufen](#)

CRecordset

- [Daten aus ODBC- Datenbanken abrufen](#)

CTime

- [MFC-Hilfsklassen](#)

Cursortypen

- [Datenbanken per ADO bearbeiten](#)

Datenquelle

- [Datenbanken per ADO bearbeiten](#)

Fehler

- [Datenbanken per ADO bearbeiten](#)

Felder mit Steuerelementen verbinden

- [Daten aus ODBC- Datenbanken abrufen](#)

flache

- [Dateizugriff](#)

in Tabellen zwischenspeichern

- [MFC-Hilfsklassen](#)

lineare

- [Dateizugriff](#)

ODBC-Schnittstelle

- [Daten aus ODBC- Datenbanken abrufen](#)

OLE DB

- [Datenbanken per ADO bearbeiten](#)

Produkte

- [Daten aus ODBC- Datenbanken abrufen](#)

relationale

- [Dateizugriff](#)

Transaktionen

- [Datenbanken per ADO bearbeiten](#)

Verbindung zu

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)
- [Datenbanken per ADO bearbeiten](#)

Datenbankfelder

Datumswerte

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenbankverbindung

CDatabase

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenquelle

ADO

- [Datenbanken per ADO bearbeiten](#)

festlegen

- [Datenbanken per ADO bearbeiten](#)

hinzufügen

- [Daten aus ODBC- Datenbanken abrufen](#)

Datensätze

aktualisieren

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

BOF

- [Datenbanken per ADO bearbeiten](#)

CRecordset

- [Daten aus ODBC- Datenbanken abrufen](#)

EOF

- [Datenbanken per ADO bearbeiten](#)

Felder

- [Datenbanken per ADO bearbeiten](#)

hinzufügen

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)
- [Datenbanken per ADO bearbeiten](#)

leeren

- [Datenbanken per ADO bearbeiten](#)

löschen

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)
- [Datenbanken per ADO bearbeiten](#)

Datensatzfeldaustausch (DFX)

- [Ausnahmebehandlung](#)

Datensatzklassen

benutzerdefinierte

- [Datenbanken per ADO bearbeiten](#)

Datentypen

benutzerdefinierte

- [C++ im Überblick](#)

Größe

- [C++ im Überblick](#)

LPCTSTR

- [Eigene ActiveX- Steuerelemente](#)

Recordset

- [Datenbanken per ADO bearbeiten](#)

Strukturen

- [C++ im Überblick](#)

Variablen

- [C++ im Überblick](#)

Variant

- [Datenbanken per ADO bearbeiten](#)

Datum/Zeit

Auflösung

- [MFC-Hilfsklassen](#)

Bereich

- [MFC-Hilfsklassen](#)

COleDateTime

- [MFC-Hilfsklassen](#)

CTime

- [MFC-Hilfsklassen](#)

CTimeSpan

- [MFC-Hilfsklassen](#)

Differenzen

- [MFC-Hilfsklassen](#)

Format

- [MFC-Hilfsklassen](#)

Formatcodes

- [MFC-Hilfsklassen](#)

GetDay

- [MFC-Hilfsklassen](#)

GetDayOfWeek

- [MFC-Hilfsklassen](#)

GetDayOfYear

- [MFC-Hilfsklassen](#)

GetHour

- [MFC-Hilfsklassen](#)

GetMinute

- [MFC-Hilfsklassen](#)

GetMonth

- [MFC-Hilfsklassen](#)

GetSecond

- [MFC-Hilfsklassen](#)

GetYear

- [MFC-Hilfsklassen](#)

Klassen

- [MFC-Hilfsklassen](#)

Ländereinstellungen

- [MFC-Hilfsklassen](#)

ParseDateTime

- [MFC-Hilfsklassen](#)

SetDateTime

- [MFC-Hilfsklassen](#)

suchen

- [MFC-Hilfsklassen](#)

Systemzeit

- [MFC-Hilfsklassen](#)

Wochentag

- [MFC-Hilfsklassen](#)

Zeitspannen

- [MFC-Hilfsklassen](#)

deaktivieren

Steuerelemente

- [Steuerelemente](#)

Debug

- [Debugger und Profiler](#)

Debugger

- [Debugger und Profiler](#)

Ablaufverfolgung

- [Debugger und Profiler](#)

Aufruf als ein Schritt

- [Debugger und Profiler](#)

Aufrufliste

- [Debugger und Profiler](#)

Ausführen

- [Debugger und Profiler](#)

Ausführen bis Cursor

- [Debugger und Profiler](#)

Ausführen bis Rücksprung

- [Debugger und Profiler](#)

Ausführung anhalten

- [Debugger und Profiler](#)

Bearbeiten und Fortsetzen

- [Debugger und Profiler](#)

Code-Änderungen zuweisen

- [Debugger und Profiler](#)

Debug beenden

- [Debugger und Profiler](#)

Disassemblierung

- [Debugger und Profiler](#)

EinzelSchrittbetrieb

- [Debugger und Profiler](#)

Erneut starten

- [Debugger und Profiler](#)

Fenster

- [Debugger und Profiler](#)

Haltepunkte

- [Debugger und Profiler](#)

In Aufruf springen

- [Debugger und Profiler](#)

Just-In-Time

- [Debugger und Profiler](#)

Optimierungen

- [Debugger und Profiler](#)

Optionen

- [Debugger und Profiler](#)

Register

- [Debugger und Profiler](#)

Remote

- [Debugger und Profiler](#)

Schnellüberwachung

- [Debugger und Profiler](#)

Speicher

- [Debugger und Profiler](#)

Variablen überwachen

- [Debugger und Profiler](#)

Warnstufen

- [Debugger und Profiler](#)

Debug-Infos

- [Debugger und Profiler](#)

DECLARE_DYNAMIC

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

DECLARE_SERIAL

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

DEFAULT_CHARSET

- [Text und Schriften](#)

DEFAULT_PITCH

- [Text und Schriften](#)

DEFAULT_QUALITY

- [Text und Schriften](#)

Definitionen

Funktionen

- [C++ im Überblick](#)

Klassen

- [C++ im Überblick](#)

Präprozessor

- [Debugger und Profiler](#)

Quellcode-Browser

- [Debugger und Profiler](#)

DeflateRect

- [MFC-Hilfsklassen](#)

Deklarationen

Klassen

- [C++ im Überblick](#)

verschachtelte

- [C++ im Überblick](#)

Delete

- [Daten aus ODBC- Datenbanken abrufen](#)

Delete (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

DeleteContents

- [SDI-Anwendungen](#)
- [Datenbanken per ADO bearbeiten](#)

DeleteContents (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

DeleteContents (CSerialDoc)

- [Dateizugriff](#)

DeleteContents (CTag10Doc)

- [SDI-Anwendungen](#)

DeleteContents (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

Deserialisierung

- [Dateizugriff](#)

Destruktor

- [C++ im Überblick](#)

Aufrufreihenfolge

- [C++ im Überblick](#)

der Basisklasse

- [C++ im Überblick](#)

Regeln

- [C++ im Überblick](#)

DEVICE_FONTTYPE

- [Text und Schriften](#)

DevMode

- [Drucken und Druckvorschau](#)

DFX (Datensatzfeldaustausch)

- [Ausnahmebehandlung](#)

Dialog einfügen

- [Dialogfelder](#)

Dialog-Editor

- [Erste Schritte mit Visual C++](#)

Dialogleisten

- [Funktionen für Webbrowser](#)

Eigenschaftsseiten

- [Eigene ActiveX- Steuerelemente](#)

Steuerelemente

- [Steuerelemente](#)

Dialogfelder

- [Dialogfelder](#)

anzeigen

- [Dialogfelder](#)

benutzerdefinierte

- [Dialogfelder](#)

Dateiauswahl

- [Dialogfelder](#)

Dateinamen ermitteln

- [Dialogfelder](#)

Datenbankoptionen

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenbanktabellen wählen

- [Daten aus ODBC- Datenbanken abrufen](#)

DoModal

- [Dialogfelder](#)

Drucken

- [Dialogfelder](#)
- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

Drucken direkt aufrufen

- [Drucken und Druckvorschau](#)

Drucken überspringen

- [Drucken und Druckvorschau](#)

Eigenschaften

- [Erste Schritte mit Visual C++](#)

Farbauswahl

- [Dialogfelder](#)

Fokus

- [Maus und Tastatur](#)

Gerätekontext

- [Maus und Tastatur](#)

Hauptfenster

- [Erste Schritte mit Visual C++](#)

Hinzufügen einer Klasse

- [Dialogfelder](#)

Info

- [Erste Schritte mit Visual C++](#)

Laufwerksbezeichnung ermitteln

- [Dialogfelder](#)

mehrere in einer Anwendung

- [Bilder, Zeichnungen und Bitmaps](#)

Meldungsfelder

- [Dialogfelder](#)

Menüs verbinden

- [Menüs](#)

MessageBox

- [Erste Schritte mit Visual C++](#)

modale

- [Dialogfelder](#)

Navigation

- [Steuerelemente](#)

nichtmodale

- [Dialogfelder](#)

ODBC Microsoft Access 97-Setup

- [Daten aus ODBC- Datenbanken abrufen](#)

ODBC-Datenquellen-Administrator

- [Daten aus ODBC- Datenbanken abrufen](#)

Öffnen

- [Dialogfelder](#)

Pfad ermitteln

- [Dialogfelder](#)

Projekteinstellungen

- [Debugger und Profiler](#)

schließen

- [Steuerelemente](#)

Schriftauswahl

- [Dialogfelder](#)

Seite einrichten

- [Dialogfelder](#)

Speichern unter

- [Dialogfelder](#)

Steuerelemente

- [Steuerelemente](#)

Suchen und Ersetzen

- [Dialogfelder](#)

Symbol

- [Erste Schritte mit Visual C++](#)

Systemmenü

- [Bilder, Zeichnungen und Bitmaps](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)

Verzeichnis vorgeben

- [Dialogfelder](#)

vordefinierte

- [Dialogfelder](#)

Dialogleiste

Adresse ausgeben

- [Funktionen für Webbrowser](#)

Dialogleisten

- [Funktionen für Webbrowser](#)

Dienstprogramme

MFC-Ablaufverfolgung

- [Debugger und Profiler](#)

OLE/COM-Objektkatalog

- [Debugger und Profiler](#)

Prozess-Viewer

- [Debugger und Profiler](#)

Spy++

- [Debugger und Profiler](#)

Direktiven

#import

- [Datenbanken per ADO bearbeiten](#)

Präprozessor

- [C++ im Überblick](#)

Disassemblierung

- [Debugger und Profiler](#)

DISPIDs

- [ActiveX-Steuerelemente](#)

Division

- [C++ im Überblick](#)

DLLs

- [DLLs](#)

AFX_EXT_CLASS

- [DLLs](#)

Funktionen exportieren

- [DLLs](#)

MFC-Klassen

- [DLLs](#)

Moduldefinitionsdatei

- [DLLs](#)

portierbare

- [DLLs](#)

Standard-

- [DLLs](#)
- [DLLs](#)

Testanwendung

- [DLLs](#)

Typen in Visual C++

- [DLLs](#)

und Bibliotheksmodule

- [DLLs](#)

DOCINFO

- [Drucken und Druckvorschau](#)

DockControlBar

- [Symbolleisten und Statusleisten](#)

Dokument

Begriff

- [SDI-Anwendungen](#)

Dokument/Ansicht-Architektur

Drucken

- [Drucken und Druckvorschau](#)

MDI-Anwendungen

- [MDI-Anwendungen](#)

SDI-Anwendungen

- [SDI-Anwendungen](#)

Dokumente

als noch nicht gespeichert markieren

- [SDI-Anwendungen](#)

Farben hinzufügen

- [SDI-Anwendungen](#)

neue öffnen

- [Dateizugriff](#)

Dokumentklasse

Inhalt löschen

- [SDI-Anwendungen](#)

Dokumentobjekte

- [SDI-Anwendungen](#)

Serialize

- [Dateizugriff](#)

Zeiger auf

- [SDI-Anwendungen](#)

DoModal

- [Dialogfelder](#)

doppelklicken

- [Maus und Tastatur](#)

DoPreparePrinting

- [Drucken und Druckvorschau](#)

DoSort

- [ActiveX-Steuerelemente](#)

DoSpin (CTaskingDoc)

- [Multitasking](#)

Download

anhalten

- [Funktionen für Webbrowser](#)

erneut starten

- [Funktionen für Webbrowser](#)

Fortschritt

- [Funktionen für Webbrowser](#)

DRAFT_QUALITY

- [Text und Schriften](#)

Drag & Drop

Spalten verschieben

- [ActiveX-Steuerelemente](#)

Draw (CLine)

- [SDI-Anwendungen](#)
- [SDI-Anwendungen](#)
- [Antworten](#)

Draw (CModArt)

- [Eigene Klassen und Module](#)

Draw (CSpinner)

- [Multitasking](#)

DrawLine (CPaintDlg)

- [Bilder, Zeichnungen und Bitmaps](#)

DrawRegion (CPaintDlg)

- [Bilder, Zeichnungen und Bitmaps](#)
- [Antworten](#)

Dropdown-Listenfelder

- [Steuerelemente](#)

Drucken

- [Dialogfelder](#)
- [Drucken und Druckvorschau](#)

Abbildungsmodi

- [Drucken und Druckvorschau](#)

abbrechen

- [Drucken und Druckvorschau](#)

anpassen

- [Drucken und Druckvorschau](#)

beschleunigen

- [Drucken und Druckvorschau](#)

CPrintInfo

- [Drucken und Druckvorschau](#)

DevMode

- [Drucken und Druckvorschau](#)

Dialogfeld direkt aufrufen

- [Drucken und Druckvorschau](#)

Dialogfeld überspringen

- [Drucken und Druckvorschau](#)

Dialogfelder

- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

DOCINFO

- [Drucken und Druckvorschau](#)

DoPreparePrinting

- [Drucken und Druckvorschau](#)

EndDoc

- [Drucken und Druckvorschau](#)

EndPage

- [Drucken und Druckvorschau](#)

erste/letzte Seite

- [Drucken und Druckvorschau](#)

Gerätekontext

- [Drucken und Druckvorschau](#)

GetDeviceCaps

- [Drucken und Druckvorschau](#)

Hochformat

- [Drucken und Druckvorschau](#)

Höhen-/Seitenverhältnis

- [Drucken und Druckvorschau](#)

Kopien

- [Drucken und Druckvorschau](#)

OnDraw

- [Drucken und Druckvorschau](#)

OnPreparePrinting

- [Drucken und Druckvorschau](#)

OnPrint

- [Drucken und Druckvorschau](#)

Papierabmessungen

- [Drucken und Druckvorschau](#)

Pinself

- [Drucken und Druckvorschau](#)

Querformat

- [Drucken und Druckvorschau](#)

Ressourcen

- [Drucken und Druckvorschau](#)

Seitenausrichtung

- [Drucken und Druckvorschau](#)

Seitennummern

- [Drucken und Druckvorschau](#)

Seitenumbruch

- [Drucken und Druckvorschau](#)

Sortieren

- [Drucken und Druckvorschau](#)

Spooler

- [Drucken und Druckvorschau](#)

StartDoc

- [Drucken und Druckvorschau](#)

StartPage

- [Drucken und Druckvorschau](#)

Stift

- [Drucken und Druckvorschau](#)

Druckvorschau

- [Drucken und Druckvorschau](#)

beschleunigen

- [Drucken und Druckvorschau](#)

Höhen-/Seitenverhältnis

- [Drucken und Druckvorschau](#)

mehrere Seiten

- [Drucken und Druckvorschau](#)

durchgestrichen

- [Text und Schriften](#)

E

E/A-Streams

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

Edit

- [Daten aus ODBC- Datenbanken abrufen](#)

Editoren

Dialog

- [Erste Schritte mit Visual C++](#)

Menü

- [Menüs](#)

Symbol

- [Erste Schritte mit Visual C++](#)

Symbolleisten

- [Symbolleisten und Statusleisten](#)

Zeichenfolgen

- [Symbolleisten und Statusleisten](#)

Eigenschaften

ambient

- [Eigene ActiveX- Steuerelemente](#)

Automatisierung

- [Eigene ActiveX- Steuerelemente](#)

benutzerdefinierte

- [Eigene ActiveX- Steuerelemente](#)

erweiterte

- [Eigene ActiveX- Steuerelemente](#)

externer Name

- [Eigene ActiveX- Steuerelemente](#)

Gruppe

- [Dialogfelder](#)

Member-Variablen

- [Eigene ActiveX- Steuerelemente](#)

Menübefehle

- [Menüs](#)

Popup

- [Menüs](#)

Schaltfläche

- [Erste Schritte mit Visual C++](#)

Steuerelemente

- [Steuerelemente](#)
- [Eigene ActiveX- Steuerelemente](#)

Variablen

- [Eigene ActiveX- Steuerelemente](#)

vordefinierte

- [Eigene ActiveX- Steuerelemente](#)

Eigenschaftsseiten

- [Eigene ActiveX- Steuerelemente](#)

ActiveX-Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Einfügeoperator

- [C++ im Überblick](#)

Eingabefelder

- [Steuerelemente](#)

Inhalt löschen

- [Steuerelemente](#)

Eingabe-Operator

- [C++ im Überblick](#)

Eintrittspunkt

Programm

- [C++ im Überblick](#)

Einzelschrittbetrieb

- [Debugger und Profiler](#)

Haltepunkte

- [Debugger und Profiler](#)

Elemente

Quellcode-Browser

- [Debugger und Profiler](#)

Ellipse

- [Bilder, Zeichnungen und Bitmaps](#)

Empty

- [MFC-Hilfsklassen](#)

EN_CHANGE

- [Timer](#)

EnableWindow

- [Steuerelemente](#)

END_ADO_BINDING

- [Datenbanken per ADO bearbeiten](#)

EndPage

- [Drucken und Druckvorschau](#)

EndWaitCursor

- [Maus und Tastatur](#)

Entwicklungsumgebung

- [Erste Schritte mit Visual C++](#)

Arbeitsbereich

- [Erste Schritte mit Visual C++](#)

Ausgabebereich

- [Erste Schritte mit Visual C++](#)

Editorbereich

- [Erste Schritte mit Visual C++](#)

Menüleisten

- [Erste Schritte mit Visual C++](#)

neu arrangieren

- [Erste Schritte mit Visual C++](#)

Symbolleisten

- [Erste Schritte mit Visual C++](#)

enum

- [C++ im Überblick](#)

EnumFontFamiliesEx

- [Text und Schriften](#)

EnumFontFamProc

- [Text und Schriften](#)

ENUMLOGFONTEX

- [Text und Schriften](#)

EOF

- [Datenbanken per ADO bearbeiten](#)

Ereignisse

benutzerdefinierte

- [Eigene ActiveX- Steuerelemente](#)

CEvent

- [Multitasking](#)

Klicken

- [Erste Schritte mit Visual C++](#)

Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

Maus

- [Maus und Tastatur](#)

OnAccept

- [Internet und Netzwerke](#)

- [Internet und Netzwerke](#)

OnClose

- [Internet und Netzwerke](#)

- [Internet und Netzwerke](#)

OnConnect

- [Internet und Netzwerke](#)

OnReceive

- [Internet und Netzwerke](#)

- [Internet und Netzwerke](#)

OnSend

- [Internet und Netzwerke](#)

Sockets

- [Internet und Netzwerke](#)

Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Tastatur

- [Maus und Tastatur](#)

Threads

- [Multitasking](#)

Timer

- [Timer](#)

ErrnoToException

- [Ausnahmebehandlung](#)

Error

ADO

- [Datenbanken per ADO bearbeiten](#)

erweiterte MFC-DLLs

- [DLLs](#)
in Standard-DLL umwandeln
 - [DLLs](#)

Exceptions Siehe Ausnahmen

- [Ausnahmebehandlung](#)

F

Fächer

- [Multitasking](#)
drehen
 - [Multitasking](#)
initialisieren
 - [Multitasking](#)
Positionen berechnen
 - [Multitasking](#)
zeichnen
 - [Multitasking](#)

Farbauswahl

- [Dialogfelder](#)

Farben

RGB

- [Maus und Tastatur](#)
- [Bilder, Zeichnungen und Bitmaps](#)

Stifte

- [Bilder, Zeichnungen und Bitmaps](#)

Tabelle erstellen

- [Bilder, Zeichnungen und Bitmaps](#)

Farbtabelle

- [SDI-Anwendungen](#)

- [Multitasking](#)

Fehler

abfangen

- [Ausnahmebehandlung](#)

Assertion

- [MFC-Hilfsklassen](#)

Ausnahmen

- [Ausnahmebehandlung](#)

nicht initialisierte Zeiger

- [C++ im Überblick](#)

Sockets

- [Internet und Netzwerke](#)

Warnungen als

- [Debugger und Profiler](#)

Zugriffsverletzungen

- [Ausnahmebehandlung](#)

Fehlercodes

AFX_DAO_ERROR_DFX_BIND

- [Ausnahmebehandlung](#)

AFX_DAO_ERROR_ENGINE_INITIALIZATION

- [Ausnahmebehandlung](#)

AFX_DAO_ERROR_OBJECT_NOT_OPEN

- [Ausnahmebehandlung](#)

Datenbankausnahmen

- [Ausnahmebehandlung](#)

ErrnoToException

- [Ausnahmebehandlung](#)

NO_AFX_DAO_ERROR

- [Ausnahmebehandlung](#)

ThrowOsError

- [Ausnahmebehandlung](#)

Fehlermeldungen

Bezeichner

- [C++ im Überblick](#)

COleDateTime

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenbanken

- [Datenbanken per ADO bearbeiten](#)

SOCKET_ERROR

- [Internet und Netzwerke](#)

Zuweisung an Konstante

- [Steuerelemente](#)

Fehlersuche

ASSERT

- [Debugger und Profiler](#)

Debugger

- [Debugger und Profiler](#)

Felder

Datensätze

- [Datenbanken per ADO bearbeiten](#)

Felder Siehe auch Arrays

- [MFC-Hilfsklassen](#)

Feldklassen

- [MFC-Hilfsklassen](#)

Fenster

Elemente neu arrangieren

- [Symbolleisten und Statusleisten](#)

Fokus

- [Maus und Tastatur](#)

Mittelpunkt

- [MFC-Hilfsklassen](#)

modale

- [Dialogfelder](#)

neu zeichnen

- [Bilder, Zeichnungen und Bitmaps](#)

nichtmodale

- [Dialogfelder](#)

OnPaint

- [Bilder, Zeichnungen und Bitmaps](#)

untergeordnete

- [MDI-Anwendungen](#)

Feststelltasten

in Statusleiste anzeigen/ausblenden

- [Symbolleisten und Statusleisten](#)

FF_xxxx-Konstanten

- [Text und Schriften](#)

Field

- [Datenbanken per ADO bearbeiten](#)

FillFontList

- [Text und Schriften](#)
- [Text und Schriften](#)

Filtern

Flags

- [Maus und Tastatur](#)

Find

- [MFC-Hilfsklassen](#)
- [MFC-Hilfsklassen](#)

FindIndex

- [MFC-Hilfsklassen](#)

FindOneOf

- [MFC-Hilfsklassen](#)

FIXED_PITCH

- [Text und Schriften](#)

Flags

Attribute

- [Maus und Tastatur](#)

FlexGrid

- [ActiveX-Steuerelemente](#)

Daten laden

- [ActiveX-Steuerelemente](#)

Eingaben

- [ActiveX-Steuerelemente](#)

Ereignisse

- [ActiveX-Steuerelemente](#)

GetFixedRows

- [ActiveX-Steuerelemente](#)

GetMouseCol

- [ActiveX-Steuerelemente](#)

GetMouseRow

- [ActiveX-Steuerelemente](#)

GetRows

- [ActiveX-Steuerelemente](#)

ID einer Zelle berechnen

- [ActiveX-Steuerelemente](#)

SetCol

- [ActiveX-Steuerelemente](#)

SetColPosition

- [ActiveX-Steuerelemente](#)

SetColSel

- [ActiveX-Steuerelemente](#)

SetMergeCol

- [ActiveX-Steuerelemente](#)

SetSort

- [ActiveX-Steuerelemente](#)

SetTextArray

- [ActiveX-Steuerelemente](#)

sortieren

- [ActiveX-Steuerelemente](#)

- [ActiveX-Steuerelemente](#)

Spalte ermitteln

- [ActiveX-Steuerelemente](#)

Spalte verschieben

- [ActiveX-Steuerelemente](#)

Spalten auswählen

- [ActiveX-Steuerelemente](#)

Spaltennummer speichern

- [ActiveX-Steuerelemente](#)

unverankertes Eingabefeld

- [ActiveX-Steuerelemente](#)

Zeile ermitteln

- [ActiveX-Steuerelemente](#)

Fokus

- [Maus und Tastatur](#)

Format

- [MFC-Hilfsklassen](#)

Formatcodes

Datum/Zeit

- [MFC-Hilfsklassen](#)

Zeichenfolgen

- [MFC-Hilfsklassen](#)

Formatieren

Text

- [MFC-Hilfsklassen](#)

Formulare

Viewport

- [Multitasking](#)

Formularklassen

- [Eigene Klassen und Module](#)

Fortschritt

Download

- [Funktionen für Webbrowser](#)

Fragezeichen-Symbol

- [Dialogfelder](#)

Friend

Funktionen

- [C++ im Überblick](#)

Klassen

- [C++ im Überblick](#)

Funktionen

Add

- [MFC-Hilfsklassen](#)

AddHead

- [MFC-Hilfsklassen](#)

AddLine (CTag10Doc)

- [SDI-Anwendungen](#)
- [SDI-Anwendungen](#)
- [Antworten](#)

AddNew

- [Daten aus ODBC- Datenbanken abrufen](#)

AddNew (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

AddNewRecord (CSerialDoc)

- [Dateizugriff](#)

AddTail

- [MFC-Hilfsklassen](#)

afx_msg (Typ)

- [MDI-Anwendungen](#)

AfxBeginThread

- [Multitasking](#)

AfxThrowMemoryException

- [Ausnahmebehandlung](#)

AfxThrowArchiveException

- [Ausnahmebehandlung](#)

AfxThrowFileException

- [Ausnahmebehandlung](#)

AfxThrowNotSupportedException

- [Ausnahmebehandlung](#)

AfxThrowOleDispatchException

- [Ausnahmebehandlung](#)

AfxThrowOleException

- [Ausnahmebehandlung](#)

AfxThrowResourceException

- [Ausnahmebehandlung](#)

AfxThrowUserException

- [Ausnahmebehandlung](#)

Attach

- [Bilder, Zeichnungen und Bitmaps](#)

Aufzählungen

- [Text und Schriften](#)

BeginWaitCursor

- [Maus und Tastatur](#)

BitBlt

- [Bilder, Zeichnungen und Bitmaps](#)

BottomRight

- [MFC-Hilfsklassen](#)

CalcPoint (CTaskingDoc)

- [Multitasking](#)

Callback

- [Timer](#)
- [Text und Schriften](#)

ClearDrawing (CModArt)

- [Eigene Klassen und Module](#)

Close

- [Daten aus ODBC- Datenbanken abrufen](#)

CoInitialize

- [Datenbanken per ADO bearbeiten](#)

CommandToIndex

- [Symbolleisten und Statusleisten](#)

Compare

- [MFC-Hilfsklassen](#)

CompareNoCase

- [MFC-Hilfsklassen](#)

CoUninitialize

- [Datenbanken per ADO bearbeiten](#)

CreateBlankRecord (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

CreateColorBar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

CreateEx

- [Symbolleisten und Statusleisten](#)

CreateFontIndirect

- [Text und Schriften](#)

CreateInstance

- [Datenbanken per ADO bearbeiten](#)

CreateProcess

- [Steuerelemente](#)

Definition

- [C++ im Überblick](#)

DeflateRect

- [MFC-Hilfsklassen](#)

deklarieren

- [C++ im Überblick](#)

Delete

- [Daten aus ODBC- Datenbanken abrufen](#)

Delete (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

DeleteContents (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

DeleteContents (CSerialDoc)

- [Dateizugriff](#)

DeleteContents (CTag10Doc)

- [SDI-Anwendungen](#)

DeleteContents (CTestAppDoc)

- [Eigene Klassen und Module](#)

- [DLLs](#)

DockControlBar

- [Symbolleisten und Statusleisten](#)

Dokument als noch nicht gespeichert markieren

- [SDI-Anwendungen](#)

DoPreparePrinting

- [Drucken und Druckvorschau](#)

DoSort

- [ActiveX-Steuerelemente](#)

DoSpin (CTaskingDoc)

- [Multitasking](#)

Draw (CLine)

- [SDI-Anwendungen](#)

- [SDI-Anwendungen](#)

- [Antworten](#)

Draw (CModArt)

- [Eigene Klassen und Module](#)

Draw (CSpinner)

- [Multitasking](#)

DrawLine (CPaintDlg)

- [Bilder, Zeichnungen und Bitmaps](#)

DrawRegion (CPaintDlg)

- [Bilder, Zeichnungen und Bitmaps](#)

- [Antworten](#)

Edit

- [Daten aus ODBC- Datenbanken abrufen](#)

Ellipse

- [Bilder, Zeichnungen und Bitmaps](#)

Empty

- [MFC-Hilfsklassen](#)

EnableWindow

- [Steuerelemente](#)

EndDoc

- [Drucken und Druckvorschau](#)

EndPage

- [Drucken und Druckvorschau](#)

EndWaitCursor

- [Maus und Tastatur](#)

EnumFontFamiliesEx

- [Text und Schriften](#)

EnumFontFamProc

- [Text und Schriften](#)

ErrnoToException

- [Ausnahmebehandlung](#)

exportieren

- [DLLs](#)

FillFontList

- [Text und Schriften](#)
- [Text und Schriften](#)

Find

- [MFC-Hilfsklassen](#)
- [MFC-Hilfsklassen](#)

FindIndex

- [MFC-Hilfsklassen](#)

FindOneOf

- [MFC-Hilfsklassen](#)

Format

- [MFC-Hilfsklassen](#)

Friend

- [C++ im Überblick](#)

GenerateError (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

GenID

- [ActiveX-Steuerelemente](#)

GenNewDrawing (CSquiggleCtrl)

- [Antworten](#)

Get

- [Dateizugriff](#)
- [Eigene ActiveX- Steuerelemente](#)

GetAt

- [MFC-Hilfsklassen](#)

GetBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

GetCapture

- [SDI-Anwendungen](#)

GetColor (CTag10Doc)

- [SDI-Anwendungen](#)

GetCopies

- [Drucken und Druckvorschau](#)

GetCount

- [MFC-Hilfsklassen](#)

GetCurrentTime

- [MFC-Hilfsklassen](#)

GetDay

- [MFC-Hilfsklassen](#)

GetDayOfWeek

- [MFC-Hilfsklassen](#)

GetDayOfYear

- [MFC-Hilfsklassen](#)

GetDC

- [Multitasking](#)

GetDeviceCaps

- [Drucken und Druckvorschau](#)

GetDlgItem

- [Steuerelemente](#)

GetDocument

- [SDI-Anwendungen](#)

GetDrawing (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

GetDriverName

- [Drucken und Druckvorschau](#)

GetFileName

- [Antworten](#)

GetFirstPosition

- [MFC-Hilfsklassen](#)

GetFirstRecord (CSerialDoc)

- [Dateizugriff](#)

GetFirstViewPosition

- [Symbolleisten und Statusleisten](#)

GetFromPage

- [Drucken und Druckvorschau](#)

GetHeadPosition

- [MFC-Hilfsklassen](#)

GetHour

- [MFC-Hilfsklassen](#)

GetItemRect

- [Symbolleisten und Statusleisten](#)

GetKeyState

- [Maus und Tastatur](#)

GetLastRecord (CSerialDoc)

- [Dateizugriff](#)

GetLength (CModArt)

- [Eigene ActiveX- Steuerelemente](#)

GetLine (CTag10Doc)

- [SDI-Anwendungen](#)

GetLineCount (CTag10Doc)

- [SDI-Anwendungen](#)

GetLocationURL

- [Funktionen für Webbrowser](#)

GetMaxID (CDBOdbcSet)

- [Daten aus ODBC- Datenbanken abrufen](#)

GetMinute

- [MFC-Hilfsklassen](#)

GetMonth

- [MFC-Hilfsklassen](#)

GetNextAssoc

- [MFC-Hilfsklassen](#)

GetNextRecord (CSerialDoc)

- [Dateizugriff](#)

GetNextRecord (CSerialView)

- [Dateizugriff](#)

GetNextView

- [Symbolleisten und Statusleisten](#)

GetPathName

- [Antworten](#)

GetPortName

- [Drucken und Druckvorschau](#)

GetPrevRecord (CSerialDoc)

- [Dateizugriff](#)

GetPrevRecord (CSerialView)

- [Dateizugriff](#)

GetPrinterDC

- [Drucken und Druckvorschau](#)

GetRecordCount

- [Daten aus ODBC- Datenbanken abrufen](#)

GetRecSet (CdbAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

GetSecond

- [MFC-Hilfsklassen](#)

GetSize

- [SDI-Anwendungen](#)

GetSquiggleLength (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

GetTailPosition

- [MFC-Hilfsklassen](#)

GetToPage

- [Drucken und Druckvorschau](#)

GetTopLevelFrame

- [Symbolleisten und Statusleisten](#)

GetTotalRecords (CSerialDoc)

- [Dateizugriff](#)

GetUpperBound

- [MFC-Hilfsklassen](#)

GetViewWnd

- [Multitasking](#)

GetWidth (CTag10Doc)

- [Antworten](#)

GetWindowText

- [Funktionen für Webbrowser](#)

GetYear

- [MFC-Hilfsklassen](#)

GoBack

- [Funktionen für Webbrowser](#)

GoForward

- [Funktionen für Webbrowser](#)

GoHome

- [Funktionen für Webbrowser](#)

GoSearch

- [Funktionen für Webbrowser](#)

Height

- [MFC-Hilfsklassen](#)

InflateRect

- [MFC-Hilfsklassen](#)

InitSpinners (CTaskingDoc)

- [Multitasking](#)

Inline

- [Dateizugriff](#)
- [Multitasking](#)
- [C++ im Überblick](#)

InsertAfter

- [MFC-Hilfsklassen](#)

InsertAt

- [MFC-Hilfsklassen](#)

InsertBefore

- [MFC-Hilfsklassen](#)

InterSectRect

- [MFC-Hilfsklassen](#)

Invalidate

- [Bilder, Zeichnungen und Bitmaps](#)

IsBOF

- [Daten aus ODBC- Datenbanken abrufen](#)

IsEmpty

- [MFC-Hilfsklassen](#)

IsEOF

- [Daten aus ODBC- Datenbanken abrufen](#)

IsLoading

- [Dateizugriff](#)

IsStoring

- [Dateizugriff](#)

Left

- [MFC-Hilfsklassen](#)

LineTo

- [Maus und Tastatur](#)

Linie hinzufügen

- [SDI-Anwendungen](#)

LoadBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

LoadCursor

- [Maus und Tastatur](#)

LoadData

- [ActiveX-Steuerelemente](#)

LoadDrawing (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

- [Eigene ActiveX- Steuerelemente](#)

LoadImage

- [Bilder, Zeichnungen und Bitmaps](#)

- [Bilder, Zeichnungen und Bitmaps](#)

LoadStandardCursor

- [Maus und Tastatur](#)

LoadToolBar

- [Symbolleisten und Statusleisten](#)

Lookup

- [MFC-Hilfsklassen](#)

main

- [C++ im Überblick](#)

MakeLower

- [MFC-Hilfsklassen](#)

MakeReverse

- [MFC-Hilfsklassen](#)

MakeUpper

- [Steuerelemente](#)

- [MFC-Hilfsklassen](#)

MessageBox

- [Erste Schritte mit Visual C++](#)

- [Dialogfelder](#)

Methoden

- [C++ im Überblick](#)

Mid

- [MFC-Hilfsklassen](#)

ModArtClearDrawing

- [DLLs](#)

ModArtDraw

- [DLLs](#)

ModArtInit

- [DLLs](#)

ModArtNewDrawing

- [DLLs](#)

- [DLLs](#)

ModArtSerialize

- [DLLs](#)

Move

- [Daten aus ODBC- Datenbanken abrufen](#)

MoveFirst

- [Daten aus ODBC- Datenbanken abrufen](#)

- [Datenbanken per ADO bearbeiten](#)

MoveFirst (CdbAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

MoveLast

- [Daten aus ODBC- Datenbanken abrufen](#)

- [Datenbanken per ADO bearbeiten](#)

MoveNext

- [Daten aus ODBC- Datenbanken abrufen](#)

- [Datenbanken per ADO bearbeiten](#)

MovePrev

- [Daten aus ODBC- Datenbanken abrufen](#)

MovePrevious

- [Datenbanken per ADO bearbeiten](#)

MoveTo

- [Maus und Tastatur](#)

Nachrichtenbehandlung

- [Erste Schritte mit Visual C++](#)

Navigate

- [Funktionen für Webbrowser](#)
- [Funktionen für Webbrowser](#)

NewDataSet (CSerialView)

- [Dateizugriff](#)

NewDrawing (CModArt)

- [Eigene Klassen und Module](#)
- [Eigene Klassen und Module](#)
- [Eigene ActiveX- Steuerelemente](#)

NewLine (CModArt)

- [Eigene Klassen und Module](#)
- [Eigene Klassen und Module](#)
- [DLLs](#)
- [DLLs](#)
- [Eigene ActiveX- Steuerelemente](#)

NormalizeRect

- [MFC-Hilfsklassen](#)

OnAccept (CMySocket)

- [Internet und Netzwerke](#)

OnAccept (CSockDlg)

- [Internet und Netzwerke](#)
- [Antworten](#)

OnBbitmap

- [Bilder, Zeichnungen und Bitmaps](#)

OnBclose (CSockDlg)

- [Internet und Netzwerke](#)

OnBconnect (CSockDlg)

- [Internet und Netzwerke](#)

OnBcustomdialog

- [Dialogfelder](#)

OnBeginPrinting (CPrintItView)

- [Drucken und Druckvorschau](#)

OnBfirst (CSerialView)

- [Dateizugriff](#)

OnBlast (CSerialView)

- [Dateizugriff](#)

OnBsend (CSockDlg)

- [Internet und Netzwerke](#)

OnBwhichoption

- [Dialogfelder](#)

OnCancel

- [Steuerelemente](#)

OnCbemployed (CSerialView)

- [Dateizugriff](#)
- [Dateizugriff](#)

OnCbitalic (CTag7Dlg)

- [Antworten](#)

OnCbondle (CTaskingView)

- [Multitasking](#)

OnCbthread1 (CTaskingView)

- [Multitasking](#)

OnCbthread2 (CTaskingView)

- [Multitasking](#)

OnCbusetext (CTag7Dlg)

- [Antworten](#)

OnChangeEsamptext

- [Text und Schriften](#)

OnChangeEsamptext (CTag7Dlg)

- [Antworten](#)

OnChangeInterval

- [Timer](#)

OnCkenblmsg

- [Steuerelemente](#)

OnCkenblpgm (CTag2Dlg)

- [Antworten](#)

OnCkshwmsg

- [Steuerelemente](#)

OnClick (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

OnClose (CSockDlg)

- [Internet und Netzwerke](#)

OnClrmsg

- [Steuerelemente](#)

OnColorBlack (CTag10Doc)

- [SDI-Anwendungen](#)

OnColorBlue (CTag10Doc)

- [SDI-Anwendungen](#)

OnColorCommand (CTag11Doc)

- [MDI-Anwendungen](#)

OnConnect (CSockDlg)

- [Internet und Netzwerke](#)

OnContextMenu (CMenuDlg)

- [Menüs](#)

- [Antworten](#)

OnContextMenu (CTag11View)

- [MDI-Anwendungen](#)

OnCreate (CMainFrame)

- [Symbolleisten und Statusleisten](#)

- [Symbolleisten und Statusleisten](#)

OnDataFirst (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

OnDbClickMsfgrid (CActiveXDlg)

- [Antworten](#)

OnDestroy (CTaskingView)

- [Multitasking](#)

OnDfltmsg (CTag2Dlg)

- [Antworten](#)

OnDocumentComplete (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnDraw (CPrintItView)

- [Drucken und Druckvorschau](#)

OnDraw (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

OnDraw (CTag10View)

- [SDI-Anwendungen](#)

OnDraw (CTestAppView)

- [DLLs](#)

OnEndPrinting

- [Drucken und Druckvorschau](#)

OnExit

- [Steuerelemente](#)

OnFileHello

- [Menüs](#)

OnFileopen

- [Dialogfelder](#)

- [Dialogfelder](#)

OnFileopen (CDialoge)

- [Antworten](#)

OnGoBack (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnGoNext (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnGoSearch (CWebBrowseView)

- [Antworten](#)

OnGoStart (CWebBrowseView)

- [Antworten](#)

OnHello

- [Erste Schritte mit Visual C++](#)

OnIdle

- [Multitasking](#)

OnIdle (CTaskingApp)

- [Multitasking](#)

- [Multitasking](#)

- [Antworten](#)

OnInitDialog

- [Steuerelemente](#)

- [Steuerelemente](#)

OnInitDialog (CActiveXDlg)

- [ActiveX-Steuerelemente](#)

OnInitDialog (CGrafikDlg)

- [Bilder, Zeichnungen und Bitmaps](#)

OnInitDialog (CMausDlg)

- [Maus und Tastatur](#)

OnInitDialog (CSockDlg)

- [Internet und Netzwerke](#)

OnInitDialog (CTag7Dlg)

- [Text und Schriften](#)

- [Text und Schriften](#)

- [Antworten](#)

OnInitDialog (CTimerDlg)

- [Timer](#)

- [Timer](#)

OnKeyDown

- [Maus und Tastatur](#)

- [Maus und Tastatur](#)

OnKeyDown (CMausDlg)

- [Antworten](#)

OnLButtonDown (CMausDlg)

- [Maus und Tastatur](#)

OnLButtonDown (CTag10View)

- [SDI-Anwendungen](#)

OnLButtonUp (CTag10View)

- [SDI-Anwendungen](#)

OnMouseDownMsfggrid

- [ActiveX-Steuerelemente](#)

OnMouseMove (CMausDlg)

- [Maus und Tastatur](#)

- [Maus und Tastatur](#)

- [Antworten](#)

OnMouseMove (CTag10View)

- [SDI-Anwendungen](#)

OnMouseUpMsfggrid

- [ActiveX-Steuerelemente](#)

OnNewAddress (CMainFrame)

- [Funktionen für Webbrowser](#)

OnNewDocument

- [SDI-Anwendungen](#)

OnNewDocument (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

OnNewDocument (CSerialDoc)

- [Dateizugriff](#)

OnNewDocument (CTag10Doc)

- [SDI-Anwendungen](#)

- [Antworten](#)

OnNewDocument (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)

OnNewDocument (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

OnOK

- [Steuerelemente](#)

OnOK (CDlgPrintDlg)

- [Drucken und Druckvorschau](#)

OnOpenDocument (CSerialDoc)

- [Dateizugriff](#)

OnPaint

- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)

OnPrepareDC (CPrintItView)

- [Drucken und Druckvorschau](#)

OnPreparePrinting

- [Drucken und Druckvorschau](#)

OnPreparePrinting (CPrintItView)

- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

OnPrint (CPrintItView)

- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

OnRButtonDown (CMausDlg)

- [Antworten](#)

OnReceive (CSockDlg)

- [Internet und Netzwerke](#)

OnRecordDelete (CDBOdbcView)

- [Daten aus ODBC- Datenbanken abrufen](#)

OnRecordMove (CDBOdbcView)

- [Antworten](#)

OnRecordNew (CDBOdbcView)

- [Daten aus ODBC- Datenbanken abrufen](#)

OnRSelection

- [Bilder, Zeichnungen und Bitmaps](#)

OnRType (CSockDlg)

- [Internet und Netzwerke](#)

OnRunpgm

- [Steuerelemente](#)

OnRunpgm (CTag2Dlg)

- [Antworten](#)

OnSelchangeLfont

- [Text und Schriften](#)

OnSelchangeLfont (CTag7Dlg)

- [Antworten](#)

OnSelChangeWidth (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnSetCursor

- [Maus und Tastatur](#)

OnShwmsg

- [Steuerelemente](#)

- [Steuerelemente](#)

OnSize (CPaintDlg)

- [Antworten](#)

OnStarttime

- [Timer](#)

- [Timer](#)

OnStarttime (CTimerDlg)

- [Antworten](#)

OnStoptimer

- [Timer](#)

- [Timer](#)

OnStoptimer (CTimerDlg)

- [Antworten](#)

OnTimer

- [Timer](#)

- [Timer](#)

OnUpdateColorBlack (CTag10Doc)

- [SDI-Anwendungen](#)

OnUpdateColorBlue (CTag10Doc)

- [SDI-Anwendungen](#)

OnUpdateDataFirst (CDBAdoDoc)

- [Antworten](#)

OnUpdateDataLast (CDBAdoDoc)

- [Antworten](#)

OnUpdateIndicatorColor (CToolBarDoc)

- [Symbolleisten und Statusleisten](#)

OnUpdateIndicatorWidth (CToolbarDoc)

- [Antworten](#)

OnUpdateViewColorbar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnUpdateViewStop (CWebBrowseView)

- [Antworten](#)

OnUpdateWidthThin (CTag10Doc)

- [Antworten](#)

OnUpdateWidthUI (CTag11Doc)

- [Antworten](#)

OnUpdateWidthVthin (CTag10Doc)

- [Antworten](#)

OnViewColorbar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnViewRefresh (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnViewStop (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnWidthCommand (CTag11Doc)

- [Antworten](#)

OnWidthThin (CTag10Doc)

- [Antworten](#)

OnWidthVthin (CTag10Doc)

- [Antworten](#)

OnWidthVthin (CToolbarDoc)

- [Symbolleisten und Statusleisten](#)

OnYesno

- [Antworten](#)

OnYesno (CDialogDlg)

- [Antworten](#)

OnYesnocancel

- [Dialogfelder](#)

Open

- [Daten aus ODBC- Datenbanken abrufen](#)

Parameter prüfen

- [Debugger und Profiler](#)

ParseDateTime

- [MFC-Hilfsklassen](#)

PopulateView (CSerialView)

- [Dateizugriff](#)

PrintAll

- [Drucken und Druckvorschau](#)

PrintCollate

- [Drucken und Druckvorschau](#)

printf

- [MFC-Hilfsklassen](#)

PrintRange

- [Drucken und Druckvorschau](#)

PrintSelection

- [Drucken und Druckvorschau](#)

Prototyp

- [C++ im Überblick](#)

prozeßexterne Aufrufe

- [Debugger und Profiler](#)

RandomStringValue

- [ActiveX-Steuerelemente](#)

RecalcLayout

- [Symbolleisten und Statusleisten](#)

Rectangle

- [Bilder, Zeichnungen und Bitmaps](#)

Refresh

- [Funktionen für Webbrowser](#)

RefreshBoundData (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

ReleaseCapture

- [SDI-Anwendungen](#)

Remote Procedure Call (RPC)

- [Debugger und Profiler](#)

RemoveAll

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

RemoveAt

- [MFC-Hilfsklassen](#)
- [MFC-Hilfsklassen](#)

RemoveKey

- [MFC-Hilfsklassen](#)

Requery

- [Daten aus ODBC- Datenbanken abrufen](#)

ResumeThread

- [Multitasking](#)

ReverseFind

- [MFC-Hilfsklassen](#)

RGB

- [Maus und Tastatur](#)

Right

- [MFC-Hilfsklassen](#)

Rückgabewerte

- [C++ im Überblick](#)

SaveDrawing (CSquiggleCtrl)

- [Antworten](#)

SelectObject

- [SDI-Anwendungen](#)

Serialize

- [Dateizugriff](#)

Serialize (CLine)

- [SDI-Anwendungen](#)
- [SDI-Anwendungen](#)
- [Antworten](#)

Serialize (CModArt)

- [Eigene Klassen und Module](#)

Serialize (CSerialDoc)

- [Dateizugriff](#)

Serialize (CTag10Doc)

- [SDI-Anwendungen](#)

Serialize (CTestAppDoc)

- [Eigene Klassen und Module](#)

- [DLLs](#)

Set

- [Dateizugriff](#)
- [Eigene ActiveX- Steuerelemente](#)

SetAbsolutePosition

- [Daten aus ODBC- Datenbanken abrufen](#)

SetAddress (CMainFrame)

- [Funktionen für Webbrowser](#)

SetAt

- [MFC-Hilfsklassen](#)

SetButtonStyle

- [Symbolleisten und Statusleisten](#)

SetButtonText

- [Symbolleisten und Statusleisten](#)

SetCapture

- [SDI-Anwendungen](#)

SetCheck

- [SDI-Anwendungen](#)

SetDateTime

- [MFC-Hilfsklassen](#)

SetIndicators

- [Symbolleisten und Statusleisten](#)

SetLength (CModArt)

- [Eigene ActiveX- Steuerelemente](#)

SetMaxPage

- [Drucken und Druckvorschau](#)

SetMinPage

- [Drucken und Druckvorschau](#)

SetModifiedFlag

- [SDI-Anwendungen](#)

SetMyFont

- [Text und Schriften](#)
- [Text und Schriften](#)

SetMyFont (CTag7Dlg)

- [Antworten](#)

SetPaneText

- [Symbolleisten und Statusleisten](#)

SetParent (CMySocket)

- [Internet und Netzwerke](#)

SetPixel

- [Maus und Tastatur](#)

SetRect (CModArt)

- [Eigene Klassen und Module](#)

SetSquiggleLength (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

SetTimer

- [Timer](#)

SetViewportExt

- [Bilder, Zeichnungen und Bitmaps](#)

SetViewportOrg

- [Multitasking](#)

SetWidth (CToolBarDoc)

- [Symbolleisten und Statusleisten](#)

SetWindowExt

- [Bilder, Zeichnungen und Bitmaps](#)

SetWindowText

- [Symbolleisten und Statusleisten](#)

ShellExecute

- [Steuerelemente](#)

ShowBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

ShowControlBar

- [Symbolleisten und Statusleisten](#)

ShowWindow

- [Steuerelemente](#)

sprintf

- [MFC-Hilfsklassen](#)

StartDoc

- [Drucken und Druckvorschau](#)

StartPage

- [Drucken und Druckvorschau](#)

StretchBlt

- [Bilder, Zeichnungen und Bitmaps](#)

Strings vergleichen

- [MFC-Hilfsklassen](#)

SubtractRect

- [MFC-Hilfsklassen](#)

suchen

- [Steuerelemente](#)

SuspendSpinner (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)
- [Antworten](#)

Systemzeit

- [MFC-Hilfsklassen](#)

Teilstrings

- [MFC-Hilfsklassen](#)

ThreadFunc (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)

ThrowErrno

- [Ausnahmebehandlung](#)

ThrowOsError

- [Ausnahmebehandlung](#)

TopLeft

- [MFC-Hilfsklassen](#)

TrackPopupMenu

- [Menüs](#)

TrimLeft

- [MFC-Hilfsklassen](#)

TrimRight

- [MFC-Hilfsklassen](#)

überladen

- [C++ im Überblick](#)

überschreiben

- [Drucken und Druckvorschau](#)

UnionRect

- [MFC-Hilfsklassen](#)

Update

- [Daten aus ODBC- Datenbanken abrufen](#)

UpdateBoundData (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

UpdateColorbar (CToolbarDoc)

- [Symbolleisten und Statusleisten](#)

UpdateData

- [Steuerelemente](#)

UpdateWidthCB (CMainFrame)

- [Symbolleisten und Statusleisten](#)

virtuelle

- [Drucken und Druckvorschau](#)

Width

- [MFC-Hilfsklassen](#)

WinExec

- [Steuerelemente](#)

Funktionen Siehe auch Member-Funktionen

- [Steuerelemente](#)

Funktionstasten

- [Menüs](#)

FW_XXXX-Konstanten

- [Text und Schriften](#)

G

GDI

- [Bilder, Zeichnungen und Bitmaps](#)

16-Bit-Versionen

- [Multitasking](#)

Objekte mit OnBeginPrinting hinzufügen

- [Drucken und Druckvorschau](#)

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Stifte

- [Bilder, Zeichnungen und Bitmaps](#)

Genauigkeit

Schriften

- [Text und Schriften](#)

GenerateError (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

GenID

- [ActiveX-Steuerelemente](#)

GenNewDrawing (CSquiggleCtrl)

- [Antworten](#)

Geräte

DevMode

- [Drucken und Druckvorschau](#)

Grafiken

- [Bilder, Zeichnungen und Bitmaps](#)

Höhen-/Seitenverhältnis

- [Drucken und Druckvorschau](#)

Gerätekontexte

- [Maus und Tastatur](#)
- [Text und Schriften](#)
- [Bilder, Zeichnungen und Bitmaps](#)
- [Multitasking](#)

anpassen beim Drucken

- [Drucken und Druckvorschau](#)

CClientDC

- [Text und Schriften](#)

Drucken

- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

kompatible

- [Bilder, Zeichnungen und Bitmaps](#)

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Stifte

- [Maus und Tastatur](#)

get

- [C++ im Überblick](#)

GetAt

Arrayklassen

- [MFC-Hilfsklassen](#)

Llistenklassen

- [MFC-Hilfsklassen](#)

Zeichenfolgen

■ [MFC-Hilfsklassen](#)

GetBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

GetCapture

- [SDI-Anwendungen](#)

GetColor (CTag10Doc)

- [SDI-Anwendungen](#)

GetCopies

- [Drucken und Druckvorschau](#)

GetCount

- [MFC-Hilfsklassen](#)

GetCurRecordNbr (CSerialDoc)

- [Dateizugriff](#)

GetCurrentTime

- [MFC-Hilfsklassen](#)

GetDay

- [MFC-Hilfsklassen](#)

GetDayOfWeek

- [MFC-Hilfsklassen](#)

GetDayOfYear

- [MFC-Hilfsklassen](#)

GetDC

- [Multitasking](#)

GetDeviceCaps

- [Drucken und Druckvorschau](#)

GetDlgItem

- [Steuerelemente](#)

GetDocument

- [SDI-Anwendungen](#)

GetDrawing (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

GetDriverName

- [Drucken und Druckvorschau](#)

GetFileName

- [Antworten](#)

GetFirstPosition

- [MFC-Hilfsklassen](#)

GetFirstRecord (CSerialDoc)

- [Dateizugriff](#)

GetFirstViewPosition

- [Symbolleisten und Statusleisten](#)

GetFixedRows

- [ActiveX-Steuerelemente](#)

GetFromPage

- [Drucken und Druckvorschau](#)

GetHeadPosition

- [MFC-Hilfsklassen](#)

GetHour

- [MFC-Hilfsklassen](#)

GetItemRect

- [Symbolleisten und Statusleisten](#)

GetKeyState

- [Maus und Tastatur](#)

GetLastError

- [Internet und Netzwerke](#)

GetLastRecord (CSerialDoc)

- [Dateizugriff](#)

GetLength (CModArt)

- [Eigene ActiveX- Steuerelemente](#)

GetLine (CTag10Doc)

- [SDI-Anwendungen](#)

GetLineCount (CTag10Doc)

- [SDI-Anwendungen](#)

GetLocationURL

- [Funktionen für Webbrowser](#)

GetMaxID (CDBOdbcSet)

- [Daten aus ODBC- Datenbanken abrufen](#)

GetMinute

- [MFC-Hilfsklassen](#)

GetMonth

- [MFC-Hilfsklassen](#)

GetMouseCol

- [ActiveX-Steuerelemente](#)

GetMouseRow

- [ActiveX-Steuerelemente](#)

GetNextAssoc

- [MFC-Hilfsklassen](#)

GetNextRecord (CSerialDoc)

- [Dateizugriff](#)

GetNextRecord (CSerialView)

- [Dateizugriff](#)

GetNextView

- [Symbolleisten und Statusleisten](#)

GetPathName

- [Antworten](#)

GetPortName

- [Drucken und Druckvorschau](#)

GetPrevRecord (CSerialDoc)

- [Dateizugriff](#)

GetPrevRecord (CSerialView)

- [Dateizugriff](#)

GetPrinterDC

- [Drucken und Druckvorschau](#)

GetRecordCount

- [Daten aus ODBC- Datenbanken abrufen](#)

GetRecSet (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

GetRows

- [ActiveX-Steuerelemente](#)

GetSecond

- [MFC-Hilfsklassen](#)

GetSize

- [SDI-Anwendungen](#)

GetSquiggleLength (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

GetTailPosition

- [MFC-Hilfsklassen](#)

GetToPage

- [Drucken und Druckvorschau](#)

GetTopLevelFrame

- [Symbolleisten und Statusleisten](#)

GetTotalRecords (CSerialDoc)

- [Dateizugriff](#)

GetUpperBound

- [MFC-Hilfsklassen](#)

GetViewWnd

- [Multitasking](#)

GetWidth (CTag10Doc)

- [Antworten](#)

GetWindowText

- [Funktionen für Webbrowser](#)

GetYear

- [MFC-Hilfsklassen](#)

Gewichtung

- [Text und Schriften](#)

Gleichheitszeichen

- [Steuerelemente](#)

GoBack

- [Funktionen für Webbrowser](#)

GoForward

- [Funktionen für Webbrowser](#)

GoHome

- [Funktionen für Webbrowser](#)

GoSearch

- [Funktionen für Webbrowser](#)

Grafiken

- [Bilder, Zeichnungen und Bitmaps](#)

Abbildungsmodi

- [Bilder, Zeichnungen und Bitmaps](#)

Funktionen erstellen

- [Bilder, Zeichnungen und Bitmaps](#)

GDI

- [Bilder, Zeichnungen und Bitmaps](#)

Gerätekontexte

- [Bilder, Zeichnungen und Bitmaps](#)

Koordinatensysteme

- [Bilder, Zeichnungen und Bitmaps](#)

OnPaint

- [Bilder, Zeichnungen und Bitmaps](#)

Viewport

- [Multitasking](#)

Grafikkarten

- [Bilder, Zeichnungen und Bitmaps](#)

Größe

Arithmetik

- [MFC-Hilfsklassen](#)

Arrays dynamisch anpassen

- [SDI-Anwendungen](#)

Ausschnitte in Statusleisten

- [Symbolleisten und Statusleisten](#)

Datentypen

- [C++ im Überblick](#)

Kombinationsfelder

- [Steuerelemente](#)

Rechtecke

- [MFC-Hilfsklassen](#)

SIZE

- [MFC-Hilfsklassen](#)

Größer als

- [C++ im Überblick](#)

Größer als oder gleich

- [C++ im Überblick](#)

Groß-/Kleinschreibung

- [C++ im Überblick](#)
- [C++ im Überblick](#)

CompareNoCase

- [MFC-Hilfsklassen](#)

in Großbuchstaben konvertieren

- [Steuerelemente](#)

MakeUpper

- [Steuerelemente](#)

Gruppenfelder

- [Dialogfelder](#)

Gültigkeitsbereiche

Klassen

- [C++ im Überblick](#)

Variablen

- [C++ im Überblick](#)

H

Haltepunkte

- [Debugger und Profiler](#)

Hash-Schlüssel

- [MFC-Hilfsklassen](#)

Header

vorkompilierter

- [C++ im Überblick](#)

Header-Dateien

einbinden

- [Dialogfelder](#)
- [C++ im Überblick](#)

Height

- [MFC-Hilfsklassen](#)

Hintergrund

- [Eigene ActiveX- Steuerelemente](#)

Hochformat

- [Drucken und Druckvorschau](#)

Höhe

Schrift

- [Text und Schriften](#)

Höhen-/Seitenverhältnis

- [Drucken und Druckvorschau](#)

hören

- [Internet und Netzwerke](#)

Hotkeys

- [Menüs](#)

HS_XXXX-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

I

I-Balken-Zeiger

- [Maus und Tastatur](#)

IDABORT

- [Dialogfelder](#)

IDC_STATIC

- [Steuerelemente](#)

IDCANCEL

- [Dialogfelder](#)

IDIGNORE

- [Dialogfelder](#)

IDispatch

- [ActiveX-Steuerelemente](#)

Ausnahmen

- [Ausnahmebehandlung](#)

Client

- [ActiveX-Steuerelemente](#)

Invoke

- [ActiveX-Steuerelemente](#)

Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

IDNO

- [Dialogfelder](#)

IDOK

- [Dialogfelder](#)
- [Funktionen für Webbrowser](#)

IDR_MAINFRAME

- [MDI-Anwendungen](#)

IDRETRY

- [Dialogfelder](#)

IDs

Timer

- [Timer](#)

Umlaute

- [Menüs](#)

IDYES

- [Dialogfelder](#)

if

- [C++ im Überblick](#)

IMPLEMENT_DYNAMIC

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

IMPLEMENT_SERIAL

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

Include

- [C++ im Überblick](#)

Indexoperator

- [MFC-Hilfsklassen](#)
Tabellenklassen
 - [MFC-Hilfsklassen](#)

Indirektionsoperator

- [C++ im Überblick](#)

InflateRect

- [MFC-Hilfsklassen](#)

Infoleisten

- [Funktionen für Webbrowser](#)

Info-Symbol

- [Dialogfelder](#)

initialisieren

- Klassenelemente
 - [C++ im Überblick](#)

InitSpinners (CTaskingDoc)

- [Multitasking](#)

Inline

- Funktionen
 - [Dateizugriff](#)

Inline-Funktionen

- [Multitasking](#)
- [C++ im Überblick](#)

Innenbereich

- [Bilder, Zeichnungen und Bitmaps](#)

InsertAfter

- [MFC-Hilfsklassen](#)

InsertAt

- [MFC-Hilfsklassen](#)

InsertBefore

- [MFC-Hilfsklassen](#)

Internet

- [Internet und Netzwerke](#)

Internet Explorer

ActiveX-Modell

- [Funktionen für Webbrowser](#)

Inforeisten

- [Funktionen für Webbrowser](#)

InterSectRect

- [MFC-Hilfsklassen](#)

Intervalle

- [Timer](#)

Invalidate

- [Bilder, Zeichnungen und Bitmaps](#)

Invoke

- [ActiveX-Steuerelemente](#)

IsBOF

- [Daten aus ODBC- Datenbanken abrufen](#)

IsEmpty

- [MFC-Hilfsklassen](#)

IsEOF

- [Daten aus ODBC- Datenbanken abrufen](#)

IsLoading

- [Dateizugriff](#)

IsStoring

- [Dateizugriff](#)

J

JavaScript

- [Funktionen für Webbrowser](#)

JScript

- [Funktionen für Webbrowser](#)

Just-In-Time-Debugging

- [Debugger und Profiler](#)

K

Kapselung

- [Eigene Klassen und Module](#)
- [C++ im Überblick](#)

Klassen

abgeleitete

- [C++ im Überblick](#)
- [Debugger und Profiler](#)

Ableitung

- [C++ im Überblick](#)

AFX_EXT_CLASS

- [DLLs](#)

allgemeine

- [SDI-Anwendungen](#)
- [Eigene Klassen und Module](#)

Auflistungs-

- [MFC-Hilfsklassen](#)

Aufzählungen

- [C++ im Überblick](#)

Ausnahme-

- [Ausnahmebehandlung](#)

Basisklassen

- [C++ im Überblick](#)

benutzerdefinierte Auflistungs-

- [MFC-Hilfsklassen](#)

benutzerdefinierte Datensatz-

- [Datenbanken per ADO bearbeiten](#)

CADORecordBinding

- [Datenbanken per ADO bearbeiten](#)

CArchive

- [Dateizugriff](#)

CAsyncSocket

- [Internet und Netzwerke](#)

CBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

CBrush

- [Bilder, Zeichnungen und Bitmaps](#)

CByteArray

- [SDI-Anwendungen](#)

CChildFrame

- [MDI-Anwendungen](#)

CClientDC

- [Maus und Tastatur](#)
- [Text und Schriften](#)

CCmdTarget

- [Maus und Tastatur](#)

CColorDialog

- [Dialogfelder](#)

CDAOException

- [Ausnahmebehandlung](#)

CDatabase

- [Daten aus ODBC- Datenbanken abrufen](#)

CDC

- [Bilder, Zeichnungen und Bitmaps](#)

CDialog

- [Steuerelemente](#)

CDocument

- [SDI-Anwendungen](#)

CEditView

- [SDI-Anwendungen](#)

CEvent

- [Multitasking](#)

CException

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

CFile

- [Dateizugriff](#)

CFileDialog

- [Dialogfelder](#)

CFileException

- [Ausnahmebehandlung](#)

CFindReplaceDialog

- [Dialogfelder](#)

CFont

- [Text und Schriften](#)

CFontDialog

- [Dialogfelder](#)

CFormView

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

CFrameView

- [SDI-Anwendungen](#)

CHtmlView

- [SDI-Anwendungen](#)
- [Funktionen für Webbrowser](#)

CLine

- [SDI-Anwendungen](#)

CListView

- [SDI-Anwendungen](#)

CMainFrame

- [MDI-Anwendungen](#)

CMDIChildWnd

- [MDI-Anwendungen](#)

CMDIFrameWnd

- [MDI-Anwendungen](#)

CMutex

- [Multitasking](#)

CNotSupportedException

- [Ausnahmebehandlung](#)

CObArray

- [SDI-Anwendungen](#)

COleDateTime

- [MFC-Hilfsklassen](#)

COleDispatchException

- [Ausnahmebehandlung](#)

CPageSetupDialog

- [Dialogfelder](#)

CPen

- [Bilder, Zeichnungen und Bitmaps](#)

CPerson

- [Dateizugriff](#)

CPoint

- [SDI-Anwendungen](#)

- [MFC-Hilfsklassen](#)

CPrintDialog

- [Dialogfelder](#)

CPtrArray

- [SDI-Anwendungen](#)

CRect

- [MFC-Hilfsklassen](#)

CRichEditView

- [SDI-Anwendungen](#)

CScrollView

- [SDI-Anwendungen](#)

CSemaphore

- [Multitasking](#)

CSize

- [MFC-Hilfsklassen](#)

CString

- [MFC-Hilfsklassen](#)

CStringArray

- [SDI-Anwendungen](#)

CTime

- [Timer](#)

- [MFC-Hilfsklassen](#)

CTimeSpan

- [MFC-Hilfsklassen](#)

CToolBar

- [Symbolleisten und Statusleisten](#)

CTreeView

- [SDI-Anwendungen](#)

CUserException

- [Ausnahmebehandlung](#)

CView

- [SDI-Anwendungen](#)

CWaitCursor

- [Maus und Tastatur](#)

CWinApp

- [SDI-Anwendungen](#)

CWinThread

- [Multitasking](#)

CWnd

- [Steuerelemente](#)

CWordArray

- [SDI-Anwendungen](#)

Datum/Zeit

- [MFC-Hilfsklassen](#)

definieren

- [Eigene Klassen und Module](#)

Definitionen

- [C++ im Überblick](#)

Deklarationen

- [C++ im Überblick](#)

Destruktor

- [C++ im Überblick](#)

Einführung

- [C++ im Überblick](#)

Elemente initialisieren

- [C++ im Überblick](#)

entwerfen

- [Eigene Klassen und Module](#)

exportieren

- [DLLs](#)

Formular-

- [Eigene Klassen und Module](#)

Friend

- [C++ im Überblick](#)

Gerätekontexte

- [Bilder, Zeichnungen und Bitmaps](#)

Gültigkeitsbereiche

- [C++ im Überblick](#)

hinzufügen

- [Dialogfelder](#)

Kapselung

- [Eigene Klassen und Module](#)
- [C++ im Überblick](#)

Komponenten

- [C++ im Überblick](#)

Konstruktor

- [C++ im Überblick](#)

Koordinaten

- [MFC-Hilfsklassen](#)

Linien

- [SDI-Anwendungen](#)

Listen-

- [MFC-Hilfsklassen](#)

Map-

- [MFC-Hilfsklassen](#)

Member-Variablen

- [Steuerelemente](#)
- [C++ im Überblick](#)

Methoden

- [C++ im Überblick](#)

MFC

- [Eigene Klassen und Module](#)

neue Klasse erstellen

- [Dialogfelder](#)

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Platzhalter

- [Bilder, Zeichnungen und Bitmaps](#)

SDI-Anwendungen

- [SDI-Anwendungen](#)

serialisierbar machen

- [SDI-Anwendungen](#)

serialisierbare erstellen

- [Dateizugriff](#)

serialisierbare implementieren

- [Dateizugriff](#)

serialisieren

- [Dateizugriff](#)

Standardkonstruktor

- [C++ im Überblick](#)

Tabellen-

- [MFC-Hilfsklassen](#)

Typ prüfen

- [Debugger und Profiler](#)

Typen in Visual C++

- [Eigene Klassen und Module](#)

Typsicherheit

- [MFC-Hilfsklassen](#)

Vererbung

- [Eigene Klassen und Module](#)

verschachtelte

- [C++ im Überblick](#)

von CAsyncSocket ableiten

- [Internet und Netzwerke](#)

vorhandene hinzufügen

- [Menüs](#)

Vorteile

- [C++ im Überblick](#)

Zeichenfolgen

- [MFC-Hilfsklassen](#)

Klassenansicht

- [Erste Schritte mit Visual C++](#)

Klassen-Assistent

- [Erste Schritte mit Visual C++](#)

Code bearbeiten

- [Erste Schritte mit Visual C++](#)

Datenbankfelder mit Steuerelementen verbinden

- [Daten aus ODBC- Datenbanken abrufen](#)

Funktionen suchen

- [Steuerelemente](#)

Hinzufügen einer Klasse

- [Dialogfelder](#)

kann Header-Datei nicht finden

- [SDI-Anwendungen](#)

Member-Variable hinzufügen

- [Timer](#)

Nachrichtenzuordnungstabellen

- [MDI-Anwendungen](#)

Steuerelementereignisse ermitteln

- [ActiveX-Steuerelemente](#)

Timer-Nachrichten

- [Timer](#)

Variablen für Ansichtsklasse

- [Dateizugriff](#)

Variablen mit Steuerelementen verbinden

- [Steuerelemente](#)

Kleiner als

- [C++ im Überblick](#)

Kleiner als oder gleich

- [C++ im Überblick](#)

Kombinationsfelder

- [Steuerelemente](#)

aktualisieren

- [Symbolleisten und Statusleisten](#)

Auswahl ermitteln

- [Symbolleisten und Statusleisten](#)

Daten

- [Steuerelemente](#)

Einträge sortieren

- [Symbolleisten und Statusleisten](#)

Ereignisse

- [Symbolleisten und Statusleisten](#)

füllen

- [Symbolleisten und Statusleisten](#)

für Symbolleisten erstellen

- [Symbolleisten und Statusleisten](#)

Größe festlegen

- [Steuerelemente](#)

harmonisch in Symbolleiste einfügen

- [Symbolleisten und Statusleisten](#)

in Symbolleisten

- [Symbolleisten und Statusleisten](#)

Position in Symbolleiste ermitteln

- [Symbolleisten und Statusleisten](#)

Projektressourcen manuell bearbeiten

- [Symbolleisten und Statusleisten](#)

Separatoren als Platzhalter

- [Symbolleisten und Statusleisten](#)

Stile

- [Symbolleisten und Statusleisten](#)

Zeichenfolgentabelle

- [Symbolleisten und Statusleisten](#)

Kommentare

- [C++ im Überblick](#)

Quellcodedateien

- [Erste Schritte mit Visual C++](#)

Kompilieren

Anwendung

- [Erste Schritte mit Visual C++](#)

Komponenten

- [Eigene ActiveX- Steuerelemente](#)

Konfigurationen

aktive

- [C++ im Überblick](#)

Debug

- [Debugger und Profiler](#)

Release

- [Debugger und Profiler](#)

Konsolenanwendungen

- [C++ im Überblick](#)

Konstruktor

- [C++ im Überblick](#)

Aufrufreihenfolge

- [C++ im Überblick](#)

der Basisklasse

- [C++ im Überblick](#)

Regeln

- [C++ im Überblick](#)

Variablen initialisieren

- [SDI-Anwendungen](#)

Kontextmenüs

- [Menüs](#)
- [Menüs](#)
- [MDI-Anwendungen](#)

Nachrichten

- [Menüs](#)

Position

- [Menüs](#)

TPM_xxxx-Konstanten

- [Menüs](#)

TrackPopupMenu

- [Menüs](#)

WM_CONTEXTMENU

- [Menüs](#)
- [MDI-Anwendungen](#)

WM_RBUTTONDOWN

- [Menüs](#)

Kontrollhäkchen

- [SDI-Anwendungen](#)

Kontrollkästchen

- [Steuerelemente](#)
 - initialisieren
 - [Steuerelemente](#)
 - Zeiger auf Variable
 - [Multitasking](#)

Konventionen

Menüs

- [Menüs](#)

Namen für globale/lokale Variablen

- [C++ im Überblick](#)

Namen für Member-Variablen

- [Steuerelemente](#)

konvertieren

Großbuchstaben

- [Steuerelemente](#)

kooperatives Multitasking

- [Multitasking](#)

Koordinaten

addieren/subtrahieren

- [MFC-Hilfsklassen](#)

Klassen

- [MFC-Hilfsklassen](#)

normalisieren

- [MFC-Hilfsklassen](#)

Koordinatensysteme

- [Bilder, Zeichnungen und Bitmaps](#)
- [Eigene Klassen und Module](#)

Kopien

Drucken

- [Drucken und Druckvorschau](#)

Kreise

- [Bilder, Zeichnungen und Bitmaps](#)

kritische Abschnitte

- [Multitasking](#)

kursiv

- [Text und Schriften](#)

L

Ländereinstellungen

- [MFC-Hilfsklassen](#)

Laufwerksbezeichnung

Dialogfeld Öffnen

- [Dialogfelder](#)

Laufzeitlizenzen

- [Eigene ActiveX- Steuerelemente](#)

Layout

Tabulator-Reihenfolge

- [Steuerelemente](#)

LBN_SELCHANGE

- [Text und Schriften](#)

Leerlaufverarbeitung

- [Multitasking](#)

ständige

- [Multitasking](#)

Left

- [MFC-Hilfsklassen](#)

LIB-Dateien

- [DLLs](#)

LineTo

- [Maus und Tastatur](#)

Linien

abrufen

- [SDI-Anwendungen](#)

Anzahl ermitteln

- [SDI-Anwendungen](#)

CPen

- [Bilder, Zeichnungen und Bitmaps](#)

Farbe auswählen

- [SDI-Anwendungen](#)

Stile

- [Bilder, Zeichnungen und Bitmaps](#)

zeichnen

- [Bilder, Zeichnungen und Bitmaps](#)

Linienklasse

- [SDI-Anwendungen](#)

Listen

- [Internet und Netzwerke](#)

verkettete

- [MFC-Hilfsklassen](#)

Listenfelder

LBN_SELCHANGE

- [Text und Schriften](#)

Nachrichten

- [Text und Schriften](#)

Listenklassen

- [MFC-Hilfsklassen](#)

AddHead

- [MFC-Hilfsklassen](#)

AddTail

- [MFC-Hilfsklassen](#)

benutzerdefinierte

- [MFC-Hilfsklassen](#)

COBList

- [MFC-Hilfsklassen](#)

CPtrList

- [MFC-Hilfsklassen](#)

CStringList

- [MFC-Hilfsklassen](#)

Elemente

- [MFC-Hilfsklassen](#)

Find

- [MFC-Hilfsklassen](#)

GetAt

- [MFC-Hilfsklassen](#)

GetHeadPosition

- [MFC-Hilfsklassen](#)

GetTailPosition

- [MFC-Hilfsklassen](#)

InsertAfter

- [MFC-Hilfsklassen](#)

InsertBefore

- [MFC-Hilfsklassen](#)

POSITION

- [MFC-Hilfsklassen](#)

Position ermitteln

- [MFC-Hilfsklassen](#)

RemoveAt

- [MFC-Hilfsklassen](#)

SetAt

- [MFC-Hilfsklassen](#)

Listings

allgemeine Hinweise

- [Steuerelemente](#)

B.11

Die modifizierte Funktion Serialize

- [Antworten](#)

B.12

Die modifizierte Funktion OnNewDocument

- [Antworten](#)

B.13

Die modifizierte Funktion AddLine

■ [Antworten](#)

B.14

Die Funktion GetWidth

■ [Antworten](#)

B.15

Die Funktionen des Menübefehls Sehr dünn

■ [Antworten](#)

B.16

Die Funktionen des Menüs Dünn

■ [Antworten](#)

B.17

Die Deklarationen der Behandlungsroutinen in Tag11Doc.h

■ [Antworten](#)

B.18

Die Einträge der Behandlungsroutinen in der Nachrichtenzuordnungstabelle in Tag11Doc.cpp

■ [Antworten](#)

B.19

Die Behandlungsroutinen für die Nachrichten des Menüs Breite

■ [Antworten](#)

B.20

Der modifizierte Konstruktor von CPerson

■ [Antworten](#)

B.21

Die modifizierte Deklaration der Klasse CPerson

■ [Antworten](#)

B.22

Die modifizierte Funktion Serialize der Klasse CPerson

■ [Antworten](#)

B.23

Die modifizierte Funktion PopulateView der Klasse CSerializeView

■ [Antworten](#)

B.24

Die Funktion OnSex der Klasse CSerializeView

■ [Antworten](#)

B.25

Die Funktion OnRecordMove der Klasse CDbOdbcView

■ [Antworten](#)

B.26

Die Include-Anweisungen der Klasse CDbOdbcView

■ [Antworten](#)

B.27

Die Funktion OnUpdateDataFirst der Klasse CDbAdoDoc

■ [Antworten](#)

B.28

Die Funktion OnUpdateDataLast der Klasse CDbAdoDoc

■ [Antworten](#)

B.29

Die Include-Anweisungen und die Farbtabelle der Klasse CModArt

■ [Antworten](#)

B.30

Die Include-Anweisungen der Klasse CModArt

■ [Antworten](#)

B.31

Der Konstruktor der Klasse CLine

■ [Antworten](#)

B.32

Die modifizierte Funktion OnIdle der Klasse CTaskingApp

■ [Antworten](#)

B.33

Die modifizierte Funktion SuspendSpinner der Klasse CTaskingDoc

■ [Antworten](#)

B.34

Die Funktion GenNewDrawing der Klasse CSquiggleCtrl

■ [Antworten](#)

B.35

Die Funktion SaveDrawing der Klasse CSquiggleCtrl

■ [Antworten](#)

B.36

Die modifizierte Funktion OnAccept der Klasse CSockDlg

■ [Antworten](#)

B.37

Die Funktion OnGoSearch der Klasse CWebBrowseView

■ [Antworten](#)

B.38

Die Funktion OnGoStart der Klasse CWebBrowseView

- [Antworten](#)

B.39

Die Funktion OnUpdateViewStop der Klasse CWebBrowseView

- [Antworten](#)

Zeilennummern

- [C++ im Überblick](#)

Lizenzen

ActiveX-Steuerelemente

- [ActiveX-Steuerelemente](#)

LoadBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

LoadCursor

- [Maus und Tastatur](#)

LoadData

- [ActiveX-Steuerelemente](#)

LoadDrawing (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)
- [Eigene ActiveX- Steuerelemente](#)

LoadImage

- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)

LoadStandardCursor

- [Maus und Tastatur](#)

LoadToolBar

- [Symbolleisten und Statusleisten](#)

LOGFONT

- [Text und Schriften](#)

Lookup

- [MFC-Hilfsklassen](#)

loopback

- [Internet und Netzwerke](#)

LPARAM

- [Text und Schriften](#)

LPCTSTR

- [Eigene ActiveX- Steuerelemente](#)

lpszFacename

- [Text und Schriften](#)

M

main

- [C++ im Überblick](#)

MAKEINTRESOURCE

- [Bilder, Zeichnungen und Bitmaps](#)

MakeLower

- [MFC-Hilfsklassen](#)

MakeReverse

- [MFC-Hilfsklassen](#)

MakeUpper

- [Steuerelemente](#)
- [MFC-Hilfsklassen](#)

Makros

ADO

- [Datenbanken per ADO bearbeiten](#)

ADO_FIXED_LENGTH_ENTRY

- [Datenbanken per ADO bearbeiten](#)

ADO_NUMERIC_ENTRY

- [Datenbanken per ADO bearbeiten](#)

ADO_VARIABLE_LENGTH_ENTRY

- [Datenbanken per ADO bearbeiten](#)

AFX_EXT_CLASS

- [DLLs](#)

AFX_MANAGE_STATE

- [DLLs](#)

ASSERT

- [Debugger und Profiler](#)
- [Debugger und Profiler](#)

ASSERT_KINDOF

- [Debugger und Profiler](#)

ASSERT_VALID

- [Debugger und Profiler](#)

BEGIN_ADO_BINDING

- [Datenbanken per ADO bearbeiten](#)

DECLARE_DYNAMIC

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

DECLARE_SERIAL

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

END_ADO_BINDING

- [Datenbanken per ADO bearbeiten](#)

IMPLEMENT_DYNAMIC

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

IMPLEMENT_SERIAL

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

MAKEINTRESOURCE

- [Bilder, Zeichnungen und Bitmaps](#)

ON_COMMAND

- [MDI-Anwendungen](#)
- [Funktionen für Webbrowser](#)

ON_UPDATE_COMMAND_UI

- [MDI-Anwendungen](#)

RGB

- [Maus und Tastatur](#)

RUNTIME_CLASS

- [Multitasking](#)
- [Ausnahmebehandlung](#)

TRACE

- [Debugger und Profiler](#)

VERIFY

- [Debugger und Profiler](#)

WinExec

- [Steuerelemente](#)

Makrosprachen

- [Funktionen für Webbrowser](#)

Map Siehe Tabellen

- [MFC-Hilfsklassen](#)

Marshaling

- [Eigene ActiveX- Steuerelemente](#)

Maskieren

Flags

- [Maus und Tastatur](#)

Maus

- [Maus und Tastatur](#)

Doppelklick

- [Maus und Tastatur](#)

Ereignisse

- [Maus und Tastatur](#)

Ereignisse auffangen

- [SDI-Anwendungen](#)

fangen

- [SDI-Anwendungen](#)

linke Maustaste

- [Maus und Tastatur](#)

Position

- [Maus und Tastatur](#)

Rad

- [Maus und Tastatur](#)

rechte Maustaste

- [Maus und Tastatur](#)

verschieben

- [Maus und Tastatur](#)

Zeiger ändern

- [Maus und Tastatur](#)

Mauszeiger

I-Balken

- [Maus und Tastatur](#)

laden

- [Maus und Tastatur](#)

Pfeil

- [Maus und Tastatur](#)

Sanduhr

- [Maus und Tastatur](#)

- [Maus und Tastatur](#)

WM_SETCURSOR

- [Maus und Tastatur](#)

Maximieren-Schaltfläche

- [Erste Schritte mit Visual C++](#)

MB_xxxx-Konstanten

- [Dialogfelder](#)

MDI-Anwendungen

- [MDI-Anwendungen](#)

Dokument/Ansicht-Architektur

- [MDI-Anwendungen](#)

Drucken

- [Drucken und Druckvorschau](#)

Kontextmenüs

- [MDI-Anwendungen](#)

Menüs

- [MDI-Anwendungen](#)

unterschiedliche Recordsets

- [Daten aus ODBC- Datenbanken abrufen](#)

Meldungsfelder

- [Dialogfelder](#)

Meldungsfelder Siehe auch MessageBox

- [Dialogfelder](#)

Member-Variablen

- [Steuerelemente](#)
- [C++ im Überblick](#)

Control (Typ)

- [ActiveX-Steuerelemente](#)

Eigenschaften

- [Eigene ActiveX- Steuerelemente](#)

hinzufügen

- [Steuerelemente](#)

Menü

Datensatz

- [Daten aus ODBC- Datenbanken abrufen](#)

Menübefehle

- [Menüs](#)

als Kontextmenü

- [Menüs](#)

Behandlungsroutinen

- [MDI-Anwendungen](#)

Behandlungsroutinen mehrfach verwenden

- [Menüs](#)

COMMAND

- [Menüs](#)

Funktionalität

- [Menüs](#)

ID-Nummern

- [SDI-Anwendungen](#)

Kontrollhäkchen

- [SDI-Anwendungen](#)

Konventionen

- [Menüs](#)

Nachrichten

- [Menüs](#)

Namen

- [Menüs](#)

Popup

- [Menüs](#)

Tastenkombination angeben

- [Menüs](#)

Trennlinien

- [Menüs](#)

UPDATE_COMMAND_UI

- [SDI-Anwendungen](#)

Menü-Editor

- [Menüs](#)

Menüleisten

MDI-Anwendungen

- [MDI-Anwendungen](#)

Menüs

- [Menüs](#)

Entwurf

- [Menüs](#)

erstellen

- [Menüs](#)

Farbe auswählen

- [SDI-Anwendungen](#)

für Kontextmenü erstellen

- [MDI-Anwendungen](#)

IDR_MAINFRAME

- [MDI-Anwendungen](#)

Kontextmenüs

- [Menüs](#)
- [Menüs](#)

Konventionen

- [Menüs](#)

MDI-Anwendungen

- [MDI-Anwendungen](#)

mit Dialogfeld verbinden

- [Menüs](#)

Numerierung

- [Menüs](#)

Popup

- [Menüs](#)
- [Menüs](#)

Pulldown

- [Menüs](#)

Statuszeilentext

- [Symbolleisten und Statusleisten](#)

Stile

- [Menüs](#)

Trennlinien

- [Menüs](#)

überlappende

- [Menüs](#)

Zugriffstasten

- [Menüs](#)

MessageBox

- [Erste Schritte mit Visual C++](#)
- [Dialogfelder](#)

IDABORT

- [Dialogfelder](#)

IDCANCEL

- [Dialogfelder](#)

IDIGNORE

- [Dialogfelder](#)

IDNO

- [Dialogfelder](#)

IDOK

- [Dialogfelder](#)

MB_XXXX-Konstanten

- [Dialogfelder](#)

Rückgabewerte

- [Dialogfelder](#)

Schaltflächen

- [Dialogfelder](#)

Symbole

- [Dialogfelder](#)

Methoden

- [C++ im Überblick](#)

Accept

- [Internet und Netzwerke](#)

benutzerdefinierte

- [Eigene ActiveX- Steuerelemente](#)

Close

- [Internet und Netzwerke](#)

Connect

- [Internet und Netzwerke](#)

get

- [C++ im Überblick](#)

GetLastError

- [Internet und Netzwerke](#)

Listen

- [Internet und Netzwerke](#)

Receive

- [Internet und Netzwerke](#)

Send

- [Internet und Netzwerke](#)

set

- [C++ im Überblick](#)

Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Methoden Siehe auch Funktionen

- [Eigene ActiveX- Steuerelemente](#)

MFC ActiveX-Steuerelement-Assistent

- [Eigene ActiveX- Steuerelemente](#)

MFC-Ablaufverfolgung

- [Debugger und Profiler](#)

MFC-Anwendungs-Assistent (dll)

- [DLLs](#)

MFC-Ausnahmetypen

- [Ausnahmebehandlung](#)

MFC-DLLs

erweiterte

- [DLLs](#)

MFC-Klassen

- [Eigene Klassen und Module](#)

DLLs

- [DLLs](#)

Mid

- [MFC-Hilfsklassen](#)

Minileiste erstellen

- [Erste Schritte mit Visual C++](#)

Minimieren-Schaltfläche

- [Erste Schritte mit Visual C++](#)

Mittelpunkt

Client-Bereich

- [MFC-Hilfsklassen](#)

Fenster

- [MFC-Hilfsklassen](#)

MK_LBUTTON

- [Antworten](#)

MM_xxxx-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

mnemonischer Code

- [Steuerelemente](#)

modale Fenster

- [Dialogfelder](#)

ModArtClearDrawing

- [DLLs](#)

ModArtDraw

- [DLLs](#)

ModArtInit

- [DLLs](#)

ModArtNewDrawing

- [DLLs](#)
- [DLLs](#)

ModArtSerialize

- [DLLs](#)

Moduldefinitionsdateien

- [DLLs](#)

Module

Bibliotheken

- [Eigene Klassen und Module](#)

Modulo-Division

- [C++ im Überblick](#)

Move

- [Daten aus ODBC- Datenbanken abrufen](#)

MoveFirst

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

MoveFirst (CdbAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

MoveLast

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

MoveNext

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

MovePrev

- [Daten aus ODBC- Datenbanken abrufen](#)

MovePrevious

- [Datenbanken per ADO bearbeiten](#)

MoveTo

- [Maus und Tastatur](#)

Multiplikation

- [C++ im Überblick](#)

Multitasking

- [Multitasking](#)

kooperatives

- [Multitasking](#)

OnIdle-Tasks

- [Multitasking](#)

präemptives

- [Multitasking](#)

Multithreading

Synchronisierungsklassen

- [Multitasking](#)

Muster

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Mutexe

- [Multitasking](#)

N

Nachrichten

angekommen

- [Internet und Netzwerke](#)

Behandlungsroutinen

- [Erste Schritte mit Visual C++](#)

BN_CLICKED

- [Steuerelemente](#)

COMMAND

- [Menüs](#)

Datenbank-

- [Debugger und Profiler](#)

DeleteContents

- [SDI-Anwendungen](#)
- [Datenbanken per ADO bearbeiten](#)

empfangen

- [Internet und Netzwerke](#)

EN_CHANGE

- [Timer](#)

Inhalt der Dokumentklasse löschen

- [SDI-Anwendungen](#)

Klicken

- [Erste Schritte mit Visual C++](#)

LBN_SELCHANGE

- [Text und Schriften](#)

Listenfelder

- [Text und Schriften](#)

MFC-Ablaufverfolgung

- [Debugger und Profiler](#)

mit NULL abtrennen

- [Internet und Netzwerke](#)

ON_CBN_SELCHANGE

- [Symbolleisten und Statusleisten](#)

OnIdle

- [Multitasking](#)

senden

- [Internet und Netzwerke](#)

Spy++

- [Debugger und Profiler](#)

Timer

- [Timer](#)

UPDATE_COMMAND_UI

- [SDI-Anwendungen](#)

Windows

- [Debugger und Profiler](#)

WM_CONTEXTMENU

- [Menüs](#)

WM_DESTROY

- [Multitasking](#)

WM_INITDIALOG

- [Steuerelemente](#)

WM_KEY

- [Menüs](#)

WM_KEYDOWN

- [Maus und Tastatur](#)

WM_KEYUP

- [Maus und Tastatur](#)

WM_LBUTTONDOWN

- [Maus und Tastatur](#)

WM_LBUTTONUP

- [Maus und Tastatur](#)

WM_MOUSEMOVE

- [Maus und Tastatur](#)

WM_MOUSEWHEEL

- [Maus und Tastatur](#)

WM_PAINT

- [Bilder, Zeichnungen und Bitmaps](#)

WM_RBUTTONDOWNBLCLK

- [Maus und Tastatur](#)

WM_RBUTTONDOWN

- [Maus und Tastatur](#)

WM_RBUTTONUP

- [Maus und Tastatur](#)

WM_SETCURSOR

- [Maus und Tastatur](#)

WM_TIMER

- [Timer](#)

Nachrichtenzuordnungstabellen

- [Erste Schritte mit Visual C++](#)

IDOK

- [Funktionen für Webbrowser](#)

Klassen-Assistent

- [MDI-Anwendungen](#)

ON_COMMAND

- [Funktionen für Webbrowser](#)

ON_COMMAND_RANGE

- [MDI-Anwendungen](#)

ON_UPDATE_COMMAND_UI_RANGE

- [MDI-Anwendungen](#)

Zugriffstasten

- [Menüs](#)

Namen

englisch/deutsch

- [Menüs](#)

Member-Variablen

- [Steuerelemente](#)

Menübefehle

- [Menüs](#)

Namensbereiche

umbenennen

- [Datenbanken per ADO bearbeiten](#)

Navigate

- [Funktionen für Webbrowser](#)

- [Funktionen für Webbrowser](#)

Navigation

Ansichtsklasse

- [Dateizugriff](#)

Dokumentklasse

- [Dateizugriff](#)

Recordset

- [Datenbanken per ADO bearbeiten](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)

Web

- [Funktionen für Webbrowser](#)

Navigationstypen

SQL_FETCH_ABSOLUTE

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_FIRST

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_LAST

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_NEXT

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_PRIOR

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_RELATIVE

- [Daten aus ODBC- Datenbanken abrufen](#)

nCharSet

- [Text und Schriften](#)

nClipPrecision

- [Text und Schriften](#)

nEscapement

- [Text und Schriften](#)

Netzwerke

- [Internet und Netzwerke](#)

Adressen

- [Internet und Netzwerke](#)
- [Internet und Netzwerke](#)

Anschlüsse

- [Internet und Netzwerke](#)

Computername

- [Internet und Netzwerke](#)

Funktionsweise

- [Internet und Netzwerke](#)

loopback

- [Internet und Netzwerke](#)

Sockets

- [Internet und Netzwerke](#)

TCP/IP

- [Internet und Netzwerke](#)

Neuzeichnen

abschalten

- [ActiveX-Steuerelemente](#)

Invalidate

- [Bilder, Zeichnungen und Bitmaps](#)

OnPaint

- [Bilder, Zeichnungen und Bitmaps](#)

new

- [Ausnahmebehandlung](#)
- [Ausnahmebehandlung](#)

NewDataSet (CSerialView)

- [Dateizugriff](#)

NewDrawing (CModArt)

- [Eigene Klassen und Module](#)
- [Eigene Klassen und Module](#)
- [Eigene ActiveX- Steuerelemente](#)

NewLine (CModArt)

- [Eigene Klassen und Module](#)
- [Eigene Klassen und Module](#)
- [DLLs](#)
- [DLLs](#)
- [Eigene ActiveX- Steuerelemente](#)

NEWTEXTMETRICEX

- [Text und Schriften](#)

nHeight

- [Text und Schriften](#)

Nicht gleich

- [C++ im Überblick](#)

nichtmodale Fenster

- [Dialogfelder](#)

NO_AFX_DAO_ERROR

- [Ausnahmebehandlung](#)

nOrientation

- [Text und Schriften](#)

Normalisieren

Koordinaten

- [MFC-Hilfsklassen](#)

NormalizeRect

- [MFC-Hilfsklassen](#)

NOT

- [C++ im Überblick](#)

nOutPrecision

- [Text und Schriften](#)

nPitchAndFamily

- [Text und Schriften](#)

nQuality

- [Text und Schriften](#)

Numerierung

- [Dialogfelder](#)

Untermenüs

- [Menüs](#)

nWeight

- [Text und Schriften](#)

nWidth

- [Text und Schriften](#)

O

Objektarray

zurücksetzen

- [SDI-Anwendungen](#)

- [Dateizugriff](#)

Objekte

_ConnectionPtr

- [Datenbanken per ADO bearbeiten](#)

Arrays

- [DLLs](#)

Command

- [Datenbanken per ADO bearbeiten](#)

Connection

- [Datenbanken per ADO bearbeiten](#)

Error

- [Datenbanken per ADO bearbeiten](#)

Field

- [Datenbanken per ADO bearbeiten](#)

Kapselung

- [Eigene Klassen und Module](#)

OLE/COM-Objektkatalog

- [Debugger und Profiler](#)

Parameter

- [Datenbanken per ADO bearbeiten](#)

prüfen

- [Debugger und Profiler](#)

Recordset

- [Datenbanken per ADO bearbeiten](#)

serialisierbar machen

- [Dateizugriff](#)

Vererbung

- [Eigene Klassen und Module](#)

objektorientierte Programmierung

- [Eigene Klassen und Module](#)

Kapselung

- [C++ im Überblick](#)

ODBC

- [Daten aus ODBC- Datenbanken abrufen](#)

- [Daten aus ODBC- Datenbanken abrufen](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

CDatabase

- [Daten aus ODBC- Datenbanken abrufen](#)

CRecordset

- [Daten aus ODBC- Datenbanken abrufen](#)

Datenquelle

- [Datenbanken per ADO bearbeiten](#)

öffentlich

- [C++ im Überblick](#)

OEM_CHARSET

- [Text und Schriften](#)

OEM-Zeichensatz

- [Text und Schriften](#)

OLE

COleDateTime

- [MFC-Hilfsklassen](#)

SCODE

- [Ausnahmebehandlung](#)

OLE 2.0

- [ActiveX-Steuerelemente](#)

OLE DB

- [Datenbanken per ADO bearbeiten](#)

OLE RPC-Debugging

- [Debugger und Profiler](#)

OLE/COM-Objektkatalog

- [Debugger und Profiler](#)

OLE-Ausnahmen

- [Ausnahmebehandlung](#)

ON_CBN_SELCHANGE

- [Symbolleisten und Statusleisten](#)

ON_COMMAND

- [MDI-Anwendungen](#)
- [Funktionen für Webbrowser](#)

ON_COMMAND_RANGE

- [MDI-Anwendungen](#)

ON_UPDATE_COMMAND_UI

- [MDI-Anwendungen](#)

ON_UPDATE_COMMAND_UI_RANGE

- [MDI-Anwendungen](#)

OnAccept

- [Internet und Netzwerke](#)

OnAccept (CMySocket)

- [Internet und Netzwerke](#)

OnAccept (CSockDlg)

- [Internet und Netzwerke](#)
- [Antworten](#)

OnBbitmap

- [Bilder, Zeichnungen und Bitmaps](#)

OnBclose (CSockDlg)

- [Internet und Netzwerke](#)

OnBconnect (CSockDlg)

- [Internet und Netzwerke](#)

OnBcustomdialog

- [Dialogfelder](#)

OnBeginPrinting

GDI-Objekte hinzufügen

- [Drucken und Druckvorschau](#)

OnBeginPrinting (CPrintItView)

- [Drucken und Druckvorschau](#)

OnBfirst (CSerialView)

- [Dateizugriff](#)

OnBlast (CSerialView)

- [Dateizugriff](#)

OnBsend (CSockDlg)

- [Internet und Netzwerke](#)

OnBwhichoption

- [Dialogfelder](#)

OnCancel

- [Steuerelemente](#)

OnCbemployed (CSerialView)

- [Dateizugriff](#)
- [Dateizugriff](#)

OnCbitalic (CTag7Dlg)

- [Antworten](#)

OnCbonidle (CTaskingView)

- [Multitasking](#)

OnCbthread1 (CTaskingView)

- [Multitasking](#)

OnCbthread2 (CTaskingView)

- [Multitasking](#)

OnCbusetext (CTag7Dlg)

- [Antworten](#)

OnChangeEsamptext

- [Text und Schriften](#)

OnChangeEsamptext (CTag7Dlg)

- [Antworten](#)

OnChangeInterval

- [Timer](#)

OnCkenblmsg

- [Steuerelemente](#)

OnCkenblpgm (CTag2Dlg)

- [Antworten](#)

OnCkshwmsg

- [Steuerelemente](#)

OnClick (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

OnClose

- [Internet und Netzwerke](#)
- [Internet und Netzwerke](#)

OnClose (CSockDlg)

- [Internet und Netzwerke](#)

OnClrmsg

- [Steuerelemente](#)

OnColorBlack (CTag10Doc)

- [SDI-Anwendungen](#)

OnColorBlue (CTag10Doc)

- [SDI-Anwendungen](#)

OnColorCommand (CTag11Doc)

- [MDI-Anwendungen](#)

OnConnect

- [Internet und Netzwerke](#)

OnConnect (CSockDlg)

- [Internet und Netzwerke](#)

OnContextMenu (CMenuDlg)

- [Menüs](#)
- [Antworten](#)

OnContextMenu (CTag11View)

- [MDI-Anwendungen](#)

OnCreate (CMainFrame)

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

OnDataFirst (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

OnDbClickMsgrid (CActiveXDlg)

- [Antworten](#)

OnDestroy (CTaskingView)

- [Multitasking](#)

OnDfltmsg (CTag2Dlg)

- [Antworten](#)

OnDocumentComplete (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnDraw

Drucken

- [Drucken und Druckvorschau](#)

OnDraw (CPrintItView)

- [Drucken und Druckvorschau](#)

OnDraw (CSquiggleCtrl)

- [Eigene ActiveX- Steuerelemente](#)

OnDraw (CTag10View)

- [SDI-Anwendungen](#)

OnDraw (CTestAppView)

- [DLLs](#)

OnEndPrinting

- [Drucken und Druckvorschau](#)

OnExit

- [Steuerelemente](#)

OnFileHello

- [Menüs](#)

OnFileopen

- [Dialogfelder](#)
- [Dialogfelder](#)

OnFileopen (CDialoge)

- [Antworten](#)

OnGoBack (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnGoNext (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnGoSearch (CWebBrowseView)

- [Antworten](#)

OnGoStart (CWebBrowseView)

- [Antworten](#)

OnHello

- [Erste Schritte mit Visual C++](#)

OnIdle

- [Multitasking](#)

OnIdle (CTaskingApp)

- [Multitasking](#)
- [Multitasking](#)
- [Antworten](#)

OnInitDialog

- [Steuerelemente](#)

OnInitDialog (CActiveXDlg)

- [ActiveX-Steuerelemente](#)

OnInitDialog (CGrafikDlg)

- [Bilder, Zeichnungen und Bitmaps](#)

OnInitDialog (CMausDlg)

- [Maus und Tastatur](#)

OnInitDialog (CSockDlg)

- [Internet und Netzwerke](#)

OnInitDialog (CTag2Dlg)

- [Steuerelemente](#)

OnInitDialog (CTag7Dlg)

- [Text und Schriften](#)
- [Text und Schriften](#)
- [Antworten](#)

OnInitDialog (CTimerDlg)

- [Timer](#)
- [Timer](#)

OnKeyDown

- [Maus und Tastatur](#)
- [Maus und Tastatur](#)

OnKeyDown (CMausDlg)

- [Antworten](#)

OnLButtonDown (CMausDlg)

- [Maus und Tastatur](#)

OnLButtonDown (CTag10View)

- [SDI-Anwendungen](#)

OnLButtonUp (CTag10View)

- [SDI-Anwendungen](#)

OnMouseDownMsgrid

- [ActiveX-Steuerelemente](#)

OnMouseMove

MK_LBUTTON

- [Antworten](#)

OnMouseMove (CMausDlg)

- [Maus und Tastatur](#)
- [Maus und Tastatur](#)
- [Antworten](#)

OnMouseMove (CTag10View)

- [SDI-Anwendungen](#)

OnMouseUpMsgrid

- [ActiveX-Steuerelemente](#)

OnNewAddress (CMainFrame)

- [Funktionen für Webbrowser](#)

OnNewDocument

- [SDI-Anwendungen](#)

OnNewDocument (CDBAdoDoc)

- [Datenbanken per ADO bearbeiten](#)

OnNewDocument (CSerialDoc)

- [Dateizugriff](#)

OnNewDocument (CTag10Doc)

- [SDI-Anwendungen](#)
- [Antworten](#)

OnNewDocument (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)

OnNewDocument (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

OnOK

- [Steuerelemente](#)

OnOK (CDlgPrintDlg)

- [Drucken und Druckvorschau](#)

OnOpenDocument (CSerialDoc)

- [Dateizugriff](#)

OnPaint

- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)
- [Bilder, Zeichnungen und Bitmaps](#)

OnPrepareDC (CPrintItView)

- [Drucken und Druckvorschau](#)

OnPreparePrinting

- [Drucken und Druckvorschau](#)

OnPreparePrinting (CPrintItView)

- [Drucken und Druckvorschau](#)
- [Drucken und Druckvorschau](#)

OnPrint

überschreiben

- [Drucken und Druckvorschau](#)

OnPrint (CPrintItView)

- [Drucken und Druckvorschau](#)

- [Drucken und Druckvorschau](#)

OnRButtonDown (CMausDlg)

- [Antworten](#)

OnReceive

- [Internet und Netzwerke](#)

OnReceive (CSockDlg)

- [Internet und Netzwerke](#)

OnRecordDelete (CDBOdbcView)

- [Daten aus ODBC- Datenbanken abrufen](#)

OnRecordMove (CDBOdbcView)

- [Antworten](#)

OnRecordNew (CDBOdbcView)

- [Daten aus ODBC- Datenbanken abrufen](#)

OnRSelection

- [Bilder, Zeichnungen und Bitmaps](#)

OnRType (CSockDlg)

- [Internet und Netzwerke](#)

OnRunpgm

- [Steuerelemente](#)

OnRunpgm (CTag2Dlg)

- [Antworten](#)

OnSelchangeLfonts

- [Text und Schriften](#)

OnSelchangeLfonts (CTag7Dlg)

- [Antworten](#)

OnSelChangeWidth (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnSend

- [Internet und Netzwerke](#)

OnSetCursor

- [Maus und Tastatur](#)

OnShowmsg

- [Steuerelemente](#)

OnShwmsg

- [Steuerelemente](#)

OnSize (CPaintDlg)

- [Antworten](#)

OnStarttime

- [Timer](#)
- [Timer](#)

OnStarttime (CTimerDlg)

- [Antworten](#)

OnStoptimer

- [Timer](#)
- [Timer](#)

OnStoptimer (CTimerDlg)

- [Antworten](#)

OnTimer

- [Timer](#)
- [Timer](#)

OnUpdateColorBlack (CTag10Doc)

- [SDI-Anwendungen](#)

OnUpdateColorBlue (CTag10Doc)

- [SDI-Anwendungen](#)

OnUpdateDataFirst (CDBAdoDoc)

- [Antworten](#)

OnUpdateDataLast (CDBAdoDoc)

- [Antworten](#)

OnUpdateIndicatorColor (CToolbarDoc)

- [Symbolleisten und Statusleisten](#)

OnUpdateIndicatorWidth (CToolbarDoc)

- [Antworten](#)

OnUpdateViewColorbar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnUpdateViewStop (CWebBrowseView)

- [Antworten](#)

OnUpdateWidthThin (CTag10Doc)

- [Antworten](#)

OnUpdateWidthUI (CTag11Doc)

- [Antworten](#)

OnUpdateWidthVthin (CTag10Doc)

- [Antworten](#)

OnViewColorbar (CMainFrame)

- [Symbolleisten und Statusleisten](#)

OnViewRefresh (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnViewStop (CWebBrowseView)

- [Funktionen für Webbrowser](#)

OnWidthCommand (CTag11Doc)

- [Antworten](#)

OnWidthThin (CTag10Doc)

- [Antworten](#)

OnWidthVthin (CTag10Doc)

- [Antworten](#)

OnWidthVthin (CToolBarDoc)

- [Symbolleisten und Statusleisten](#)

OnYesno

- [Antworten](#)

OnYesno (CDialogDlg)

- [Antworten](#)

OnYesnocancel

- [Dialogfelder](#)

Open

- [Daten aus ODBC- Datenbanken abrufen](#)

OPENFILENAME

- [Dialogfelder](#)

Operatoren

Addition

- [C++ im Überblick](#)

Adreß-

- [C++ im Überblick](#)

Ausgabe-

- [C++ im Überblick](#)

Bereichs-Auflösungs-

- [C++ im Überblick](#)

Division

- [C++ im Überblick](#)

Doppelpunkt

- [C++ im Überblick](#)

Einfüge-

- [C++ im Überblick](#)

Eingabe-

- [C++ im Überblick](#)

Index-

- [MFC-Hilfsklassen](#)

Indirektions-

- [C++ im Überblick](#)

Koordinaten addieren/subtrahieren

- [MFC-Hilfsklassen](#)

logische

- [C++ im Überblick](#)

Modulo

- [C++ im Überblick](#)

Multiplikation

- [C++ im Überblick](#)

Punkt-

- [C++ im Überblick](#)

Referenz-

- [C++ im Überblick](#)

relationale

- [C++ im Überblick](#)

Strukturelement-

- [C++ im Überblick](#)

Subtraktion

- [C++ im Überblick](#)

Zugriffs-

- [C++ im Überblick](#)

Zuweisung

- [C++ im Überblick](#)

Optimierungen

- [Debugger und Profiler](#)

Optionsfelder

- [Steuerelemente](#)

gewähltes ermitteln

- [Dialogfelder](#)

Gruppe

- [Dialogfelder](#)

mehrere Gruppen

- [Bilder, Zeichnungen und Bitmaps](#)

Variablen

- [Dialogfelder](#)

OR

- [Maus und Tastatur](#)
- [C++ im Überblick](#)

OUT_XXXX-Konstanten

- [Text und Schriften](#)

P

Parameter

ADO

- [Datenbanken per ADO bearbeiten](#)

Funktionen

- [C++ im Überblick](#)

Liste

- [C++ im Überblick](#)

ParseDateTime

- [MFC-Hilfsklassen](#)

Pfad

Dialogfeld Öffnen

- [Dialogfelder](#)

Pfeilcursor

- [Maus und Tastatur](#)

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Bitmaps

- [Bilder, Zeichnungen und Bitmaps](#)

CBrush

- [Bilder, Zeichnungen und Bitmaps](#)

HS_XXXX-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

Muster

- [Bilder, Zeichnungen und Bitmaps](#)

Platzhalter

Klassen

- [Bilder, Zeichnungen und Bitmaps](#)

Separatoren für Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

POINT

- [MFC-Hilfsklassen](#)

Polymorphismus

- [C++ im Überblick](#)

PopulateView (CSerialView)

- [Dateizugriff](#)

Popup

- [Menüs](#)

Popup-Menüs Siehe Kontextmenüs

- [Menüs](#)

POSITION

- [MFC-Hilfsklassen](#)

Position

Maus

- [Maus und Tastatur](#)

Präcompiler

#import-Direktive

- [Datenbanken per ADO bearbeiten](#)

präemptives Multitasking

- [Multitasking](#)

Präprozessor

#include-Direktive

- [C++ im Überblick](#)

Definitionen

- [Debugger und Profiler](#)

Direktiven

- [C++ im Überblick](#)

PrintAll

- [Drucken und Druckvorschau](#)

PrintCollate

- [Drucken und Druckvorschau](#)

printf

- [MFC-Hilfsklassen](#)

PrintRange

- [Drucken und Druckvorschau](#)

PrintSelection

- [Drucken und Druckvorschau](#)

Prioritätsebenen

- [Multitasking](#)

private

- [Dialogfelder](#)
- [C++ im Überblick](#)

Profiler

- [Debugger und Profiler](#)

Programme

ausführen

- [C++ im Überblick](#)

beenden

- [Steuerelemente](#)

Eintrittspunkt

- [C++ im Überblick](#)

Fehlersuche

- [Debugger und Profiler](#)

Optimierungen

- [Debugger und Profiler](#)

Programmierung

objektorientierte

- [Eigene Klassen und Module](#)

Projekt-Arbeitsbereich

- [Erste Schritte mit Visual C++](#)

anlegen

- [Erste Schritte mit Visual C++](#)

Projekte

ActiveX

- [ActiveX-Steuerelemente](#)

ActiveX-Steuerelement

- [Eigene ActiveX- Steuerelemente](#)

aktive Konfiguration

- [C++ im Überblick](#)

Dateien hinzufügen

- [C++ im Überblick](#)

DLLs

- [DLLs](#)

Drucken

- [Drucken und Druckvorschau](#)

Einstellungen

- [Debugger und Profiler](#)

Fächer

- [Multitasking](#)

Hello

- [Erste Schritte mit Visual C++](#)

Menüressource einbinden

- [Menüs](#)

Multitasking

- [Multitasking](#)

Netzwerk

- [Internet und Netzwerke](#)

Serialisierung

- [Dateizugriff](#)

Webbrowser

- [Funktionen für Webbrowser](#)

PROOF_QUALITY

- [Text und Schriften](#)

protected

- [C++ im Überblick](#)

Protokolle

TCP/IP

- [Internet und Netzwerke](#)

Prototypen

- [C++ im Überblick](#)

Prozesse

Prozess-Viewer

- [Debugger und Profiler](#)

Prozessorauslastung

- [Multitasking](#)

Prozess-Viewer

- [Debugger und Profiler](#)

PS_xxxx-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

public

- [Dialogfelder](#)
- [C++ im Überblick](#)

Destruktor

- [C++ im Überblick](#)

Konstruktor

- [C++ im Überblick](#)

Pulldown-Menüs

- [Menüs](#)

Punkte

addieren/subtrahieren

- [MFC-Hilfsklassen](#)

Arithmetik

- [MFC-Hilfsklassen](#)

verbinden

- [Maus und Tastatur](#)

Punktoperator

- [C++ im Überblick](#)

Q

Quadrate

- [Bilder, Zeichnungen und Bitmaps](#)

Quellcode-Browser

- [Debugger und Profiler](#)

Quellcodedateien

Kommentare

- [Erste Schritte mit Visual C++](#)
- [C++ im Überblick](#)

Querformat

- [Drucken und Druckvorschau](#)

QuickInfos

- [Symbolleisten und Statusleisten](#)

R

Rahmenfenster

MDI-Anwendungen

- [MDI-Anwendungen](#)

Zeiger auf

- [Symbolleisten und Statusleisten](#)

RandomStringValue

- [ActiveX-Steuerelemente](#)

RASTER_FONTTYPE

- [Text und Schriften](#)

RecalcLayout

- [Symbolleisten und Statusleisten](#)

Receive

- [Internet und Netzwerke](#)

Rechtecke

Arithmetik

- [MFC-Hilfsklassen](#)

Breite

- [MFC-Hilfsklassen](#)

Größe ändern

- [MFC-Hilfsklassen](#)

Höhe

- [MFC-Hilfsklassen](#)

Mittelpunkt

- [MFC-Hilfsklassen](#)

subtrahieren

- [MFC-Hilfsklassen](#)

Vereinigung/Schnitt

- [MFC-Hilfsklassen](#)

Recordset

ADO

- [Datenbanken per ADO bearbeiten](#)

Änderungen übernehmen

- [Dateizugriff](#)

aktuelle Datensatznummer

- [Dateizugriff](#)

aktuelle Position

- [Dateizugriff](#)

aktuellen Datensatz anzeigen

- [Dateizugriff](#)

Anzahl Datensätze

- [Dateizugriff](#)

BOF

- [Datenbanken per ADO bearbeiten](#)

Datensätze

- [Datenbanken per ADO bearbeiten](#)

Datensätze hinzufügen

- [Dateizugriff](#)

Datentypen

- [Datenbanken per ADO bearbeiten](#)

Dynaset

- [Daten aus ODBC- Datenbanken abrufen](#)

EOF

- [Datenbanken per ADO bearbeiten](#)

navigieren

- [Dateizugriff](#)
- [Dateizugriff](#)
- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

neuen anzeigen

- [Dateizugriff](#)

öffnen

- [Daten aus ODBC- Datenbanken abrufen](#)

schließen

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

serialisieren

- [Dateizugriff](#)

Snapshot

- [Daten aus ODBC- Datenbanken abrufen](#)

Spalten

- [Datenbanken per ADO bearbeiten](#)

Typen

- [Daten aus ODBC- Datenbanken abrufen](#)

verschiedene in MDI-Anwendungen

- [Daten aus ODBC- Datenbanken abrufen](#)

RECT

- [MFC-Hilfsklassen](#)

Rectangle

- [Bilder, Zeichnungen und Bitmaps](#)

Referenzen

- [C++ im Überblick](#)

Quellcode-Browser

- [Debugger und Profiler](#)

Referenzoperator

- [C++ im Überblick](#)

Refresh

- [Funktionen für Webbrowser](#)

RefreshBoundData (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

Register

- [Debugger und Profiler](#)

Registrierdatenbank

ActiveX-Objekte

- [ActiveX-Steuerelemente](#)

COM-Objekte

- [ActiveX-Steuerelemente](#)

registrieren

ActiveX-Steuerelemente

- [ActiveX-Steuerelemente](#)
- [Eigene ActiveX- Steuerelemente](#)

regsvr32

- [ActiveX-Steuerelemente](#)

regsvr32

- [ActiveX-Steuerelemente](#)

Reihenfolge

lexikografische

- [MFC-Hilfsklassen](#)

Release

- [Debugger und Profiler](#)

ASSERT

- [Debugger und Profiler](#)

Optimierungen

- [Debugger und Profiler](#)

ReleaseCapture

- [SDI-Anwendungen](#)

Remote-Debugging

- [Debugger und Profiler](#)

RemoveAll

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

RemoveAt

- [MFC-Hilfsklassen](#)
- [MFC-Hilfsklassen](#)

RemoveKey

- [MFC-Hilfsklassen](#)

Requery

- [Daten aus ODBC- Datenbanken abrufen](#)

Ressourcen

Accelerator-Tabelle

- [Funktionen für Webbrowser](#)

Ausnahmen

- [Ausnahmebehandlung](#)

Bitmaps

- [Bilder, Zeichnungen und Bitmaps](#)

Drucken

- [Drucken und Druckvorschau](#)

einfügen

- [Antworten](#)

freigeben

- [Ausnahmebehandlung](#)

gemeinsam genutzte

- [Multitasking](#)

Lücke

- [Bilder, Zeichnungen und Bitmaps](#)

manuell bearbeiten

- [Symbolleisten und Statusleisten](#)

Menü einfügen

- [MDI-Anwendungen](#)

Menüs

- [Menüs](#)

Pinsel

- [Bilder, Zeichnungen und Bitmaps](#)

Stifte

- [Bilder, Zeichnungen und Bitmaps](#)

Zugriffstasten

- [Menüs](#)
- [Funktionen für Webbrowser](#)

Ressourcenansicht

- [Erste Schritte mit Visual C++](#)

Ressourcensymbole

- [Timer](#)

ResumeThread

- [Multitasking](#)

return

- [C++ im Überblick](#)

ReverseFind

- [MFC-Hilfsklassen](#)

RGB

- [Maus und Tastatur](#)

Stifte

- [Bilder, Zeichnungen und Bitmaps](#)

Right

- [MFC-Hilfsklassen](#)

Rückgabewerte

Funktionen

- [C++ im Überblick](#)

MessageBox

- [Dialogfelder](#)

RUNTIME_CLASS

- [Multitasking](#)
- [Ausnahmebehandlung](#)

S

Sanduhrzeiger

- [Maus und Tastatur](#)
 - automatisch anzeigen
 - [Maus und Tastatur](#)

BeginWaitCursor

- [Maus und Tastatur](#)

EndWaitCursor

- [Maus und Tastatur](#)

SaveDrawing (CSquiggleCtrl)

- [Antworten](#)

Schaltflächen

- [Steuerelemente](#)

aktivieren/deaktivieren

- [Timer](#)

Maximieren

- [Erste Schritte mit Visual C++](#)

Meldungsfelder

- [Dialogfelder](#)

Minimieren

- [Erste Schritte mit Visual C++](#)

schließen

Anwendungen

- [Steuerelemente](#)

Schlüsselwörter

catch

- [Ausnahmebehandlung](#)

class

- [C++ im Überblick](#)

enum

- [C++ im Überblick](#)

new

- [Ausnahmebehandlung](#)

- [Ausnahmebehandlung](#)

private

- [C++ im Überblick](#)

protected

- [C++ im Überblick](#)

public

- [C++ im Überblick](#)

struct

- [C++ im Überblick](#)

- [C++ im Überblick](#)

throw

- [Ausnahmebehandlung](#)

try

- [Ausnahmebehandlung](#)

typedef

- [C++ im Überblick](#)

Schnellkasten

- [Menüs](#)

Schnellüberwachung

- [Debugger und Profiler](#)

Schnittstellen

ActiveX-Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

IDispatch

- [ActiveX-Steuerelemente](#)
- [Eigene ActiveX- Steuerelemente](#)

IUnknown

- [ActiveX-Steuerelemente](#)

ODBC

- [Daten aus ODBC- Datenbanken abrufen](#)

Schriftauswahl

- [Dialogfelder](#)

Schriften

- [Text und Schriften](#)

Attribute

- [Text und Schriften](#)

auflisten

- [Text und Schriften](#)

Ausgabegenauigkeit

- [Text und Schriften](#)

Ausgabequalität

- [Text und Schriften](#)

auswählen

- [Text und Schriften](#)

Breite

- [Text und Schriften](#)

CFont

- [Text und Schriften](#)

Clipping-Genauigkeit

- [Text und Schriften](#)

CreateFont

- [Text und Schriften](#)

DEVICE_FONTTYPE

- [Text und Schriften](#)

doppelte Einträge in Liste vermeiden

- [Text und Schriften](#)

durchgestrichen

- [Text und Schriften](#)

Erscheinung

- [Text und Schriften](#)

Familie

- [Text und Schriften](#)

Gewichtung

- [Text und Schriften](#)

Höhe

- [Text und Schriften](#)

- [Text und Schriften](#)

kursiv

- [Text und Schriften](#)

Liste erstellen

- [Text und Schriften](#)

Name

- [Text und Schriften](#)

- [Text und Schriften](#)

RASTER_FONTTYPE

- [Text und Schriften](#)

Schriftprobe

- [Text und Schriften](#)

SetFont

- [Text und Schriften](#)

Skript

- [Text und Schriften](#)

Stärke

- [Text und Schriften](#)

Stil

- [Text und Schriften](#)

Textwinkel

- [Text und Schriften](#)

TrueType

- [Text und Schriften](#)

TRUETYPE_FONTTYPE

- [Text und Schriften](#)

unterstrichen

- [Text und Schriften](#)

verwenden

- [Text und Schriften](#)

Zeichenabstand

- [Text und Schriften](#)

- [Text und Schriften](#)

Zeichenbreite

- [Text und Schriften](#)

Zeichensatz

- [Text und Schriften](#)

Zeichenwinkel

- [Text und Schriften](#)

Schriftfamilie

- [Text und Schriften](#)

SCODE

- [Ausnahmebehandlung](#)

SDI-Anwendungen

- [SDI-Anwendungen](#)

Dokument/Ansicht-Architektur

- [SDI-Anwendungen](#)

Drucken

- [Drucken und Druckvorschau](#)

Klassen

- [SDI-Anwendungen](#)

Symbolleisten

- [Symbolleisten und Statusleisten](#)

Seite einrichten

- [Dialogfelder](#)

Seitenansicht Siehe Druckvorschau

- [Drucken und Druckvorschau](#)

Seitenausrichtung

- [Drucken und Druckvorschau](#)

Seitennummern

- [Drucken und Druckvorschau](#)

Seitenumbruch

- [Drucken und Druckvorschau](#)

SelectObject

- [SDI-Anwendungen](#)

Semaphoren

- [Multitasking](#)

Send

- [Internet und Netzwerke](#)

Separator

Symbolleisten

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

Serialisierung

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

Ausnahmebehandlung

- [Dateizugriff](#)

CArchive

- [Dateizugriff](#)

DECLARE_SERIAL

- [Dateizugriff](#)

dialogfeldbasierende Anwendungen

- [Dateizugriff](#)

E/A-Streams

- [SDI-Anwendungen](#)
- [Dateizugriff](#)

IMPLEMENT_SERIAL

- [Dateizugriff](#)

Objekte serialisierbar machen

- [Dateizugriff](#)

Zeichenanwendung

- [Eigene Klassen und Module](#)

Serialize

- [Dateizugriff](#)
in Klasse einbinden
 - [Dateizugriff](#)

Serialize (CLine)

- [SDI-Anwendungen](#)
- [SDI-Anwendungen](#)
- [Antworten](#)

Serialize (CModArt)

- [Eigene Klassen und Module](#)

Serialize (CSerialDoc)

- [Dateizugriff](#)

Serialize (CTag10Doc)

- [SDI-Anwendungen](#)

Serialize (CTestAppDoc)

- [Eigene Klassen und Module](#)
- [DLLs](#)

Server

ActiveX

- [ActiveX-Steuerelemente](#)

Sockets

- [Internet und Netzwerke](#)

set

- [C++ im Überblick](#)

SetAbsolutePosition

- [Daten aus ODBC- Datenbanken abrufen](#)

SetAddress (CMainFrame)

- [Funktionen für Webbrowser](#)

SetAt

Arrayklassen

- [MFC-Hilfsklassen](#)

Indexoperator

- [MFC-Hilfsklassen](#)

Listenklassen

- [MFC-Hilfsklassen](#)

Tabellenklassen

- [MFC-Hilfsklassen](#)

Zeichenfolgen

- [MFC-Hilfsklassen](#)

SetButtonStyle

- [Symbolleisten und Statusleisten](#)

SetButtonText

- [Symbolleisten und Statusleisten](#)

SetCapture

- [SDI-Anwendungen](#)

SetCheck

- [SDI-Anwendungen](#)

SetCol

- [ActiveX-Steuerelemente](#)

SetColPosition

- [ActiveX-Steuerelemente](#)

SetColSel

- [ActiveX-Steuerelemente](#)

SetDateTime

- [MFC-Hilfsklassen](#)

SetFont

- [Text und Schriften](#)

SetIndicators

- [Symbolleisten und Statusleisten](#)

SetLength (CModArt)

- [Eigene ActiveX- Steuerelemente](#)

SetMaxPage

- [Drucken und Druckvorschau](#)

SetMergeCol

- [ActiveX-Steuerelemente](#)

SetMinPage

- [Drucken und Druckvorschau](#)

SetModifiedFlag

- [SDI-Anwendungen](#)

SetMyFont

- [Text und Schriften](#)
- [Text und Schriften](#)

SetMyFont (CTag7Dlg)

- [Antworten](#)

SetPaneText

- [Symbolleisten und Statusleisten](#)

SetParent (CMySocket)

- [Internet und Netzwerke](#)

SetPixel

- [Maus und Tastatur](#)

SetRect (CModArt)

- [Eigene Klassen und Module](#)

SetSort

- [ActiveX-Steuererelemente](#)

SetSquiggleLength (CSquiggleCtrl)

- [Eigene ActiveX- Steuererelemente](#)

SetTextArray

- [ActiveX-Steuererelemente](#)

SetTimer

- [Timer](#)

SetViewportExt

- [Bilder, Zeichnungen und Bitmaps](#)

SetViewportOrg

- [Multitasking](#)

SetWidth (CToolBarDoc)

- [Symbolleisten und Statusleisten](#)

SetWindowExt

- [Bilder, Zeichnungen und Bitmaps](#)

SetWindowText

- [Symbolleisten und Statusleisten](#)

ShellExecute

- [Steuererelemente](#)

SHIFTJIS_CHARSET

- [Text und Schriften](#)

ShowBitmap

- [Bilder, Zeichnungen und Bitmaps](#)

ShowControlBar

- [Symbolleisten und Statusleisten](#)

ShowWindow

- [Steuerelemente](#)

Sicherheitsprofile

- [Multitasking](#)

Singlethreading

- [Multitasking](#)

SIZE

- [MFC-Hilfsklassen](#)

SOCKET_ERROR

- [Internet und Netzwerke](#)

Sockets

- [Internet und Netzwerke](#)

Accept

- [Internet und Netzwerke](#)

CAsyncSocket

- [Internet und Netzwerke](#)

Client

- [Internet und Netzwerke](#)

Close

- [Internet und Netzwerke](#)

Connect

- [Internet und Netzwerke](#)

Create

- [Internet und Netzwerke](#)

Ereignisse

- [Internet und Netzwerke](#)

erstellen

- [Internet und Netzwerke](#)

Fehler erkennen

- [Internet und Netzwerke](#)

GetLastError

- [Internet und Netzwerke](#)

Listen

- [Internet und Netzwerke](#)

Nachrichten senden/empfangen

- [Internet und Netzwerke](#)

OnClose

- [Internet und Netzwerke](#)

Receive

- [Internet und Netzwerke](#)

Send

- [Internet und Netzwerke](#)

Server

- [Internet und Netzwerke](#)

SOCKET_ERROR

- [Internet und Netzwerke](#)

Verbindungen

- [Internet und Netzwerke](#)

Verbindungen herstellen

- [Internet und Netzwerke](#)

Verbindungen schließen

- [Internet und Netzwerke](#)

von CAsyncSocket ableiten

- [Internet und Netzwerke](#)

sortieren

Drucken

- [Drucken und Druckvorschau](#)

FlexGrid

- [ActiveX-Steuerelemente](#)

Sortierreihenfolge

FlexGrid

- [ActiveX-Steuerelemente](#)

Spalten

Recordset

- [Datenbanken per ADO bearbeiten](#)

Speicher

- [Debugger und Profiler](#)

Ausnahmen

- [Ausnahmebehandlung](#)

CObArray

- [MFC-Hilfsklassen](#)

CPtrArray

- [MFC-Hilfsklassen](#)

Zugriffsverletzungen

- [Ausnahmebehandlung](#)

Spooler

Drucken

- [Drucken und Druckvorschau](#)

sprintf

- [MFC-Hilfsklassen](#)

Spy++

- [Debugger und Profiler](#)

SQL

- [Daten aus ODBC- Datenbanken abrufen](#)

Abfragen

- [Daten aus ODBC- Datenbanken abrufen](#)
- [Datenbanken per ADO bearbeiten](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

Befehle ausführen (ADO)

- [Datenbanken per ADO bearbeiten](#)

SQL_FETCH_ABSOLUTE

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_FIRST

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_LAST

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_NEXT

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_PRIOR

- [Daten aus ODBC- Datenbanken abrufen](#)

SQL_FETCH_RELATIVE

- [Daten aus ODBC- Datenbanken abrufen](#)

Squiggle

ActiveX-Steuerelement

- [Eigene ActiveX- Steuerelemente](#)

Squiggles

Anzahl

- [Eigene ActiveX- Steuerelemente](#)

Hintergrund

- [Eigene ActiveX- Steuerelemente](#)

Stack-Größe

- [Multitasking](#)

Standarddialogfelder

- [Dialogfelder](#)

verwenden

- [Dialogfelder](#)

Standard-DLLs

- [DLLs](#)

- [DLLs](#)

Standardkonstruktor

- [C++ im Überblick](#)

Standardsymbolleiste

- [Erste Schritte mit Visual C++](#)

starten

Anwendungen

- [Steuerelemente](#)

OnIdle-Tasks

- [Multitasking](#)

Threads

- [Multitasking](#)

Timer

- [Timer](#)

StartPage

- [Drucken und Druckvorschau](#)

Status

COleDateTime

- [MFC-Hilfsklassen](#)

COleDateTimeSpan

- [MFC-Hilfsklassen](#)

Statusleisten

- [Symbolleisten und Statusleisten](#)

Ausschnitte

- [Symbolleisten und Statusleisten](#)

Ausschnitte aktualisieren

- [Symbolleisten und Statusleisten](#)

Elemente einfügen

- [Symbolleisten und Statusleisten](#)

Feststelltasten anzeigen/ausblenden

- [Symbolleisten und Statusleisten](#)

Text für Ausschnitte

- [Symbolleisten und Statusleisten](#)

Text in Ausschnitten festlegen

- [Symbolleisten und Statusleisten](#)

Statuszeilentext

- [Symbolleisten und Statusleisten](#)

Steuerelemente

- [Steuerelemente](#)

aktivieren/deaktivieren

- [Steuerelemente](#)

aktualisieren

- [Steuerelemente](#)

anzeigen/ausblenden

- [Steuerelemente](#)

DataGrid

- [Datenbanken per ADO bearbeiten](#)

datengebundene

- [Datenbanken per ADO bearbeiten](#)

Datenquelle

- [Datenbanken per ADO bearbeiten](#)

Datumswerte

- [Daten aus ODBC- Datenbanken abrufen](#)

Dialogfelder

- [Steuerelemente](#)

Dialogleisten

- [Funktionen für Webbrowser](#)

Dropdown-Listenfelder

- [Steuerelemente](#)

Eigenschaften

- [Steuerelemente](#)

Eigenschaftsseiten

- [Eigene ActiveX- Steuerelemente](#)

Eingabefelder

- [Steuerelemente](#)

Ereignisse

- [Eigene ActiveX- Steuerelemente](#)

Fokus

- [Maus und Tastatur](#)

Get-/Set-Methoden

- [Eigene ActiveX- Steuerelemente](#)

IDC_STATIC

- [Steuerelemente](#)

Kombinationsfelder

- [Steuerelemente](#)

Kontrollkästchen

- [Steuerelemente](#)

Methoden

- [Eigene ActiveX- Steuerelemente](#)

mit Datenbankfeldern verbinden

- [Daten aus ODBC- Datenbanken abrufen](#)

Optionsfelder

- [Steuerelemente](#)

registrieren

- [ActiveX-Steuerelemente](#)
- [Eigene ActiveX- Steuerelemente](#)

Schaltflächen

- [Steuerelemente](#)

Stile für Symbolleisten

- [Symbolleisten und Statusleisten](#)

Symbol in der Toolbox

- [Eigene ActiveX- Steuerelemente](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)
- [Antworten](#)

testen

- [Eigene ActiveX- Steuerelemente](#)

Text

- [Steuerelemente](#)

UpdateData

- [Steuerelemente](#)

Variablen in SDI-/MDI-Anwendungen

- [Dateizugriff](#)

Variablen initialisieren

- [Steuerelemente](#)

Variablen zuordnen

- [Steuerelemente](#)

Webbrowser

- [Funktionen für Webbrowser](#)

Zugriffstasten

- [Steuerelemente](#)

Steuerelemente Siehe auch ActiveX-Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Steuerelementpalette

ActiveX-Steuerelemente

- [ActiveX-Steuerelemente](#)

Stifte

- [Maus und Tastatur](#)
- [Bilder, Zeichnungen und Bitmaps](#)

Breite

- [Bilder, Zeichnungen und Bitmaps](#)

CPen

- [Bilder, Zeichnungen und Bitmaps](#)

Farben

- [Bilder, Zeichnungen und Bitmaps](#)

PS_XXXX-Konstanten

- [Bilder, Zeichnungen und Bitmaps](#)

RGB

- [Bilder, Zeichnungen und Bitmaps](#)

Stile

Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

Stifte

- [Bilder, Zeichnungen und Bitmaps](#)

Symbolleisten

- [Symbolleisten und Statusleisten](#)

Symbolleistenschaltflächen

- [Symbolleisten und Statusleisten](#)

- [Symbolleisten und Statusleisten](#)

Stop-Symbol

- [Dialogfelder](#)

StretchBlt

- [Bilder, Zeichnungen und Bitmaps](#)

Strichstärke

- [Text und Schriften](#)

Strings

CString

- [Steuerelemente](#)

in Großbuchstaben konvertieren

- [Steuerelemente](#)

nullterminierte

- [MFC-Hilfsklassen](#)

suchen

- [MFC-Hilfsklassen](#)

vergleichen

- [MFC-Hilfsklassen](#)

Strings Siehe auch Zeichenfolgen

- [Steuerelemente](#)

struct

- [C++ im Überblick](#)
- [C++ im Überblick](#)

Strukturelementoperator

- [C++ im Überblick](#)

Strukturen

- [C++ im Überblick](#)

AFX_MODULE_STATE

- [DLLs](#)

BITMAP

- [Bilder, Zeichnungen und Bitmaps](#)

CDaoErrorInfo

- [Ausnahmebehandlung](#)

DATE

- [MFC-Hilfsklassen](#)

DevMode

- [Drucken und Druckvorschau](#)

DOCINFO

- [Drucken und Druckvorschau](#)

Einführung

- [C++ im Überblick](#)

ENUMLOGFONTEX

- [Text und Schriften](#)

LOGFONT

- [Text und Schriften](#)

Nachteile

- [C++ im Überblick](#)

NEWTEXTMETRICEX

- [Text und Schriften](#)

OPENFILENAME

- [Dialogfelder](#)

POINT

- [MFC-Hilfsklassen](#)

RECT

- [MFC-Hilfsklassen](#)

SIZE

- [MFC-Hilfsklassen](#)

Threads

- [Multitasking](#)

VARIANT

- [MFC-Hilfsklassen](#)

SubtractRect

- [MFC-Hilfsklassen](#)

Subtraktion

- [C++ im Überblick](#)

Suchen

Funktionen

- [Steuerelemente](#)

Strings

- [MFC-Hilfsklassen](#)

Zeichenfolgen

- [MFC-Hilfsklassen](#)

Suchen und Ersetzen

- [Dialogfelder](#)

SuspendSpinner (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)

- [Antworten](#)

SYMBOL_CHARSET

- [Text und Schriften](#)

Symbole

bearbeiten

- [Erste Schritte mit Visual C++](#)

Dialogfelder

- [Erste Schritte mit Visual C++](#)

Meldungsfelder

- [Dialogfelder](#)

Ressourcen

- [Timer](#)

Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Symbol-Editor

- [Erste Schritte mit Visual C++](#)

Symbolleisten

- [Symbolleisten und Statusleisten](#)

andocken

- [Symbolleisten und Statusleisten](#)

anzeigen/ausblenden

- [Symbolleisten und Statusleisten](#)

CreateEx

- [Symbolleisten und Statusleisten](#)

Debug

- [Debugger und Profiler](#)

Dialogleisten

- [Funktionen für Webbrowser](#)

erstellen

- [Symbolleisten und Statusleisten](#)

Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

laden

- [Symbolleisten und Statusleisten](#)

LoadToolBar

- [Symbolleisten und Statusleisten](#)

mit Gerüst verbinden

- [Symbolleisten und Statusleisten](#)

neu erstellen

- [Symbolleisten und Statusleisten](#)

Platzhalter für Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

QuickInfos

- [Symbolleisten und Statusleisten](#)

Rahmengröße

- [Symbolleisten und Statusleisten](#)

Separator

- [Symbolleisten und Statusleisten](#)

- [Symbolleisten und Statusleisten](#)

ShowControlBar

- [Symbolleisten und Statusleisten](#)

Statuszeilentext

- [Symbolleisten und Statusleisten](#)

Steuerelemente

- [Steuerelemente](#)

Steuerelementnummer ermitteln

- [Symbolleisten und Statusleisten](#)

Stile

- [Symbolleisten und Statusleisten](#)

verankern

- [Symbolleisten und Statusleisten](#)

Zwischenraum

- [Symbolleisten und Statusleisten](#)

Symbolleisten-Editor

- [Symbolleisten und Statusleisten](#)

Symbolleistenschaltflächen

Nummer ermitteln

- [Symbolleisten und Statusleisten](#)

Stile

- [Symbolleisten und Statusleisten](#)

- [Symbolleisten und Statusleisten](#)

Stile festlegen

- [Symbolleisten und Statusleisten](#)

Text festlegen

- [Symbolleisten und Statusleisten](#)

Verhalten festlegen

- [Symbolleisten und Statusleisten](#)

Synchronisierungsklassen

- [Multitasking](#)

Synchronisierungsmechanismen

- [Multitasking](#)

Systemdialogfelder

- [Dialogfelder](#)

Systemmenü

Dialogfelder

- [Bilder, Zeichnungen und Bitmaps](#)

T

Tabellenklassen

- [MFC-Hilfsklassen](#)

Anzahl der Elemente

- [MFC-Hilfsklassen](#)

benutzerdefinierte

- [MFC-Hilfsklassen](#)

CMapPtrToPtr

- [MFC-Hilfsklassen](#)

CMapPtrToWord

- [MFC-Hilfsklassen](#)

CMapStringToOb

- [MFC-Hilfsklassen](#)

CMapStringToPtr

- [MFC-Hilfsklassen](#)

CMapStringToString

- [MFC-Hilfsklassen](#)

CMapWordToOb

- [MFC-Hilfsklassen](#)

CMapWordToPtr

- [MFC-Hilfsklassen](#)

Elemente

- [MFC-Hilfsklassen](#)

erstes Element

- [MFC-Hilfsklassen](#)

GetCount

- [MFC-Hilfsklassen](#)

GetNextAssoc

- [MFC-Hilfsklassen](#)

Hash-Schlüssel

- [MFC-Hilfsklassen](#)

Indexoperator

- [MFC-Hilfsklassen](#)

Liste durchlaufen

- [MFC-Hilfsklassen](#)

Lookup

- [MFC-Hilfsklassen](#)

Objekte löschen

- [MFC-Hilfsklassen](#)

POSITION

- [MFC-Hilfsklassen](#)

RemoveKey

- [MFC-Hilfsklassen](#)

SetAt

- [MFC-Hilfsklassen](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)
- [Antworten](#)

Textfelder

- [Steuerelemente](#)

Tasks

fortlaufend arbeitende

- [Multitasking](#)

Tastatur

- [Maus und Tastatur](#)

Ereignisse

- [Maus und Tastatur](#)

Fokus

- [Maus und Tastatur](#)

GetKeyState

- [Maus und Tastatur](#)

Hotkeys

- [Menüs](#)

Schnellasten

- [Menüs](#)

Strg-Taste

- [Maus und Tastatur](#)

Tastencode ermitteln

- [Maus und Tastatur](#)

Umschalt-Taste

- [Maus und Tastatur](#)

virtuelle Tastenbezeichner

- [Funktionen für Webbrowser](#)

virtuelle Tastencodes

- [Maus und Tastatur](#)

Zugriffstasten

- [Menüs](#)
- [Funktionen für Webbrowser](#)

Tasten

Alt-Taste

- [Maus und Tastatur](#)

TBBS_CHECKGROUP

- [Symbolleisten und Statusleisten](#)

TBSTYLE_xxxx-Konstanten

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

TCP/IP

- [Internet und Netzwerke](#)

Teilstrings

- [MFC-Hilfsklassen](#)

Templates Siehe Vorlagen

- [MFC-Hilfsklassen](#)

Testcontainer

für ActiveX-Steuerelemente

- [Eigene ActiveX- Steuerelemente](#)

Methoden aufrufen

- [Eigene ActiveX- Steuerelemente](#)

Testen

Release-Modus

- [Debugger und Profiler](#)

Text

- [Text und Schriften](#)

formatieren

- [MFC-Hilfsklassen](#)

Steuerelement

- [Steuerelemente](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)

Textwinkel

- [Text und Schriften](#)

Thread

Stack-Größe

- [Multitasking](#)

ThreadFunc (CTaskingDoc)

- [Multitasking](#)
- [Multitasking](#)

Threads

anhalten

- [Multitasking](#)

Benutzeroberflächen

- [Multitasking](#)

CREATE_SUSPENDED

- [Multitasking](#)

CWinThread

- [Multitasking](#)

Ereignisse

- [Multitasking](#)

gemeinsam genutzte Ressourcen

- [Multitasking](#)

kritische Abschnitte

- [Multitasking](#)

Leerlaufverarbeitung

- [Multitasking](#)

Mutexe

- [Multitasking](#)

Prioritätsebenen

- [Multitasking](#)

Prozessorauslastung

- [Multitasking](#)

ResumeThread

- [Multitasking](#)

RUNTIME_CLASS

- [Multitasking](#)

sauberes Herunterfahren

- [Multitasking](#)

Semaphoren

- [Multitasking](#)

Sicherheitsprofile

- [Multitasking](#)

starten

- [Multitasking](#)

Strukturen

- [Multitasking](#)

Synchronisierungsmechanismen

- [Multitasking](#)

unabhängige

- [Multitasking](#)

- [Multitasking](#)

vom Ansichtsbjekt auslösen

- [Multitasking](#)

Worker-

- [Multitasking](#)

throw

- [Ausnahmebehandlung](#)

ThrowErrno

- [Ausnahmebehandlung](#)

ThrowOsError

- [Ausnahmebehandlung](#)

Tilde

- [C++ im Überblick](#)

Timer

- [Timer](#)

anhalten

- [Timer](#)

Anzahl

- [Timer](#)

auslösenden ermitteln

- [Timer](#)

Bereich des Intervalls

- [Timer](#)

starten

- [Timer](#)

Titelleiste

Maximieren-Schaltfläche

- [Erste Schritte mit Visual C++](#)

Minimieren-Schaltfläche

- [Erste Schritte mit Visual C++](#)

TODO

- [Steuerelemente](#)

TopLeft

- [MFC-Hilfsklassen](#)

TPM_xxxx-Konstanten

- [Menüs](#)

TRACE

- [Debugger und Profiler](#)

TrackPopupMenu

- [Menüs](#)

Transaktionen

- [Datenbanken per ADO bearbeiten](#)

Trennlinien

Menübefehle

- [Menüs](#)

TrimLeft

- [MFC-Hilfsklassen](#)

TrimRight

- [MFC-Hilfsklassen](#)

TrueType

- [Text und Schriften](#)

TRUETYPE_FONTTYPE

- [Text und Schriften](#)

try-Blöcke

- [SDI-Anwendungen](#)
- [Ausnahmebehandlung](#)

Typdefinition

- [C++ im Überblick](#)

typedef

- [C++ im Überblick](#)

Typsicherheit

- [MFC-Hilfsklassen](#)

Typumwandlung

CLine

- [SDI-Anwendungen](#)

U

überladen

Funktionen

- [C++ im Überblick](#)

überschreiben

Funktionen

- [Drucken und Druckvorschau](#)

OnPrint

- [Drucken und Druckvorschau](#)

Uhr

- [Timer](#)

Zeitformat

- [Timer](#)

Umlaute

Menübefehle

- [Menüs](#)

UnionRect

- [MFC-Hilfsklassen](#)

Untermenüs

- [Menüs](#)

unterstrichen

- [Text und Schriften](#)

Update

- [Daten aus ODBC- Datenbanken abrufen](#)

UPDATE_COMMAND_UI

- [SDI-Anwendungen](#)

UpdateBoundData (CDBAdoView)

- [Datenbanken per ADO bearbeiten](#)

UpdateColorbar (CToolBarDoc)

- [Symbolleisten und Statusleisten](#)

UpdateData

- [Steuerelemente](#)

UpdateWidthCB (CMainFrame)

- [Symbolleisten und Statusleisten](#)

URL

- [Funktionen für Webbrowser](#)

aktuellen anzeigen

- [Funktionen für Webbrowser](#)

in Dialogleiste anzeigen

- [Funktionen für Webbrowser](#)

Ursachencodes

Archivausnahmen

- [Ausnahmebehandlung](#)

Dateiausnahmen

- [Ausnahmebehandlung](#)

USER

16-Bit-Versionen

- [Multitasking](#)

V

VARIABLE_PITCH

- [Text und Schriften](#)

Variablen

- [C++ im Überblick](#)

Datentypen

- [C++ im Überblick](#)

Eigenschaften

- [Eigene ActiveX- Steuerelemente](#)

globale

- [C++ im Überblick](#)

Gültigkeitsbereich

- [C++ im Überblick](#)

initialisieren

- [Steuerelemente](#)
- [SDI-Anwendungen](#)

lokale

- [C++ im Überblick](#)

Member-

- [Steuerelemente](#)

mit Steuerelementen verbinden

- [Steuerelemente](#)

Optionsfelder

- [Dialogfelder](#)

Präfix

- [Steuerelemente](#)

Referenzen

- [C++ im Überblick](#)

schreiben/lesen

- [Dateizugriff](#)

Typdefinition

- [C++ im Überblick](#)

überwachen

- [Debugger und Profiler](#)

VARIANT

- [MFC-Hilfsklassen](#)

Variant

- [Datenbanken per ADO bearbeiten](#)

VBScript

- [Funktionen für Webbrowser](#)

ADO

- [Datenbanken per ADO bearbeiten](#)

verankern

Symbolleisten

- [Symbolleisten und Statusleisten](#)

Verbindungen

Accept

- [Internet und Netzwerke](#)

Adressen

- [Internet und Netzwerke](#)

annehmen

- [Internet und Netzwerke](#)

Anwendungen

- [Internet und Netzwerke](#)

beenden

- [Internet und Netzwerke](#)

Datenbanken

- [Datenbanken per ADO bearbeiten](#)

herstellen

- [Internet und Netzwerke](#)

hören

- [Internet und Netzwerke](#)

schließen

- [Internet und Netzwerke](#)

Sockets

- [Internet und Netzwerke](#)

Vererbung

- [Eigene Klassen und Module](#)
- [C++ im Überblick](#)

Vergleiche

- [Steuerelemente](#)
- [C++ im Überblick](#)

Zeichenfolgen

- [MFC-Hilfsklassen](#)

VERIFY

- [Debugger und Profiler](#)

Verknüpfungen

binäre

- [Maus und Tastatur](#)

Versionsnummer

IMPLEMENT_SERIAL

- [Dateizugriff](#)

Verzeichnisse

im Dialogfeld Öffnen vorgeben

- [Dialogfelder](#)

Viewport

- [Multitasking](#)

virtuelle Tastencodes

- [Maus und Tastatur](#)

Visual C++

Arbeitsbereich

- [C++ im Überblick](#)

Ausgabefenster

- [C++ im Überblick](#)

DLL-Typen

- [DLLs](#)

Entwicklungsumgebung

- [Erste Schritte mit Visual C++](#)

Klassentypen

- [Eigene Klassen und Module](#)

Visual Studio

- [Erste Schritte mit Visual C++](#)

Arbeitsbereich

- [Erste Schritte mit Visual C++](#)

Vorlagen

Auflistungsklassen

- [MFC-Hilfsklassen](#)

CArray

- [MFC-Hilfsklassen](#)

CList

- [MFC-Hilfsklassen](#)

CMap

- [MFC-Hilfsklassen](#)

Dateien

- [MFC-Hilfsklassen](#)

Syntax

- [MFC-Hilfsklassen](#)

Vorwärts

Webbrowser

- [Funktionen für Webbrowser](#)

Warnstufen

- [Debugger und Profiler](#)

Warnungen

Debugger

- [Debugger und Profiler](#)

Webbrowser

- [Funktionen für Webbrowser](#)

Download anhalten

- [Funktionen für Webbrowser](#)

Download-Fortschritt

- [Funktionen für Webbrowser](#)

Navigate

- [Funktionen für Webbrowser](#)

navigieren

- [Funktionen für Webbrowser](#)

Refresh

- [Funktionen für Webbrowser](#)

Status

- [Funktionen für Webbrowser](#)

steuern

- [Funktionen für Webbrowser](#)

URL

- [Funktionen für Webbrowser](#)

URL abrufen

- [Funktionen für Webbrowser](#)

Vorwärts

- [Funktionen für Webbrowser](#)

Zurück

- [Funktionen für Webbrowser](#)

Webseiten

navigieren

- [Funktionen für Webbrowser](#)

neu laden

- [Funktionen für Webbrowser](#)

Width

- [MFC-Hilfsklassen](#)

Windows

Standardsteuerelemente

■ [Steuerelemente](#)

Windows_NT

16-Bit-Anwendungen

■ [Multitasking](#)

WinExec

○ [Steuerelemente](#)

WM_CONTEXTMENU

○ [Menüs](#)

○ [MDI-Anwendungen](#)

WM_DESTROY

○ [Multitasking](#)

WM_INITDIALOG

○ [Steuerelemente](#)

WM_KEY

○ [Menüs](#)

WM_KEYDOWN

○ [Maus und Tastatur](#)

WM_KEYUP

○ [Maus und Tastatur](#)

WM_LBUTTONDOWNBLCLK

○ [Maus und Tastatur](#)

WM_LBUTTONDOWN

○ [Maus und Tastatur](#)

WM_LBUTTONUP

○ [Maus und Tastatur](#)

WM_MOUSEMOVE

○ [Maus und Tastatur](#)

WM_MOUSEWHEEL

○ [Maus und Tastatur](#)

WM_PAINT

○ [Bilder, Zeichnungen und Bitmaps](#)

WM_RBUTTONDOWNBLCLK

○ [Maus und Tastatur](#)

WM_RBUTTONDOWN

○ [Maus und Tastatur](#)

Kontextmenüs

- [Menüs](#)

WM_RBUTTONUP

- [Maus und Tastatur](#)

WM_SETCURSOR

- [Maus und Tastatur](#)

WM_TIMER

- [Timer](#)

Wochentag

- [MFC-Hilfsklassen](#)

Worker-Threads

- [Multitasking](#)

WS_xxxx-Konstanten

- [Symbolleisten und Statusleisten](#)
- [Symbolleisten und Statusleisten](#)

Z

Zeichenabstand

- [Text und Schriften](#)

Zeichenanwendung

ActiveX-Steuerelement

- [Eigene ActiveX- Steuerelemente](#)

Bibliotheksmodul

- [Eigene Klassen und Module](#)

DLL

- [DLLs](#)

Hintergrund

- [Eigene ActiveX- Steuerelemente](#)

neue Zeichnung

- [Eigene Klassen und Module](#)

serialisieren

- [Eigene Klassen und Module](#)

Symbolleiste Farben

- [Symbolleisten und Statusleisten](#)

Zeichenbereich

- [Eigene Klassen und Module](#)

Zeichenanwendungen

Linienklasse

- [SDI-Anwendungen](#)

Zeichenbreite

- [Text und Schriften](#)

Zeichenfolgen

Empty

- [MFC-Hilfsklassen](#)

Formatcodes

- [MFC-Hilfsklassen](#)

GetAt

- [MFC-Hilfsklassen](#)

IsEmpty

- [MFC-Hilfsklassen](#)

Klassen

- [MFC-Hilfsklassen](#)

konvertieren in Groß-/Kleinbuchstaben

- [MFC-Hilfsklassen](#)

Leerzeichen entfernen

- [MFC-Hilfsklassen](#)

Left

- [MFC-Hilfsklassen](#)

MakeUpper

- [MFC-Hilfsklassen](#)

Mid

- [MFC-Hilfsklassen](#)

nullterminierte

- [MFC-Hilfsklassen](#)

Right

- [MFC-Hilfsklassen](#)

SetAt

- [MFC-Hilfsklassen](#)

suchen

- [MFC-Hilfsklassen](#)

testen

- [MFC-Hilfsklassen](#)

TrimLeft

- [MFC-Hilfsklassen](#)

TrimRight

- [MFC-Hilfsklassen](#)

umkehren

- [MFC-Hilfsklassen](#)

vergleichen

- [MFC-Hilfsklassen](#)

Zeichenfolgen Siehe auch Strings

- [Steuerelemente](#)

Zeichenfolgen-Editor

- [Symbolleisten und Statusleisten](#)

Zeichenfolgentabelle

bearbeiten

- [Symbolleisten und Statusleisten](#)

für Statusleistenausschnitt

- [Symbolleisten und Statusleisten](#)

Kombinationsfelder

- [Symbolleisten und Statusleisten](#)

Zeichensatz

- [Text und Schriften](#)

Zeichenwinkel

- [Text und Schriften](#)

zeichnen

- [Maus und Tastatur](#)

Gerätekontext

- [Maus und Tastatur](#)

Kreise

- [Bilder, Zeichnungen und Bitmaps](#)

Linien

- [Bilder, Zeichnungen und Bitmaps](#)

Punkte verbinden

- [Maus und Tastatur](#)

Quadrate

- [Bilder, Zeichnungen und Bitmaps](#)

Stifte

- [Maus und Tastatur](#)

Zeichnungen

- [Bilder, Zeichnungen und Bitmaps](#)

anzeigen

- [SDI-Anwendungen](#)

darstellen

- [SDI-Anwendungen](#)

laden

- [SDI-Anwendungen](#)

löschen

- [SDI-Anwendungen](#)

Serialisierung

- [SDI-Anwendungen](#)

speichern

- [SDI-Anwendungen](#)

Zeiger

- [SDI-Anwendungen](#)

Adreßoperator

- [C++ im Überblick](#)

auf Ansichtsfenster

- [Multitasking](#)

auf Dokumentobjekt

- [SDI-Anwendungen](#)

auf Kontrollkästchenvariable

- [Multitasking](#)

I-Balken

- [Maus und Tastatur](#)

Indirektionsoperator

- [C++ im Überblick](#)

nicht initialisierte

- [C++ im Überblick](#)

Zeilennummern

- [C++ im Überblick](#)

Zeit

Format

- [Timer](#)

Zeit Siehe auch Datum/Zeit

- [MFC-Hilfsklassen](#)

Zeitspannen

- [MFC-Hilfsklassen](#)

ZU ERLEDIGEN

- [Steuerelemente](#)

Zufallszahlengenerator

- [ActiveX-Steuerelemente](#)

initialisieren

- [DLLs](#)

Zugriffsoperator

- [C++ im Überblick](#)

Zugriffsspezifizierer

- [Dialogfelder](#)

Zugriffsstatus

explizit festlegen

- [Dateizugriff](#)

Zugriffstasten

- [Menüs](#)

Accelerator-Tabelle

- [Funktionen für Webbrowser](#)

Funktionstasten

- [Menüs](#)

Konflikte

- [Steuerelemente](#)

Menüs

- [Menüs](#)

mnemonischer Code

- [Steuerelemente](#)

Nachrichtenzuordnungstabellen

- [Menüs](#)

Tabulator-Reihenfolge

- [Steuerelemente](#)

Umlaute

- [Menüs](#)

virtuelle Tastenbezeichner

- [Funktionen für Webbrowser](#)

WM_KEY

- [Menüs](#)

Zusatztasten

- [Funktionen für Webbrowser](#)

Zurück

Webbrowser

- [Funktionen für Webbrowser](#)

Zusatztasten

Zugriffstasten

- [Funktionen für Webbrowser](#)

Zuweisungen

- [Steuerelemente](#)
 - [C++ im Überblick](#)
-

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Tag G

Die CD zum Buch

Die Service-CD-ROM, die diesem Buch beiliegt, hat drei Unterverzeichnisse:

Unter EBOOKS finden Sie dieses Ihnen vorliegende Buch komplett im HTML-Format. So können Sie z.B. eine Lektion auch mal am Laptop durcharbeiten. Daneben gibt's zwei andere Titel aus dieser Reihe als Bonus: *Active Server Pages in 21 Tagen* und *Visual InterDev in 21 Tagen*.

Die Listings im Unterverzeichnis SOURCE sind getrennt nach Tagen bzw. Anhängen in den Dateien *Lnn.doc* untergebracht, wobei *nn* für die Nummer des Tages (von 01 bis 21) bzw. den Anhang (AA bis AF) steht. Innerhalb der Dateien sind die Listings fortlaufend genau wie im Buch nummeriert. Bei verschiedenen Tagen sind auch die Listings der Übungen enthalten.

Wenn Sie die Beispiele des Buches nachvollziehen wollen, öffnen Sie am besten parallel zu Visual Studio die Datei mit den Listings des jeweiligen Tages in einem Editor, markieren die betreffenden Codezeilen, kopieren sie mit (Strg)+(C) in die Zwischenablage und fügen sie mit (Strg)+(V) an die richtige Stelle im Visual Studio ein. Die Zeilennummern wurden zur besseren Orientierung auch in den Dateien mit den Listings beibehalten und müssen im Visual Studio noch manuell entfernt werden.



Versuchen Sie doch mal zur Übung, in Visual Studio ein Makro zu schreiben, das die Zeilennummern aus dem kopierten Quelltext automatisch entfernt.

Das Verzeichnis DIENSTE schließlich bietet Ihnen u.a. aktuelle Browser-Software.

Tag F

MFC-Hilfsklassen

Auflistungsklassen

Sie werden schnell feststellen, daß in jedem objektorientierten Programm, das Sie schreiben, die Objekte gruppiert und in Auflistungen unterschiedlicher Typen und Größen gespeichert werden müssen. Auch hier kommt wieder die MFC zu Hilfe und stellt Gruppen einfach anzuwendender Klassen und Vorlagen bereit.

Die Auflistungsklassen fallen in drei große Kategorien - Felder, Listen und Tabellen - mit jeweils speziellen Aufgabengebieten.

Felder (oder Arrays, diese Begriffe werden hier gleichberechtigt verwendet) bilden die Hauptstütze der Auflistungsklassen und eignen sich für die Implementierung von Objektcontainern. Jedes Objekt in einem Array hat eine bei null beginnende Position (Index), über die man sich auf das Objekt bezieht.

Die Listenklasse organisiert Elemente in Form einer doppelt verketteten Liste, bei der die Daten sequentiell miteinander verbunden sind. Listen bieten sich vor allem an, wenn man Elemente am Anfang (dem »Kopf«) oder am Ende der Liste schnell hinzufügen oder entfernen muß. Man kann die Liste auch vorwärts oder rückwärts von einem Element zum nächsten durchlaufen.

Bei Tabellen (Maps) verknüpft man ein Schlüsselobjekt, etwa einen String oder eine Zahl, mit einem Wertobjekt, wo Verknüpfungen von Haus aus nur selten oder zufällig vorkommen. Beispielsweise kann man eine Tabelle verwenden, um Objekte mit Postleitzahlen zu verknüpfen. Tabellen eignen sich hervorragend bei schnellen Abfragen von Objekten, wenn der Verknüpfungsschlüssel gegeben ist, und können für die temporäre Speicherung von Daten bei großen Datenbanken verwendet werden.

Feldklassen

MFC stellt mehrere vordefinierte Feldklassen und eine allgemeine Feldvorlage bereit, so daß man Felder erstellen kann, die eigene benutzerdefinierte Objekte aufnehmen. (Auf den letzten Punkt gehen wir später in diesem Kapitel ein.)

Verschiedene vordefinierte Feldklassen bieten einen schnellen und einfachen Zugriff auf gebräuchliche Typen von Variablen und Objekten (siehe Tabelle F.1).

Tabelle F.1: Vordefinierte Array-basierte Klassen

Feldklasse	Typen der gespeicherten Variablen	Numerischer Bereich des Typs

CByteArray	BYTE - vorzeichenlose 8 Bit-Werte	0 bis 255
CWordArray	WORD - vorzeichenlose 16 Bit-Werte	0 bis 65535
CUIntArray	UINT - vorzeichenlose Ganzzahlen mit 32 Bit	0 bis 4 294 967 295
CDWordArray	DWORD - vorzeichenlose Ganzzahlen mit 32 Bit	0 bis 4 294 967 295
CStringArray	CString - Textobjekte	
CObArray	CObject - von CObject abgeleitete Objekte	
CPtrArray	void* - Objektzeiger oder Speicheradressen	

Für jede Feldklasse gibt es mehrere Member-Funktionen, die sich nur durch den Typ der aufgenommenen Variablen unterscheiden. Jede hier behandelte Funktion läßt sich mit allen Feldklassen verwenden, um die Variablen des entsprechenden Typs zu behandeln.

Einer der nützlichsten Aspekte dieser Feldklassen ist deren Fähigkeit, dynamisch zu wachsen. Normale C/C++-Arrays sind in ihrer Größe vordefiniert und lassen sich nur durch umfangreiche Neuzuweisungen von Speicher erweitern. Die Auflistungsklassen verbergen diese Neuzuweisungen, so daß man einfach die Member-Funktion `Add` eines Feldobjekts aufrufen kann, um einen neuen Wert hinzuzufügen. Will man beispielsweise Strings in ein `CStringArray` aufnehmen, verwendet man etwa folgenden Code:

```
CStringArray myStringArray;
myStringArray.Add("Rot");
myStringArray.Add("Grün");
myStringArray.Add("Blau");
```

Die Größe eines Arrays läßt sich dann mit der Funktion `GetSize` ermitteln. Führt man im Anschluß an die obigen Zeilen die folgende Zeile aus, erhält man in `nNumberOfItems` drei Elemente zurück:

```
int nNumberOfItems = myStringArray.GetSize();
```

Ein Array kann man mit der korrespondierenden Funktion `SetSize` auch auf eine bestimmte Größe einstellen. Dabei wird das Array entweder abgeschnitten oder erweitert, um der übergebenen Größe zu entsprechen.

Mit der Funktion `SetAt`, der man einen bei null beginnenden Index und den zu speichernden Wert übergibt, kann man Werte in das Array eintragen. Man muß sicherstellen, daß der Index innerhalb der Feldgrenzen liegt, da `SetAt` ansonsten einen Assertion-Fehler auslöst. Mit der Funktion `GetAt` kann man Werte aus dem Array abrufen. Die Funktion liefert den Wert an der angegebenen Indexposition zurück. Bei einem `CWordArray` setzt man diese Funktionen zum Beispiel folgendermaßen ein:

```
CWordArray myWordArray;
myWordArray.SetSize(20);
myWordArray.SetAt(0,200);
myWordArray.SetAt(19,500);
TRACE("Der Wert an Indexposition 19 lautet %d\n",
myWordArray.GetAt(19));
```

Diese Zeilen setzen das erste Element eines 20elementigen Arrays auf 200 und das letzte auf 500. Die

letzte Zeile zeigt dann das letzte Element - den Wert 500 - an. Mit der Funktion `Add` lässt sich das Array weiter vergrößern. Den größten gültigen Index ermittelt man mit der Funktion `GetUpperBound`. Die Funktion liefert einen bei null beginnenden Index zurück, oder -1, wenn keine Elemente vorhanden sind.

Mit dem Indexoperator `[ ]` kann man Werte an einer bestimmten Indexposition wie bei einem normalen C++-Array setzen und abrufen. Beispielsweise lassen sich die Funktionen `GetAt` und `SetAt` in den obigen Zeilen folgendermaßen durch den Indexoperator ersetzen:

```
myWordArray[0] = 200;
myWordArray[19] = 500;
TRACE("Der Wert an Indexposition 19 lautet %d\n",
myWordArray.GetAt[19]);
```

Mit den Funktionen `InsertAt` und `RemoveAt` kann man Elemente an einer bestimmten Position einfügen bzw. entfernen. Dabei werden alle Elemente ab dieser Position um ein oder mehrere Elemente nach oben bzw. unten verschoben.

Die Funktion `InsertAt` hat zwei Formen: die erste benötigt eine Indexposition und das an dieser Stelle einzufügende Element. Optional kann man eine Anzahl übergeben, um mehrere Kopien des angegebenen Elements einzufügen. Die zweite Form erlaubt es, ein anderes komplettes Array an der angegebenen Indexposition einzufügen.

Die Funktion `RemoveAt` braucht nur einen Parameter, den Index des zu entfernenden Elements. Man kann auch hier einen optionalen zweiten Parameter übergeben, um die entsprechende Anzahl von Elementen zu entfernen. Die verbleibenden Feldelemente werden dann nach unten verschoben, um die Lücke aufzufüllen.

Alle Elemente lassen sich aus einem Array mit der Funktion `RemoveAll` entfernen.



Den Speicher mit `CObArray` und `CPtrArray` verwalten

Bei Objekten, die Sie mit `new` erzeugt und in einem `CObArray` oder `CPtrArray` gespeichert haben, müssen Sie beim Löschen sorgfältig vorgehen, da diese Arrays lediglich Zeiger auf die Elemente (und nicht die Elemente selbst) speichern. Demzufolge entfernt ein Aufruf von `RemoveAll` die Zeiger auf die Objekte, gibt aber nicht den von den Objekten selbst belegten Speicher frei.

Listenklassen

Wie Tabelle F.2 zeigt, gibt es nur drei Kategorien von Listen und eine Vorlage (Template) für Ihre eigenen - später noch zu behandelnden - Typen. Listen mit einfachen Ganzzahlen benötigt man nur selten. Statt dessen haben Sie es eher mit verketteten Listen zu tun, in denen Sie Ihre von `CObject` abgeleiteten Klassen, Zeiger auf C++- Klassen oder Strukturen verwalten.

Tabelle F.2: Die Auflistungsklassen für Listen

Klassenname	Typ der gespeicherten Variablen
<code>CObList</code>	<code>CObject</code> - Zeiger auf von <code>CObject</code> abgeleitete Objekte.
<code>CPtrList</code>	<code>void*</code> - Zeiger auf Speicheradressen, die beliebige Daten aufnehmen.

CStringList	CString - Zeichenfolgen.

Bei verketteten Listen sind mehrere Objekte miteinander in sequentieller Art wie die Waggon eines Zuges verkoppelt. Es gibt eine eindeutige Anfangs- und Endposition, aber jedes andere Element kennt nur seinen unmittelbaren Nachbarn. Eine **POSITION**-Variable protokolliert eine aktuelle Position in einer Liste. Man kann mehrere **POSITION**-Variablen deklarieren, um verschiedene Stellen in derselben Liste zu verfolgen.

Die Member-Funktionen der Listen verwenden dann eine **POSITION**-Variable, um den Anfang (den »Kopf«), das Ende oder das nächste bzw. vorherige Element in der Liste zu ermitteln.

Mit den Funktionen **AddHead** und **AddTail** kann man Elemente in eine Liste am Anfang bzw. am Ende hinzufügen oder mit den Funktionen **InsertBefore** und **InsertAfter** vor bzw. nach einer bestimmten Position einfügen. Alle Funktionen liefern dann einen **POSITION**-Wert zurück, der die Position des neu hinzugefügten Elements angibt.

Das folgende Beispiel konstruiert eine vierelementige Liste von **CString**-Elementen:

```
CStringList listMyStrings;
POSITION pos;
pos = listMyStrings.AddHead("Hand");
listMyStrings.AddTail("Unterarm");
listMyStrings.InsertBefore(pos, "Finger");
listMyStrings.AddTail("Ellbogen");
```

Diese Codezeilen liefern eine verknüpfte Liste von **CStrings**, die von Anfang bis Ende folgendes Aussehen hat:

Finger-Hand-Unterarm-Ellbogen

Man kann auch andere gleichartige Listenobjekte an die Funktionen **AddHead** und **AddTail** übergeben, um eine weitere Liste am Beginn oder Ende der aktuellen Liste hinzuzufügen.

Wenn Sie eine Liste aufgebaut haben, können Sie mit Hilfe einer **POSITION**-Variablen durch die Elemente der Liste laufen. Die Anfangs- oder Endposition läßt sich mit den Funktionen **GetHeadPosition** bzw. **GetTailPosition** ermitteln. Diese Funktionen geben einen **POSITION**-Wert zurück, der die aktuelle Position in der Liste bezeichnet. Dann kann man die **POSITION**-Variable als Referenz an die Funktionen **GetNext** oder **GetPrev** übergeben, um das nächste bzw. vorherige Element in der Liste zu finden. Diese Funktionen liefern dann das betreffende Objekt zurück und passen die aktuelle Position an. Wenn das Ende der Liste erreicht ist, wird die Variable **POSITION** auf **NULL** gesetzt.

Die Zeilen im folgenden Beispiel durchlaufen die oben erzeugte **listMyStrings** und zeigen die Elemente nacheinander an:

```
POSITION posCurrent = listMyStrings.GetHeadPosition();
while(posCurrent) TRACE("%s\n", listMyStrings.GetNext(posCurrent));
```

Bestimmte Listenelemente lassen sich mit der Funktion **Find** aufsuchen, die einen **POSITION**-Wert zurückliefert, wenn der übergebene Suchparameter gefunden wurde. Optional können Sie einen Positionswert übergeben, von dem aus die Suche beginnen soll.

Um zum Beispiel nach dem String **Finger** der oben angelegten Liste zu suchen, ruft man die

Funktion `Find` wie folgt auf:

```
POSITION posFinger = Find("Finger");
```

Die Funktion `Find` liefert einen `NULL`-Wert zurück, wenn das gesuchte Element nicht vorhanden ist.

Mit der Funktion `FindIndex` kann man das *n*te Element vom Beginn der Liste an suchen (wobei man *n* als Parameter übergibt).

Die Anzahl der Elemente in der Liste liefert die Member-Funktion `GetCount`. Die Funktion benötigt keine Parameter und gibt die Anzahl der Elemente zurück.

Der Wert von Elementen an einer bestimmten Position lässt sich mit den Funktionen `GetAt` und `SetAt` abrufen bzw. zurücksetzen. Die Funktionen setzt man ähnlich ein wie die äquivalenten Feldfunktionen, übergibt aber einen `POSITION`-Wert statt eines Feldindexes.

Mit der Funktion `RemoveAt` entfernt man Elemente aus der Liste. Der Funktion übergibt man den `POSITION`-Wert, um das zu entfernende Element zu kennzeichnen. Mit dem folgenden Code löschen Sie zum Beispiel den Eintrag `Finger` aus der Liste des obigen Beispiels:

```
RemoveAt(posFinger);
```

Tabellenklassen

Tabellenklassen (oder Map-Klassen) verbinden einen Typenwert (oder Element) mit einem Schlüsselwert, mit dem man nach dem Element suchen kann. Die verschiedenen Tabellenklassen sowie deren Schlüsselwerte und zugeordneten Elementtypen sind in Tabelle F.3 aufgeführt.

Tabelle F.3: Die Auflistungsklassen für Tabellen (Maps)

Klassenname	Schlüsseltyp	Elementtyp
<code>CMapWordToOb</code>	<code>WORD</code> - vorzeichenloser 16-Bit-Wert	<code>CObject</code> - von <code>CObject</code> abgeleitete Objekte
<code>CMapWordToPtr</code>	<code>WORD</code> - vorzeichenloser 16-Bit-Wert	<code>void*</code> - Zeiger auf Speicher
<code>CMapPtrToPtr</code>	<code>void*</code> - Zeiger auf Speicher	<code>void*</code> - Zeiger auf Speicher
<code>CMapPtrToWord</code>	<code>void*</code> - Zeiger auf Speicher	<code>WORD</code> - vorzeichenloser 16-Bit-Wert
<code>CMapStringToOb</code>	<code>CString</code> - Zeichenfolgen	<code>CObject</code> - von <code>CObject</code> abgeleitete Objekte
<code>CMapStringToPtr</code>	<code>CString</code> - Zeichenfolgen	<code>void*</code> - Zeiger auf Speicher
<code>CMapStringToString</code>	<code>CString</code> - Zeichenfolgen	<code>CString</code> - Textstrings

In eine Tabelle kann man mit der Funktion `SetAt` Elemente einfügen. Der Funktion übergibt man einen Schlüsselwert als ersten Parameter und den Wert des Elementes als zweiten. Wollen Sie zum Beispiel Ihre von `CObject` abgeleiteten Objekte, die mit einem Zeichenfolgenwert indiziert sind, speichern, können Sie die Klasse `CMapStringToOb` verwenden und Elemente wie folgt hinzufügen:

```
CMapStringToOb mapPlanetDetails;
mapPlanetDetails.SetAt("Merkur", new CPlanetDets (4878, 0.054,
```

```
57.91, 87.969));
mapPlanetDetails.SetAt("Venus", new CPlanetDets (12100, 0.815,
108.21, 224.701));
mapPlanetDetails.SetAt("Erde", new CPlanetDets (12756, 1.000,
149.60, 365.256));
```

In diesem Beispiel ist `CPlanetDets` eine von `CObject` abgeleitete Klasse mit einem Konstruktor, der vier Detailparameter für Planeten übernimmt. Den neuen Objekten sind dann die Planetennamen als Schlüssel zugeordnet.

Statt der Funktion `SetAt` können Sie auch den Indexoperator `[ ]` verwenden, indem Sie den Schlüsselwert in den eckigen Klammern angeben:

```
mapPlanetDetails["Mars"] = new CPlanetDets (6762, 0.107, 227.94,
686.98);
```

Nachdem Sie Daten in eine Tabelle eingetragen haben, können Sie sie mit der Member-Funktion `Lookup` wieder abrufen. Der Funktion übergeben Sie den Schlüsselwert als Referenz auf eine Variable, die das zugehörige Element aufnimmt, falls es existiert. Ist das Element nicht vorhanden, liefert die Funktion `Lookup` den Wert `FALSE` zurück. Um zum Beispiel die Angaben über einen Planet aus dem vorherigen Beispiel abzurufen, können Sie etwa folgende Zeilen verwenden:

```
CPlanetDets* pMyPlanet = NULL;
if (mapPlanetDetails.Lookup("Erde", (CObject*)&pMyPlanet))
TRACE("Umlaufzeit = %d Tage\n", pMyPlanet->m_dSidereal);
```

Die Typumwandlung `(CObject*)&` dient dazu, den Objektzeiger `pMyPlanet` in eine allgemeine Zeigerreferenz auf `CObject` umzuwandeln.

Der Funktion `GetCount` liefert die Anzahl der momentan in der Tabelle vorhandenen Elemente zurück. Die Elemente lassen sich durch Aufruf der Funktion `RemoveKey` entfernen. Der Funktion übergibt man den Schlüssel des zu entfernenden Elements:

```
mapPlanetDetails.RemoveKey("Jupiter");
```

Denken Sie daran, die reservierten Objekte zu löschen. `RemoveKey` entfernt einfach den Zeiger auf das Objekt - und nicht das Objekt selbst - und gibt den verwendeten Speicher nicht frei. Durch Aufruf der Funktion `RemoveAll` können Sie alle Elemente auf einmal entfernen.

Mit Hilfe der Funktion `GetNextAssoc` kann man die Liste der Zuordnungen durchlaufen. Die Funktion benötigt Parameter, die eine Variable mit der aktuellen Position referenzieren, eine Schlüsselvariable und eine Elementvariable. Die Position des ersten Elements ermittelt man mit der Funktion `GetFirstPosition`. Die Funktion liefert den `POSITION`-Wert für das erste Element zurück. Um durch die Zuordnungen zu gehen, kann man etwa den folgenden Code verwenden:

```
POSITION pos = mapPlanetDetails.GetStartPosition();
while(pos!=NULL)
{
CString strPlanet;
CPlanet* pMyPlanet;
mapPlanetDetails.GetNextAssoc(pos, strPlanet, CObject*)&pMyPlanet);
TRACE("%s hat einen Durchmesser von %d km\n", strPlanet, pMyPlanet-
>m_dDiameter);
}
```

Nach Rückkehr aus `GetNextAssoc` enthält `pos` die Position für die nächste Zuordnung oder `NULL`, wenn es keine Zuordnungen mehr gibt. Die Schlüssel- und Elementwerte (`strPlanet` und `pMyPlanet` im obigen Beispiel) werden nacheinander auf das jeweilige Schlüssel/Element-Paar gesetzt.

Durch die Fähigkeit der Tabelle, schwach besetzte Datenbestände schnell und effizient abzurufen, ist es oftmals von Vorteil, eine Tabelle als Zwischenspeicher bei einer langsamen Datenbanksuche zu verwenden.

Beispielsweise sind in den folgenden Zeilen die mit `strPlanetName` verbundenen Angaben erforderlich. Beim ersten Aufruf verfügt dieser Code noch nicht über eine abgebildete Version des angeforderten Planeten, so daß man ihn mit `GetPlanetFromSlowDB` suchen muß. Da der Code dann den abgerufenen Planeten in der Tabelle `mapPlanetDetails` speichert, lassen sich beim nächsten Aufruf mit demselben `strPlanetName` die Angaben schnell aus der zwischengespeicherten Version abrufen:

```
CPlanetDets* pMyPlanet = NULL;
if (mapPlanetDetails.Lookup(strPlanetName, (CObject*)&pMyPlanet) ==
FALSE)
{
pMyPlanet = GetPlanetFromSlowDB(strPlanetName);
mapPlanetDetails.SetAt(strPlanetName, pMyPlanet);
}
return pMyPlanet;
```

Dieses Verfahren ist leicht zu implementieren und kann die Geschwindigkeit der Anwendung erhöhen, wenn man mit langsamen Abfragegeräten wie etwa Datenbanken oder Dateien arbeitet.

Benutzerdefinierte Auflistungsklassen

Vielleicht möchten Sie die Auflistungsklassen anpassen, um Ihre eigenen Objekte anstelle der allgemeinen, von `CObject` abgeleiteten Klassen zu verwenden. Die Anpassung bietet mehrere Vorteile, da man ein Feld, eine Liste oder eine Tabelle erstellen kann, die nur den speziellen Typ des Objekts akzeptiert und zurückgibt. Wenn man versehentlich versucht, die falsche Objektart in ein benutzerdefiniertes Feld, eine Liste oder eine Tabelle hinzuzufügen, löst der Compiler eine Fehlermeldung aus, um Sie darüber zu informieren. Der andere Vorteil besteht darin, daß man keine Typumwandlungen von allgemeinen Zeigern auf `CObject*` (das heißt, von einem `CObArray`) zurück auf das verwendete Objekt vornehmen muß.

Diese Art der Anpassung bezeichnet man als ***Typsicherheit***. In großen Programmen kann es unschätzbar sein, bei versehentlichen Zuweisungen der falschen Klasse zu stoppen. Mit einer Gruppe von Vorlagen, `CArray`, `CList` und `CMap`, kann man in einfacher Weise ein Array, eine Liste oder eine Tabelle erstellen, um Objekte nur des spezifizierten Typs zu speichern, zu verwenden und zurückzugeben. Vorlagen (Templates) sind ein kompliziertes Thema, aber Sie brauchen Vorlagen nicht selbst zu schreiben. Die in der Header-Datei `afxtempl.h` definierten und von MFC bereitgestellten Vorlagen genügen für diese typsicheren Auflistungsklassen. In bezug auf den vorliegenden Abschnitt stellt man sich Templates einfach als große Makros vor, die beim Kompilieren auf der Basis Ihrer Parameter eine Menge Code generieren.

Die Vorlagen bieten den Zugriff auf alle normalen Funktionen in den Feld-, Listen- oder Tabellenklassen, die in den vorherigen Abschnitten behandelt wurden. Statt allerdings die auf dem

allgemeinen Objekt `CObject` basierenden Parameter und Rückgabewerte zu verwenden, können Sie Ihre eigenen Typen als Parameter und Rückgabewerte definieren.

Um die Vorlagen in Ihrem Programm einzusetzen, müssen Sie die folgende Header- Zeile in alle Module (`.cpp/.h`-Dateien) einbinden, die auf die Vorlagendefinitionen zurückgreifen:

```
#include "afxtempl.h"
```

Sie können dann Ihre eigene benutzerdefinierte, typsichere Klasse gemäß der folgenden Vorlagensyntax für ein Array von benutzerdefinierten Objekten definieren:

```
CArray<CMYCustomClass*, CMYCustomClass *>myCustomClassArray;
```

Die Symbole `<` und `>` in der obigen Definition sind als spitze Klammern (und nicht als Bedingungsoperatoren für Größer als bzw. Kleiner als) zu interpretieren. Die obige Zeile verwendet die Vorlage `CArray`, um eine Instanz von `myCustomClassArray` zu erzeugen. Der erste Parameter `CMYCustomClass*` spezifiziert Typen von Objektzeigern, die das Array zurückgeben soll, wenn man mit `GetAt` und anderen Zugriffsfunktionen arbeitet. Der zweite Parameter `CMYCustomClass*` legt den Typ fest, der für die Definitionen der Eingabeparameter zu verwenden ist. Dann akzeptieren alle Funktionen, die Objekte speichern, wie etwa `SetAt` und `Add`, nur Zeiger auf Objekte der speziellen `CMYCustomClass`.

Zum Beispiel kann man ein Array erzeugen, das nur Zeiger auf die spezielle Klasse `CPlanetDets` übernimmt und zurückgibt. Die Klasse ist wie folgt definiert (und implementiert):

```
class CPlanetDets : public CObject
{
public:
CPlanetDets(double dDiameter, double dGravity, double dDistFromSun,
double dSidereal):
m_dDiameter(dDiameter), m_dGravity(dGravity),
m_dDistFromSun(dDistFromSun), m_dSidereal(dSidereal) {}
double m_dDiameter,m_dGravity,m_dDistFromSun,m_dSidereal;
};
```

Um ein typsicheres auf `CArray` basierendes Array namens `myPlanetArray` zu deklarieren, können Sie dann den folgenden Code schreiben:

```
CArray<CPlanetDets*, CPlanetDets*> myPlanetArray;
```

Diese Zeile deklariert, daß `myPlanetArray` nur Zeiger auf ein `CPlanetDets`-Objekt akzeptiert und Zeiger auf ein `CPlanetDets`-Objekt zurückgibt. Das neue Array kann man dann wie folgt einsetzen:

```
myPlanetArray.Add(new CPlanetDets (4878, 0.054, 57.91, 87.969));
myPlanetArray.Add(new CPlanetDets (12100, 0.815, 108.21, 224.701));
myPlanetArray.Add(new CPlanetDets (12756, 1.000, 149.60, 365.256));
for(int i=0;i<myPlanetArray.GetSize();i++)
TRACE("Durchmesser = %f\n", myPlanetArray[i]->m_dDiameter);
```

Diese Zeilen erzeugen drei neue Objekte vom Typ `CPlanetDets` und fügen sie in das Array ein. Die letzte Zeile zeigt den Durchmesser im Makro `TRACE` an, ohne daß man den Typ des Rückgabewerts von `myPlanetArray[i]` umwandeln muß, da es sich bereits um einen Zeiger auf den Typ `CPlanetDets*` handelt.

Allerdings vergessen Sie vielleicht im Zuge der Programmentwicklung die genaue Natur von `myPlanetArray` und versuchen, statt dessen ein `CStatic`-Objekt hinzuzufügen:

```
myPlanetArray.Add(new CStatic());
```

Zum Glück bemerkt der Compiler diese Übertretung und löst einen Compiler-Fehler wie etwa den folgenden aus:

```
... error C2664: 'Add' : Konvertierung des Parameters 1 von 'class CStatic *' in 'class CPlanetDets *' nicht moeglich
```

Der Fehler bliebe jedoch unbemerkt, wenn man ein `CObArray` verwendet hätte, um die Planetendaten zu speichern:

```
CObArray myPlanetArray;
```

Das `CStatic`-Objekt läßt sich ohne weiteres zusammen mit den `CPlanetDets`-Objekten speichern, bewirkt aber unvorhersehbares Unheil, wenn man versucht, das `CStatic`-Objekt abzurufen und annimmt, daß es sich um ein `CPlanetDets`-Objekt handelt.

Die Vorlage zur Generierung von typsicheren Listen ist `CList`. Sie weist die gleiche allgemeine Form wie `CArray` auf:

```
CList<CMyCustomClass*, CMyCustomClass *> myCustomClassList;
```

Auch hier ist der erste Parameter der erforderliche Rückgabetypp des Objekts. Der zweite Parameter spezifiziert die akzeptierten Objekttypen für Funktionen, die Elemente für die Speicherung akzeptieren.

Alle Funktionen, die es für Listen gibt, sind auch für Ihre eigenen speziellen typsicheren benutzerdefinierten Listen verfügbar. Auch diese Funktionen prüfen und geben die spezifizierten Typen zurück. Demzufolge sieht der äquivalente Code mit einer Listenklasse für das Speichern der Planetendaten etwa folgendermaßen aus:

```
CList<CPlanetDets*, CPlanetDets*> myPlanetList;
myPlanetList.AddTail(new CPlanetDets (4878, 0.054, 57.91, 87.969));
myPlanetList.AddTail(new CPlanetDets (12100, 0.815, 108.21,
224.701));
myPlanetList.AddTail(new CPlanetDets (12756, 1.000, 149.60,
365.256));
POSITION pos = myPlanetList.GetHeadPosition();
while(pos) TRACE("Durchmesser = %f\n",
myPlanetList.GetNext(pos)->m_dDiameter);
```

Die Vorlage für benutzerdefinierte Tabellen unterscheidet sich von Listen und Arrays darin, daß vier Parameter erforderlich sind: ein Eingabe- und ein Rückgabewert sowohl für den Schlüssel als auch den Elementwert. Die allgemeine Form lautet damit:

```
CMap<MyType, MyArgType, CMyCustomClass *, CMyCustomClassArg *>
myCustomClassMap;
```

Der erste Parameter, `MyType`, legt den intern gespeicherten Schlüsselwert für jede Tabellenzuordnung fest. Das kann einer der Basistypen wie `int`, `WORD`, `DWORD`, `double`, `float` oder `CString` sein oder ein Zeiger auf einen eigenen speziellen Typ.

Der zweite Parameter, `MyArgType`, spezifiziert den Argumenttyp, der bei der Übergabe der

Schlüsselwerte in und aus Tabellenfunktionen zu verwenden ist.

Mit dem dritten Parameter, `CMyCustomClasss *`, legt man fest, wie die internen Elementwerte zu speichern sind (als spezielle typsichere Zeiger auf Ihre Objekte).

Der vierte Parameter, `CMyCustomClassArg *`, spezifiziert den Argumenttyp, der für die Übergabe der Elementwerte in und aus den Tabellenfunktionen zu verwenden ist. Um beispielsweise die Daten der Planeten mit deren Namen zu verbinden, könnte man folgendes kodieren:

```
CMap<CString, LPCSTR, CPlanetDets*, CPlanetDets*> myPlanetMap;
myPlanetMap.SetAt("Merkur",
new CPlanetDets(4878, 0.054, 57.91, 87.969));
myPlanetMap.SetAt("Venus",
new CPlanetDets(12100, 0.815, 108.21, 224.701));
myPlanetMap.SetAt("Erde",
new CPlanetDets(12756, 1.000, 149.60, 365.256));
CPlanetDets* pPlanet = NULL;
if (myPlanetMap.Lookup("Venus", pPlanet))
TRACE("Durchmesser = %f\n", pPlanet->m_dDiameter);
```

Die Tabellendeklaration gibt an, daß die Objekte intern als `CStrings` zu speichern sind, aber `LPCSTR` (Zeiger auf konstante Zeichenarrays) als Übergabewerte in und aus der Tabelle zu verwenden sind. Die Angaben zu den Planeten selbst werden sowohl intern gespeichert als auch als Zeiger auf `CPlanetDets`-Objekte zugänglich gemacht (wie zum Beispiel als `CPlanetDets*`).



Mögliche Probleme beim Einsatz von Hash-Schlüsseln für Tabellen

Beachten Sie die Umwandlung der übergebenen Parameter in das interne Speichersystem auf der Basis von Hash-Schlüsseln. Wenn Sie zum Beispiel den `CString` im obigen Beispiel durch einen anderen `LPCSTR` für das interne Speicherobjekt ersetzen müssen, scheitert die Funktion `Lookup` bei der Suche nach "Venus", da sie die Zeigerwerte (auf verschiedene Instanzen von "Venus") und nicht den Inhalt der Strings vergleicht.

Klassen zur Behandlung von Koordinaten

Da Windows eine grafisch orientierte Umgebung darstellt, muß man oftmals Punktpositionen, Rechtecke und Größen speichern. Drei MFC-Klassen unterstützen das Speichern und Manipulieren dieser Koordinaten: `CPoint`, `CRect` und `CSize`. Jede Klasse verfügt über mehrere Elementfunktionen und überladene Operatoren, die den größten Teil der Arbeit beim Addieren, Konstruieren und Finden von Ableitungen dieser Koordinaten realisieren.

Darüber hinaus verstehen verschiedene MFC- und GDI-Funktionen diese Typen oder die zugrundeliegenden Typen als Parameterwerte, so daß man keine umständlichen Operationen auszuführen hat, um Werte an diese Funktionen zu übergeben.

Die Klasse `CPoint`

`CPoint` kapselt eine `POINT`-Struktur, die lediglich eine x- und eine y-Position speichert, um einen Punkt auf einer zweidimensionalen Oberfläche darzustellen. Auf die Elemente x und y kann man immer direkt zugreifen, um deren aktuelle Werte zu erhalten oder zu setzen:

```
CPoint ptEins;
ptEins.x = 5;
ptEins.y = 20;
TRACE("Koordinate = (%d, %d)\n", ptEins.x, ptEins.y);
```

Diese Werte setzen Sie, wenn Sie ein `CPoint`-Objekt konstruieren, indem Sie die Werte an einen der verschiedenen Konstruktoren von `CPoint` übergeben, wie sie Tabelle F.4 zeigt.

Tabelle F.4: Konstruktortypen für die Klasse `CPoint`

Konstruktordefinition	Beschreibung
<code>CPoint()</code>	Konstruiert ein nicht initialisiertes Objekt.
<code>CPoint(POINT ptInit)</code>	Kopiert die Einstellungen aus einer <code>POINT</code> -Struktur oder einem anderen <code>CPoint</code> -Objekt.
<code>CPoint(int x, int y)</code>	Initialisiert das Objekt mit den Parameterwerten von <code>x</code> und <code>y</code> .
<code>CPoint(DWORD dwInit)</code>	Verwendet die unteren 16 Bit für den <code>x</code> -Wert und die höherwertigen 16 Bit für den <code>y</code> -Wert.
<code>CPoint(SIZE sizeInit)</code>	Kopiert die Einstellungen aus einer <code>SIZE</code> -Struktur oder einem <code>CSize</code> -Objekt.

Zum Beispiel können Sie die letzten Beispielzeilen durch die folgenden ersetzen und das gleiche Ergebnis erzielen:

```
CPoint ptEins(5,20);
TRACE("Koordinate = (%d, %d)\n", ptEins.x, ptEins.y);
```

Einer der nützlichsten Aspekte der Klasse `CPoint` sind die vielen überladenen Operatoren. Indem man die Operatoren `+`, `-`, `+=` und `-=` mit anderen `CPoint`-, `CRect`- oder `CSize`-Objekten verwendet, kann man Koordinatenpaare zu/von anderen Koordinatenpaaren, Rechtecken oder Größen addieren/subtrahieren. Der längere herkömmliche Weg, um zwei Punkte voneinander zu subtrahieren, um einen dritten zu erhalten, sieht zum Beispiel folgendermaßen aus:

```
CPoint ptEins(5,20);
CPoint ptZwei(25,40);
CPoint ptDrei;
ptDrei.x = ptZwei.x - ptEins.x;
```

Das Ganze läßt sich mit den überladenen Operatoren folgendermaßen vereinfachen:

```
CPoint ptEins(5,20);
CPoint ptZwei(25,40);
CPoint ptDrei = ptZwei - ptEins;
```

Man kann auch die Koordinaten des einen Punktes zu einem anderen addieren:

```
ptZwei += ptEins;
```

Mit den überladenen logischen Operatoren `==` und `!=` lassen sich auch Vergleiche ausführen. Um zum Beispiel zu prüfen, ob `ptZwei` sowohl im `x`- als auch im `y`-Wert mit `ptEins` gleich ist, schreibt man folgenden Code:

```
if(ptEins == ptZwei) TRACE("Die Punkte sind identisch.");
```


Die Funktion `Offset` addiert einen Verschiebungswert (Offset), der durch die übergebenen `x`- und `y`-Werte, eine `CPoint`-Klasse, eine `POINT`-Struktur, eine `CSize`-Klasse oder eine `SIZE`-Struktur spezifiziert ist. Demzufolge sind die beiden folgenden Zeilen funktionell identisch:

```
ptEins.Offset(75, -15);
ptEins-=CPoint(-75, 15);
```

Die Klasse `CRect`

Die Klasse `CRect` kapselt eine `RECT`-Struktur, um zwei Koordinatenpaare aufzunehmen, die ein Rechteck durch die Punkte der linken oberen und unteren rechten Ecke beschreiben. Ein `CRect`-Objekt können Sie mit mehreren Konstruktoren erstellen, wie sie Tabelle F.5 zeigt.

Tabelle F.5: Konstruktortypen für die Klasse `CRect`

Konstruktordefinition	Beschreibung
<code>CRect()</code>	Konstruiert ein nicht initialisiertes Objekt.
<code>CRect(const RECT& rcInit)</code>	Kopiert die Einstellungen aus einer anderen <code>RECT</code> -Struktur oder einem <code>CRect</code> -Objekt.
<code>CRect(LPRECT lprcInit)</code>	Kopiert die Einstellungen über einen <code>RECT</code> - oder <code>CRect</code> -Zeiger.
<code>CRect(int l, int t, int r, int b)</code>	Initialisiert die Koordinaten des linken, oberen, rechten und unteren Parameters.
<code>CRect(POINT point, SIZE size)</code>	Initialisiert von einer <code>POINT</code> -Struktur oder einem <code>CPoint</code> -Objekt und einer <code>SIZE</code> -Struktur oder einem <code>CSize</code> -Objekt.
<code>CRect(POINT ptTL, POINT ptBR)</code>	Initialisiert von einem <code>POINT</code> links oben und einem <code>POINT</code> rechts unten.

Nachdem Sie ein `CRect`-Objekt konstruiert haben, können Sie einzeln auf die Elemente `top`, `left`, `bottom` und `right` zugreifen, indem Sie die Typumwandlung (`LPRECT`) verwenden, um die Werte in eine `RECT`-Struktur wie in den folgenden Zeilen umzuwandeln:

```
CRect rcEins(15, 15, 25, 20);
((LPRECT)rcEins)->bottom += 20;
TRACE("Rechteck: (%d,%d) - (%d,%d)",
((LPRECT)rcEins)->left, ((LPRECT)rcEins)->top,
((LPRECT)rcEins)->right, ((LPRECT)rcEins)->bottom);
```

Alternativ können Sie auf die Elemente entweder über den `CPoint` für oben links oder den `CPoint` für unten rechts zugreifen. Die Funktionen `TopLeft` und `BottomRight` liefern Referenzen auf diese Elementobjekte zurück. Wenn man entweder auf den Punkt oben links oder unten rechts zugreift, kann man die Punkte dann mit einer der `CPoint`-Funktionen aus dem vorherigen Abschnitt manipulieren. Beispielsweise sind die folgenden Zeilen funktionell zu den vorherigen identisch, unterscheiden sich aber darin, daß sie das Rechteck mit `CPoint`-Objekten konstruieren und darauf zugreifen:

```
CRect rcEins(CPoint(15,15),CPoint(25,20));
rcEins.BottomRight().y += 20;
TRACE("Rechteck: (%d,%d) - (%d,%d)",
rcOne.TopLeft().x,rcOne.TopLeft().y,
rcOne.BottomRight().x,rcOne.BottomRight().y);
```

Mit der Funktion `SetRect` können Sie auch die Koordinaten setzen, wobei Sie vier Integer-Werte für die x- und y-Koordinaten der oberen linken und der unteren rechten Ecke übergeben. Die Funktion `SetRectEmpty` setzt alle Koordinaten auf null, um ein NULL-Rechteck zu erzeugen. Die Funktion `IsRectNull` liefert `TRUE`, wenn sie auf einem derartigen NULL-Rechteck aufgerufen wird, und `IsRectEmpty` gibt `TRUE` zurück, wenn sowohl die Breite als auch die Höhe gleich null sind (selbst wenn einzelne Werte ungleich null sind).

Mehrere Hilfsfunktionen unterstützen die Berechnung verschiedener Aspekte der Rechteckgeometrie. Die Breite und Höhe kann man mit den Funktionen `Width` bzw. `Height` ermitteln. Beide Funktionen liefern den diesbezüglichen Integer-Wert zurück. Alternativ kann man ein `CSize` suchen, das sowohl Breite als auch Höhe repräsentiert, indem man die Funktion `Size` aufruft. Beispielsweise zeigt die folgende Zeile die Breite und Höhe des Rechtecks `rcEins` an:

```
TRACE("Breite = %d, Höhe = %d\n", rcEins.Width(), rcEins.Height());
```

Oftmals muß man den Punkt im Zentrum des Rechtecks kennen. Dazu kann man die Funktion `CenterPoint` aufrufen, die ein `CPoint`-Objekt zurückgibt, das den Mittelpunkt des Rechtecks darstellt.

Das folgende Beispiel bestimmt mit dieser Funktion den Mittelpunkt des Client-Bereichs eines Fensters und zeichnet an dieser Stelle einen Punkt:

```
CRect rcClient;
GetClientRect(&rcClient);
dc.SetPixel(rcClient.CenterPoint(),0);
```

Mit den Funktionen `UnionRect` und `IntersectRect` kann man auch die Vereinigungs- bzw. Schnittmenge zweier Rechtecke ermitteln. Beide Funktionen übernehmen zwei Quellrechtecke als Parameter und setzen die Koordinaten des aufrufenden `CRect`-Objekts auf die Vereinigung oder den Schnitt. Die Vereinigung ist das kleinste Rechteck, das die beiden Quellrechtecke umschließt. Der Schnitt ist das größte Rechteck, daß von beiden Quellrechtecken umschlossen wird. Die Zeichnung in Abbildung F.1 zeigt Vereinigung und Schnitt zweier Quellrechtecke mit den Bezeichnern A und B.

Zeichnung kommt nach.

Abbildung F.1: Vereinigungs- und Schnittmenge zweier Rechtecke

Die folgenden Zeilen berechnen den Schnitt und die Vereinigung der Quellrechtecke `rcEins` und `rcZwei`:

```
CRect rcEins(10,10,100,100);
CRect rcZwei(50,50,150,200);
CRect rcUnion, rcIntersect;
rcUnion.UnionRect(rcEins, rcZwei);
rcIntersect.IntersectRect(rcEins, rcZwei);
```

Wenn Sie diesen Code ausführen, wird `rcUnion` auf die Koordinaten (10,10)- (150,200) und `rcIntersect` auf die Koordinaten (50,50)-(100,100) gesetzt.

Mit der Funktion `SubtractRect` kann man ein Rechteck von einem anderen abziehen. Das Ergebnis ist das kleinste Rechteck, das alle Punkte enthält, die nicht von den beiden Quellrechtecken geschnitten werden (oder der kleinste nicht überlappende Bereich). Wenn Sie beispielsweise die folgenden Zeilen in eine `OnPaint`-Behandlungsroutine einfügen, können Sie sich die Wirkung von `SubtractRect` ansehen. Der Code zieht `rcZwei` von `rcEins` ab und liefert `rcDrei`. Das Ergebnis der Subtraktion ist der Bereich, der am unteren Rand der Zeichnung in Blau dargestellt ist, wie es Abbildung F.2 zeigt.

```
CRect rcEins(10,10,220,220), rcZwei(50,50,150,260), rcDrei;
rcDrei.SubtractRect(rcZwei, rcEins);
dc.FillSolidRect(rcEins, RGB(255,0,0)); // Rot
dc.FillSolidRect(rcZwei, RGB(0,255,0)); // Grün
dc.FillSolidRect(rcDrei, RGB(0,0,255)); // Blau
```

Wenn Sie diesen Code ausführen, enthält das resultierende Rechteck `rcDrei` die Koordinaten (50,220)-(150,26).

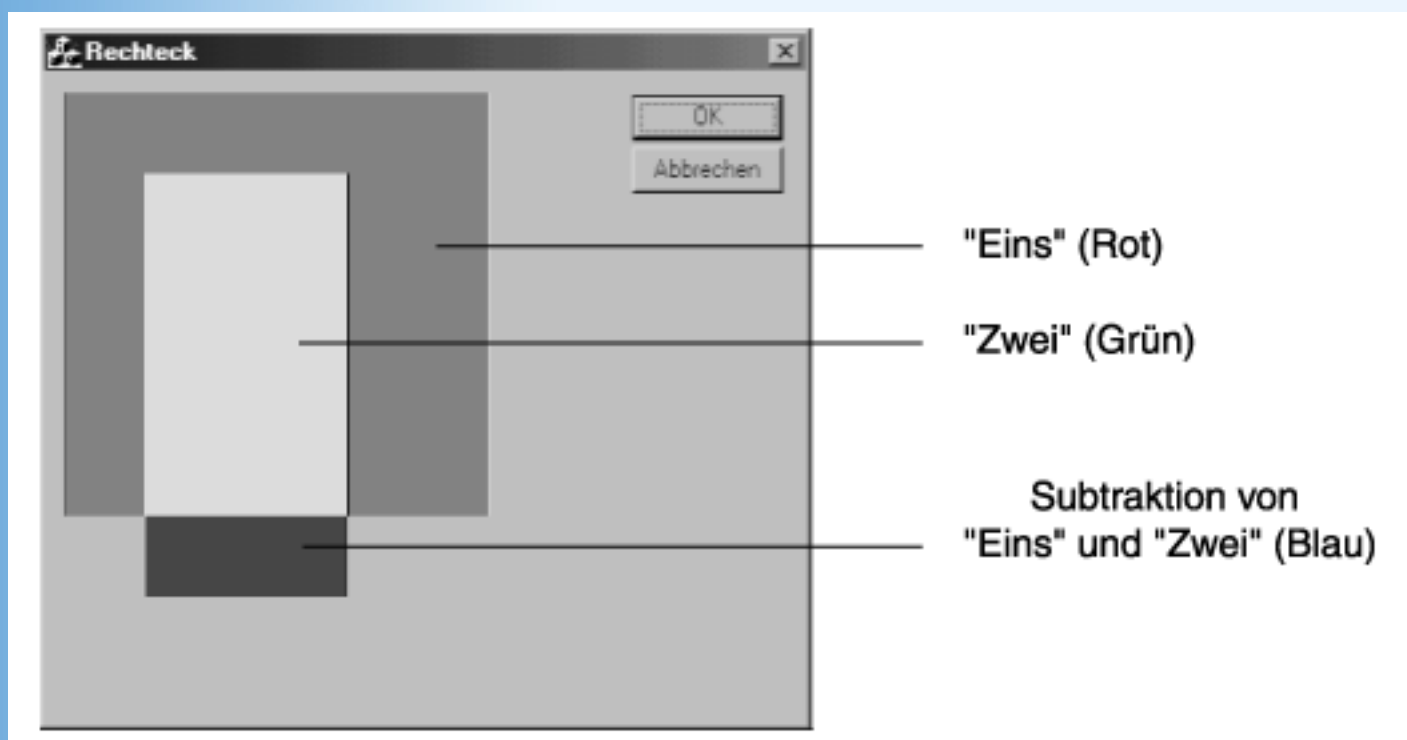


Abbildung F.2:
Die Wirkung einer Subtraktionsoperation auf zwei teilweise überlappende Rechtecke

Die Größe eines Rechtecks lässt sich mit den Funktionen `InflateRect` und `DeflateRect` vergrößern bzw. verkleinern. Beide Funktionen haben mehrere Formen, die verschiedene Typen von Parametern gemäß Tabelle F.6 akzeptieren.

Tabelle F.6: Parameterformen für `InflateRect` und `DeflateRect`

Parameter	Beschreibung
(int x, int y)	Vergrößert oder verkleinert die linke und rechte Seite nach dem Wert x sowie die obere und untere Seite nach dem Wert y.
(SIZE size)	Vergrößert oder verkleinert die linke und rechte Seite nach <code>size.cx</code> sowie die obere und untere Seite nach dem Wert von <code>size.cy</code> .

(LPCRECT lpRect)	Vergrößert jede Seite um die entsprechenden Werte von <code>left</code> , <code>top</code> , <code>right</code> und <code>bottom</code> der Struktur <code>lpRect</code> .
(int l, int t, int r, int b)	Vergrößert jede Seite um die entsprechenden Werte für links (<code>l</code>), oben (<code>t</code>), rechts (<code>r</code>) und unten (<code>b</code>).

Der folgende Beispielcode vergrößert `rcEins` und verkleinert `rcZwei`:

```
CRect rcEins(10,10,100,100);
CRect rcZwei(50,50,150,200);
rcEins.InflateRect(5,5);
rcZwei.DeflateRect(10,20,30,40);
```

Die Ausführung dieser Zeilen setzt `rcEins` auf die Koordinaten (5,5)-(105,105) und `rcZwei` auf die Koordinaten (60,70)-(120,160).

Eine Trefferprüfung läßt sich ausführen, indem man ermittelt, ob ein bestimmter Punkt (etwa bei einem Mausklick) innerhalb der Grenzen eines Rechtecks liegt. Dazu ruft man die Funktion `PtInRect` auf und übergibt den zu testenden Punkt. Wenn der Punkt innerhalb des Rechtecks liegt, gibt die Funktion `TRUE` zurück, andernfalls `FALSE`.

In den folgenden Zeilen wird die Meldung `Treffer!` - `ptTest1` angezeigt, da `ptTest1` innerhalb des Testbereichs von `rcTestArea` liegt, während `ptTest2` nicht in diesen Bereich fällt. Demzufolge liefert `PtInRect` den Wert `TRUE` für `ptTest1` und `FALSE` für `ptTest2`:

```
CRect rcTestArea(10,20,440,450);
CPoint ptTest1(200,200), ptTest2(500,500);
if (rcTestArea.PtInRect(ptTest1)) AfxMessageBox("Treffer! - ptTest1");
if (rcTestArea.PtInRect(ptTest2)) AfxMessageBox("Treffer! - ptTest2");
```

Für `CRect`-Objekte gibt es verschiedene überladene Operatoren, die in Tabelle F.7 aufgeführt sind.

Tabelle F.7: Überladene Operatoren für CRect-Objekte

Operator	Beschreibung
=	Kopiert wie bei einer normalen numerischen Zuweisung alle Koordinaten vom rechten Operanden (Rechteck) in das linke Rechteck.
+	Der Operator führt entweder eine Verschiebung der Rechteckposition aus, wenn zum Rechteck ein <code>CPoint</code> - oder <code>CSize</code> -Objekt addiert wird, oder vergrößert die Koordinaten für die entsprechenden Seiten, wenn ein <code>CRect</code> -Objekt addiert wird.
-	Analog zu <code>+</code> , außer daß die Koordinaten in negativer Richtung verschoben oder bei einem <code>CRect</code> verkleinert werden.
+=	Wirkung wie bei <code>+</code> , beeinflusst aber nur das aktuelle Rechteck.
-=	Wirkung wie bei <code>-</code> , beeinflusst aber nur das aktuelle Rechteck.
&	Erzeugt ein Rechteck als Schnittmenge zweier Rechteckoperanden.
	Erzeugt ein Rechteck als Vereinigungsmenge zweier Rechteckoperanden.

&=	Wirkung wie &, beeinflusst aber nur das aktuelle Rechteck.
=	Wirkung wie bei , beeinflusst aber nur das aktuelle Rechteck.
==	Liefert TRUE, wenn die Rechtecke identisch sind, ansonsten FALSE.
!=	Liefert FALSE, wenn die Rechtecke identisch sind, andernfalls TRUE.

Die folgenden Zeilen zeigen, wie man mit überladenen **CRect**-Operatoren das Rechteck **rcStart** manipuliert:

```
CRect rcStart(10,10,100,100);
rcStart = rcStart + CPoint(5,5);
rcStart -= CSize(5,5);
rcStart += CRect(1,2,3,4);
if (rcStart == CRect(9,8,103,104)) AfxMessageBox("TRUE");
```

Die letzte Bedingung liefert TRUE, weil die Koordinaten nach Ausführung der Anweisungen auf (9,8)-(103,104) gesetzt sind. Demzufolge erscheint das Meldungsfeld.



Die Funktion **NormalizeRect**

Bei manchen Operationen entstehen Werte für den Punkt oben links, die größer als für den Punkt unten rechts sind. In diesem Fall kann die Breite oder Höhe einen negativen Wert annehmen, so daß andere Funktionen einen Fehler liefern. Wenn Sie mit derartigen Ergebnissen rechnen müssen, rufen Sie die Funktion **NormalizeRect** auf, die die Koordinaten korrigiert, so daß die Koordinaten für den Punkt oben links kleinere Werte als die Koordinaten für den Punkt unten rechts aufweisen.

Die Klasse **CSize**

Die Klasse **CSize** kapselt die Struktur **SIZE** und bietet verschiedene Konstruktoren und überladene Operatoren, mit denen sich die internen Werte **cx** und **cy**, die eine Größe definieren, manipulieren lassen. Tabelle F.8 zeigt die Konstruktoren, mit denen man eine Instanz eines **CSize**-Objekts erzeugen kann.

Tabelle F.8: Konstruktortypen für die Klasse **CSize**

Konstruktordefinition	Beschreibung
CSize()	Erzeugt ein nicht initialisiertes CSize -Objekt.
CSize(SIZE sizeInit)	Kopiert die Werte cx und cy aus einem anderen CSize -Objekt oder einer SIZE -Struktur.
CSize(initCX, initCY)	Initialisiert das Objekt mit initCX für die horizontale Größe und initCY für die vertikale.
CSize(POINT ptInit)	Initialisiert das Objekt mit den Werten x und y aus einem CPoint -Objekt oder einer POINT -Struktur.

<code>CSize(DWORD dwSize)</code>	Setzt den Wert <code>cx</code> auf das niederwertige Wort (die unteren 16 Bit) von <code>dwSize</code> und <code>cy</code> auf das höherwertige Wort (die oberen 16 Bit).

Die Elemente `cx` und `cy` kann man direkt manipulieren:

```
CSize tstSize(10,10);
tstSize.cx = tstSize.cy * 2;
```

Die einzigen Funktionen, die die Klasse `CSize` bietet, sind die überladenen Operatoren gemäß Tabelle F.9.

Tabelle F.9: Überladene Operatoren für `CSize`

Operator	Beschreibung
+	Addiert zwei <code>SIZE</code> -Objekte.
-	Subtrahiert ein <code>SIZE</code> -Objekt von einem anderen.
+=	Addiert ein <code>SIZE</code> -Objekt.
-=	Subtrahiert ein <code>SIZE</code> -Objekt.
==	Bestimmt, ob die beiden Größen gleich sind, und liefert in diesem Fall <code>TRUE</code> zurück.
!=	Bestimmt, ob die beiden Größen unterschiedlich sind, und liefert in diesem Fall <code>TRUE</code> zurück.

Diese Operatoren kann man wie normale arithmetische Operatoren einsetzen. Sie beeinflussen sowohl `cx` als auch `cy`, wie es die folgenden Zeilen zeigen, die den Inhalt von `tstSize` manipulieren:

```
CSize tstSize(10,15);
tstSize += tstSize + tstSize - CSize(1,2);
if (tstSize == CSize(29,43)) AfxMessageBox("TRUE");
```

Bei Ausführung des Codes zeigt das Meldungsfeld `TRUE`, weil `tstSize` am Ende eine Größe von 29 mal 43 hat.

Klassen zur Behandlung der Zeit

In vielen Anwendungen sind Datums- und Zeitwerte zu speichern. Manchmal muß man die verstrichene Zeit und Zeitspannen zwischen gespeicherten Datums-/Zeitwerten berechnen und diese Werte in verständliche Zeichenfolgen formatieren können.

MFC stellt vier Klassen bereit, um Datum und Uhrzeit manipulieren und speichern zu können. Ursprünglich gab es nur zwei Klassen: `CTime` und `CTimeSpan`, die auf dem unter UNIX üblichen System `time_t` (langer 4 Byte-Wert) basieren (die Anzahl der seit 1970 verstrichenen Sekunden). Allerdings erweist sich die Auflösung von 1 Sekunde und der begrenzte Datumsbereich zwischen 1970 und 2038 bei vielen Anwendungen als unzureichend. Aus diesen Gründen wurden die beiden neuen Klassen `COleDateTime` und `COleDateTimeSpan` eingeführt, auf die man vorzugsweise statt `CTime` und `CTimeSpan` in neueren Anwendungen zurückgreifen sollte.

`COleDateTime` basiert auf einer zugrundeliegenden `DATE`-Struktur (die eigentlich nur ein

`double`-Wert ist). Durch die größere Kapazität dieses Typs kann `COleDateTime` einen Datumsbereich zwischen dem 1. Januar 100 und dem 31. Dezember 9999 bei einer Auflösung von 1 Millisekunde abdecken. Die Differenz zwischen zwei `COleDateTime`-Werten läßt sich mit dem `COleDateTimeSpan`-Objekt darstellen und manipulieren.

Aufgrund der Ähnlichkeit zwischen der Klasse `CTime` und der neueren `COleDateTime` beschreiben die folgenden Abschnitte lediglich `COleDateTime`, auch wenn viele der Funktionen in den Versionen von `CTime` identisch sind.



CTime bei Datenbanken

Bei ODBC-basierten Datenbanken kann es bequemer sein, mit `CTime` zu arbeiten, da die RFX-Transfermakros des Recordsets direkt nur mit `CTime`-Objekten umgehen und `COleDateTime`-Objekte ohne Umwandlungen nicht bearbeiten können. Wenn Sie mit DAO-Datenbanken arbeiten, können Sie `COleDateTime` ohne Umwege einsetzen.

Die Klasse COleDateTime

Die Verbindung von `COleDateTime` mit OLE ergibt sich aus der Tatsache, daß man diese Klasse mit der Struktur `VARIANT` einsetzen kann, die man häufig in der OLE-Automatisierung antrifft. Durch den breiten Bereich von Datums-/Zeitwerten muß `COleDateTime` insbesondere in OLE-Umgebungen in der Lage sein, zwischen all diesen verschiedenen Typen konvertieren zu können. Diese Unterstützung spiegelt sich in den zahlreichen Konstruktorformen wider, die Tabelle F.10 zeigt.

Tabelle F.10: Mit COleDateTime verwendete Konstruktortypen

Konstruktordefinition	Beschreibung
<code>COleDateTime()</code>	Erzeugt ein nicht initialisiertes <code>COleDateTime</code> -Objekt.
<code>COleDateTime(const COleDateTime& datesrc)</code>	Kopiert die Werte aus einem anderen <code>COleDateTime</code> -Objekt.
<code>COleDateTime(int nYear, int nMonth, int nDay, int nHour, int nMinute, int nSecond)</code>	Initialisiert Datum und Uhrzeit mit den übergebenen Werten.
<code>COleDateTime(const VARIANT& varSrc)</code>	Konvertiert einen Datums-/Zeitwert aus einer <code>VARIANT</code> -Struktur.
<code>COleDateTime(DATE dtSrc)</code>	Kopiert einen Datums-/Zeitwert aus einer <code>DATE</code> -Struktur.
<code>COleDateTime(time_t timeSrc)</code>	Kopiert einen Datums-/Zeitwert aus einer <code>time_t</code> -Struktur im Stil von UNIX.
<code>COleDateTime(WORD wDosDate)</code>	Kopiert einen Datums-/Zeitwert aus den <code>WORD</code> -Werten <code>wDosTime</code> im Stil von MS-DOS.
<code>COleDateTime(const SYSTEMTIME& systimeSrc)</code>	Kopiert einen Datums-/Zeitwert aus einer <code>SYSTEMTIME</code> -Struktur.

COleDateTime(const FILETIME& filetimeSrc)

Kopiert einen Datums-/Zeitwert aus einer **FILETIME**-Struktur.

Wenn Sie **COleDateTime** mit einem gültigen Datums-/Zeitwert konstruiert haben, ist das Objekt mit einem Statusflag für den gültigen Zustand markiert (**COleDateTime::valid**). Andernfalls ist das Statusflag ungültig (**COleDateTime::invalid**). Dieser Status läßt sich mit der Elementfunktion **GetStatus** ermitteln. Die Funktion liefert den relevanten Flagwert zurück. Man kann das Flag auch explizit durch Übergabe des gewünschten Wertes an die Funktion **SetStatus** setzen.

Das Statusflag wird auch aktualisiert, wenn man Datums-/Zeitwerte in das Objekt mit Hilfe der Funktion **SetDateTime** setzt. Die Funktion übernimmt sechs Parameter für das Jahr (100 bis 9999), den Monat (1 bis 12), den Tag (1 bis 31), die Stunde (0 bis 23), die Minute (0 bis 59) und die Sekunde (0 bis 59). Die Datums- oder Zeitkomponenten lassen sich auch separat festlegen: mit der Funktion **SetDate**, der man Jahr, Monat und Tag übergibt, bzw. mit der Funktion **SetTime**, der man nur die Werte für Stunde, Minute und Sekunde übergibt.

Die aktuelle Systemzeit läßt sich mit der statischen Funktion **GetCurrentTime** ermitteln und mit dem überladenen Operator **=** in ein **COleDateTime**-Objekt übertragen:

```
COleDateTime dtCurrent;
dtCurrent = COleDateTime::GetCurrentTime();
```

Nach Ausführung dieser Zeilen enthält **dtCurrent** die aktuelle Systemzeit (Datum und Uhrzeit) des Computers.

Die gleichen Werte (in den gleichen Bereichen) lassen sich mit den Funktionen **GetYear**, **GetMonth**, **GetDay**, **GetHour**, **GetMinute** und **GetSecond** abrufen. Weiterhin gibt es die abgeleiteten Funktionen **GetDayOfWeek** und **GetDayOfYear**. Die Funktion **GetDayOfWeek** liefert den Wochentag mit den Werten 1 bis 7 zurück, wobei die 1 für Sonntag steht. Die Funktion **GetDayOfYear** liefert einen Wert im Bereich von 1 bis 366, beginnend mit dem 1. Januar.

Mit der Funktion **Format** läßt sich ein **CString** erstellen, der die Werte im gewünschten Anzeigeformat ausgibt. Diese Funktion gehört mit zu den nützlichsten **COleDateTime**-Funktionen, da man verschiedenartige - in Tabelle F.11 aufgeführte - Formatcodes übergeben kann, um das genaue Format festzulegen. Die Codes übergibt man entweder als String oder als Bezeichner von Stringressourcen. Um verschiedene Aspekte der Formatierung zu berücksichtigen, kann man verschiedene Einzelcodes verbinden.

Die Werte werden außerdem durch die Ländereinstellungen modifiziert. Das bezieht sich auf die Namen von Tagen und Monaten, die Schreibweise des Datums und die Angabe der Uhrzeit (24 Stunden, AM/PM).

Tabelle F.11: Formatcodes für die Formatierung der Textausgaben von COleDateTime

Code	Beispiel	Beschreibung
%a	Di	abgekürzter Wochentag
%A	Dienstag	Wochentag
%b	Apr	abgekürzter Monat

%B	April	Monat
%c	20.04.99 18:05:01	Datum und Uhrzeit gemäß aktueller Ländereinstellung
%d	04	Tag des Monats (01 bis 31)
%H	18	Stunde (00 bis 23) im 24-Stunden-Format
%I	06	Stunde (01 bis 12) im 12-Stunden-Format
%j	110	Tag des Jahres
%m	04	Monat (01 bis 12)
%M	05	Minute (01 bis 59)
%p	PM	AM/PM bei lokalen Einstellungen
%S	01	Sekunde (01 bis 59)
%U	16	Woche des Jahres (00 bis 51), wobei der Sonntag als erster Tag der Woche gilt
%w	2	Wochentag (0 bis 6), wobei 0 für Sonntag steht
%W	16	Woche des Jahres (00 bis 51), wobei Montag als erster Tag der Woche gilt
%x	20.04.99	Datum im aktuellen lokalen Format
%X	18:05:01	Zeit im aktuellen lokalen Format
%y	99	zweistelliges Jahr (00 bis 99)
%Y	1999	vierstelliges Jahr (0100 bis 9999)
%z oder %Z	MEZ (Mitteleurop. Sommerzeit)	Name der Zeitzone/Abkürzung
% %	%	Prozentzeichen
%#c	Dienstag, 20. April 1999 18:05:01	Langes Datum und Uhrzeit im aktuellen lokalen Format
%#x	Dienstag, 20. April 1999	Langes Datum im aktuellen lokalen Format

Die folgenden Zeilen generieren ein Meldungsfeld, das Datum und Uhrzeit des Computers anzeigt:

```
COleDateTime dtCurrent;
dtCurrent = COleDateTime::GetCurrentTime();
AfxMessageBox(dtCurrent.Format("Heute ist %a %b %d, %Y"));
```

Die Ausführung dieser Zeilen zeigt die Systemzeit im Meldungsfeld mit dem folgenden Format an:

Heute ist Di Apr 20, 1999

Mit `COleDateTime` können Sie versuchen, ein Datum und eine Uhrzeit zu bestimmen, indem Sie `ParseDateTime` aufrufen und einen zu analysierenden String sowie ein Flag übergeben, um festzulegen, daß nur das Datum bzw. die Uhrzeit gefordert wird. `ParseDateTime` untersucht dann den String nach der Uhrzeit im Format `HH:MM:SS` und einem Datum im Format `DD/MM/YYYY` oder

in einem langen Format wie January 18th, 1998. Wenn man nur nach der Uhrzeit suchen möchten, kann man als zweiten Flag- Parameter den Wert `VAR_TIMEVALUEONLY` übergeben oder alternativ `VAR_DATEVALUEONLY` nur für das Datum. Soll lediglich das für den Computer gültige Standardformat in die Prüfung einbezogen werden und nicht die vom Benutzer überschriebenen Formate, übergeben Sie als Flag den Wert `LOCALE_NOUSEROVERRIDE`.

Es gibt auch mehrere überladene Operatoren, mit denen man `COleDateTimeSpans` addieren und subtrahieren sowie Datums-/Zeitwerte mit anderen Datums-/Zeitwerten vergleichen kann, wie es aus Tabelle F.12 hervorgeht.

Tabelle F.12: Überladene Operatoren bei COleDateTime

Operator	Beschreibung
=	Kopiert einen Datums-/Zeitwert aus einem anderen <code>COleDateTime</code> -Objekt, einer <code>VARIANT</code> -Struktur, einer <code>DATE</code> -Struktur, einer <code>time_t</code> -Struktur, einer <code>SYSTEMTIME</code> -Struktur oder einer <code>FILETIME</code> -Struktur.
+	Addiert einen <code>COleDateTimeSpan</code> -Wert zu einem <code>COleDateTime</code> -Wert.
-	Subtrahiert einen <code>COleDateTimeSpan</code> -Wert von einem <code>COleDateTime</code> -Wert oder zwei <code>COleDateTime</code> -Objekte voneinander und liefert ein <code>COleDateTimeSpan</code> -Ergebnis.
+=	Addiert einen <code>COleDateTimeSpan</code> -Wert zum aktuellen <code>COleDateTime</code> -Objekt.
-=	Subtrahiert einen <code>COleDateTimeSpan</code> -Wert vom aktuellen <code>COleDateTime</code> -Objekt.
==	Prüft, ob zwei <code>COleDateTime</code> -Objekte identische Datums-/Zeitwerte speichern.
!=	Prüft, ob zwei <code>COleDateTime</code> -Objekte unterschiedliche Datums-/Zeitwerte speichern.
<	Prüft, ob ein <code>COleDateTime</code> -Objekt kleiner als ein anderes ist.
>	Prüft, ob ein <code>COleDateTime</code> -Objekt größer als ein anderes ist.
<=	Prüft, ob ein <code>COleDateTime</code> -Objekt größer oder gleich einem anderen ist.
>=	Prüft, ob ein <code>COleDateTime</code> -Objekt größer oder gleich einem anderen ist.

Die Klasse COleDateTimeSpan

Ein `COleDateTimeSpan`-Objekt kann Differenzen zwischen zwei `COleDateTime`-Objekten speichern. Die Differenz lässt sich bilden, indem man ein `COleDateTime`-Objekt von einem anderen subtrahiert oder eine der `COleDateTimeSpan`-Konstruktorformen gemäß Tabelle F.13 verwendet.

Tabelle F.13: Konstruktortypen bei COleDateTimeSpan

Konstruktordefinition	Beschreibung
<code>COleDateTimeSpan()</code>	Erzeugt eine Zeitspanne, die auf null gesetzt ist.
<code>COleDateTimeSpan(const COleDateTimeSpan& srcSpan)</code>	Kopiert die Zeitspanne aus einem anderen <code>COleDateTimeSpan</code> -Objekt.

<code>COleDateTimeSpan(long lDays, int nHours, int nMins, int nSecs)</code>	Initialisiert die Zeitspanne mit den als Parameter übergebenen Werten.
<code>COleDateTimeSpan(double dSpanSrc)</code>	Initialisiert die Zeitspanne mit der übergebenen Anzahl der Tage.

Nachdem Sie ein `COleDateTimeSpan`-Objekt erzeugt haben, können Sie dessen Status mit den Funktionen `GetStatus` und `SetStatus` genau wie beim `COleDateTime`-Objekt prüfen bzw. setzen. Die einzigen Unterschiede bestehen darin, daß die Flags mit `COleDateTimeSpan::valid` und `COleDateTimeSpan::invalid` benannt sind.

Eine Zeitspanne kann man auch setzen, indem man die Anzahl der Tage, Stunden, Minuten und Sekunden als Integer-Parameter an die Funktion `SetDateTimeSpan` übergibt. Diese Werte lassen sich dann aus einem gültigen `COleDateTimeSpan`-Objekt mit den Funktionen `GetDays`, `GetHours`, `GetMinutes` und `GetSeconds` abrufen, die alle Long-Werte zurückgeben, um die einzelnen Teile des Wertes für die Zeitspanne darzustellen. Die komplette Zeitspanne, ausgedrückt in Tagen, Stunden, Minuten oder Sekunden, kann man mit den Funktionen `GetTotalDays`, `GetTotalHours`, `GetTotalMinutes` bzw. `GetTotalSeconds` in einem Double-Wert abrufen.

`COleDateTimeSpan`-Werte kann man als Zeichenfolgen in der gleichen Weise wie `COleDateTime`-Werte formatieren, indem man einen Formatstring mit den passenden Codes für Zeitspannen gemäß Tabelle F.11 übergibt.

Arithmetische Berechnungen mit `COleDateTimeSpan`-Objekten unterstützen verschiedene überladene Operatoren, mit denen man Zeitspannen addieren, subtrahieren und in Bedingungen testen kann, wie es aus Tabelle F.14 hervorgeht.

Tabelle F.14: Überladene Operatoren in `COleDateTimeSpan`

Operator	Beschreibung
<code>=</code>	Kopiert Zeitspannen aus anderen Zeitspannenwerten.
<code>+</code>	Addiert zwei Zeitspannen.
<code>-</code>	Subtrahiert eine Zeitspanne von einer anderen.
<code>+=</code>	Addiert eine Zeitspanne zum aktuellen Objekt.
<code>-=</code>	Subtrahiert eine Zeitspanne vom aktuellen Objekt.
<code>==</code>	Prüft, ob zwei Zeitspannen identisch sind.
<code>!=</code>	Prüft, ob zwei Zeitspannen verschieden sind.
<code><</code>	Prüft, ob eine Zeitspanne kleiner als eine andere ist.
<code>></code>	Prüft, ob eine Zeitspanne größer als eine andere ist.
<code><=</code>	Prüft, ob eine Zeitspanne kleiner oder gleich einer anderen ist.
<code>>=</code>	Prüft, ob eine Zeitspanne größer oder gleich einer anderen ist.

Der folgende Beispielcode zeigt, wie man zwei `COleDateTime`-Ojekte mit dem überladenen

Minusoperator (-) voneinander subtrahiert, um als Ergebnis eine `COleDateTimeSpan` zu erhalten:

```
COleDateTime = dtMoonwalk;
dtMoonwalk = COleDateTime(1969, 7, 20, 0, 0, 0);
COleDateTimeSpan dtDiff = COleDateTime::GetCurrentTime() -
dtMoonwalk;
CString strMessage;
strMessage.Format("Tage seit dem ersten Mondspaziergang: %d ",
(int)dtDiff.GetTotalDays());
AfxMessageBox(strMessage);
```

Die Klasse zur Zeichenfolgenbehandlung

Vor mehreren Jahren beneideten die C-Programmierer heimlich ein (und nur ein) Werkzeug, das BASIC-Programmierer zur Verfügung hatten: ausgeklügelte und einfache Zeichenfolgenbehandlung. Bei C++ läßt sich diese Funktionalität natürlich nachbilden und steht mit der MFC-Klasse `CString` zur Verfügung.

Die Zeichenfolgenbehandlung ist in Anwendungen häufig gefragt, und Visual C++- Anwendungen tendieren dazu, mit Instanzen von Objekten auf Basis der Klasse `CString` durchsetzt zu sein, um diese Aufgabe zu realisieren.

Die Klasse `CString`

`CString`-Objekte kann man in einfacher Weise als leerer String konstruieren oder initialisieren, indem man einen der vielen verschiedenen Typen von Textdarstellungssystemen an den Konstruktor übergibt. Die verschiedenen Formen der `CString`-Konstruktion sind in Tabelle F.15 aufgeführt.

Tabelle F.15: Mit `CString` verwendete Konstruktortypen

Konstruktordefinition	Beschreibung
<code>CString()</code>	Erzeugt einen leeren String der Länge null.
<code>CString(const CString& strSrc)</code>	Kopiert den Inhalt aus einem anderen String.
<code>CString(LPCSTR lpsz)</code>	Kopiert den Inhalt aus einem nullterminierten String.
<code>CString(const unsigned char* psz)</code>	Kopiert den Inhalt aus einem nullterminierten String.
<code>CString(LPCSTR lpch, int nLength)</code>	Kopiert <code>nLength</code> Zeichen aus einem Zeichenfeld.
<code>CString(TCHAR ch, int nRepeat = 1)</code>	Füllt den String mit <code>nRepeat</code> Kopien des Zeichens <code>ch</code> .
<code>CString(LPCWSTR lpsz)</code>	Kopiert einen nullterminierten Unicode-String.

Wenn Sie ein `CString`-Objekt konstruiert haben, gibt es viele Möglichkeiten, um Text hinzuzufügen oder zuzuweisen. Die überladenen Operatoren erlauben einfache Zuweisungen über den Operator `=` oder die Verkettung von zwei Strings mit den Operatoren `+` und `+=`, wie es die folgenden Zeilen zeigen:

```
CString strTest;
```



```
strTest = "Mr Gorsky";
strTest = "Viel Glück " + strTest;
AfxMessageBox(strTest);
```

Dieses Beispiel setzt den String anfänglich auf "Mr Gorsky". Anschließend wird mit dem Operator + der Text "Viel Glück " vorangestellt.

Die Länge eines Strings läßt sich mit der Elementfunktion `GetLength` ermitteln. Die Funktion liefert eine Ganzzahl zurück, die die momentane Anzahl der Zeichen im String darstellt. Mit der Funktion `IsEmpty` kann man auch testen, ob ein String leer ist. Die Funktion liefert `TRUE` zurück, wenn der String keine Zeichen enthält.

Die Funktion `Empty` dient dazu, den Inhalt eines `CString`-Objekts zu löschen. Das Objekt weist dann die Länge null auf.

Viele Funktionen erfordern Strings im alten Stil von C und keine `CString`-Objekte. Die Typumwandlungen (`const char *`) oder `LPCTSTR` erlauben diesen Funktionen den Zugriff auf den internen Puffer des `CString`-Objekts als würde es sich um einen nullterminierten C-String handeln (allerdings nur für Lesezugriff). Visual C++ wandelt implizit den `CString` in einen nullterminierten String um, wenn der Prototyp einer bestimmten Funktion das erfordert. Da jedoch einige Funktionen einen `void*` Prototyp haben, übergibt der Compiler einen Zeiger auf das `CString`-Objekt statt den erwarteten nullterminierten String. Somit müssen Sie die Umwandlung (`LPCTSTR`) auf dem `CString`-Objekt in eigener Regie erledigen.



Nullterminierte Strings

Bei **nullterminierten Strings** handelt es sich um Zeichenfelder, die das Ende des Strings mit einem Null-Zeichen markieren. Daher ist die Länge des Strings die Anzahl der Zeichen vor dem Null-Zeichen. Es gibt verschiedene Funktionen im Stil von C wie `strlen`, `strcpy` (die Länge ermitteln und einen String kopieren) und viele andere, die man für die Behandlung von nullterminierten Strings verwendet.

Mit den Funktionen `GetAt` und `SetAt` kann man auf einen String als Array von Zeichen zugreifen. `GetAt` liefert das Zeichen an der als Parameter übergebenen (von 0 an gerechneten) Position zurück. Die Funktion `SetAt` setzt ein Zeichen (als zweiten Parameter) an eine Position (erster Parameter) innerhalb der Länge des Strings. Anstelle der Funktion `GetAt` kann man auch mit dem Indexoperator `[ ]` einen Zeichenwert von einer bestimmten Position ermitteln. Beispielsweise tauschen die folgenden Zeilen die Zeichen `b` und `g` aus, um den Rechtschreibfehler zu korrigieren:

```
CString strText("Rechtschreigunb ");
TCHAR ch1 = strText.GetAt(11);
strText.SetAt(11, strText[14]);
strText.SetAt(14, ch1);
```

Mit den überladenen Operatoren `<`, `<=`, `==`, `!=`, `>=` und `>` kann man Strings gemäß der lexikographischen Reihenfolge miteinander vergleichen. Bei Zeichenfolgenvergleichen werden die ASCII-Codes miteinander verglichen, so daß Ziffern kleiner als Buchstaben und Großbuchstaben kleiner als Kleinbuchstaben sind. Demzufolge bewirken die folgenden Codezeilen, daß im Meldungsfeld `TRUE` erscheint:

```
CString str1("123");
CString str2("ABC");
```

```
CString str3("abc");
CString str4("bcd");
if (str1 < str2 && str2 < str3 && str3 < str4)
AfxMessageBox("TRUE");
```

Um den aktuellen String mit einem anderen zu vergleichen, kann man auch die Funktion `Compare` einsetzen. Die Funktion liefert null zurück, wenn zwei Strings gleich sind, einen negativen Wert, wenn der aktuelle String kleiner als der getestete String ist, oder einen positiven Wert, wenn der aktuelle String größer als der Teststring ist. Beispielsweise bewirken die folgenden Zeilen, daß ein Meldungsfeld mit dem Text TRUE erscheint:

```
CString strName("Peter");
if (strName.Compare("Piper")<0) AfxMessageBox("TRUE");
```

Dieser Vergleich beachtet auch die Groß-/Kleinschreibung der zu vergleichenden Strings. Vergleiche ohne Unterscheidung der Groß-/Kleinschreibung lassen sich mit der äquivalenten Funktion `CompareNoCase` durchführen.

Zeichenfolgenmanipulation

In der Sprache BASIC stehen drei nützliche Funktionen zur Manipulation von Strings zur Verfügung: `Mid$`, `Left$` und `Right$`. Diese Funktionen sind nun auch als `Mid`, `Left` und `Right` in der Klasse `CString` verfügbar. Als Ergebnis liefern die Funktionen Kopien von Teilstrings zurück. Der Funktion `Mid` übergibt man eine Anfangsposition und optional die Anzahl der zu kopierenden Zeichen (fehlt das optionale Argument, liefert die Funktion alle Zeichen). Die Funktion gibt dann einen anderen `CString` mit dem angegebenen Teilstring zurück. Mit der Funktion `Left` extrahiert man eine Anzahl Zeichen vom linken Teil eines Strings. Dabei übergibt man der Funktion die Anzahl der gewünschten Zeichen. Die Funktion `Right` liefert die angegebene Anzahl der Zeichen von der rechten Seite des Strings (d.h. vom Ende des Strings an gerechnet) zurück. Dazu das folgende Beispiel:

```
CString strText("Ich habe heute drei Segelschiffe gesehen");
TRACE("%s\n", strText.Left(8));
TRACE("%s\n", strText.Mid(15, 17));
TRACE("%s\n", strText.Right(7));
```

Die Ausgabe der drei TRACE-Makros liefert:

```
Ich habe
drei Segelschiffe
gesehen
```

Das Wort `heute` zwischen `habe` und `drei` erscheint nicht in der Trace-Ausgabe, da dieser Teil des Strings überhaupt nicht extrahiert wird.

Mit den Funktionen `MakeUpper` und `MakeLower` ändert man alle Zeichen in einem String in Groß- bzw. Kleinbuchstaben, während die Funktion `MakeReverse` den String in umgekehrter Zeichenfolge liefert.

Leerzeichen, Zeichen für neue Zeile und Tabulatoren lassen sich vom linken Teil eines Strings mit der Funktion `TrimLeft` entfernen. Auf der rechten Seite realisiert die Funktion `TrimRight` diese Aufgabe.

Zeichenfolgen suchen

Mit den Funktionen `Find`, `ReverseFind` und `FindOneOf` kann man nach bestimmten Teilstrings oder Zeichen in einem String suchen.

Der Elementfunktion `Find` übergibt man ein einzelnes Zeichen oder einen String, um nach einem Vorkommen dieses Zeichens oder des Strings im Kontextstring zu suchen. Wenn das Zeichen oder der Teilstring gefunden wurde, liefert die Funktion die Position (gerechnet ab 0) zurück. Andernfalls lautet das Ergebnis `-1`. Es kennzeichnet, daß der Teilstring oder das Zeichen nicht vorhanden ist. Die folgenden Zeilen suchen nach dem Wort »du« und zeigen den Teilstring ab dieser Position an:

```
CString strTest("Fang an, wenn du denkst");
int nPosn = strTest.Find("du");
if (nPosn!=-1) TRACE(strTest.Mid(nPosn) + "?");
```

Die Funktion `ReverseFind` sucht nach einem bestimmten Zeichen (Teilstrings nicht möglich) vom Ende eines Strings. Wird das Zeichen gefunden, liefert die Funktion die Position bezüglich des Anfangs vom String (gerechnet ab Position 0) zurück, andernfalls `-1`.

Bei `FindOneOf` kann man eine Anzahl von Zeichen übergeben, die in einem String zu suchen sind. Zurückgegeben wird die Position des ersten Suchzeichens der Gruppe. Das Ergebnis `-1` gibt an, daß kein Zeichen gefunden wurde. Die folgenden Beispielzeilen suchen nach den Zeichen d, e, k und w. Das w wird zuerst gefunden, so daß der String »wenn du denkst« in der Trace-Ausgabe erscheint:

```
CString strTest("Fang an, wenn du denkst");
int nPosn = strTest.FindOneOf("dekw");
if (nPosn!=-1) TRACE(strTest.Mid(nPosn));
```

Text zur Anzeige formatieren

Häufig setzt man `CString`-Objekte ein, um Text vor der Ausgabe zu formatieren. Dazu verwendet man die Funktion `Format`, der man im ersten Parameter eine Gruppe von Formatanweisungen als Prozentzeichencodes übergibt. Daran schließt sich die Übergabe von Wertparametern an. Die Anzahl der Wertparameter muß der Anzahl der angegebenen Formatcodes entsprechen. Einige dieser Formatflags und die jeweiligen Parametertypen sind in Tabelle F.16 aufgelistet. Mehrere Codes kann man zu einem Formatstring zusammenfassen, der dann im aufrufenden `CString`-Objekt gespeichert wird. Diesen Formatstring kann man auch als Stringressourcen-ID aus den Ressourcen der Anwendung übergeben.

Tabelle F.16: Formatcodes für die Funktion `Format`

Flag	Parametertyp	Beschreibung
%c	int	Zeigt ein einzelnes Textzeichen an.
%d	int	Zeigt eine Dezimalzahl mit Vorzeichen an.
%u	int	Zeigt eine Ganzzahl ohne Vorzeichen an.
%o	int	Zeigt eine Oktalzahl ohne Vorzeichen an.
%x	int	Zeigt eine ganze Hexadezimalzahl ohne Vorzeichen in Kleinbuchstaben an.
%X	int	Zeigt eine ganze Hexadezimalzahl ohne Vorzeichen in Großbuchstaben an.

%f	double	Zeigt eine Gleitkommazahl mit Vorzeichen an.
%s	string	Zeigt bei Übergabe eines Stringzeigers wie etwa <code>char *</code> einen Textstring an.
%%	keiner	Zeigt das Prozentzeichen % an.

Das folgende Beispiel kombiniert einige der gebräuchlichen Typen, um ein Meldungsfeld mit dem Text »Es kommen 1609.340000 Meter auf eine Meile.« anzuzeigen:

```
CString strFormatMe;
char *szM = "Meter";
double dMPerMile = 1609.34;
strFormatMe.Format("Es kommen %f %s auf %d %s",
dMPerMile, szM, 1, "Meile.");
AfxMessageBox(strFormatMe);
```

Mit zusätzlichen Formatangaben kann man für jedes Ausgabefeld die Breite, Genauigkeit und Ausrichtung spezifizieren. Diese Attribute sind nach dem Prozentzeichen aber noch vor dem (in Tabelle F.16 angegebenen) Typzeichen in folgendem Format zu schreiben:

%[Flag][Breite][.Genauigkeit]Typ

Für den Flag-Wert kann man die folgenden Zeichen angeben:

- Richtet das Feld linksbündig aus (ansonsten rechtsbündig).

+ Zeigt vor einer Zahl ein Plus- oder Minuszeichen an (per Vorgabe erscheint nur das Minuszeichen bei negativen Zahlen).

0 Füllt die Feldbreite mit Nullen auf.

Mit **Breite** gibt man dann die Mindestzahl von Zeichen an, die das Feld anzeigen soll. Der Parameter **.Genauigkeit** bestimmt, wie viele Zeichen nach dem Dezimalpunkt stehen sollen.

Den Text im vorherigen Beispiel kann man zum Beispiel so formatieren, daß »Es kommen +1609.3 Meter auf 1 Meile.« in der Ausgabe erscheint:

```
strFormatMe.Format("Es kommen %+.1f %s auf %d %s",
dMPerMile, szM, 1, "Meile");
```



Zeichenfolgenformatierung

Die Funktion `Format` der Klasse `CString` verwendet die gleichen Codes zur Festlegung der Formate wie die C-Funktionen `printf` und `sprintf`. C-Programmierer sind wahrscheinlich mit den Formatcodes der Funktion `printf` vertrauter und können die Codes ohne Umzulernen bei der Funktion `Format` der Klasse `CString` verwenden.

Einführung

Willkommen zu Visual C++. In den nächsten 21 Tagen lernen Sie, wie man die Funktionsmerkmale einsetzt, die Microsoft seiner C++-Entwicklungsumgebung spendiert hat, damit Sie fortschrittliche Anwendungen für die Plattformen Windows und Windows NT erstellen können. Als sich die Microsoft-Entwickler mit den Konzepten von Visual C++ auseinandersetzten, entschieden sie sich dafür, ihren C++-Compiler mit einer Entwicklungsumgebung und einem Satz von Dienstprogrammen auszustatten, die den Entwicklern das Erstellen von Windows-Anwendungen mit einer Leichtigkeit und Geschwindigkeit erlauben, die unter den C++-Entwicklungsumgebungen ihresgleichen sucht. Seit der ersten Version hat Microsoft die Werkzeuge, die Teil von Visual C++ sind, ständig verbessert, um die Anwendungsentwicklung für Windows noch einfacher zu gestalten. Sobald Microsoft neue Technologien für die Windows- Plattform eingeführt hat, wurden diese Werkzeuge auch Bestandteil der Visual C++- Suite, um auch die neusten Technologien sofort in Anwendungen einbinden zu können.

Wenn Sie heute erstmals mit C++ Bekanntschaft schließen, sollte Sie das nicht abschrecken. Dieses Buch ermöglicht es Ihnen gleichermaßen, die Programmiersprache C++ und das Erstellen von Anwendungen mit den Werkzeugen von Visual C++ zu erlernen. Falls Sie mit bestimmten Aspekten von C++ noch nicht so richtig vertraut sind, empfiehlt sich der Anhang A, der einen kurzen Überblick über die Programmiersprache bietet.

Sollten Sie bereits die früheren Versionen dieses Buchs kennen, fällt Ihnen auf, daß es vollständig neu gefaßt wurde. Das Ziel der neuen Version besteht darin, nicht nur eine Einführung in die verschiedenen Tools und Ressourcen, die Sie für das Erstellen von Anwendungen mit Visual C++ verwenden, zu geben, sondern auch ein tiefergehendes Verständnis zu den Möglichkeiten, die sich Ihnen mit den behandelten Funktionsmerkmalen erschließen, zu vermitteln. Auf diese Weise bringt Ihnen das Buch weit über die anfänglichen 21 Tage hinaus noch eine Menge Nutzen.

Wie dieses Buch organisiert ist

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

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

Mit Beginn der zweiten Woche werden die Themen komplizierter, und Sie steigen tiefer in die Programmierung ein. Sie erstellen zwar Ihre Anwendungen mit den gleichen Werkzeugen, der

Programmcode wird aber anspruchsvoller. In der zweiten Woche beschäftigen Sie sich bereits mit etwas komplizierteren Themen wie der Anzeige von Grafiken und dem Erstellen von SDI- und MDI-Anwendungen. Gegen Ende dieser Woche arbeiten Sie erstmals mit Datenbanken. Dieses Thema setzt sich in der dritten und damit letzten Woche fort.

In der dritten Woche erstellen Sie eigene Module, DLLs und ActiveX-Steuerelemente. Sie erfahren, wie man Multitasking-Anwendungen realisiert, die mehrere Aufgaben gleichzeitig ausführen. Schließlich integrieren Sie den Microsoft Internet Explorer und die von ihm bereitgestellten ActiveX-Steuerelemente in Ihre Anwendungen, um diese für das Internet zu erweitern.

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

Konventionen

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

Der gesamte Quellcode ist in **Schreibmaschinenschrift** gesetzt, wie es Listing 0.1 zeigt. Dazu gehört der vollständige Quellcode aus den Anwendungen, die Sie erstellen, und die Demonstrationsbeispiele, die den Einsatz der verschiedenen Funktionen zeigen. Wenn Sie neuen Code in eine Funktion hinzufügen oder Änderungen an bereits bestehendem Code vornehmen, helfen Ihnen die Zeilennummern im Listing, auf die der Text dann Bezug nimmt.

Listing 0.1: Etwas Beispielcode

```
1: void main()
2: {
3: // Wenn Sie Code in einem vorhandenen Codeabschnitt
4: // hinzufügen oder ändern, nimmt der Text mit Hilfe
5: // der Zeilennummern darauf Bezug.
6: }
```

Wenn ein Thema besondere Beachtung verdient, ist der betreffende Abschnitt vom übrigen Text hervorgehoben und mit speziellen Symbolen markiert:

- Hinweise
- Tips
- Achtung



Hinweise bringen eine tiefergehende Erläuterung eines Themas oder erklären interessante oder wichtige Punkte.



Tips sollen Ihnen helfen, sich das Leben - sprich die Programmierung - leichter zu machen.



Unter dieser Rubrik sind Stolpersteine angeführt, die Sie tunlichst umgehen sollten.

Am Ende der einzelnen Tage finden Sie ein paar Kontrollfragen und ein oder zwei Übungen, mit denen Sie das behandelte Thema vertiefen können. Falls Sie nicht alles auf Anhieb wissen, ist das auch nicht weiter schlimm. Die Antworten zu den Kontrollfragen und kurze Anleitungen zu den Übungen finden Sie im Anhang B.

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

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1 im Überblick

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

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

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

Der dritte Tag zeigt, wie sich Maus- und Tastaturereignisse in einer Anwendung abfangen lassen und wie man darauf reagiert. Sie ermitteln, wo sich der Mauszeiger im Fenster Ihrer Anwendung befindet. Weiterhin erfahren Sie, wie man herausfindet, welche Tasten der Benutzer auf der Tastatur betätigt und wie man auf derartige Benutzeraktionen reagieren kann.

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

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

In der Lektion 6 erstellen Sie Menüs, fügen sie in Ihre Anwendung ein und lernen, wie man Funktionen in der Anwendung über die hinzugefügten Menüs aufrufen kann.

Der siebente Tag beschäftigt sich mit der Infrastruktur der Schriften unter Windows und zeigt, wie man darauf in einer Visual C++-Anwendung zugreift. In diesem Zusammenhang erstellen Sie eine Liste der verfügbaren Schriften und bringen Text in ausgewählten Schriften auf den Bildschirm.

Damit sind Sie am Ende der ersten Woche in diesem Buch angelangt. Vielleicht blicken Sie noch einmal auf den Stoff dieser Woche zurück und wiederholen, welche Schritte für das Erstellen von

Anwendungen erforderlich sind. Ohne weitere Verzögerung geht es dann unmittelbar weiter.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1**Tag 1****Erste Schritte mit Visual C++**

Willkommen zu Visual C++ 6 in 21 Tagen. In den nächsten drei Wochen lernen Sie, wie man die verschiedenartigsten Anwendungen mit Visual C++ von Microsoft erstellt. Vor allem aber vollziehen Sie alle Schritte selbst nach und gewinnen auf diese Weise eine gehörige Portion Programmiererfahrung. Deshalb frisch ans Werk!

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

Diese Lektion beschäftigt sich mit folgenden Themen:

- Die Hauptbereiche der Entwicklungsumgebung von Visual C++.
- Der Anwendungs-Assistent - wie man ihn einsetzt, um die Infrastruktur für Anwendungen zu erstellen.
- Der Dialog-Editor - wie man ihn einsetzt, um Dialogfenster zu zeichnen, genau wie man Fenster mit Visual Basic, PowerBuilder oder Delphi erstellt.
- Der Klassen-Assistent - wie man damit arbeitet, um die Anwendungsfenster mit Funktionalität auszustatten

Die Entwicklungsumgebung von Visual C++

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

Nach dem Start von Visual Studio (die visuelle Entwicklungsumgebung von Microsoft) erscheint ein Fenster wie in Abbildung 1.1. Jeder Bereich erfüllt in der Umgebung des Visual Studios einen bestimmten Zweck. Man kann diese Bereiche neu anordnen, um die Entwicklungsumgebung des Visual Studios an die eigenen Erfordernisse anzupassen.

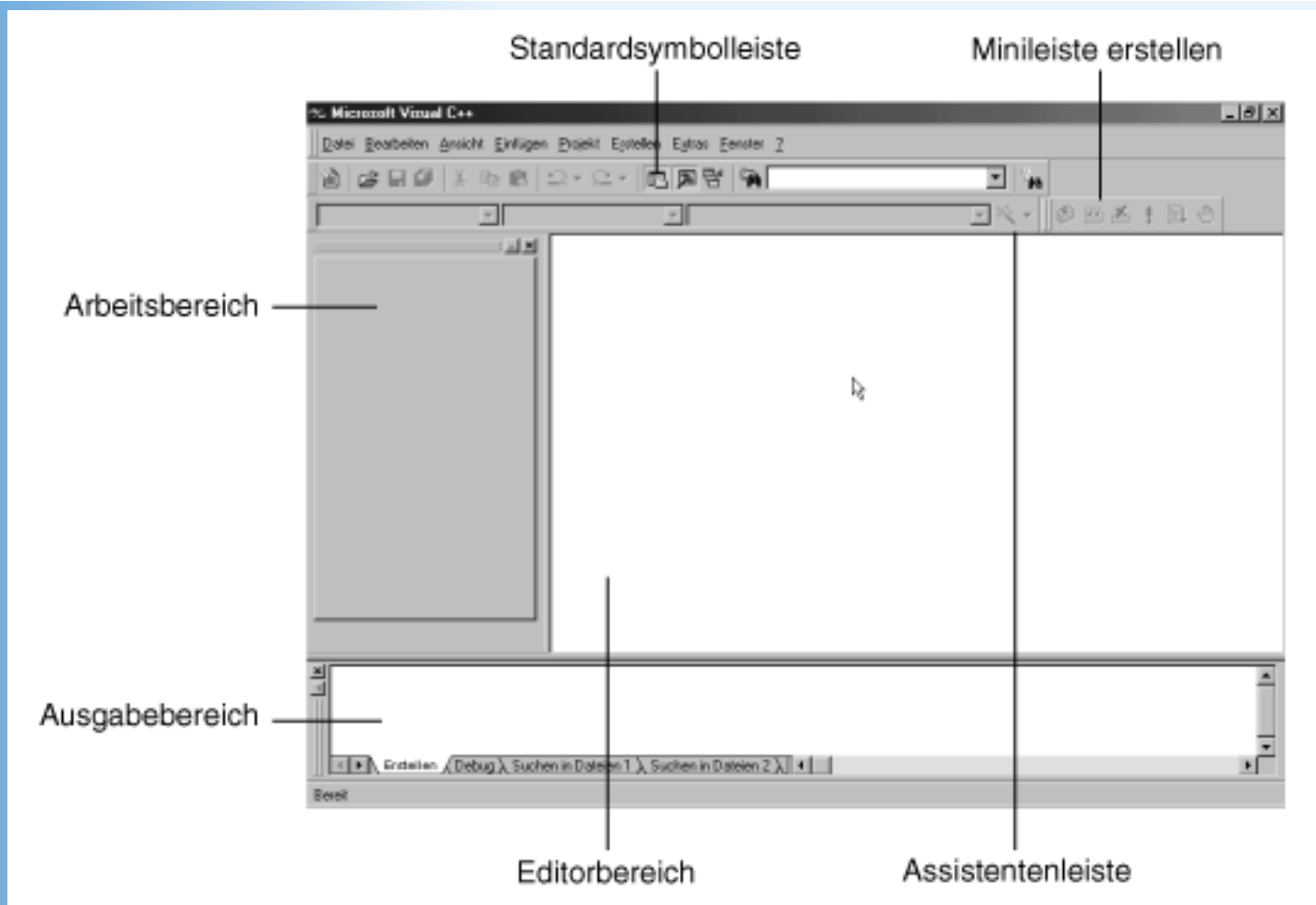


Abbildung 1.1:
Der Eröffnungsbildschirm von Visual C++

Der Arbeitsbereich

Beim ersten Start von Visual C++ sieht es so aus, daß der große Bereich auf der linken Seite des Visual Studios zwar eine Menge Platz beansprucht, aber wenig Informationen bietet. Dieser sogenannte Projekt-Arbeitsbereich (kurz Arbeitsbereich) bildet den Ausgangspunkt, um zu den verschiedenen Teilen der Entwicklungsprojekte zu navigieren. Der Arbeitsbereich gestattet es, die Teile einer Anwendung in drei verschiedenen Modi zu betrachten:

- In der Klassenansicht können Sie auf C++-Klassenebene durch den Quellcode navigieren und ihn bearbeiten.
- Die Ressourcenansicht erlaubt es, die verschiedenen Ressourcen in der Anwendung aufzusuchen und zu bearbeiten. Dazu gehören die Entwürfe von Dialogfenstern, Symbolen und Menüs.
- Die Dateiansicht bietet eine Übersicht über die Dateien, aus denen eine Anwendung besteht. Man kann die Dateien anzeigen und in ihnen navigieren.

Der Ausgabebereich

Der Ausgabebereich ist wahrscheinlich beim ersten Start von Visual C++ noch nicht sichtbar. Nachdem Sie Ihre erste Anwendung kompiliert haben, erscheint der Ausgabebereich am unteren Rand der Visual Studio-Umgebung und bleibt geöffnet, bis man ihn explizit schließt. Im Ausgabebereich stellt Visual Studio alle Informationen bereit, die für Sie relevant sind. Hier können Sie den Fortschritt beim Kompilieren verfolgen, Warnungen und Fehlermeldungen einsehen. Auch der Debugger von Visual C++ zeigt im Ausgabebereich alle Variablen mit ihren aktuellen Werten an, wenn man den

Code schrittweise abarbeitet. Nachdem Sie den Ausgabebereich geschlossen haben, wird er automatisch geöffnet, wenn Visual C++ eine Meldung anzeigen muß, die für Sie von Interesse sein könnte.

Der Editorbereich

... befindet sich im rechten Teil der Visual Studio-Umgebung. Hier erledigen Sie alle Bearbeitungsaufgaben beim Einsatz von Visual C++, hier erscheinen die Fenster des Quellcode-Editors, wenn Sie den C++-Quellcode bearbeiten, und hier wird auch der Dialog-Editor angezeigt, wenn Sie ein Dialogfeld entwerfen. Der Editorbereich dient auch dem Symbol-Editor zur Anzeige, wenn man Symbole für den Einsatz in den Anwendungen gestaltet. Grundsätzlich gehört der gesamte Bereich des Visual Studios, der nicht anderweitig durch Fensterbereiche, Menüs oder Symbolleisten eingenommen wird, zum Editorbereich.

Menüleisten

Beim ersten Start von Visual C++ erscheinen unmittelbar unterhalb der Menüleiste drei Symbolleisten. In Visual C++ sind außerdem weitere Symbolleisten verfügbar, und man kann die Symbolleisten entsprechend dem eigenen Arbeitsstil anpassen und neu erzeugen. Zu Beginn sind folgende Symbolleisten zu sehen:

- Die *Standardsymbolleiste* enthält die meisten der Standardwerkzeuge für das Öffnen und Speichern von Dateien, Ausschneiden, Kopieren, Einfügen und eine Vielzahl allgemein gebräuchlicher Befehle.
- Über die *Assistentenleiste* lassen sich eine Reihe von Aktionen in bezug auf den Klassen-Assistenten realisieren, ohne den Klassen-Assistenten öffnen zu müssen.
- *Minileiste erstellen* hält Befehle zum Erstellen und Ausführen bereit, auf die man höchstwahrscheinlich zurückgreift, wenn man Anwendungen entwickelt und testet. Die vollständige Symbolleiste *Erstellen* erlaubt es auch, zwischen mehreren Erstellen-Konfigurationen umzuschalten (beispielsweise zwischen den Konfigurationen Debug und Release).

Die Umgebung von Visual Studio neu arrangieren

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

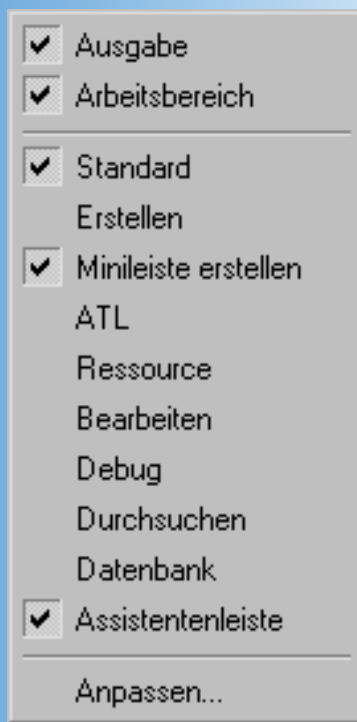


Abbildung 1.2:
Kontextmenü zum Ein-/Ausschalten von Symbolleisten

Ein anderer Weg, um die Entwicklungsumgebung in einfacher Weise neu anzuordnen, besteht darin, die Doppelbalken am linken Ende der Symbolleisten oder Fensterbereiche mit der Maus zu ziehen. Die Symbolleisten kann man von der Position, an der sie momentan verankert sind, wegziehen und unverankerte Symbolleisten wie in Abbildung 1.3 erzeugen. Man kann diese Symbolleisten (und die Fensterbereiche) an jeden Rand des Visual Studios ziehen, um sie an der neuen Position anzudocken. Selbst wenn die Symbolleisten verankert sind, kann man sie mit Hilfe der Doppelbalken nach links oder rechts an den gewünschten Platz verschieben.

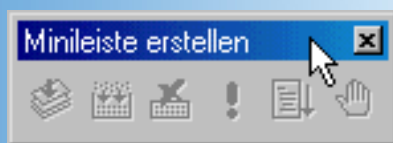
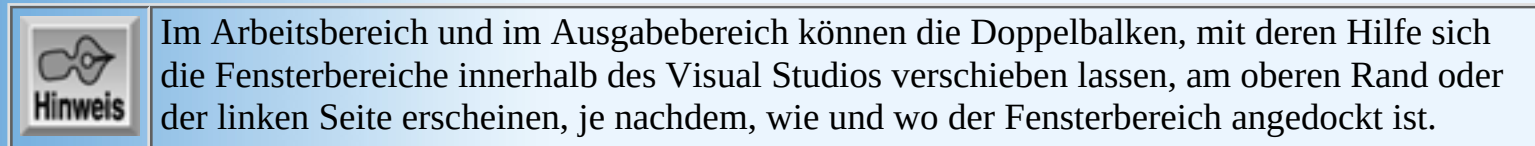


Abbildung 1.3:
Beispiel einer unverankerten Minileiste

Das erste Projekt

Als Einstieg in Visual C++ erstellen Sie eine einfache Anwendung mit zwei Schaltflächen, wie es Abbildung 1.4 zeigt. Die erste Schaltfläche präsentiert dem Benutzer eine einfache Begrüßungsmeldung (siehe Abbildung 1.5), während die zweite Schaltfläche die Anwendung schließt. Um diese Anwendung zu erstellen, sind folgende Dinge zu realisieren:

1. Einen neuen Arbeitsbereich anlegen.
2. Mit dem Anwendungs-Assistenten das Anwendungsgerüst erstellen.

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

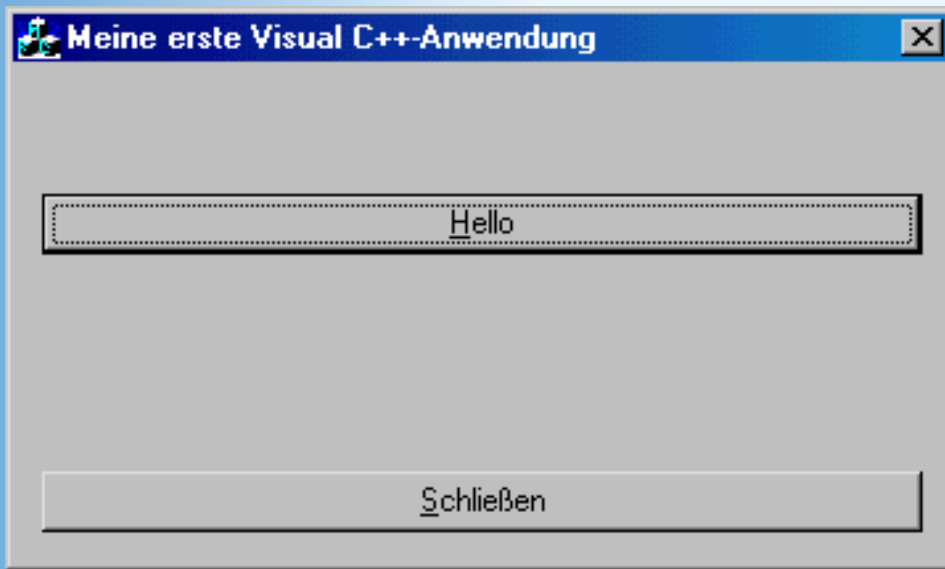


Abbildung 1.4:
Die erste Anwendung mit Visual C++



Abbildung 1.5:
Wenn der Benutzer auf die erste Schaltfläche klickt, erscheint eine einfache Begrüßung.

Den Projekt-Arbeitsbereich anlegen

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

1. Wählen Sie **Datei / Neu**. Es wird der in Abbildung 1.6 dargestellte Assistent **Neu** geöffnet.

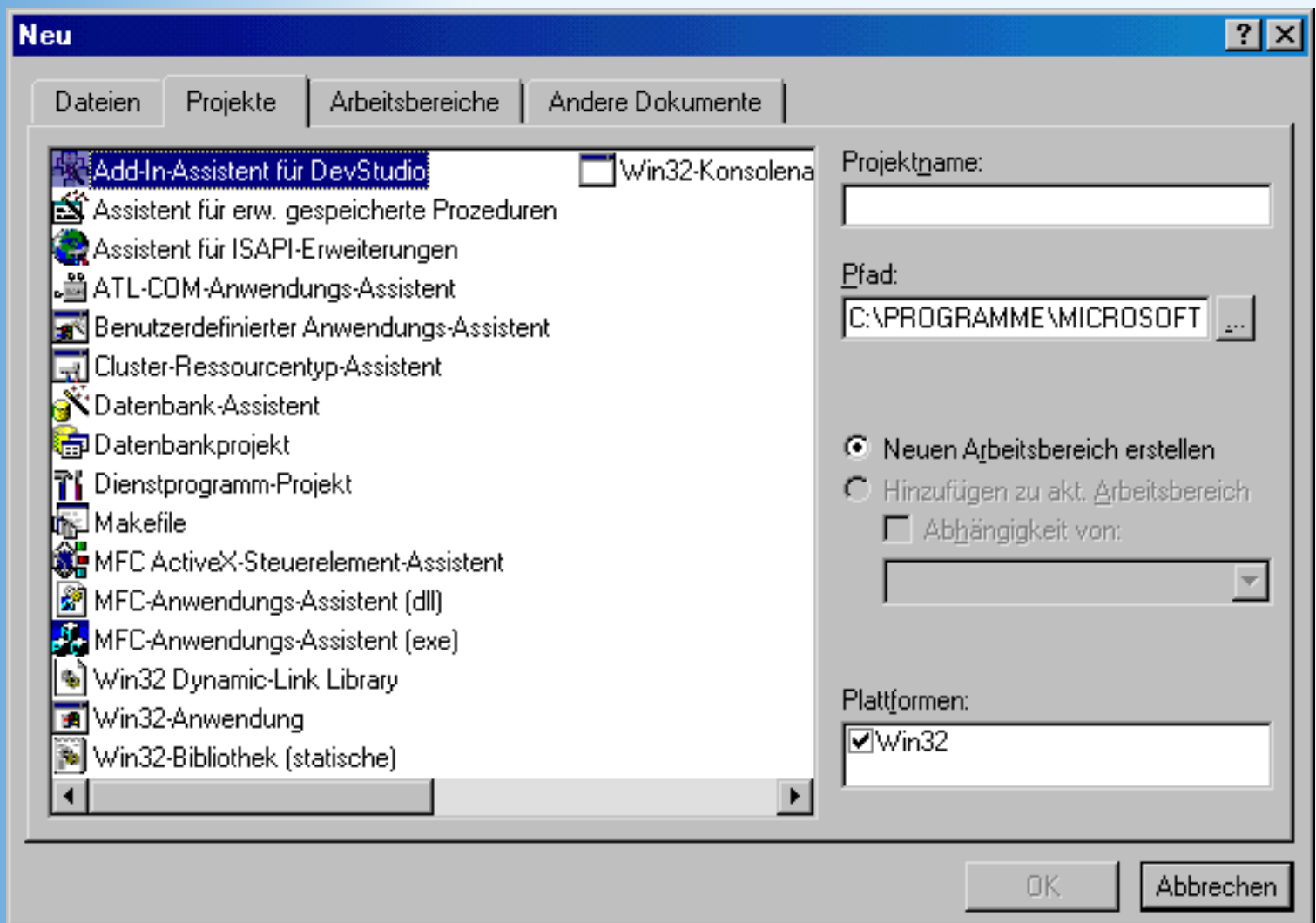


Abbildung 1.6:
Der Assistent Neu

2. Auf der Registerkarte **Projekte** markieren Sie *MFC-Anwendungs-Assistent (exe)*.
3. Tippen Sie in das Feld **Projektname** einen Namen für Ihr Projekt ein, beispielsweise *Hello*.
4. Klicken Sie auf **OK**. Daraufhin erledigt der Assistent **Neu** zwei Dinge: er legt ein Projektverzeichnis an (das im Feld **Pfad** spezifiziert ist) und startet dann den Anwendungs-Assistenten.



Alle Beispiele in diesem Buch wurden im Pfad F : \VCP621\Tagxx\ erstellt, wobei VCP621 für Visual C++ 6 in 21 Tagen und xx für die jeweilige Nummer des Tages (von 01 bis 21) stehen.

Den Anwendungsrahmen mit dem Anwendungs-Assistenten erstellen

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

1. Im Schritt 1 des Anwendungs-Assistenten legen Sie fest, daß Sie eine dialogfeldbasierende

Anwendung erstellen möchten. Klicken Sie am unteren Rand des Assistenten auf **Weiter**.

2. Im zweiten Schritt fragt der Anwendungs-Assistent, welche Merkmale Sie in Ihre Anwendung einbinden wollen. Schalten Sie die Option für die ActiveX-Steuerelemente aus, wenn Sie in der Anwendung - wie es für das heutige Beispiel zutrifft - keine ActiveX-Steuerelemente einsetzen möchten.

3. Im Feld am unteren Rand des Assistenten löschen Sie den Projektnamen (*Hello*) und geben den Titel ein, der in der Titelleiste des Hauptfensters der Anwendung - d.h. des Dialogfelds - erscheinen soll, beispielsweise *Meine erste Visual C++-Anwendung*. Klicken Sie dann auf die Schaltfläche **Weiter**.

4. Im Schritt 3 des Anwendungs-Assistenten übernehmen Sie die Vorgaben, um Kommentare für die Quellcodedateien erzeugen zu lassen und die MFC-Bibliothek als gemeinsam genutzte DLL zu verwenden. Klicken Sie auf **Weiter**, um zum letzten Schritt des Anwendungs-Assistenten zu gelangen.

5. Das letzte Fenster des Anwendungs-Assistenten zeigt die C++-Klassen an, die der Assistent für die Anwendung erstellt. Klicken Sie auf **Fertigstellen**, um den Anwendungs-Assistenten das Anwendungsgerüst erstellen zu lassen.

6. Bevor der Anwendungs-Assistent das Projektgerüst erstellt, zeigt er anhand der Angaben, die Sie in den einzelnen Schritten des Assistenten festgelegt haben, alle Komponenten an, die das Projektgerüst erhält (siehe Abbildung 1.7).

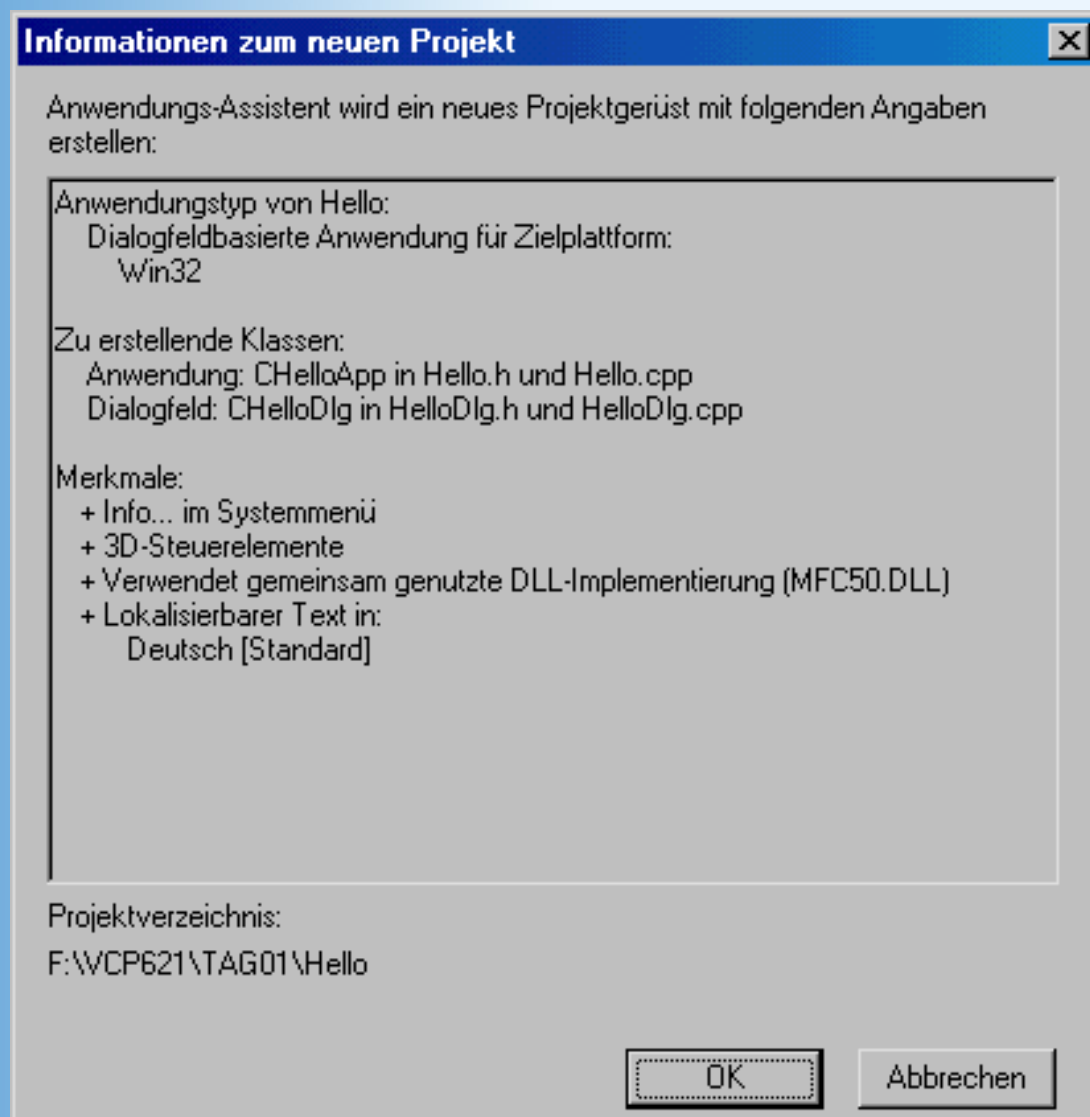


Abbildung 1.7:
Das Fenster Informationen zum neuen Projekt

7. Nachdem der Anwendungs-Assistent das Projektgerüst erstellt hat, gelangen Sie in die Umgebung des Visual Studios zurück. Der Arbeitsbereich zeigt nun eine Baumansicht der Klassen in Ihrem Projektgerüst, wie es aus Abbildung 1.8 hervorgeht. Es erscheint auch das Hauptfenster (ein Dialogfeld) im Editorbereich des Visual Studios.

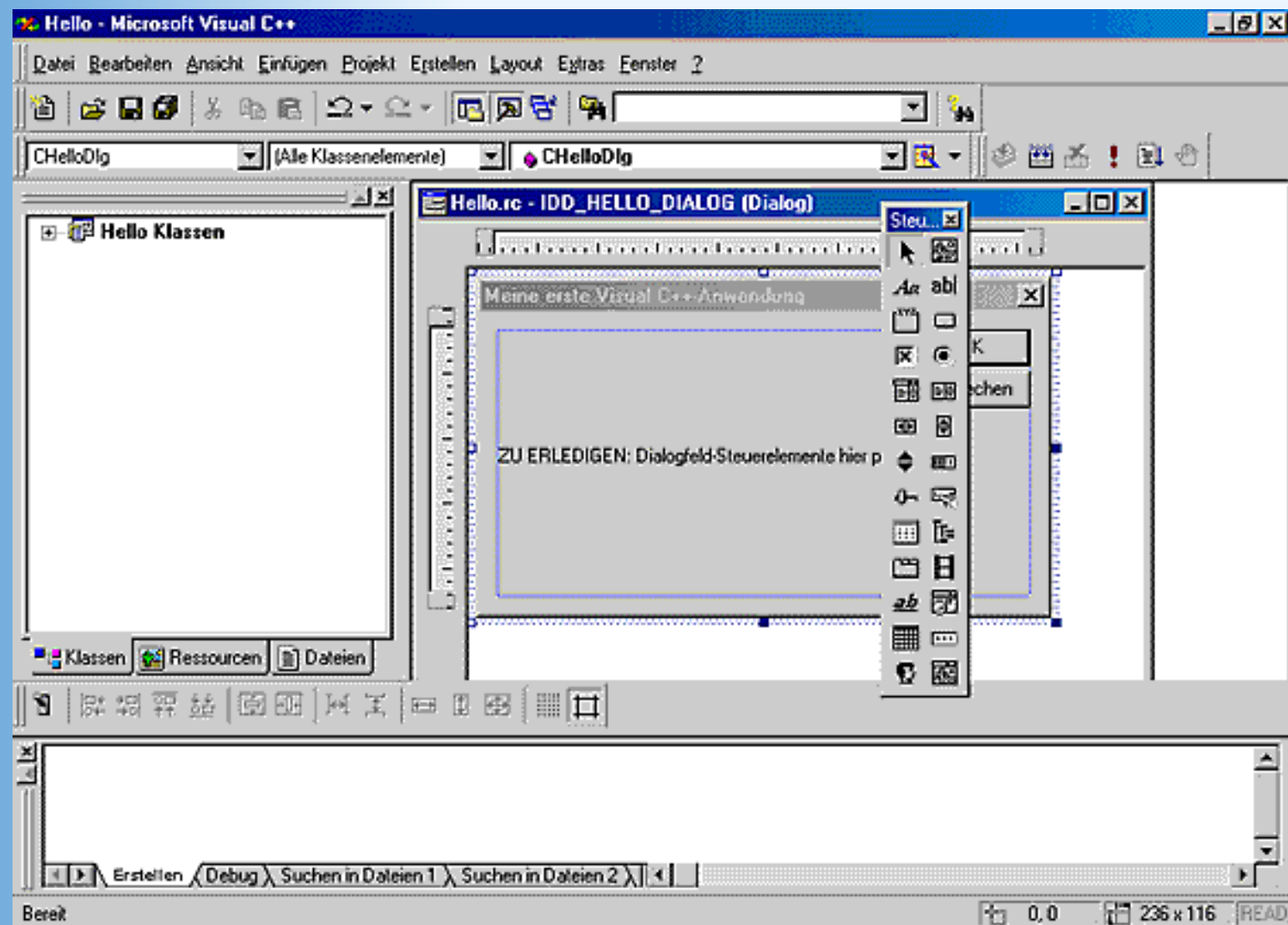


Abbildung 1.8:
Der Arbeitsbereich mit einer Baumansicht der Projektklassen

8. Wählen Sie **Erstellen / Hello.exe erstellen**, um die Anwendung zu kompilieren.

9. Wenn der Visual C++-Compiler die Anwendung erstellt, erscheinen Fortschrittsanzeigen und andere Compiler-Meldungen im Ausgabebereich. Nachdem die Anwendung erstellt ist, sollte der Ausgabebereich eine Meldung enthalten, daß keine Fehler oder Warnungen vorhanden sind (siehe Abbildung 1.9).

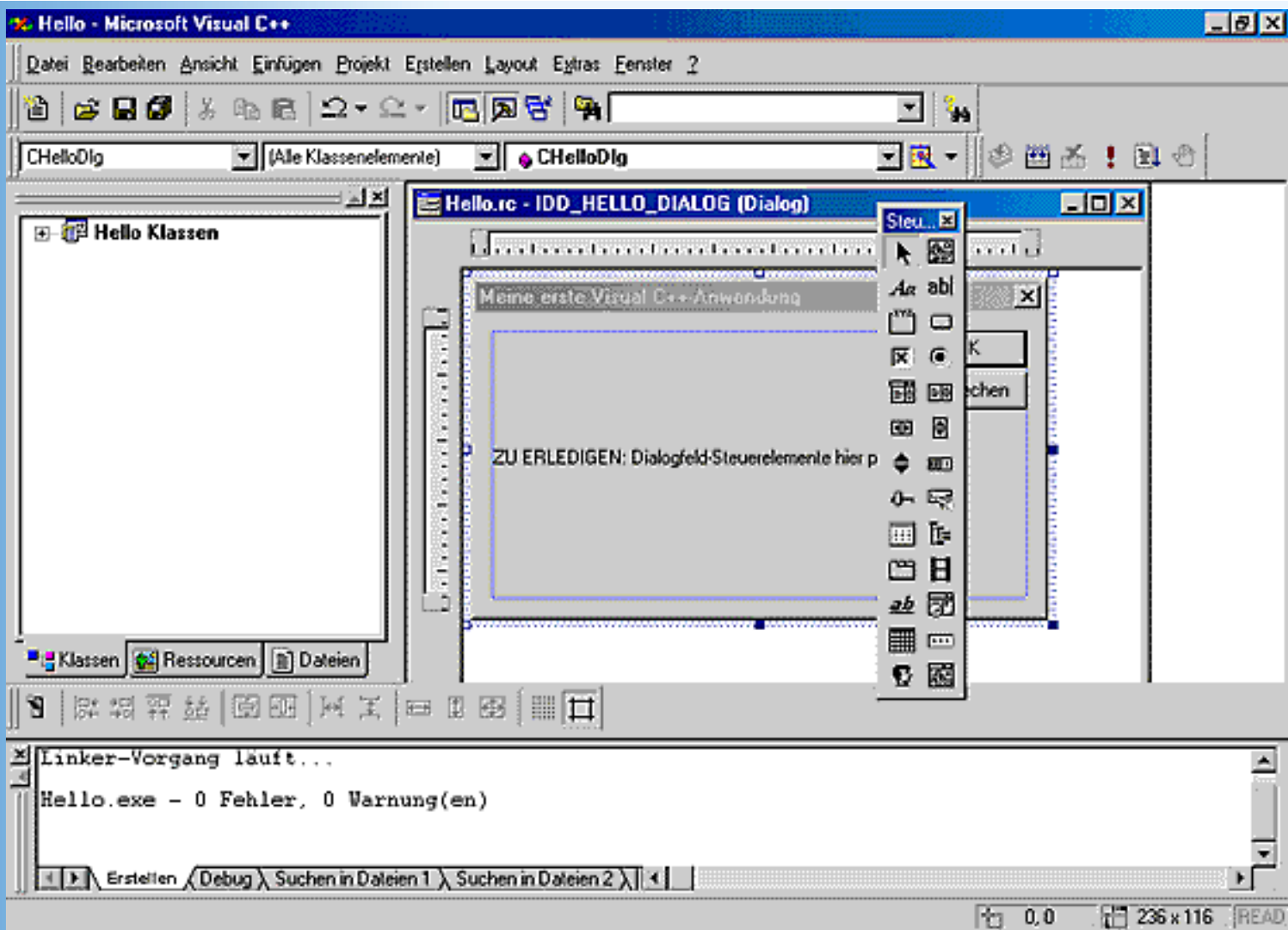


Abbildung 1.9:
Der Ausgabebereich zeigt alle Compilerfehler an.

10. Wählen Sie **Erstellen / Ausführen von Hello.exe**, um Ihre Anwendung zu starten.

11. Die Anwendung präsentiert ein Dialogfeld mit einer ZU ERLEDIGEN-Meldung und den beiden Schaltflächen **OK** und **Abbrechen**, wie es Abbildung 1.10 zeigt. Um die Anwendung zu schließen, können Sie auf eine der beiden Schaltflächen klicken.

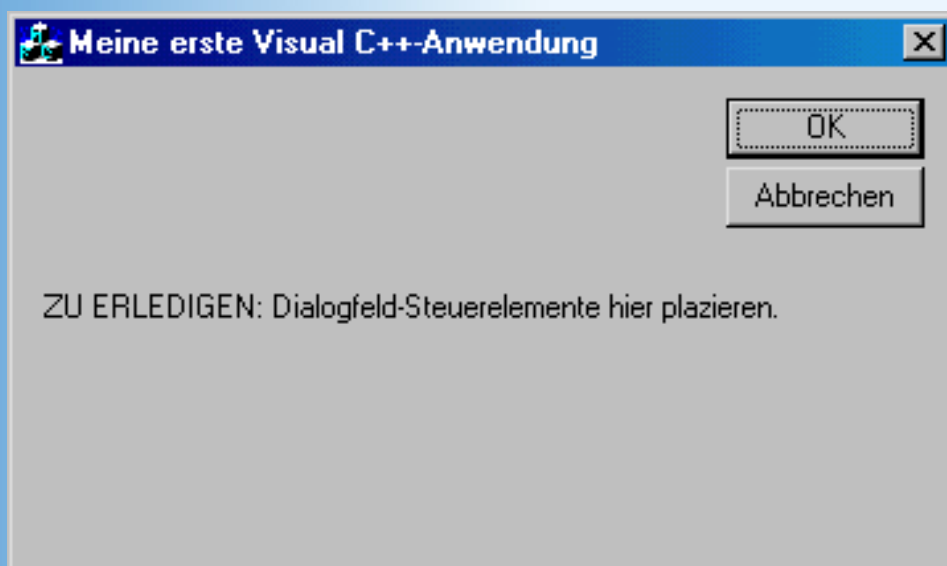


Abbildung 1.10:

Die ausgeführte Anwendung ohne Änderungen des Projektgerüsts

Das Anwendungsfenster gestalten

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

1. Aktivieren Sie im Arbeitsbereich die Registerkarte **Ressourcen**, wie es Abbildung 1.11 zeigt.

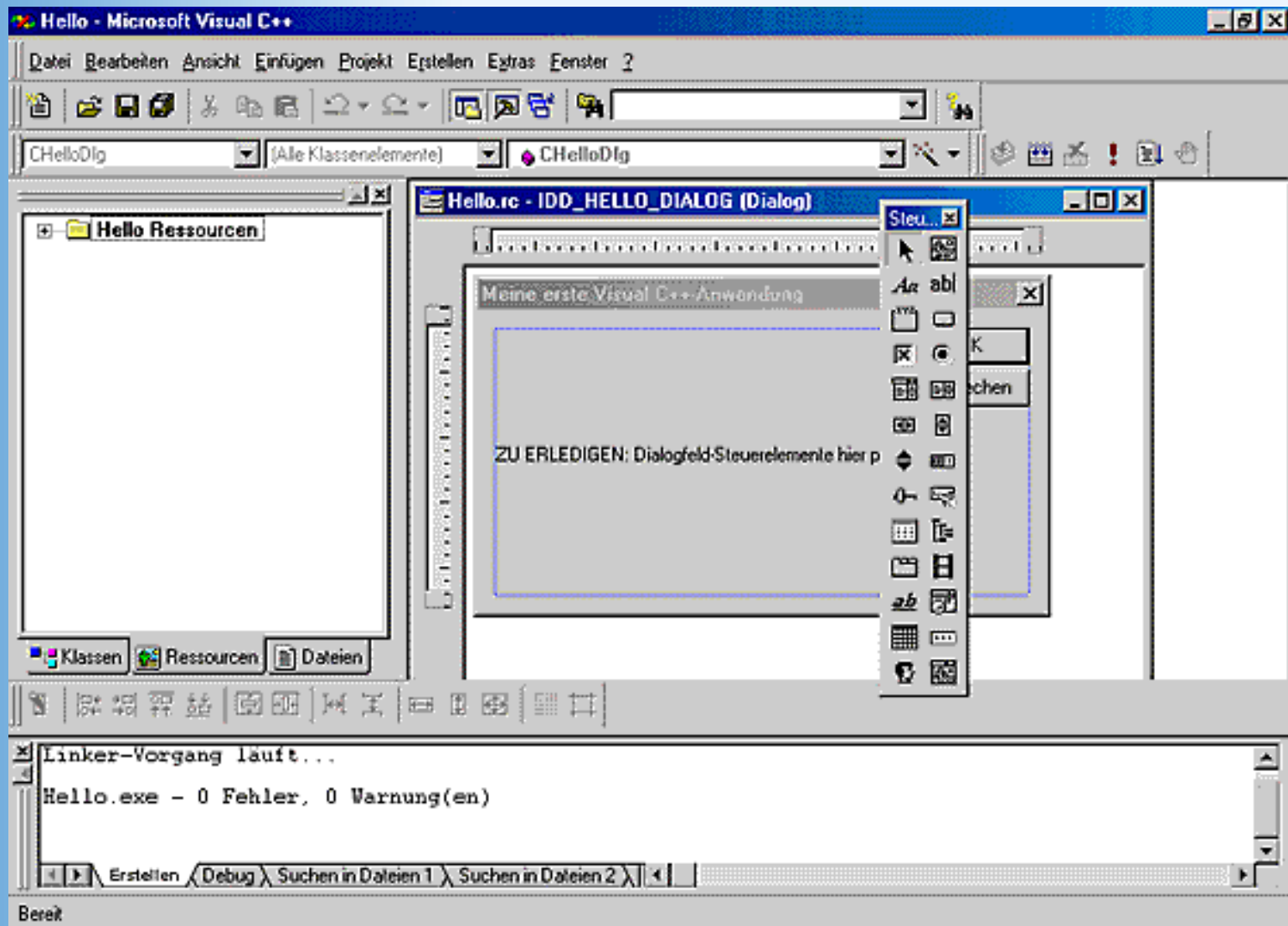


Abbildung 1.11:
Die Registerkarte Ressourcen im Arbeitsbereich

2. Erweitern Sie die Baumansicht der Ressourcen, um die verfügbaren Dialogfelder anzuzeigen. Jetzt können Sie auf das Dialogfeld `IDD_HELLO_DIALOG` doppelklicken, um das Fenster im Editorbereich von Visual Studio zu öffnen.

3. Markieren Sie den im Dialogfeld angezeigten Text, und löschen Sie ihn durch Drücken der (Entf)-Taste.

4. Markieren Sie die Schaltfläche **Abbrechen**, ziehen Sie sie an den unteren Rand des Dialogfelds, und verändern Sie die Größe der Schaltfläche, so daß sie die gesamte Breite des Layoutbereichs des

Fensters einnimmt (siehe Abbildung 1.12).

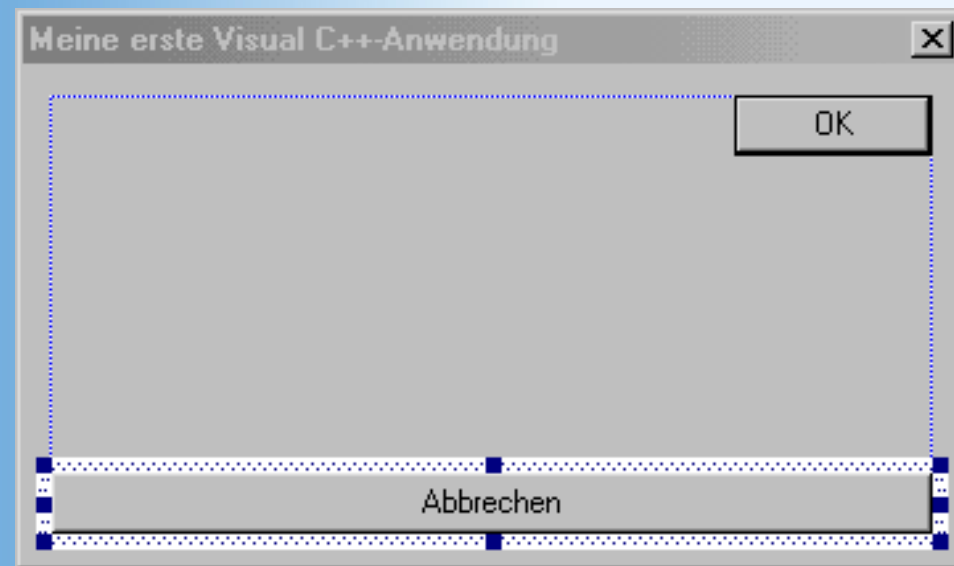


Abbildung 1.12:
Die Schaltfläche Abbrechen platzieren

5. Klicken Sie mit der rechten Maustaste über der Schaltfläche **Abbrechen**. Es erscheint das in Abbildung 1.13 dargestellte Kontextmenü. Wählen Sie aus dem Kontextmenü den Befehl **Eigenschaften**, um das in Abbildung 1.14 gezeigte Eigenschaftsdialogfeld zu öffnen.

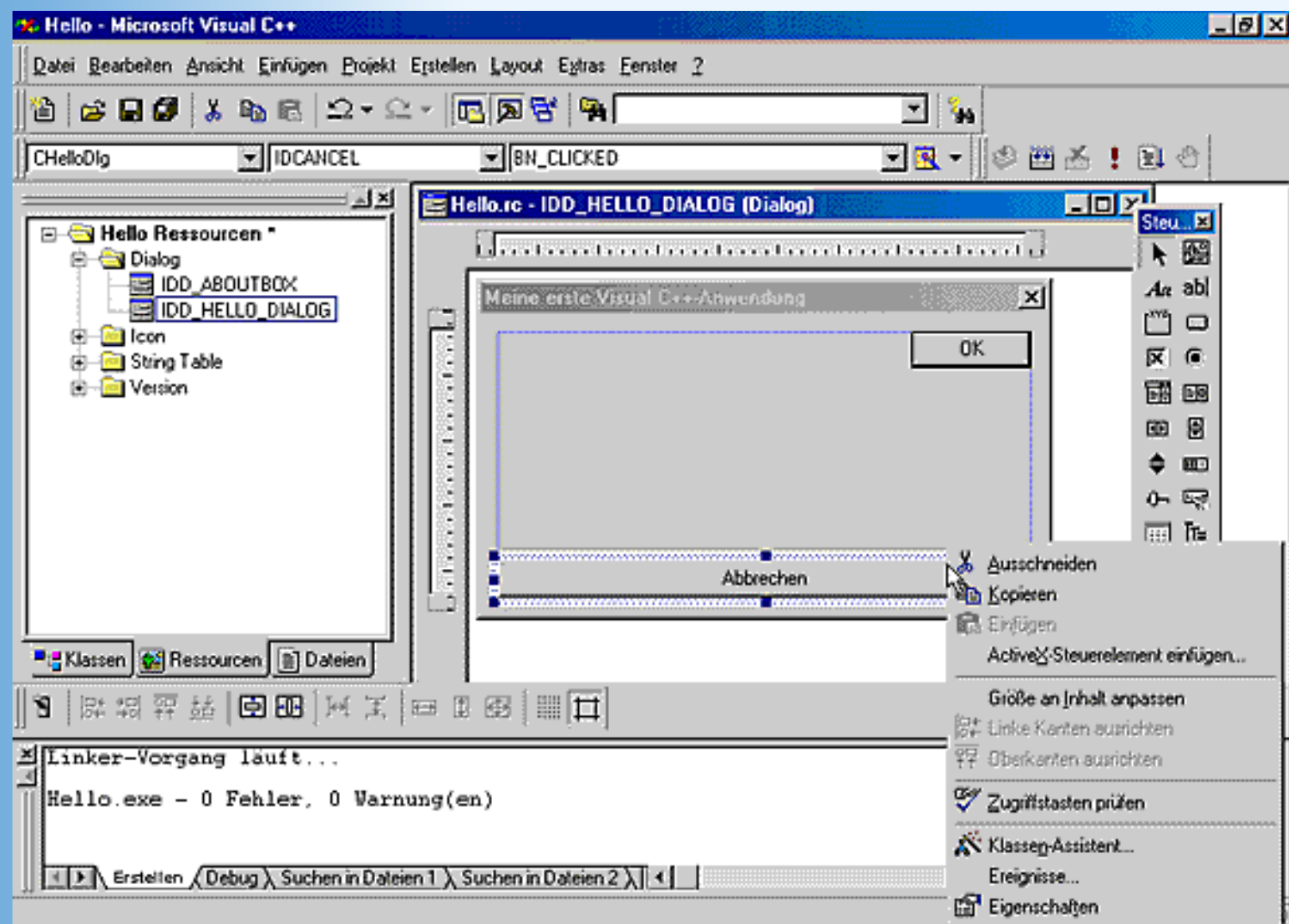


Abbildung 1.13:

Mit der rechten Maustaste klicken, um ein Kontextmenü zu öffnen



Abbildung 1.14:
Das Dialogfeld Schaltfläche Eigenschaften für die Schaltfläche Abbrechen

6. Ändern Sie den Wert im Feld **Titel** in **&Schließen**. Schließen Sie das Eigenschaftsdialogfeld, indem Sie auf das **Schliessen**-Symbol in der oberen rechten Ecke des Dialogfelds klicken.

7. Verschieben Sie die Schaltfläche **OK** in die Mitte des Fensters, und ändern Sie die Größe der Schaltfläche entsprechend Abbildung 1.15.

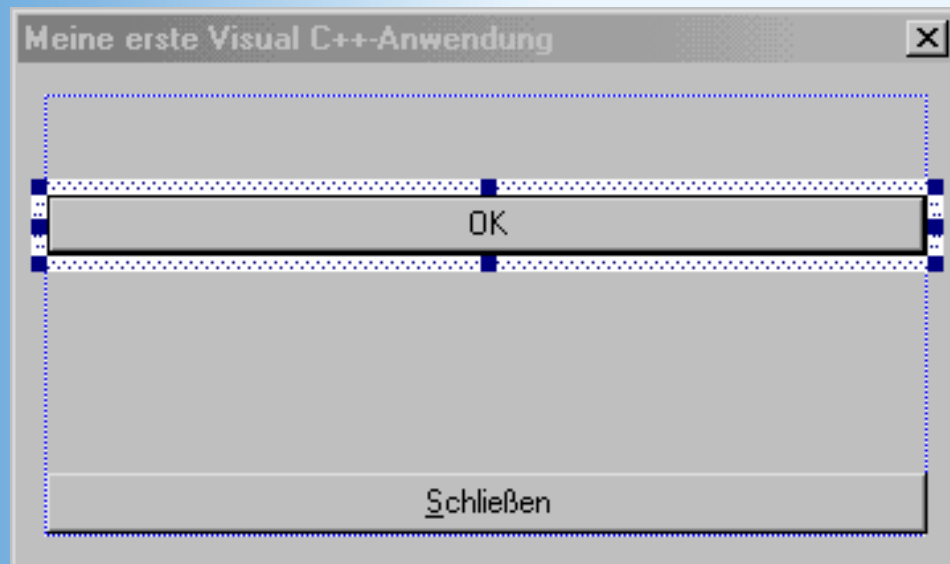


Abbildung 1.15:
Die Schaltfläche OK platzieren

8. Im Eigenschaftsdialogfeld der Schaltfläche **OK** ändern Sie die ID in **IDHELLO** und den Titel in **&Hello**.

9. Wenn Sie nun die Anwendung kompilieren und ausführen, entspricht sie dem gerade vorgenommenen Entwurf, wie es Abbildung 1.16 zeigt.



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

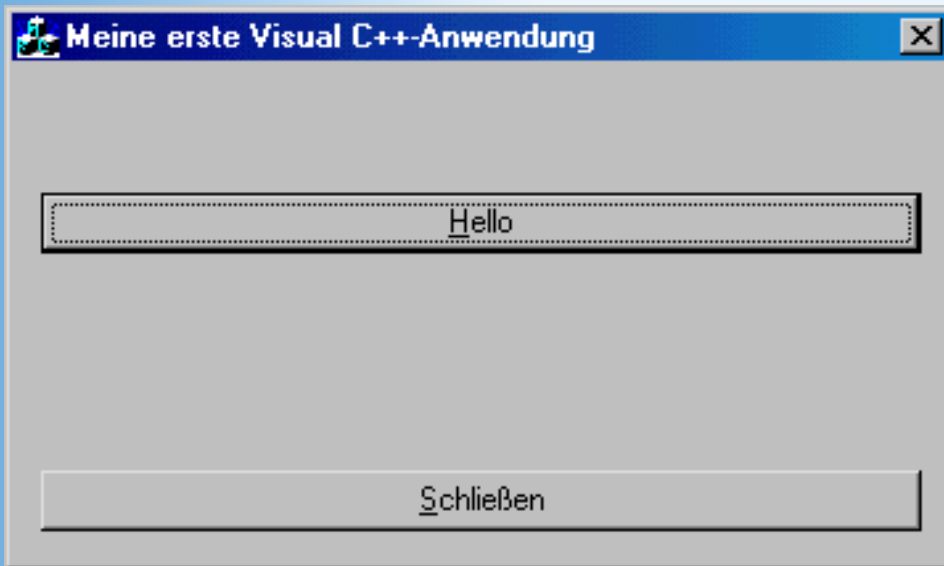


Abbildung 1.16:
Die neu gestaltete Anwendung ausführen

Code in die Anwendung aufnehmen

Über den Klassen-Assistenten von Visual C++ können Sie das Dialogfeld mit Code verbinden. Mit dem Klassen-Assistenten erstellen Sie die Tabelle der Windows-Nachrichten, die die Anwendung empfangen kann, einschließlich der zu verarbeitenden Funktionen. Auf diese Angaben greifen die MFC-Makros zurück, um die Funktionalität mit Windows-Steuerelementen zu verbinden. Die Funktionalität für Ihre erste Beispielanwendung weisen Sie in folgenden Schritten zu:

1. Um der Schaltfläche **Hello** eine Funktion zuzuordnen, klicken Sie mit der rechten Maustaste über der Schaltfläche und wählen **Klassen-Assistent** aus dem Kontextmenü.
2. War die Schaltfläche **Hello** bereits markiert, als Sie den Klassen-Assistenten geöffnet haben, ist sie bereits in der Liste der verfügbaren Objekt-IDs ausgewählt, wie es Abbildung 1.17 zeigt.

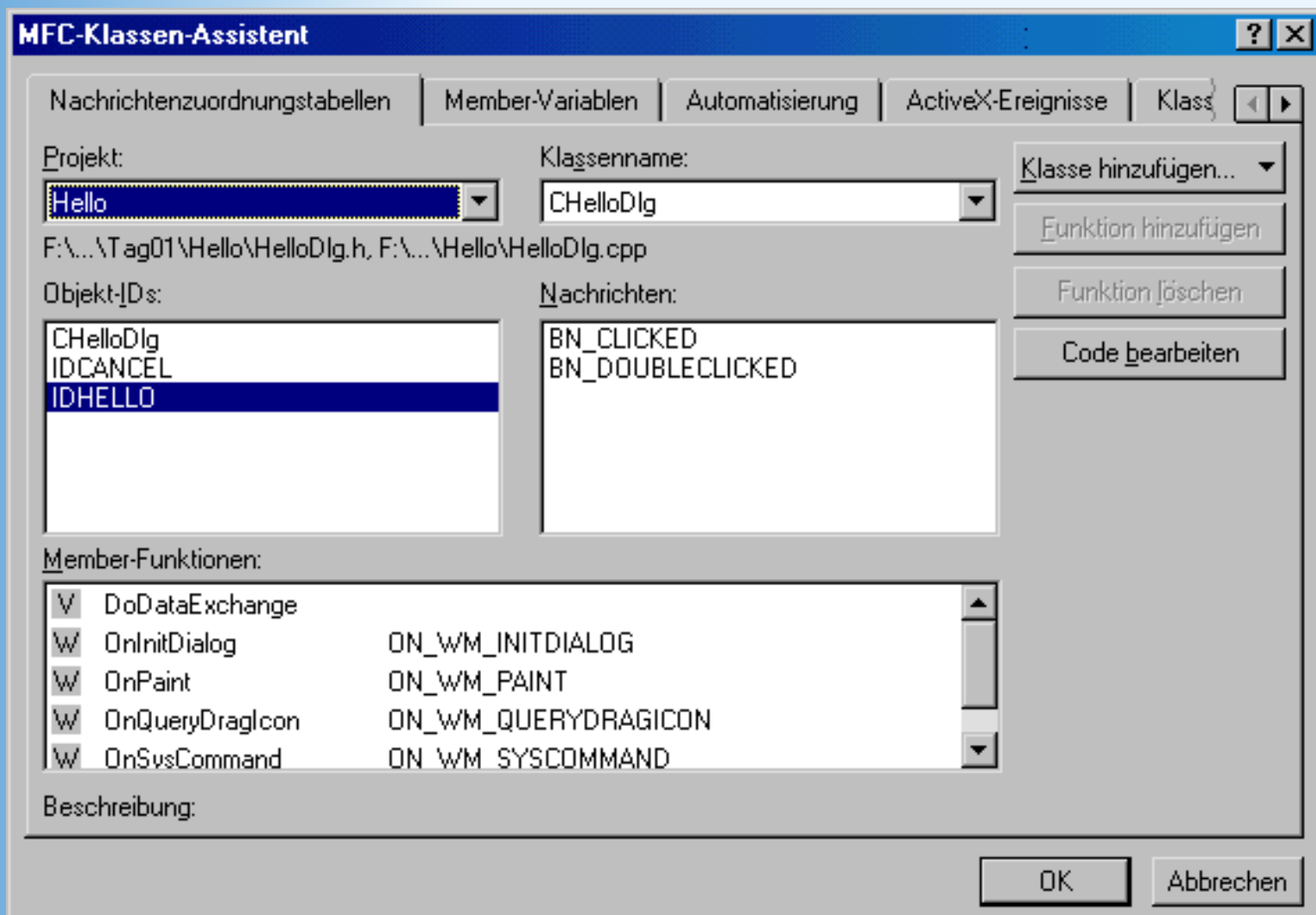


Abbildung 1.17:
Der Klassen-Assistent

3. Bei markiertem Eintrag **IDHELLO** in der Liste **Objekt-IDs** wählen Sie **BN_CLICKED** in der Liste der Nachrichten und klicken auf die Schaltfläche **Funktion hinzufügen**. Daraufhin öffnet sich das in Abbildung 1.18 dargestellte Dialogfeld **Member-Funktion hinzufügen**. Dieses Dialogfeld enthält einen Vorschlag für den Funktionsnamen. Klicken Sie auf **OK**, um die Funktion zu erzeugen und sie in die Nachrichtenzuordnungstabelle aufzunehmen.

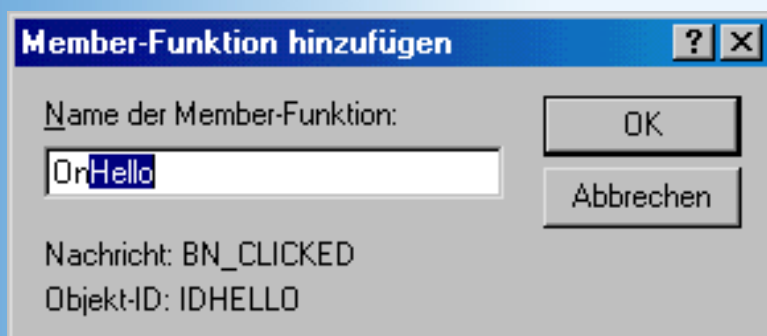


Abbildung 1.18:
Das Dialogfeld Member-Funktion hinzufügen des Klassen-Assistenten

4. Nachdem Sie die Funktion für die Klicknachricht der Schaltfläche **Hello** hinzugefügt haben, markieren Sie die Funktion **OnHello** in der Liste der verfügbaren Funktionen, wie es aus Abbildung 1.19 hervorgeht. Klicken Sie auf die Schaltfläche **Code bearbeiten**. Der Cursor steht nun im Quellcode der Funktion, und zwar genau an dem Punkt, an dem Sie den Code für die gewünschte

Funktionalität eintragen sollten.

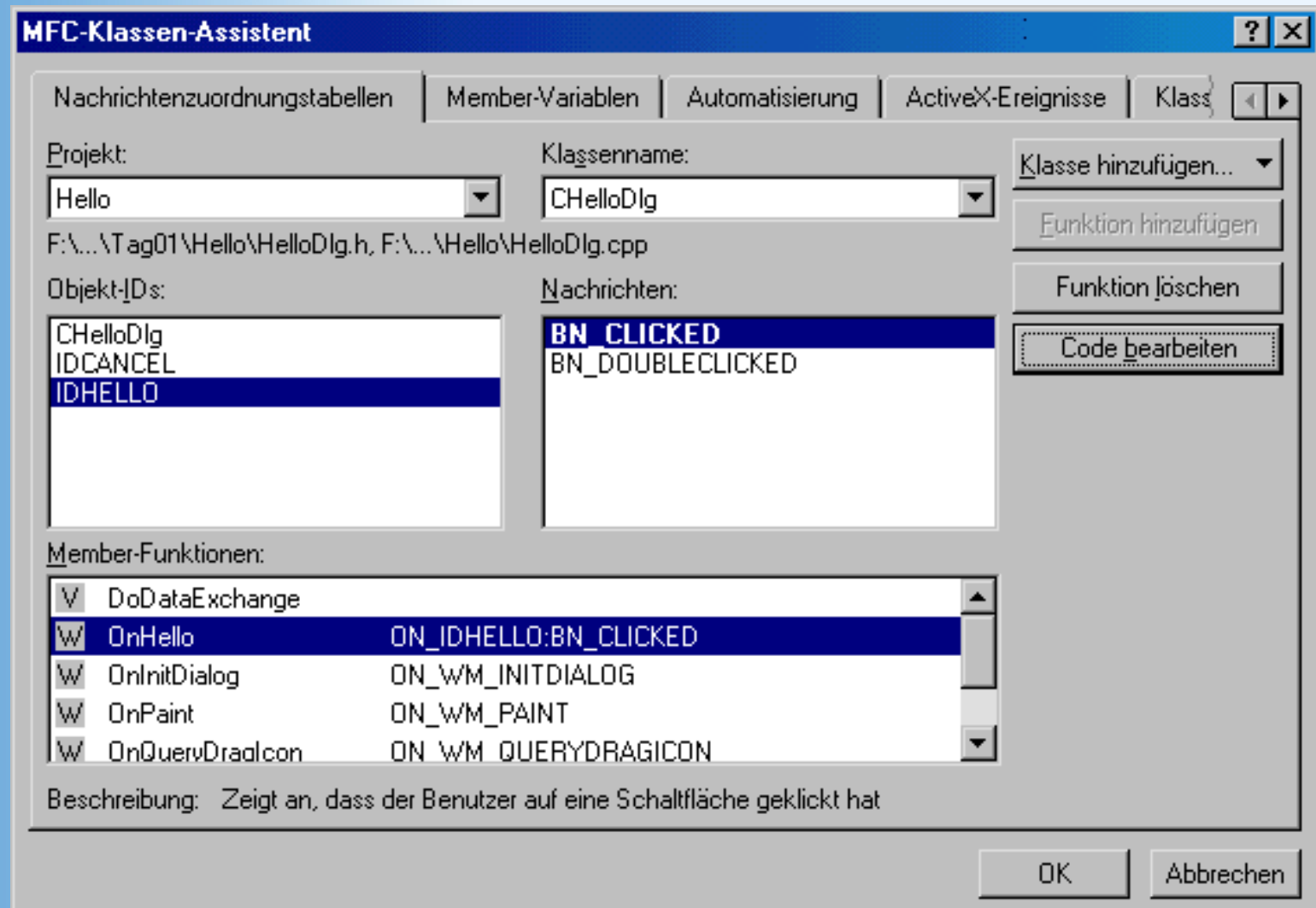


Abbildung 1.19:
Die Liste der verfügbaren Funktionen im Klassen-Assistenten

5. Tragen Sie den Code aus Listing 1.1 unmittelbar unter der TODO-Kommentarzeile ein, wie es Abbildung 1.20 verdeutlicht.

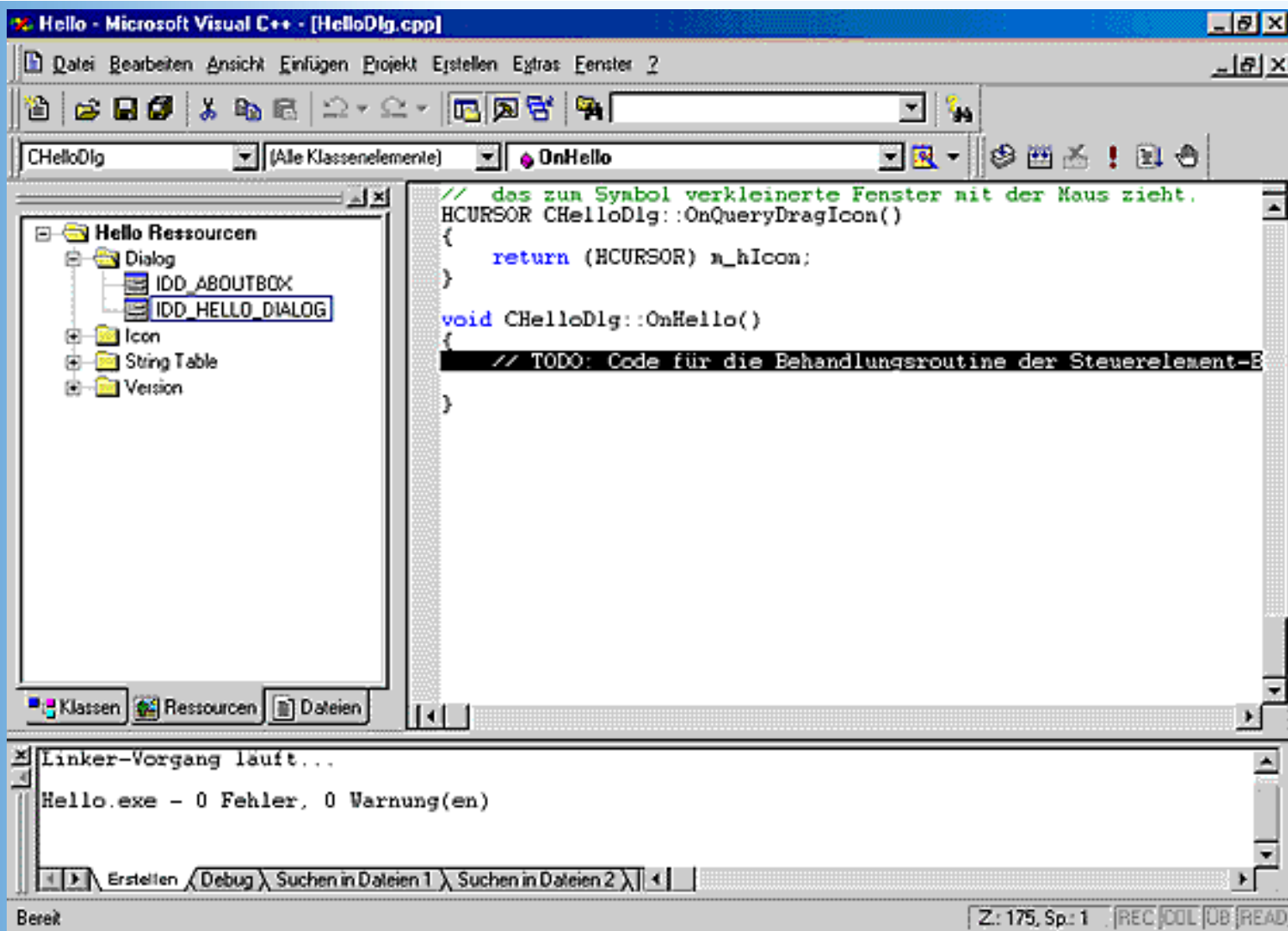


Abbildung 1.20:

In der Quellcode-Ansicht fügen Sie Listing 1.1 ein.

Listing 1.1: HELLODLG.CPP - Die Funktion OnHello

```

1: void CHelloDlg::OnHello()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Âbenachrichtigung hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Benutzer begrüßen
10: MessageBox("Hello. Das ist meine erste Visual C++-Anwendung!");
11:
12: ///////////////////////////////////
13: // EIGENER CODE, ENDE
14: ///////////////////////////////////
15: }

```

6. Wenn Sie die Anwendung kompilieren und ausführen, sollte nach Anklicken der Schaltfläche **Hello** die in Abbildung 1.21 gezeigte Nachricht erscheinen.

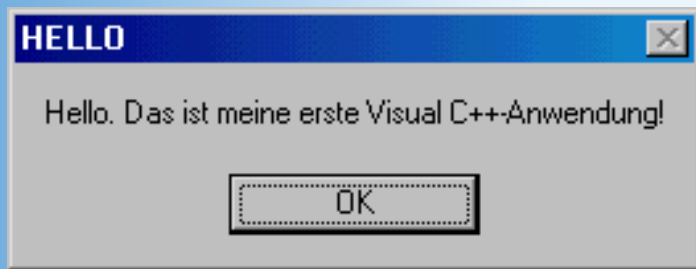


Abbildung 1.21:
Die Anwendung begrüßt Sie nun.

Der letzte Schliff

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

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

Das Symbol für das Dialogfeld

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

1. In der Baumansicht der Ressourcen im Arbeitsbereich erweitern Sie den Zweig *Icon* und markieren das Symbol `IDR_MAINFRAME`, wie es Abbildung 1.22 zeigt. Daraufhin wird das Anwendungssymbol in den Editorbereich von Visual Studio gebracht.

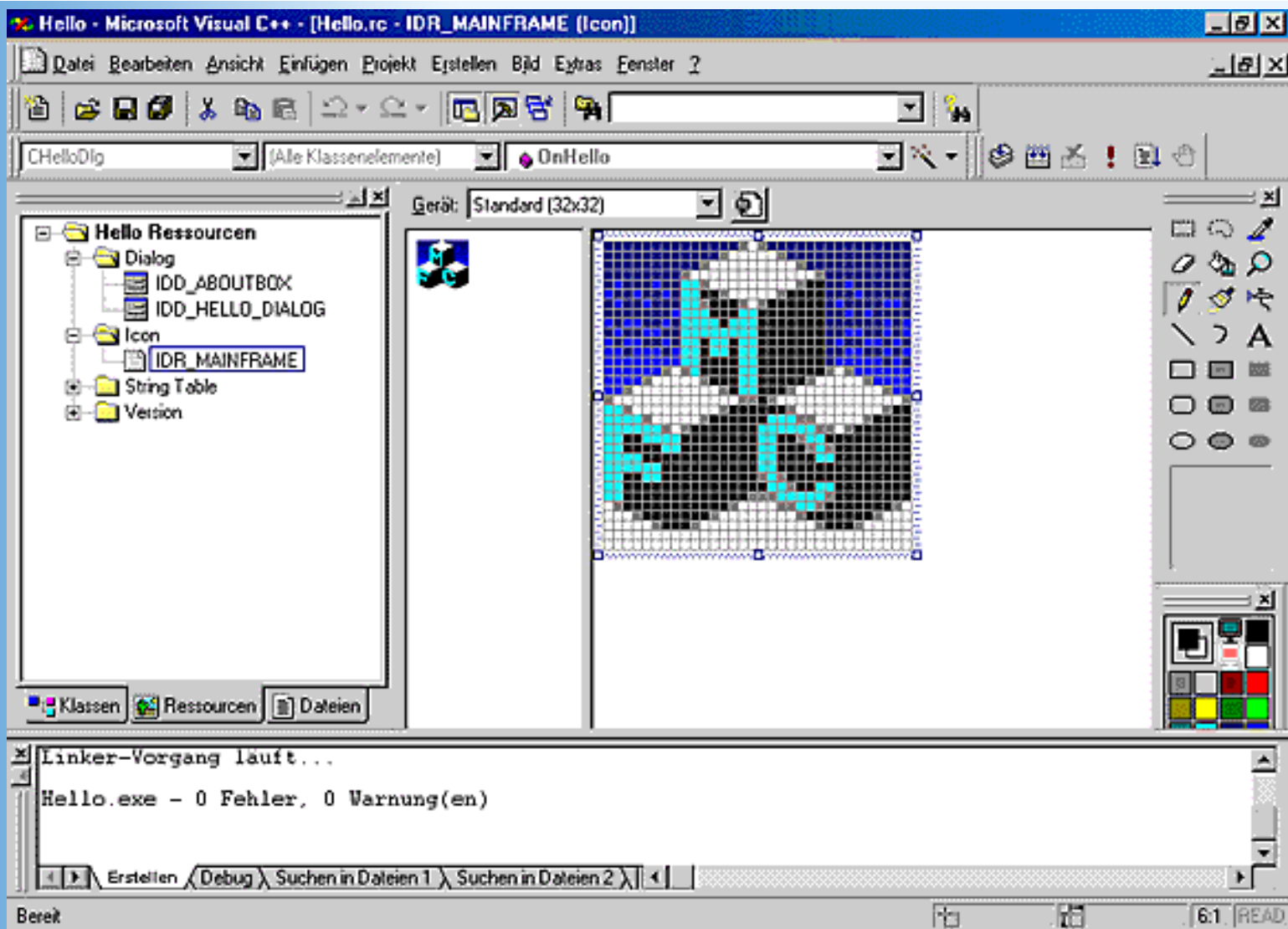


Abbildung 1.22:
Das MFC-Standardsymbol

2. Mit den zur Verfügung stehenden Zeichenwerkzeugen gestalten Sie das Symbol um, so daß ein Bild entsteht, mit dem Sie Ihre Anwendung präsentieren möchten (siehe Abbildung 1.23).

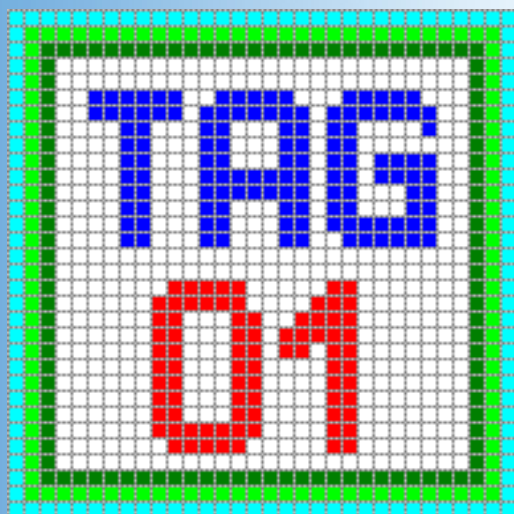


Abbildung 1.23:
Ein eigenes Symbol für die Anwendung

3. Wenn Sie die Anwendung kompilieren und ausführen, sehen Sie Ihr Symbol in der oberen linken Ecke des Anwendungsfensters. Klicken Sie auf das Symbol, und wählen Sie den Befehl **Info über**

Hello aus dem Dropdown-Menü.

4. Im Info-Dialogfeld, das Visual C++ automatisch erstellt hat, ist eine große Version Ihres Symbols in all seiner Schönheit zu sehen (siehe Abbildung 1.24).



Hinweis

Wenn man ein Anwendungssymbol im Symbol-Editor öffnet, wird die Größe des Symbols per Vorgabe auf 32 mal 32 Pixel eingestellt. Aus der Dropdown-Liste unmittelbar über dem Zeichenbereich können Sie auch die Größe 16 mal 16 wählen. Die Symbole sollten Sie in beiden Größen zeichnen, da es Fälle gibt, in denen das große Symbol angezeigt wird, während in bestimmten anderen Fällen das kleine Symbol zu sehen ist. Beide Symbole sollten aber mit dem gleichen Bild die Anwendung repräsentieren.

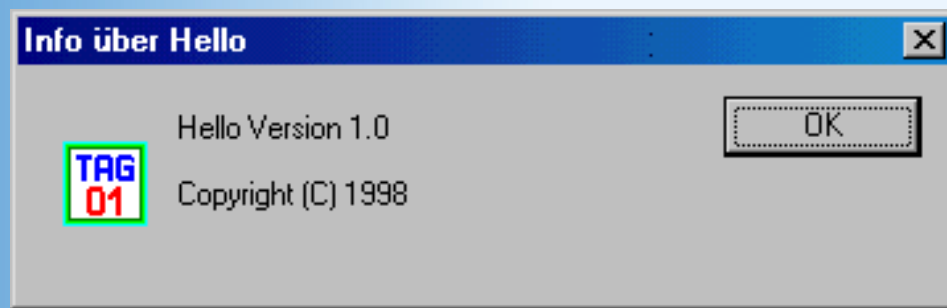


Abbildung 1.24:
Das Info-Fenster der Anwendung

Die Schaltflächen Maximieren und Minimieren hinzufügen

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

1. Markieren Sie das Dialogfenster selbst, als wollten Sie das Fenster in der Größe verändern.
2. Klicken Sie mit der rechten Maustaste, und wählen Sie aus dem Kontextmenü den Befehl **Eigenschaften**.
3. Gehen Sie auf die Registerkarte **Formate**, wie es Abbildung 1.25 zeigt.

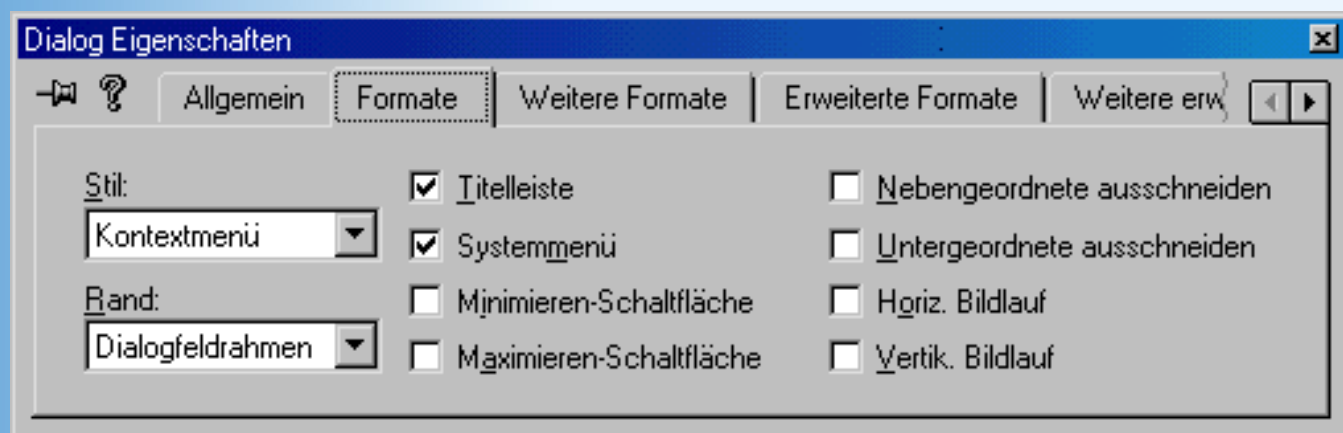


Abbildung 1.25:
Hier aktivieren/deaktivieren Sie die Schaltflächen Minimieren und Maximieren.

4. Schalten Sie die Kontrollkästchen **Minimieren-Schaltfläche** und **Maximieren-Schaltfläche** ein. Kompilieren und starten Sie nun die Anwendung. Die Schaltflächen **Minimieren** und **Maximieren**

erscheinen in der Titelleiste, wie es in Abbildung 1.26 zu sehen ist.

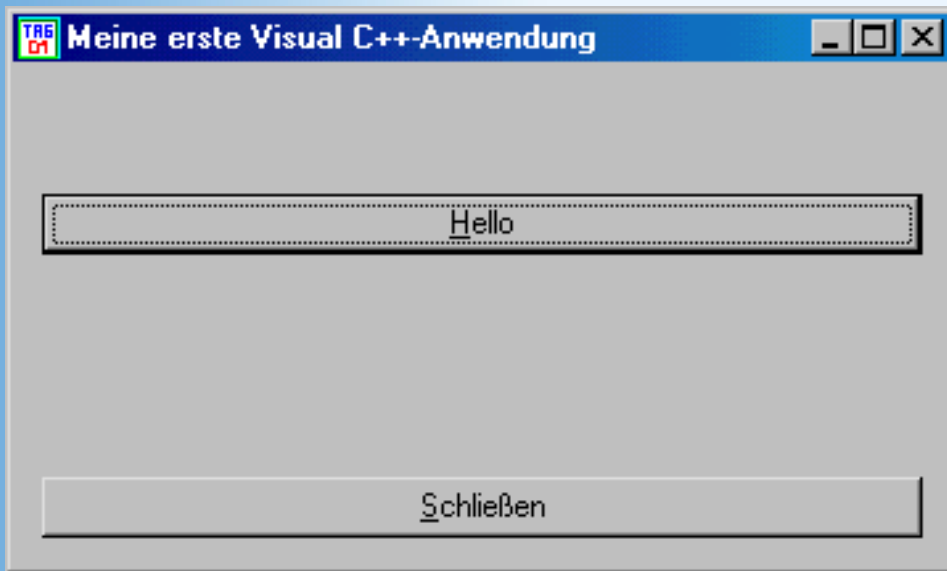


Abbildung 1.26:
Das Anwendungsfenster mit den Schaltflächen Minimieren und Maximieren

Zusammenfassung

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

Fragen und Antworten

Frage:

Wie kann man statt dem Anwendungsnamen einen anderen Titel im Meldungsfeld anzeigen?

Antwort:

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

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

Frage:

Kann ich den Text im Info-Fenster ändern, um den Namen meiner Firma und weiterführende Copyright-Informationen anzugeben?

Antwort:

Ja. Das Info-Fenster befindet sich im Ordner `Dialog` auf der Registerkarte `Ressourcen` des Arbeitsbereichs. Wenn Sie auf das Dialogfeld `IDD_ABOUTBOX` doppelklicken, wird das

Info-Dialogfeld im Dialog-Editor geöffnet. Sie können nun das Dialogfeld in jeder gewünschten Weise gestalten.

Workshop

Die Kontrollfragen im Workshop sollen Ihnen helfen, die neu erworbenen Kenntnisse zu den behandelten Themen zu festigen. Die Übungen geben Ihnen die Möglichkeit, praktische Erfahrungen mit dem gelernten Stoff zu sammeln. Die Antworten zu den Kontrollfragen finden Sie im Anhang mit den Antworten zu den Tagen.

Kontrollfragen

1. Wie ändert man den Titel (die Aufschrift) einer Schaltfläche?
2. Welche Aufgaben erledigt man mit dem Anwendungs-Assistenten von Visual C++?
3. Wie weist man dem Klickereignis einer Schaltfläche Funktionalität zu?

Übung

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

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](http://www.mut.com/lesecke/buecher/vcplus6/kap01.htm).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1**Tag 2****Steuerelemente**

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

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

Standardsteuerelemente von Windows

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

- Text
- Eingabefeld
- Schaltfläche
- Kontrollkästchen
- Optionsfeld
- Kombinationsfeld (auch als Dropdown-Listenfeld bezeichnet)

Diese und andere Steuerelemente stehen zur sofortigen Nutzung in Visual C++-Anwendungen bereit. Sie sind in der Steuerelementpalette im Dialog-Editor des Visual Studios untergebracht (siehe Abbildung 2.1).

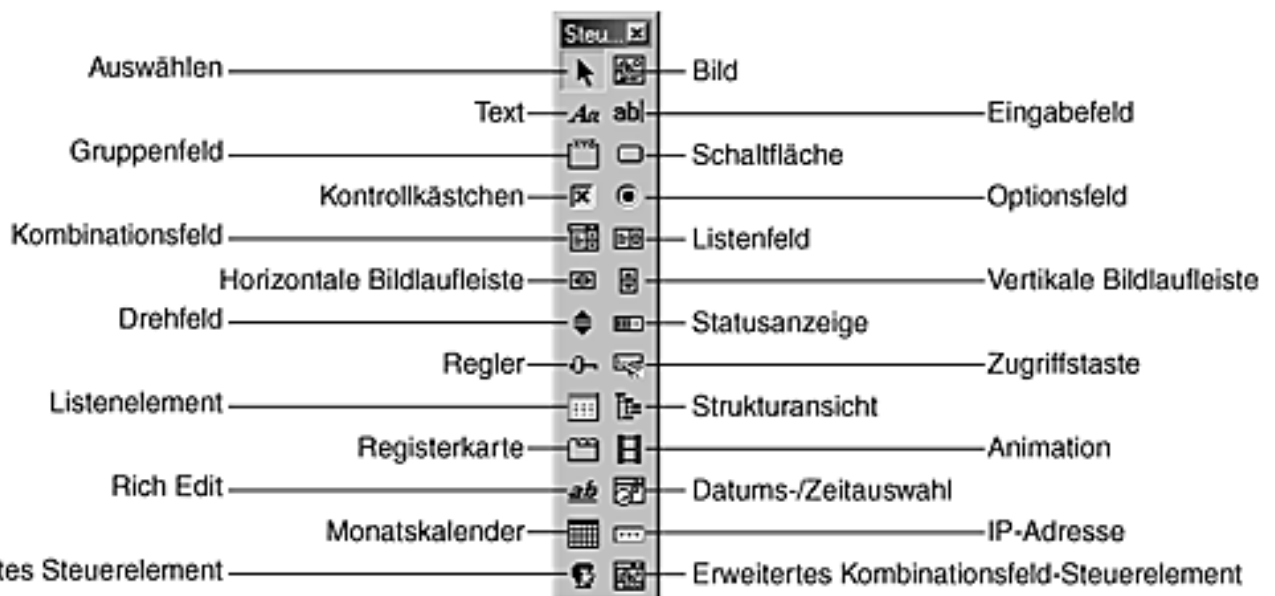


Abbildung 2.1:
Die in der Steuerelementpalette verfügbaren Standardsteuerelemente

Text

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

Eingabefelder

In einem Eingabefeld kann der Benutzer Text eingeben oder ändern. Das Steuerelement gehört zu den hauptsächlichen Werkzeugen, die dem Benutzer die Möglichkeit bieten, einer Anwendung bestimmte erforderliche Informationen bereitzustellen. Es ist in der Lage, beliebigen Text aufzunehmen, der sich auslesen und nach Bedarf weiterverarbeiten lässt. Das Eingabefeld akzeptiert ausschließlich reinen Text, Formatierungen stehen dem Benutzer nicht zur Verfügung.

Schaltflächen

Über eine Schaltfläche löst der Benutzer eine bestimmte Aktion aus. Die Beschriftung - der Titel - der Schaltfläche sollte einen Hinweis auf die Aktion liefern, die beim Klicken auf die Schaltfläche stattfindet. Eine Schaltfläche kann auch Bilder enthalten, die man - allein oder zusammen mit einer Textbeschreibung - benutzt, um den Zweck der Schaltfläche zu vermitteln.

Kontrollkästchen

Als Kontrollkästchen bezeichnet man die kleinen quadratischen Elemente, die der Benutzer durch Anklicken ein- (X) bzw. ausschaltet und damit einen bestimmten Wert setzt bzw. zurücksetzt. Grundsätzlich handelt es sich um Umschalter, gelegentlich mit einem dritten Zwischenzustand. Normalerweise verwendet man Kontrollkästchen, um diskrete Variablen vom Typ An/Aus zu steuern.

Optionsfelder

Ein Optionsfeld wird als Kreis dargestellt. Klickt der Benutzer ein Optionsfeld an, erscheint ein Punkt im Kreis. Das Optionsfeld ähnelt dem Kontrollkästchen, wird aber in einer Gruppe verwendet, in der nur ein Optionsfeld eingeschaltet sein kann. Normalerweise setzt man Optionsfelder in Gruppen mit mindestens drei Optionen ein, wobei die Optionsfelder von einem Gruppenfeld umgeben sind. Das Gruppenfeld gewährleistet die Unabhängigkeit der einzelnen Optionsfeldgruppen, wobei gleichzeitig in jeder Gruppe jeweils ein Optionsfeld eingeschaltet sein kann.

Kombinationsfelder

Ein Kombinationsfeld (oder Dropdown-Listenfeld) besteht aus einem Eingabefeld, dem eine Liste mit verfügbaren Werten zugeordnet ist. Mit einem Kombinationsfeld kann man eine Liste von Auswahlen bereitstellen, wobei der Benutzer einen Wert aus der Liste auswählen kann. Manchmal hat der Benutzer die Möglichkeit, selbst einen Wert einzutippen, wenn die Liste keinen passenden Wert enthält.

Steuerelemente in ein Fenster aufnehmen

Die heute zu erstellende Anwendung soll eine Reihe von Steuerelementen in einem Dialogfeld enthalten, wie es aus Abbildung 2.2 hervorgeht. Diese Steuerelemente haben verschiedene Aufgaben. Am oberen Rand des Fensters befindet sich ein Eingabefeld, in das der Benutzer eine Nachricht eintippen kann, die in einem Meldungsfeld erscheint, wenn er auf die Schaltfläche neben dem Feld klickt. Unterhalb dieses Eingabefelds sind zwei Schaltflächen angeordnet, die entweder das Eingabefeld mit einer Standardmeldung füllen oder einen vorhandenen Eintrag löschen. Unter diesen Schaltflächen befindet sich ein Kombinationsfeld mit einer Liste von Windows-Standardanwendungen. Wählt der Benutzer eines dieser Programme aus und klickt auf die Schaltfläche neben der Dropdown-Liste, startet das ausgewählte Programm. Darunter sind zwei Gruppen von Kontrollkästchen angeordnet. Diese wirken auf die Steuerelemente im oberen Teil des Dialogfelds: die Steuerelemente für die Anzeige einer Benutzermeldung und die Steuerelemente für die Ausführung eines anderen Programms. Die Kontrollkästchen auf der linken Seite aktivieren und deaktivieren die einzelnen Gruppen von Steuerelementen. Mit den rechten Kontrollkästchen lassen sich die Gruppen der Steuerelemente anzeigen und ausblenden. Im unteren Teil des Dialogfelds befindet sich eine Schaltfläche, die das Schließen der Anwendung bewirkt.

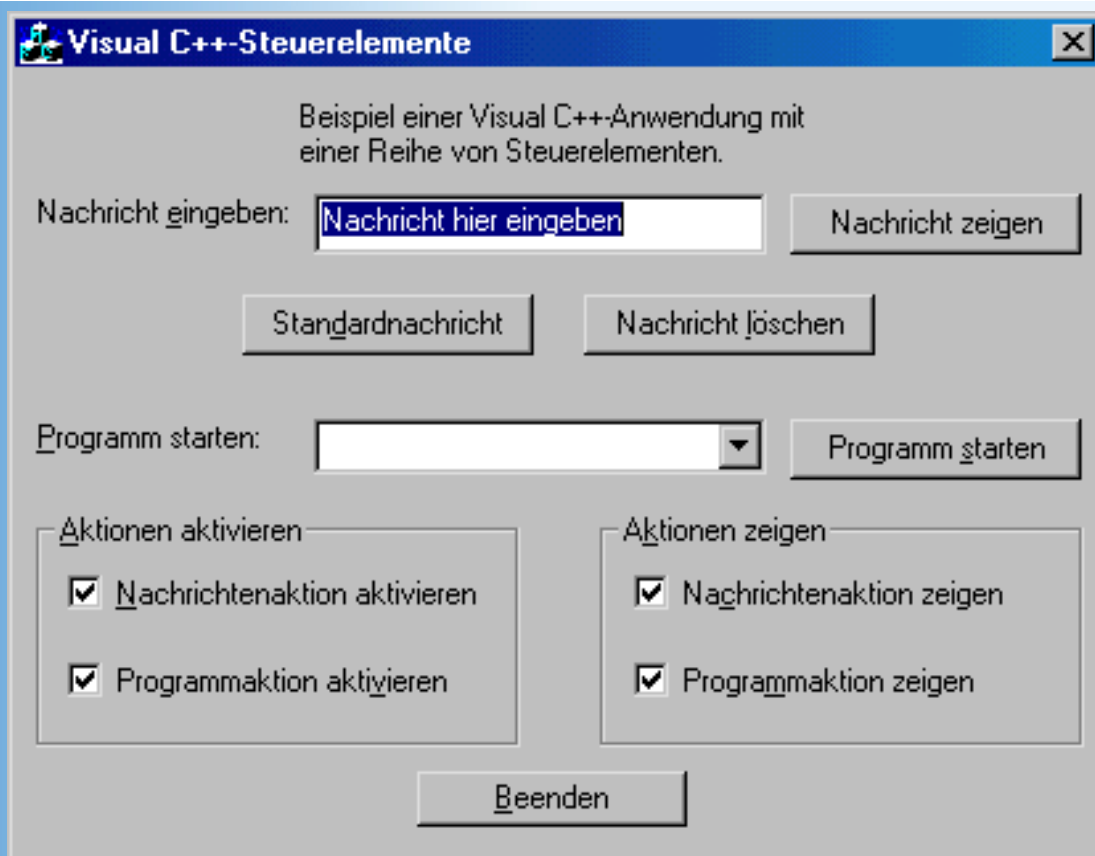


Abbildung 2.2:
Die heutige Anwendung verwendet mehrere Standardsteuerelemente.

Anwendungsgerüst und Layout des Dialogfelds erstellen

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

1. Erstellen Sie mit dem Anwendungs-Assistenten einen neuen Projekt-Arbeitsbereich, und nennen Sie das Projekt Tag2.
2. Verwenden Sie im Anwendungs-Assistenten die gleichen Einstellungen wie gestern. Legen Sie den Titel des Dialogfelds mit `Visual C++-Steuerelemente` fest.
3. Nachdem Sie das Anwendungsgerüst erstellt haben, gestalten Sie das Hauptdialogfeld wie in der weiter oben gezeigten Abbildung 2.2.
4. Konfigurieren Sie die Eigenschaften der Steuerelemente gemäß Tabelle 2.1.

Tabelle 2.1: Eigenschaftseinstellungen für die Steuerelemente im Anwendungsdialogfeld

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Beispiel einer Visual C++-Anwendung mit einer Reihe von Steuerelementen.
Text	ID	IDC_STATICMSG
	Titel	Nachricht &eingeben:

Text	ID	IDC_STATICPGM
	Titel	&Programm starten:
Eingabefeld	ID	IDC_MSG
Schaltfläche	ID	IDC_SHWMSG
	Titel	Nachricht zei&gen
Schaltfläche	ID	IDC_DFLTMSG
	Titel	&Stan&dardnachricht
Schaltfläche	ID	IDC_CLRMSG
	Titel	Nachricht &löschen
Schaltfläche	ID	IDC_RUNPGM
	Titel	Programm &starten
Schaltfläche	ID	IDC_EXIT
	Titel	&Beenden
Kombinationsfeld	ID	IDC_PROGTORUN
Gruppenfeld	ID	IDC_STATIC
	Titel	&Aktionen aktivieren
Gruppenfeld	ID	IDC_STATIC
	Titel	A&ktionen zeigen
Kontrollkästchen	ID	IDC_CKENBLMSG
	Titel	&Nachrichtenaktion aktivieren
Kontrollkästchen	ID	IDC_CKENBLPGM
	Titel	Programmaktion akti&vieren
Kontrollkästchen	ID	IDC_CKSHWMSG
	Titel	Na&chrichtenaktion zeigen
Kontrollkästchen	ID	IDC_CKSHWPGM
	Titel	Progra&mmaktion zeigen



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

5. Nachdem Sie alle genannten Steuerelemente im Dialogfeld platziert und deren Eigenschaften konfiguriert haben, öffnen Sie das Eigenschaftsdialogfeld für das Kombinationsfeld erneut. Auf der Registerkarte **Daten** tragen Sie die folgenden Werte ein, wobei Sie zum zweiten bzw. dritten Eintrag mit der Tastenkombination (Strg)+(¢) weiterschalten (siehe Abbildung 2.3).

- Editor
- Paint
- Solitär

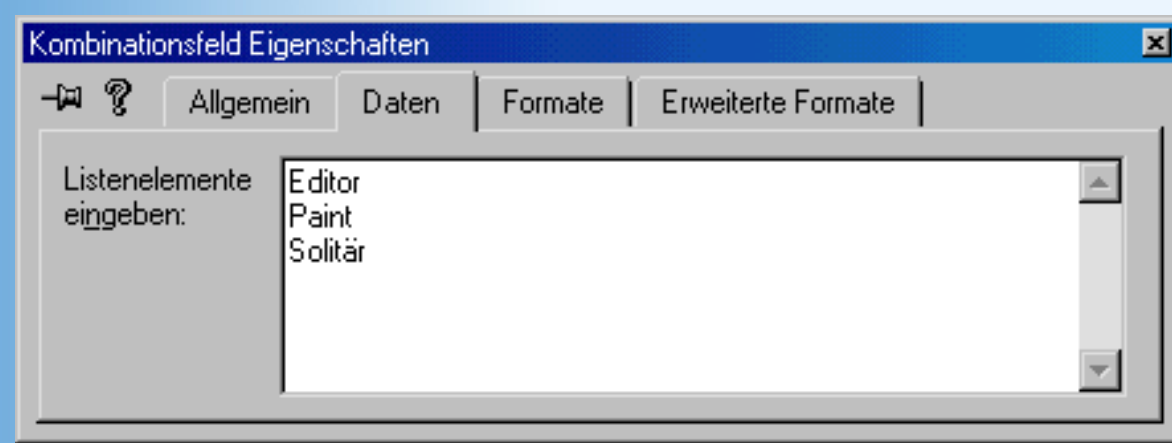


Abbildung 2.3:
Die Einträge in der Dropdown-Liste des Kombinationsfeldes legen Sie über das Eigenschaftsdialogfeld fest.

Die Tabulator-Reihenfolge von Steuerelementen festlegen

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

1. Markieren Sie im Bearbeitungsbereich von Visual Studio entweder das Dialogfeld oder eines der Steuerelemente im Fenster.
2. Wählen Sie **Layout / Tabulator-Reihenfolge**. Daraufhin erscheinen im Fenster neben den Steuerelementen Nummern. Diese kennzeichnen die Reihenfolge, in der die Navigation durch das Dialogfeld verläuft (siehe Abbildung 2.4).

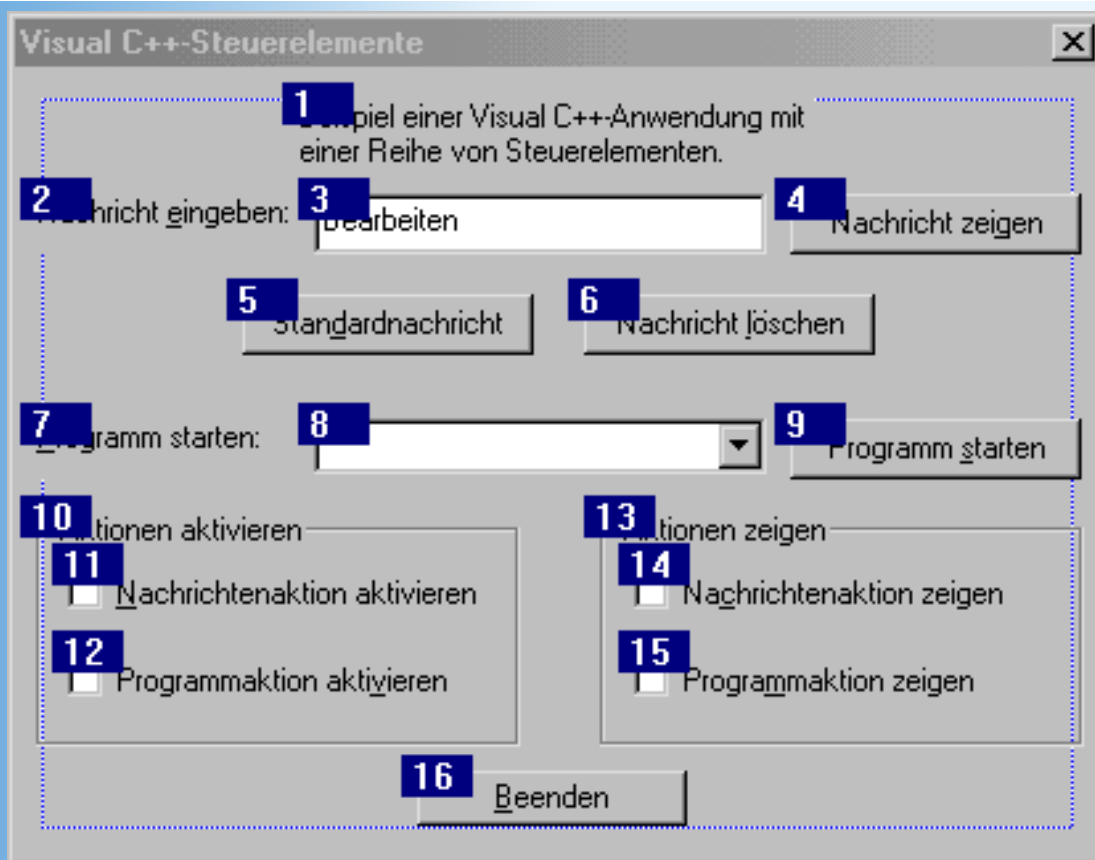


Abbildung 2.4:
Das Einschalten der Tabulator-Reihenfolge zeigt die Reihenfolge der Navigation durch das Dialogfeld an.

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

4. Nachdem Sie die Tabulator-Reihenfolge festgelegt haben, wählen Sie **Layout / Tabulator-Reihenfolge** erneut, um zum Dialog-Editor zurückzukehren.



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

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

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

1. Markieren Sie im Dialog-Editor das Dialogfeld oder eines der Steuerelemente. Klicken Sie mit der rechten Maustaste, und wählen Sie **Zugriffstasten prüfen**.
2. Wenn keine Konflikte bei Ihren mnemonischen Codes aufgetreten sind, zeigt Visual C++ ein entsprechendes Meldungsfeld an (siehe Abbildung 2.5).

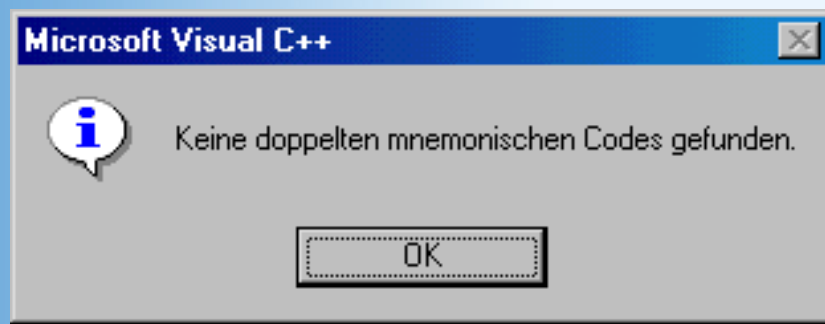


Abbildung 2.5:
Eine Prüfung der Zugriffstasten zeigt, ob Konflikte vorliegen.

3. Falls Konflikte vorhanden sind, kennzeichnet das Dialogfeld den betreffenden Buchstaben und gibt Ihnen die Möglichkeit, die Steuerelemente mit den gegensätzlichen Einträgen auswählen zu lassen (siehe Abbildung 2.6).

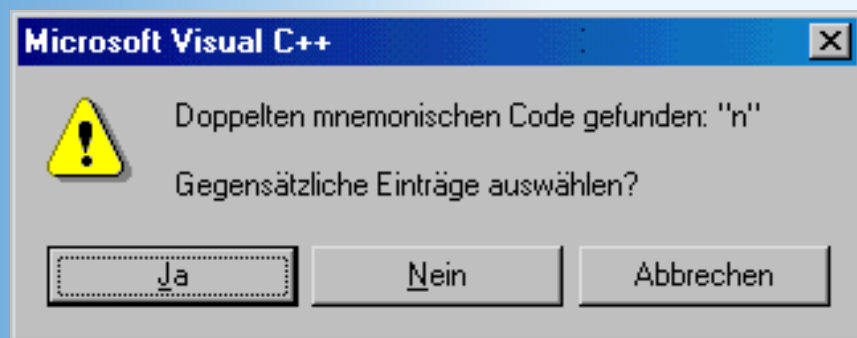


Abbildung 2.6:
Doppelte mnemonische Codes lassen sich automatisch auswählen.

Variablen mit Steuerelementen verbinden

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

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

1. Öffnen Sie den Klassen-Assistenten, wie Sie es gestern gelernt haben.
2. Aktivieren Sie die Registerkarte **Member-Variablen** (siehe Abbildung 2.7).

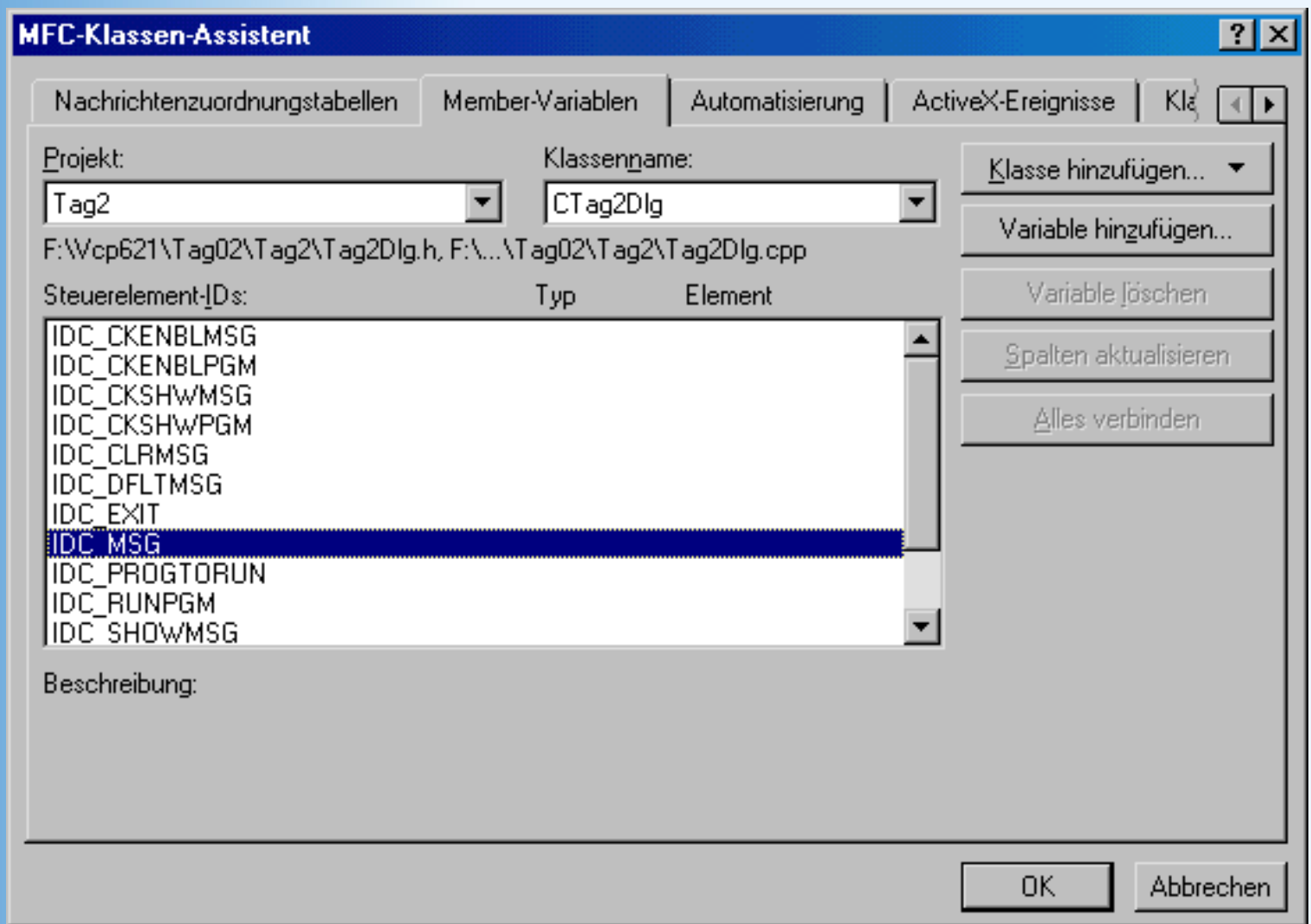


Abbildung 2.7:
Über die Registerkarte **Member-Variablen** des **Klassen-Assistenten** fügen Sie den **Steuerelementen Variablen** hinzu.

3. Markieren Sie die ID eines der Steuerelemente, denen Sie eine Variable zuordnen möchten, beispielsweise **IDC_MSG**.

4. Klicken Sie auf **Variable hinzufügen**.

5. Im Dialogfeld **Member-Variable hinzufügen** geben Sie den Variablennamen ein und legen die Kategorie sowie den Typ der Variablen fest wie es Abbildung 2.8 zeigt. Klicken Sie auf **OK**.

Member-Variable hinzufügen

Name der Member-Variablen:

Kategorie:

Variablentyp:

OK

Abbrechen

Beschreibung:
CString mit Längenprüfung

Abbildung 2.8:
Einem Steuerelement eine Variable zuordnen

6. Wiederholen Sie die Schritte 3 bis 5 für alle anderen Steuerelemente, für die Sie Variablen zuordnen müssen. In der heutigen Anwendung betrifft das die Variablen gemäß Tabelle 2.2.

Tabelle 2.2: Variablen für die Steuerelemente der Anwendung

Steuerelement	Variablenname	Kategorie	Typ
IDC_MSG	m_strMessage	Wert	CString
IDC_PROGTORUN	m_strProgToRun	Wert	CString
IDC_CKENBLMSG	m_bEnableMsg	Wert	BOOL
IDC_CKENBLPGM	m_bEnablePgm	Wert	BOOL
IDC_CKSHWMSG	m_bShowMsg	Wert	BOOL
IDC_CKSHWPGM	m_bShowPgm	Wert	BOOL



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

7. Nachdem Sie alle erforderlichen Variablen hinzugefügt haben, klicken Sie auf **OK**, um den Klassen-Assistenten zu schließen.

Steuerelemente mit Funktionalität ausstatten

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

1. Starten Sie den Klassen-Assistenten, gehen Sie auf die Registerkarte

Nachrichtenzuordnungstabellen, und markieren Sie die Funktion `OnInitDialog` in der Liste der Member-Funktionen. Die Funktion suchen Sie im Listenfeld **Member- Funktionen** auf, oder Sie wählen das Objekt `CTag2Dlg` in der Liste **Objekt-IDs** aus und markieren dann `WM_INITDIALOG` im Listenfeld **Nachrichten**, wie es Abbildung 2.9 zeigt.

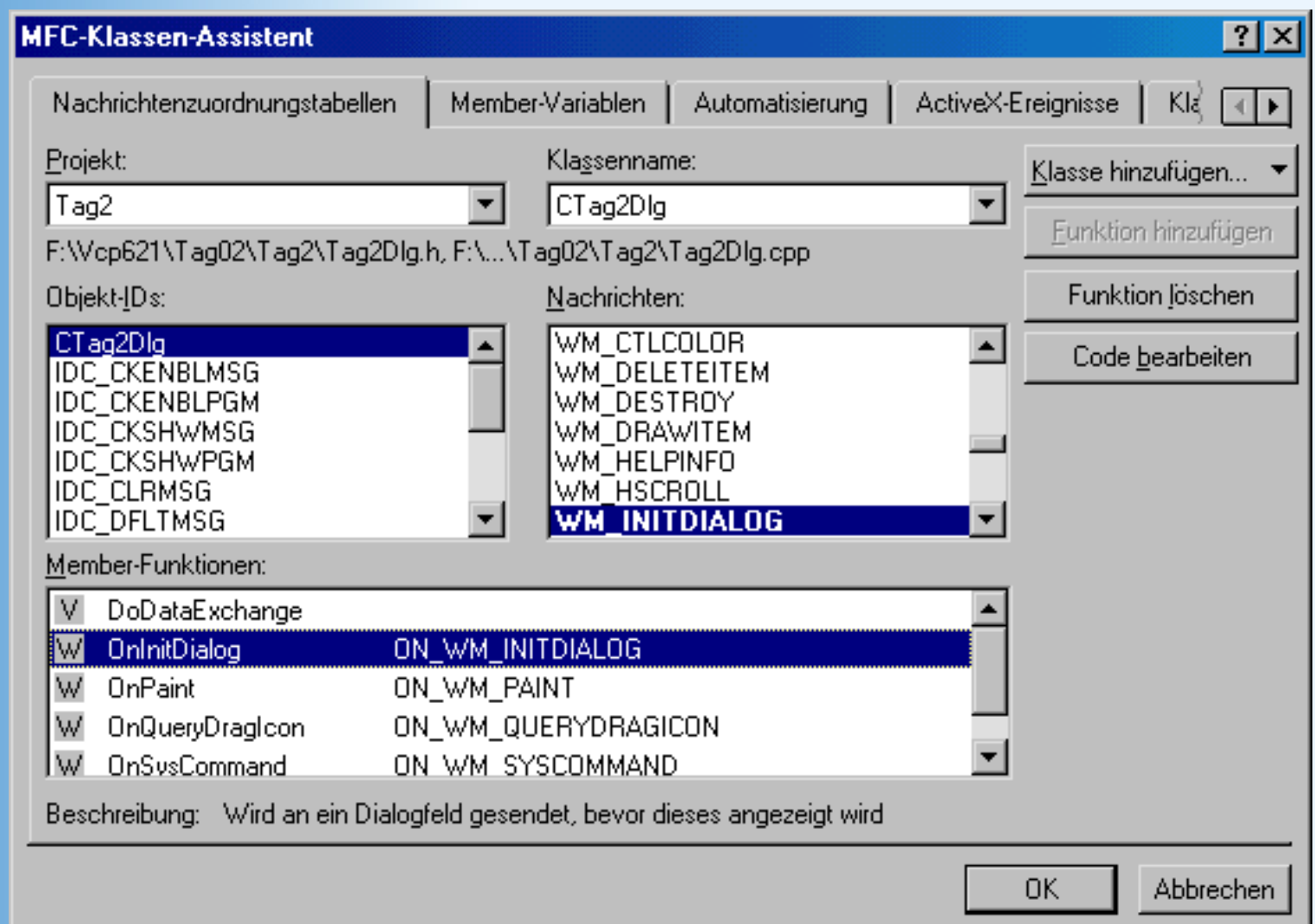


Abbildung 2.9:

Vorhandene Funktionen lassen sich mit dem Klassen-Assistenten auffinden.

2. Klicken Sie auf die Schaltfläche **Code bearbeiten**. Daraufhin gelangen Sie in den Quellcode für die Funktion `OnInitDialog`.

3. Gehen Sie zur Markierung **ZU ERLEDIGEN**, die die Stelle kennzeichnet, wo Sie mit der Codeeingabe beginnen. Geben Sie hier den Code aus Listing 2.1 ein.

Listing 2.1: TAG2DLG.CPP - In die Funktion OnInitDialog geben Sie den Initialisierungscode ein.

```

1: BOOL CTag2Dlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:     .
6:     .
7:     .
8:
9:     // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
10:
11:     //////////////////////////////////////
12:     // EIGENER CODE, ANFANG
13:     //////////////////////////////////////
14:
15:     // Standardtext in Eingabefeld eintragen
16:     m_strMessage = "Nachricht hier eingeben";
17:
18:     // Alle Kontrollkästchen einschalten
19:     m_bShowMsg = TRUE;
20:     m_bShowPgm = TRUE;
21:     m_bEnableMsg = TRUE;
22:     m_bEnablePgm = TRUE;
23:
24:     // Dialogfeld mit den Werten aktualisieren
25:     UpdateData(FALSE);
26:
27:     //////////////////////////////////////
28:     // EIGENER CODE, ENDE
29:     //////////////////////////////////////
30:
31:     return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
    soll den Fokus erhalten
32: }

```



Listing 2.1 zeigt nur einen Ausschnitt der Funktion `OnInitDialog`. Die Listings im gesamten Buch konzentrieren sich nur auf den Code, der hinzuzufügen oder zu modifizieren ist, und zeigen der Übersichtlichkeit wegen nicht den gesamten Code aller Funktionen. (Damit bleibt auch der Umfang des Buches in einem vernünftigen Rahmen.) Um Ihre Kenntnisse zu MFC und Visual C++ zu erweitern, sollten Sie sich auch den Code ansehen, der hier ausgelassen wurde. Versuchen Sie zu verstehen, was der Code bewirkt.



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

Der Initialisierungscode ist nicht weiter kompliziert. Das Programm stellt als erstes in das Eingabefeld eine anfängliche Nachricht, die Sie dem Benutzer anzeigen wollen, und setzt dann alle Kontrollkästchen in den eingeschalteten Zustand. Die letzte Zeile des hinzugefügten Codes verdient etwas mehr Beachtung.

Die Funktion `UpdateData` bildet den Schlüssel für die Arbeit mit Steuerelementvariablen in Visual C++. Die Funktion übernimmt die Daten aus den Variablen und aktualisiert mit den Werten die Steuerelemente auf dem Bildschirm. Umgekehrt übernimmt die Funktion die Daten aus den Steuerelementen und füllt die zugeordneten Variablen mit allen vom Benutzer geänderten Werten. Die Richtung der Datenübertragung steuert man mit dem an die Funktion `UpdateData` übergebenen Argument. Ist das Argument auf `FALSE` gesetzt, werden die Werte in den Variablen an die Steuerelemente im Fenster übertragen. Übergibt man `TRUE` als Argument, erhalten die Variablen die aktuellen Werte der Steuerelemente im Fenster. Welcher Wert an die Funktion zu übergeben ist, hängt also davon ab, in welcher Richtung die Aktualisierung stattfinden soll. Nachdem Sie eine oder mehrere Variablen in Ihrem Code aktualisiert haben, müssen Sie `UpdateData` aufrufen und `FALSE` als Argument übergeben. Wenn Sie die Variablen lesen müssen, um deren aktuellen Wert zu erhalten, ist `UpdateData` mit dem Argument `TRUE` aufzurufen, bevor Sie irgendeinen Wert aus den Variablen verarbeiten. Ein Gefühl für diese Vorgehensweise werden Sie entwickeln, wenn Sie mehr Code in Ihre Anwendung aufnehmen.

Die Anwendung schließen

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

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion zur Objekt-ID `IDC_EXIT` für die Nachricht `BN_CLICKED` hinzu, wie Sie es gestern kennengelernt haben.
2. Klicken Sie auf die Schaltfläche **Code bearbeiten**, um in den Quelltext der neu hinzugefügten Funktion zu gelangen.
3. Geben Sie den Code von Listing 2.2 ein.



Die Markierungen `TODO` und `ZU ERLEDIGEN` sind vollkommen identisch. An manchen Stellen ist im Visual Studio offensichtlich nur die deutsche Übersetzung verlorengegangen - oder vergessen worden. Lassen Sie sich durch diesen Schönheitsfehler nicht verwirren.

Listing 2.2: TAG2DLG.CPP - Die Funktion OnExit

```

1: void CTag2Dlg::OnExit()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Programm beenden
10: OnOK();
11:
12: //////////////////////////////////
13: // EIGENER CODE, ENDE
14: //////////////////////////////////
15: }

```

Ein einziger Funktionsaufruf in der Funktion `OnExit` schließt das Fenster und beendet die Anwendung. Woher kommt diese `OnOK`-Funktion, und warum mußten Sie sie nicht in der gestrigen Anwendung aufrufen? Zwei Funktionen, `OnOK` und `OnCancel`, sind in der Basisklasse `CDialog` definiert, von der Ihre Klasse `CTag2Dlg` abgeleitet ist. In der Klasse `CDialog` hat die Nachrichtenzuordnungstabelle bereits die Objekt-IDs der Schaltflächen OK und Abbrechen mit den Funktionen `OnOK` bzw. `OnCancel` verbunden, so daß Schaltflächen mit diesen IDs automatisch die entsprechenden Funktionen aufrufen. Wenn Sie die Objekt-ID der Schaltfläche Beenden mit `IDOK` festgelegt hätten, müßten Sie der Schaltfläche keinerlei Code zuordnen, solange Sie nicht die grundlegende Funktionalität von `OnOK` überschreiben möchten.

Die Nachricht des Benutzers anzeigen

Es sollte ein Leichtes sein, die vom Benutzer in das Eingabefeld eingegebene Nachricht anzuzeigen, da dieser Vorgang den gestern beschriebenen Abläufen ähnelt. Sie können der Schaltfläche Nachricht zeigen eine Funktion zuordnen und die Funktion `MessageBox` aufrufen, wie es aus Listing 2.3 hervorgeht.

Listing 2.3: TAG2DLG.CPP - Die Funktion OnShwmsg zeigt die Benutzernachricht an.

```

1: void CTag2Dlg::OnShwmsg()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Nachricht anzeigen
10: MessageBox(m_strMessage);
11:
12: //////////////////////////////////

```

```

13: // EIGENER CODE, ENDE
14: //////////////////////////////////////
15: }

```

Wenn Sie die Anwendung in dieser Phase kompilieren und ausführen, tritt ein Problem mit diesem Code zutage. Es erscheint der String, mit dem Sie die Variable `m_strMessage` in der Funktion `OnInitDialog` initialisiert haben, und nicht die Nachricht, die der Benutzer in das Eingabefeld eingibt.

Das ist darauf zurückzuführen, daß Sie die Variable noch nicht mit dem Inhalt des Steuerelements im Fenster aktualisiert haben. Es ist `UpdateData` mit auf `TRUE` gesetztem Argument aufzurufen, um die Werte der Steuerelemente zu lesen und die Variablen damit zu initialisieren, bevor Sie die Funktion `MessageBox` aufrufen. Ändern Sie die Funktion `OnShwmsg` gemäß Listing 2.4 ab.

Listing 2.4: TAG2DLG.CPP - Aktualisierte Version der Funktion `OnShwmsg`

```

1: void CTag2Dlg::OnShowmsg()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Nachrichtenvariable mit Benutzereingabe aktualisieren
10: UpdateData(TRUE);
11:
12: // Nachricht anzeigen
13: MessageBox(m_strMessage);
14:
15: //////////////////////////////////////
16: // EIGENER CODE, ENDE
17: //////////////////////////////////////
18: }

```

Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, sollte die in das Eingabefeld eingegebene Meldung angezeigt werden, wie es Abbildung 2.10 verdeutlicht.

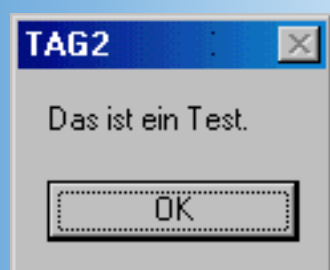


Abbildung 2.10:
Die in das Eingabefeld eingetippte Nachricht wird dem Benutzer angezeigt.

Die Nachricht des Benutzers löschen

Falls der Benutzer ein leeres Eingabefeld vorfinden möchte, bevor er eine Nachricht eintippt, können Sie der Schaltfläche Nachricht löschen eine Funktion zuordnen, um den Inhalt des Eingabefeldes zu leeren. Die Funktion fügen Sie in der gewohnten Weise über den Klassen-Assistenten hinzu. Die Funktionalität realisieren Sie ganz einfach, indem Sie die Variable `m_strMessage` auf einen leeren String setzen und dann die Steuerelemente im Fenster aktualisieren, um diesen Wert wiederzugeben. Der entsprechende Code ist in Listing 2.5 zu sehen.

Listing 2.5: TAG2DLG.CPP - Die Funktion OnClrmsg

```

1: void CTag2Dlg::OnClrmsg()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Nachricht löschen
10: m_strMessage = "";
11:
12: // Bildschirm aktualisieren
13: UpdateData(FALSE);
14:
15: //////////////////////////////////////
16: // EIGENER CODE, ENDE
17: //////////////////////////////////////
18: }
```

Die Nachrichtensteuerelemente deaktivieren und ausblenden

In bezug auf die Nachrichtensteuerelemente ist als letztes noch die Funktionalität für die Kontrollkästchen Nachrichtenaktion aktivieren und Nachrichtenaktion zeigen zu implementieren. Das erste dieser Kontrollkästchen aktiviert oder deaktiviert die Steuerelemente, die sich auf die Anzeige der Benutzernachricht beziehen. Wenn das Kontrollkästchen eingeschaltet ist, sind alle Steuerelemente aktiviert. Weist das Kontrollkästchen den ausgeschalteten Zustand auf, sind die betreffenden Steuerelemente deaktiviert. Analog dazu dient das zweite Kontrollkästchen dazu, dieselbe Gruppe der Steuerelemente anzuzeigen bzw. auszublenden. Listing 2.6 zeigt den Code für beide Funktionen.

Listing 2.6: TAG2DLG.CPP - Die Funktionen für die Kontrollkästchen Nachrichtenaktion aktivieren/zeigen.

```

1: void CTag2Dlg::OnCkenblmsg()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
```

```

7: ///////////////////////////////////////////////////
8:
9: // Aktuelle Werte vom Bildschirm holen
10: UpdateData(TRUE);
11:
12: // Kontrollkästchen 'Nachrichtenaktion aktivieren' eingeschaltet?
13: if (m_bEnableMsg == TRUE)
14: {
15: // Ja, dann alle Steuerelemente aktivieren,
16: // die für Anzeige der Nachricht relevant sind.
17: GetDlgItem(IDC_MSG)->EnableWindow(TRUE);
18: GetDlgItem(IDC_SHWMSG)->EnableWindow(TRUE);
19: GetDlgItem(IDC_DFLTMSG)->EnableWindow(TRUE);
20: GetDlgItem(IDC_CLRMSG)->EnableWindow(TRUE);
21: GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
22: }
23: else
24: {
25: // Nein, dann alle Steuerelemente deaktivieren,
26: // die für Anzeige der Nachricht relevant sind.
27: GetDlgItem(IDC_MSG)->EnableWindow(FALSE);
28: GetDlgItem(IDC_SHWMSG)->EnableWindow(FALSE);
29: GetDlgItem(IDC_DFLTMSG)->EnableWindow(FALSE);
30: GetDlgItem(IDC_CLRMSG)->EnableWindow(FALSE);
31: GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
32: }
33:
34: ///////////////////////////////////////////////////
35: // EIGENER CODE, ENDE
36: ///////////////////////////////////////////////////
37: }
38:
39: void CTag2Dlg::OnCkshwmsg()
40: {
41: // TODO: Code für die Behandlungsroutine der Steuerelement-
42: // Benachrichtigung hier einfügen
43:
44: ///////////////////////////////////////////////////
45: // EIGENER CODE, ANFANG
46: ///////////////////////////////////////////////////
47: // Aktuelle Werte vom Bildschirm holen
48: UpdateData(TRUE);
49:
50: // Kontrollkästchen 'Nachrichtenaktion zeigen' eingeschaltet?
51: if (m_bShowMsg == TRUE)
52: {
53: // Ja, dann alle Steuerelemente anzeigen, die
54: // für Anzeige der Nachricht relevant sind.

```

```

55: GetDlgItem(IDC_MSG)->ShowWindow(TRUE);
56: GetDlgItem(IDC_SHWMSG)->ShowWindow(TRUE);
57: GetDlgItem(IDC_DFLTMSG)->ShowWindow(TRUE);
58: GetDlgItem(IDC_CLRMSG)->ShowWindow(TRUE);
59: GetDlgItem(IDC_STATICMSG)->ShowWindow(TRUE);
60: }
61: else
62: {
63: // Nein, dann alle Steuerelemente ausblenden, die
64: // für Anzeige der Nachricht relevant sind
65: GetDlgItem(IDC_MSG)->ShowWindow(FALSE);
66: GetDlgItem(IDC_SHWMSG)->ShowWindow(FALSE);
67: GetDlgItem(IDC_DFLTMSG)->ShowWindow(FALSE);
68: GetDlgItem(IDC_CLRMSG)->ShowWindow(FALSE);
69: GetDlgItem(IDC_STATICMSG)->ShowWindow(FALSE);
70: }
71:
72: //////////////////////////////////////
73: // EIGENER CODE, ENDE
74: //////////////////////////////////////
75: }

```

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

Von jetzt an ist der Code nicht mehr ganz so durchsichtig. Die erste Funktion, `GetDlgItem`, erhält als Parameter die ID des zu ändernden Steuerelements. Die Funktion liefert das Objekt für das Steuerelement zurück. Mit dieser Funktion läßt sich das Objekt für beliebige Steuerelemente im Fenster bei laufender Anwendung abrufen. Der nächste Teil jedes Befehls ruft eine Member-Funktion des Steuerelementobjekts auf. Die zweite Funktion ist eine Member-Funktion des von der ersten Funktion zurückgegebenen Objekts. Falls Ihnen dieser Ablauf nicht ganz klar sein sollte, empfiehlt sich ein Studium von Anhang A.

Die zweiten Funktionen in diesen Aufrufen, `EnableWindow` und `ShowWindow`, sehen eher wie Funktionen für Fenster aber nicht für Steuerelemente aus. Natürlich sind sie für Fenster vorgesehen. Zufällig sind es aber auch Elemente der Klasse `CWnd`, die der Vorfahr der Klasse `CDialog` ist, von der Sie Ihre Klasse `Ctag2Dlg` abgeleitet haben. In Windows sind nun mal alle Steuerelemente selbst Fenster, die völlig unabhängig von dem Fenster sind, in dem sie sich befinden. Damit kann man Steuerelemente als Fenster behandeln und Fensterfunktionen auf ihnen aufrufen. In der Tat sind alle Steuerelementklassen von der Klasse `CWnd` abgeleitet, was ihr wahres Gesicht als Fenster offenbart.

Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, können Sie die Kontrollkästchen Nachrichtenaktion aktivieren und Nachrichtenaktion zeigen ausprobieren. Die Funktionsweise sollte Abbildung 2.11 entsprechen.

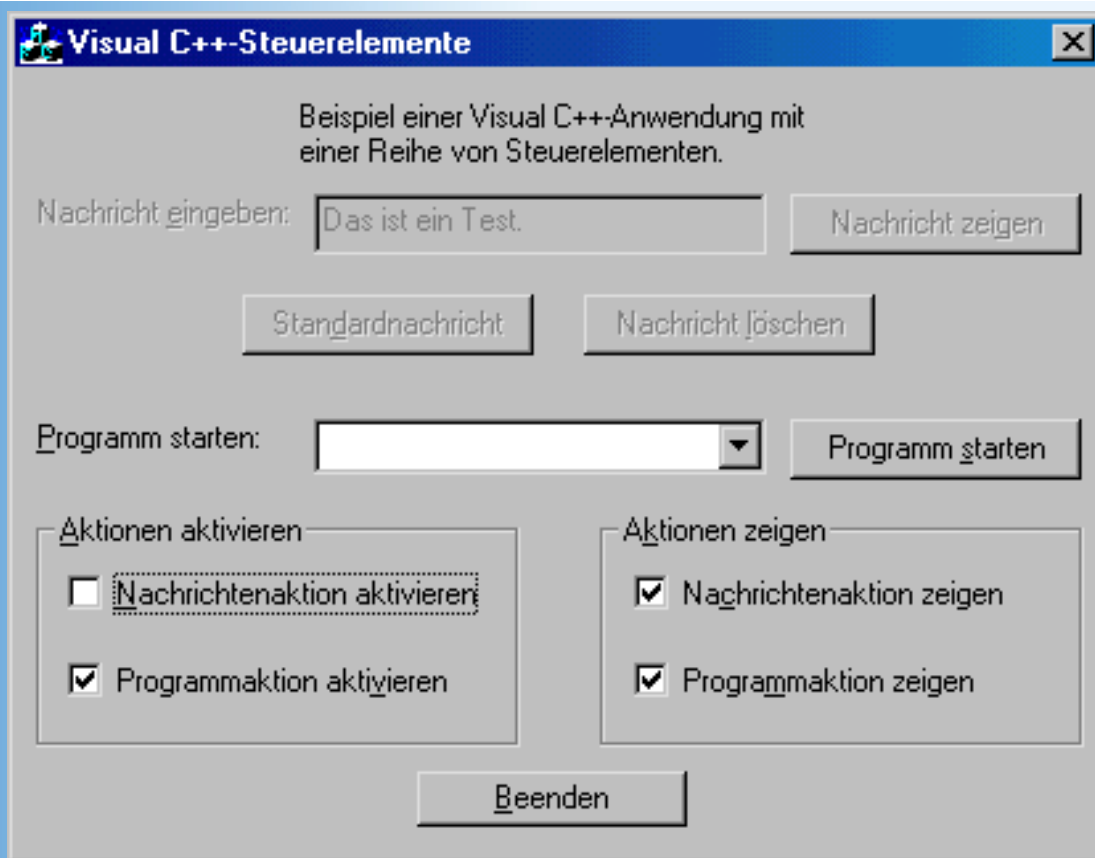


Abbildung 2.11:
Die Steuerelemente für Benutzernachrichten lassen sich nun deaktivieren.

Eine andere Anwendung starten

Als letzte größere Aufgabe müssen wir noch die Funktionalität für die Steuerelemente implementieren, die dem Start eines anderen Programms dienen. Weiter vorn haben Sie die Namen von drei Windows-Anwendungen in das Kombinationsfeld aufgenommen. Wenn Sie die Anwendung starten, erscheinen diese Namen in der Dropdown-Liste. Man kann einen Eintrag auswählen, und im Wertbereich des Kombinationsfelds erscheint der Name der jeweiligen Anwendung. Damit dieser Teil auch funktioniert, müssen Sie lediglich für die Schaltfläche Programm starten Code hinzufügen, um tatsächlich den Wert aus dem Kombinationsfeld zu ermitteln und das passende Programm zu starten. Nachdem Sie das Funktionsgerüst für die Schaltfläche Programm starten mit dem Klassen-Assistenten erstellt haben, fügen Sie den Code aus Listing 2.7 in die Funktion ein.

Listing 2.7: TAG2DLG.CPP - Die Funktion OnRunpgm startet andere Windows-Anwendungen.

```

1: void CTag2Dlg::OnRunpgm()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Nachrichtigung hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Aktuelle Werte vom Bildschirm holen
10: UpdateData(TRUE);
11:

```

```

12: // Lokale Variable zur Aufnahme des Programmnamens deklarieren
13: CString strPgmName;
14:
15: // Programmname in die lokale Variable kopieren
16: strPgmName = m_strProgToRun;
17:
18: // Programmname in Großbuchstaben umwandeln
19: strPgmName.MakeUpper();
20:
21: // Programm Paint gewählt?
22: if (strPgmName == "PAINT")
23: // Ja, Paint starten
24: WinExec("pbrush.exe", SW_SHOW);
25:
26: // Editor (Notepad) gewählt?
27: if (strPgmName == "EDITOR")
28: // Ja, Editor starten
29: WinExec("notepad.exe", SW_SHOW);
30:
31: // Solitär gewählt?
32: if (strPgmName == "SOLITÄR")
33: // Ja, Solitär starten
34: WinExec("sol.exe", SW_SHOW);
35:
36: //////////////////////////////////////
37: // EIGENER CODE, ENDE
38: //////////////////////////////////////
39: }

```

Wie zu erwarten, findet in dieser Funktion zunächst der Aufruf von `UpdateData` statt, um die Variablen mit den Werten der Steuerelemente im Fenster zu aktualisieren. Der nächste Teil erscheint allerdings ein wenig eigentümlich. Es wird eine neue `CString`- Variable deklariert und in diese der Wert des Kombinationsfeldes kopiert. Ist es wirklich notwendig, wenn der Wert bereits in einer `CString`-Variablen steht? Das hängt davon ab, wie sich Ihre Anwendung verhalten soll. Die nächste Codezeile enthält einen Aufruf der `CString`-Funktion `MakeUpper`, die den String in Großbuchstaben konvertiert. Wenn man die `CString`-Variable verwendet, die mit dem Kombinationsfeld verbunden ist, wird beim nächsten Aufruf von `UpdateData` mit dem Argument `FALSE` der Wert im Kombinationsfeld ebenfalls in Großbuchstaben angezeigt. Unter diesen Umständen ist das wahrscheinlich ein ungünstiger Zeitpunkt, und das Verhalten entspricht nicht den Vorstellungen. Deshalb kommt in der Funktion ein zusätzlicher `CString` zum Einsatz.

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



In C und C++ ist der Unterschied zwischen einem einfachen Gleichheitszeichen (=) und einem doppelten Gleichheitszeichen (==) zu beachten. Das einfache Gleichheitszeichen führt eine Zuweisung des Wertes auf der rechten Seite des Gleichheitszeichens an die Variable auf der linken Seite durch. Steht auf der linken Seite des Gleichheitszeichens eine Konstante, wird das Programm nicht kompiliert, und Sie erhalten die Fehlermeldung, daß man keinen Wert der rechten Seite an eine Konstante auf der linken Seite zuweisen kann. Das doppelte Gleichheitszeichen ist für Vergleiche vorgesehen. Achten Sie darauf, das doppelte Gleichheitszeichen zu verwenden, wenn Sie zwei Werte miteinander vergleichen möchten. Wenn Sie in diesem Fall versehentlich das einfache Gleichheitszeichen verwenden, ändern Sie den Wert der Variablen auf der linken Seite. Hier liegt eine der größten Quellen für logische Fehler in C/C++-Programmen.



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

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

Zusammenfassung

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

Fragen und Antworten

Frage:

Nachdem ich die Objekt-IDs der Steuerelemente im Fenster festgelegt habe, weisen drei Steuerelemente dieselbe ID, `IDC_STATIC`, auf. Bei diesen Steuerelementen handelt es sich um den Text im oberen Teil des Fensters und die beiden Gruppentfelder. Die zwei anderen statischen

Textsteuerelemente hatten zuerst die gleiche ID, bis ich sie geändert habe. Wieso haben diese Steuerelemente die gleiche ID, und warum mußte ich die IDs der beiden statischen Textsteuerelemente an dieser Stelle ändern?

Antwort:

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

Die Anwendung verhält sich in unvorhergesehener Weise, wenn Sie eines der statischen Steuerelemente, die die gleiche ID haben, verändern. Als Faustregel gilt, daß man statischen Steuerelementen die gleiche ID zuweisen kann, wenn man die Steuerelemente überhaupt nicht ändert. Macht sich eine Interaktion mit den Steuerelementen erforderlich, müssen Sie jedem Steuerelement eine eindeutige Objekt-ID zuordnen.

Frage:

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

Antwort:

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

Workshop

Kontrollfragen

1. Warum muß man die Tabulator-Reihenfolge der Steuerelemente im Anwendungsfenster festlegen?
2. Wie kann man eine Zugriffstaste in ein statisches Textfeld einbinden, das den Benutzer zum Eingabefeld oder Kombinationsfeld neben dem Textsteuerelement bringt?
3. Warum muß man den statischen Textfeldern vor dem Eingabefeld und den Kombinationsfeldern eindeutige Objekt-IDs zuweisen?
4. Warum muß man die Funktion `UpdateData` aufrufen, bevor man den Wert eines Steuerelements

überprüft?

Übungen

- Fügen Sie für die Schaltfläche Standardnachricht Code hinzu, um den Text im Eingabefeld auf »Nachricht hier eingeben« zurückzusetzen.
- Nehmen Sie Code auf, um die Steuerelemente zur Auswahl und zum Start einer anderen Anwendung zu aktivieren oder zu deaktivieren sowie anzuzeigen oder auszublenden.
- Erweitern Sie den Code in der Funktion `OnRunpgm`, damit der Benutzer den Namen des auszuführenden Programms selbst eingeben kann.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1**Tag 3****Maus und Tastatur**

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

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

In der heutigen Lektion lernen Sie, ...

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

Mausereignisse

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

Wenn Sie den Klassen-Assistenten in einem Ihrer Projekte öffnen, das Dialogfeld in der Liste der Objekt-IDs markieren und dann durch die Liste der verfügbaren Nachrichten blättern, finden Sie eine Reihe von Ereignissen, die sich auf die Maus beziehen. Mit diesen Ereignissen kann man alle Aufgaben erledigen, die in einer Anwendung anfallen. Tabelle 3.1 bringt eine Übersicht dieser Ereignisse.

Tabelle 3.1: Nachrichten für Mausereignisse

Nachricht	Beschreibung
WM_LBUTTONDOWN	Die linke Maustaste wurde gedrückt.
WM_LBUTTONUP	Die linke Maustaste wurde losgelassen.
WM_LBUTTONDBLCLK	Mit der linken Maustaste wurde ein Doppelklick ausgeführt.
WM_RBUTTONDOWN	Die rechte Maustaste wurde gedrückt.
WM_RBUTTONUP	Die rechte Maustaste wurde losgelassen.
WM_RBUTTONDBLCLK	Mit der rechten Maustaste wurde ein Doppelklick ausgeführt.
WM_MOUSEMOVE	Die Maus wird über den Fensterbereich der Anwendung verschoben.
WM_MOUSEWHEEL	Das Mousrad wird bewegt.

Mit der Maus zeichnen

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

1. Legen Sie mit dem MFC-Anwendungs-Assistenten einen neuen Projekt-Arbeitsbereich an, und nennen Sie das Projekt **Maus**.
2. Legen Sie im ersten Schritt des Anwendungs-Assistenten für das Projekt fest, daß Sie eine dialogfeldbasierende Anwendung erstellen.
3. Übernehmen Sie die Standardeinstellungen des Anwendungs-Assistenten. Im zweiten Schritt geben Sie einen passenden Titel für das Dialogfeld wie zum Beispiel **Maus und Tastatur** an.
4. Nachdem das Anwendungsgerüst erstellt ist, entfernen Sie alle Steuerelemente aus dem Dialogfeld. Damit steht die gesamte Fensterfläche des Dialogfelds zum Zeichnen zur Verfügung. Dieser Schritt ist auch erforderlich, damit die Anwendung alle Tastaturereignisse auffangen kann.



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

5. Öffnen Sie den Klassen-Assistenten. Wählen Sie `WM_MOUSEMOVE` aus der Liste der Nachrichten, und klicken Sie auf die Schaltfläche **Funktion hinzufügen**, um eine neue Funktion zu erstellen. Klicken Sie auf die Schaltfläche **OK**, um den vorgeschlagenen Funktionsnamen zu übernehmen.
6. Klicken Sie auf die Schaltfläche **Code bearbeiten**, um in die gerade erstellte Funktion

OnMouseMove den Code aus Listing 3.1 einzugeben.

Listing 3.1: Die Funktion OnMouseMove

```

1: void CMausDlg::OnMouseMove(UINT nFlags, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
   einfügen Äund/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Linke Maustaste gedrückt?
10: if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
11: {
12: // Gerätekontext holen
13: CClientDC dc(this);
14:
15: // Pixel zeichnen
16: dc.SetPixel(point.x, point.y, RGB(0, 0, 0));
17: }
18:
19: //////////////////////////////////////
20: // EIGENER CODE, ENDE
21: //////////////////////////////////////
22:
23: CDialog::OnMouseMove(nFlags, point);
24: }

```

Sehen Sie sich die Funktionsdefinition am Beginn des Listings an. An die Funktion werden zwei Argumente übergeben. Das erste ist eine Gruppe von Flags, über die man ermitteln kann, ob und welche Maustaste niedergedrückt ist. Das geschieht in der ersten Codezeile mit der `if`-Anweisung: `if ((nFlags & MK_LBUTTON) == MK_LBUTTON)`

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

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

Bevor man aber Punkte im Dialogfeld zeichnen kann, muß man den Gerätekontext für das Dialogfeld holen. Dazu wird eine neue Instanz der Klasse `CClientDC` erzeugt. Diese Klasse kapselt den Gerätekontext und die meisten darauf ausführbaren Operationen, einschließlich aller Operationen, die auf den Bildschirm zeichnen. Praktisch ist der Gerätekontext die Leinwand, auf der Sie mit Ihrer Anwendung zeichnen. Keine Leinwand - kein Gemälde. Nachdem das Gerätekontextobjekt erzeugt ist, können Sie dessen Funktion `SetPixel` aufrufen. Diese Funktion färbt Pixel (Bildpunkte) an der Position, die in den beiden ersten Argumenten angegeben ist, mit der im dritten Argument

spezifizierten Farbe. Wenn Sie Ihr Programm kompilieren und ausführen, können Sie sich davon überzeugen, daß man mit der Maus in die Fensterfläche des Dialogfelds zeichnen kann (siehe Abbildung 3.1).



In Windows spezifiziert man Farben in Form eines Zahlenwertes, der eine Kombination aus drei Zahlen darstellt. Diese drei Zahlen bezeichnen die Helligkeit der roten, grünen und blauen Pixel auf dem Computerbildschirm. Die Funktion RGB ist ein Makro, das die drei separaten Werte in eine einzige Zahl umwandelt. Diese Zahl läßt sich an die Funktion SetPixel oder eine andere Funktion, die einen Farbwert erfordert, übergeben. Die Werte für die Farben Rot, Grün und Blau können im Bereich zwischen 0 und 255 (jeweils einschließlich) liegen.

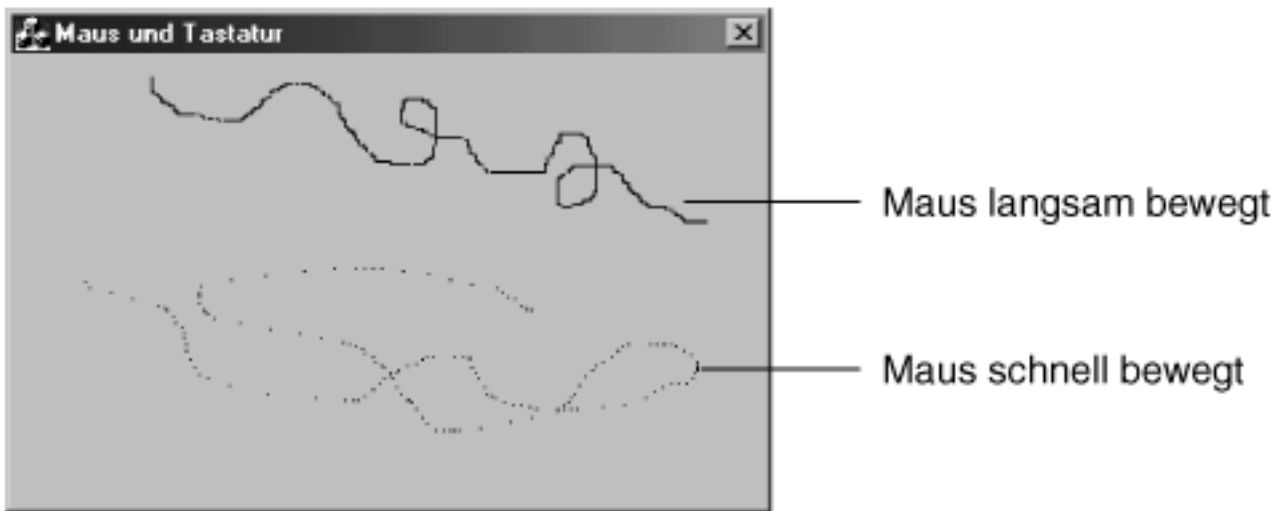


Abbildung 3.1:
Mit der Maus im Fenster zeichnen

Die binären Verknüpfungen AND und OR

Falls Sie ein Einsteiger in C++ sind, sollten Sie sich mit der Arbeitsweise der verschiedenen Typen von AND (UND) und OR (ODER) vertraut machen. Man unterscheidet logische und binäre Verknüpfungen mit AND und OR. Die erste Kategorie verwendet man in logischen oder Bedingungsanweisungen, beispielsweise `if`- oder `while`-Anweisungen, die den Programmfluß steuern. Die binären AND- und OR-Verknüpfungen kombinieren zwei Werte auf Bitebene.

Die AND-Verknüpfung wird durch das kaufmännische Und-Zeichen (&) repräsentiert. Ein einzelnes Und-Zeichen (&) steht für ein binäres AND, ein doppeltes Und-Zeichen (&&) kennzeichnet ein logisches AND. Die Arbeitsweise der logischen AND-Verknüpfung entspricht dem Schlüsselwort AND in Visual Basic oder PowerBuilder. Diese Verknüpfung verwendet man beispielsweise in einer `if`-Anweisung, um auszudrücken »Wenn (IF) diese Bedingung UND (AND) die andere Bedingung erfüllt sind, dann ...«. Beide Bedingungen müssen erfüllt (TRUE) sein, damit die gesamte Anweisung das Ergebnis TRUE liefert. Eine binäres AND setzt man dagegen ein, um Bits zu setzen oder rückzusetzen. Wenn man zwei Werte mit einem binären AND verknüpft, behalten nur diejenigen Bitpositionen eine 1, die in beiden Werten auf 1 gesetzt waren. Alle anderen Bits werden auf 0 gesetzt. Sehen wir uns zur Verdeutlichung zwei 8-Bit-Werte an:

Wert1 01011001

Wert2 00101001

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

Wert1 AND Wert2 00001001

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

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

Wert1 01011001

Wert2 00101001

erhält man das Ergebnis einer binären OR-Verknüpfung zu:

Wert1 OR Wert2 01111001

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

Binäre Attributflags

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

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

Flag1 00001000

Flag2 00100000

OR-Kombination 00101000

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

sich verhalten soll.

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

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

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

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

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

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

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

Das Zeichenprogramm verbessern

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

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

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

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**.
2. Markieren Sie die Dialogfeldklasse - in diesem Fall `CMausDlg`.
3. Klicken Sie mit der rechten Maustaste, und wählen Sie **Member-Variable hinzufügen** aus dem Kontextmenü.

4. Geben Sie im Dialogfeld **Member-Variable hinzufügen** als Variablentyp `int` und als Variablenname `m_iPrevY` ein, und legen Sie den Zugriff mit *Privat* fest, wie es Abbildung 3.2 zeigt.

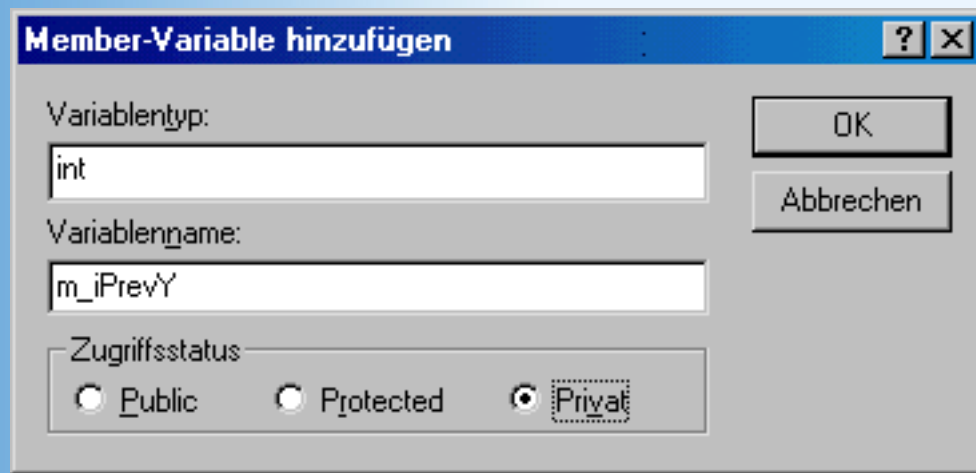


Abbildung 3.2:
Das Dialogfeld Member-Variable hinzufügen

5. Klicken Sie auf **OK**, um die Variable hinzuzufügen.

6. Wiederholen Sie die Schritte 3 bis 5, wobei Sie als Variablenname `m_iPrevX` für die zweite Variable eingeben.

Nachdem Sie die Variablen zum Speichern der vorherigen Mausposition hinzugefügt haben, können Sie die notwendigen Änderungen an der Funktion `OnMouseMove` gemäß Listing 3.2 vornehmen.

Listing 3.2: Die überarbeitete Funktion `OnMouseMove`

```

1: void CMAusDlg::OnMouseMove(UINT nFlags, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
   einfügen Äund/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Linke Maustaste gedrückt?
10: if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
11: {
12: // Gerätekontext holen
13: CClientDC dc(this);
14:
15: // Linie vom letzten zum aktuellen Punkt zeichnen
16: dc.MoveTo(m_iPrevX, m_iPrevY);
17: dc.LineTo(point.x, point.y);
18:
19: // Aktuellen Punkt als letzten Punkt speichern
20: m_iPrevX = point.x;
21: m_iPrevY = point.y;
22: }
23:

```

```

24: ///////////////////////////////////
25: // EIGENER CODE, ENDE
26: ///////////////////////////////////
27:
28: CDialog::OnMouseMove(nFlags, point);
29: }

```

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

```

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

```

Es ist also zunächst eine Bewegung zur ersten Position auszuführen (**MoveTo**) und dann eine Linie zum zweiten Punkt zu ziehen (**LineTo**). Der erste Schritt ist deshalb wichtig, da man sonst Windows nicht mitteilen könnte, wo der Startpunkt liegt. Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, läßt es sich etwas besser zeichnen. Allerdings tritt nun ein eigentümliches Verhalten zutage. Sobald Sie die linke Maustaste drücken, um etwas mehr zu zeichnen, zieht die Anwendung zu diesem Punkt eine Linie vom Endpunkt der letzten gezeichneten Linie, wie es Abbildung 3.3 verdeutlicht.

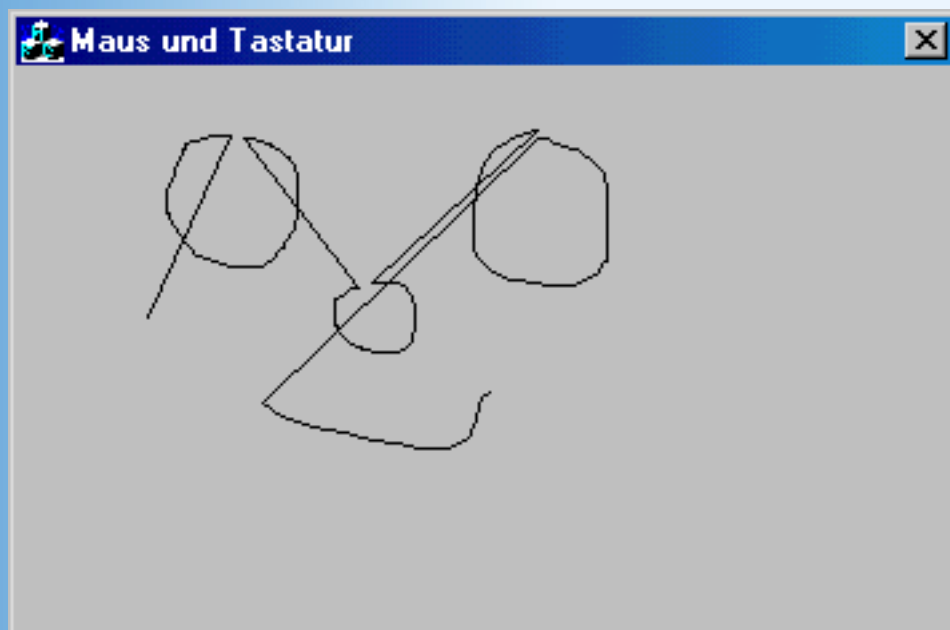


Abbildung 3.3:
Das Zeichenprogramm mit einem eigentümlichen Verhalten

Letzte Anpassungen

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

1. Nehmen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht **WM_LBUTTONDOWN** in das Dialogfeldobjekt auf.
2. Fügen Sie in die eben erstellte Funktion **OnLButtonDown** den Code von Listing 3.3 ein.

Listing 3.3: Die Funktion OnLButtonDown

```

1: void CMausDlg::OnLButtonDown(UINT nFlags, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
   einfügen und/oder Standard aufrufen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Aktuellen Punkt als Anfangspunkt setzen
10: m_iPrevX = point.x;
11: m_iPrevY = point.y;
12:
13: ///////////////////////////////////
14: // EIGENER CODE, ENDE
15: ///////////////////////////////////
16:
17: CDialog::OnLButtonDown(nFlags, point);
18: }

```

Wenn Sie die Anwendung kompilieren und ausführen, sollten Sie schon fast so zeichnen können, wie man es von einem Zeichenprogramm erwartet (siehe Abbildung 3.4).

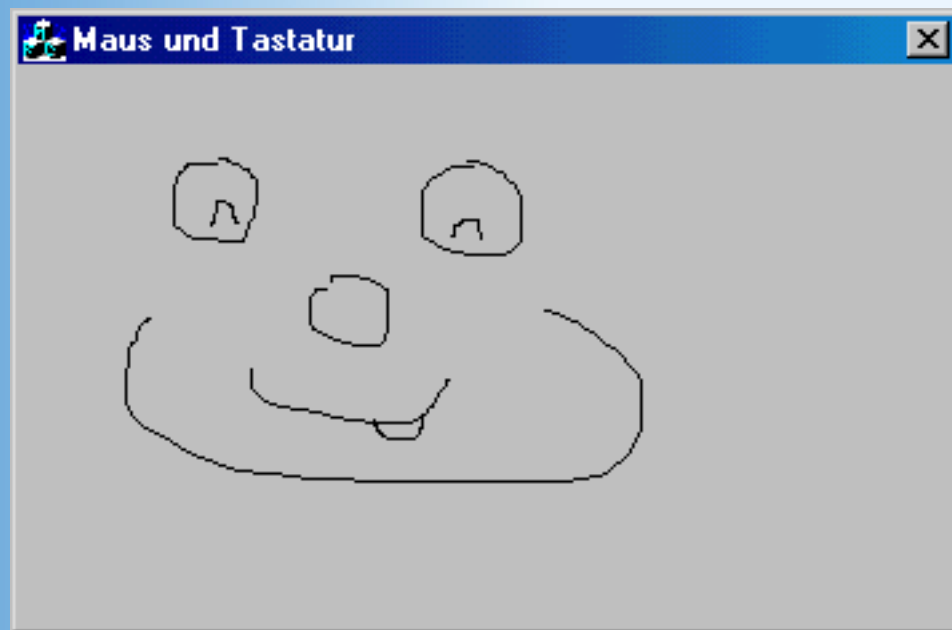


Abbildung 3.4:
Das fertiggestellte Zeichenprogramm

Tastaturereignisse auffangen

Tastaturereignisse lassen sich fast genauso lesen wie Mausereignisse. Analog zur Maus werden Nachrichten verschickt, wenn man eine Taste drückt und wenn man sie wieder losläßt. Diese Ereignisse sind in Tabelle 3.2 aufgeführt.

Tabelle 3.2: Nachrichten für Tastaturereignisse

Nachricht	Beschreibung
WM_KEYDOWN	Eine Taste wurde gedrückt.
WM_KEYUP	Eine Taste wurde losgelassen.

Offensichtlich gibt es für die Tastatur weniger Nachrichten als für die Maus, und man kann mit der Tastatur aus programmtechnischer Sicht dementsprechend weniger anfangen. Die Nachrichten sind im Dialogfeldobjekt verfügbar und werden nur ausgelöst, wenn keine aktivierten Steuerelemente im Fenster vorhanden sind. Alle aktivierten Steuerelemente im Fenster haben den Eingabefokus, so daß alle Tastaturereignisse zu ihnen gelangen. Aus diesem Grund sollten Sie für die Zeichenanwendung alle Steuerelemente aus dem Hauptdialogfeld entfernen.

Den Mauszeiger zum Zeichnen ändern

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

1. Nehmen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht WM_KEYDOWN in das Dialogfeldobjekt auf.
2. In die eben erstellte Funktion OnKeyDown fügen Sie den Code gemäß Listing 3.4 ein.

Listing 3.4: Die Funktion OnKeyDown

```

1: void CMAusDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
   einfügen Âund/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: char lsChar; // Zeichen der gedrückten Taste
10: HCURSOR lhCursor; // Handle zum anzuzeigenden Cursor
11:
12: // Code der gedrückten Taste in Zeichen umwandeln
13: lsChar = char(nChar);
14:
15: // Ist Zeichen ein "A"?
16: if (lsChar == 'A')
17: {
18: // Pfeilcursor laden
19: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);

```

```

20: // Bildschirmcursor setzen
21: SetCursor(lhCursor);
22: }
23:
24: // Ist Zeichen ein "B"?
25: if (lsChar == 'B')
26: {
27: // Balkencursor laden
28: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
29: // Bildschirmcursor setzen
30: SetCursor(lhCursor);
31: }
32:
33: // Ist Zeichen ein "C"?
34: if (lsChar == 'C')
35: {
36: // Sanduhrzeiger laden
37: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
38: // Bildschirmcursor setzen
39: SetCursor(lhCursor);
40: }
41:
42: // Ist Zeichen ein "X"?
43: if (lsChar == 'X')
44: {
45: // Pfeilcursor laden
46: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
47: // Bildschirmcursor setzen
48: SetCursor(lhCursor);
49: // Anwendung beenden
50: OnOK();
51: }
52:
53: //////////////////////////////////////
54: // EIGENER CODE, ENDE
55: //////////////////////////////////////
56:
57: CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
58: }

```

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

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

Das zweite Argument an die Funktion `OnKeyDown` liefert die Anzahl, wie oft die Taste gedrückt wurde. Normalerweise ist dieser Wert gleich 1, wenn man die Taste drückt und wieder losläßt. Hält man die Taste dagegen nieder, tritt ein Wiederholungszähler für diese Taste in Kraft. Der

entsprechende Wert sagt also aus, wie oft Windows in diesem Fall ein wiederholtes Drücken der Taste annimmt.

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

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



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

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

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

Nachdem die Funktion den Cursor in den Speicher geladen hat, wird der Handle auf diesen Cursor an die Funktion **SetCursor** übergeben. Damit nimmt der Mauszeiger die Form des Cursors an, auf den der Handle zeigt. Wenn Sie die Anwendung kompilieren und ausführen, können Sie eine der festgelegten Tasten drücken, um die Form des Cursors zu ändern, wie es Abbildung 3.5 zeigt. Sobald Sie aber die Maus verschieben, um etwas zu zeichnen, nimmt der Cursor wieder die Form des Standardpfeils an. Der folgende Abschnitt beschreibt, wie man die Änderung dauerhaft machen kann.

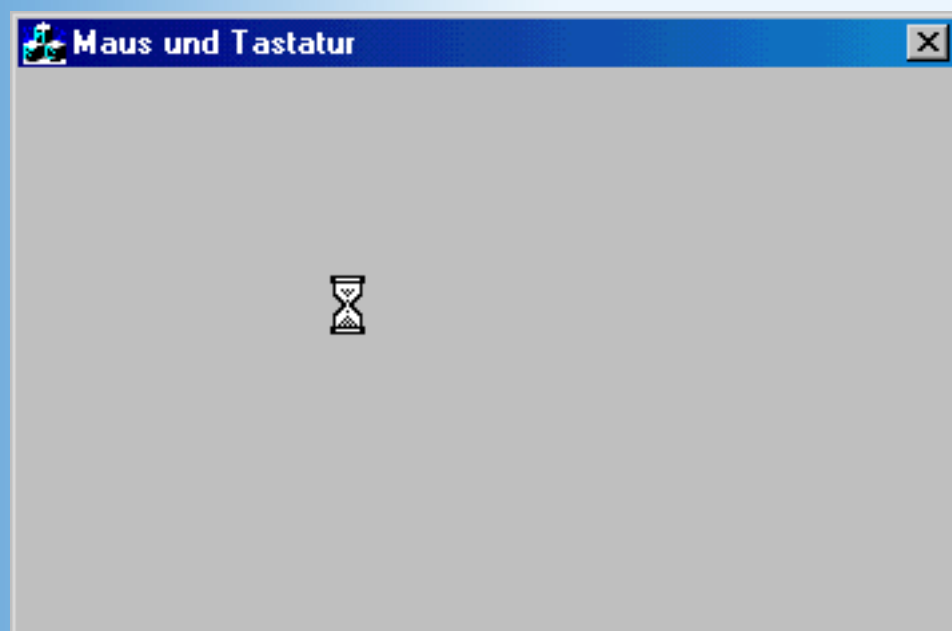


Abbildung 3.5:

Die Form des Cursors läßt sich mit festgelegten Tasten ändern.

Änderung beibehalten

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

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

1. Nehmen Sie in die Klasse CMAusDlg eine neue Variable auf, wie Sie es für die Variablen der vorherigen Position ausgeführt haben. Dieses Mal deklarieren Sie den Typ als **BOOL** und nennen die Variable `m_bCursor`, wie es Abbildung 3.6 zeigt.

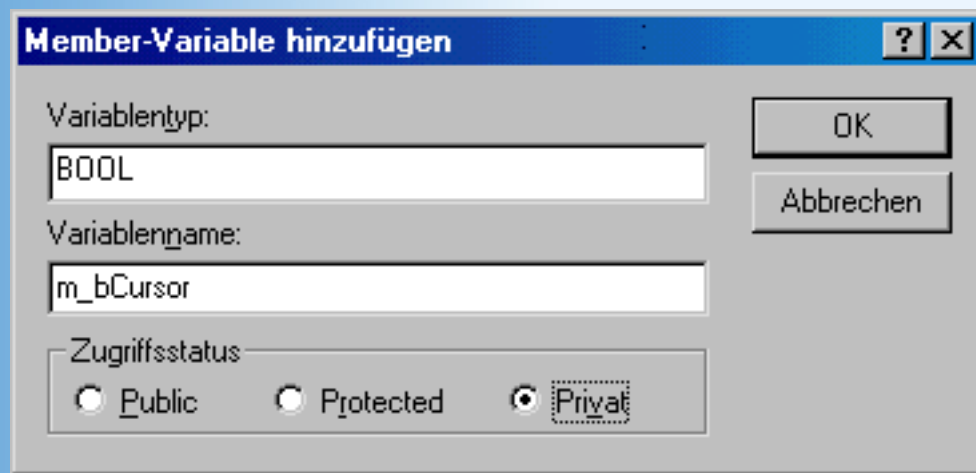


Abbildung 3.6:
Eine Member-Variable definieren

2. Initialisieren Sie die Variable `m_bCursor` in der Funktion `OnInitDialog` mit dem Code in Listing 3.5.

Listing 3.5: Die Funktion `OnInitDialog`

```
1: BOOL CMAusDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:     .
6:     .
7:     .
8:     // Symbol für dieses Dialogfeld festlegen. Wird automatisch
    erledigt
9:     // wenn das Hauptfenster der Anwendung kein Dialogfeld ist
10:    SetIcon(m_hIcon, TRUE); // Großes Symbol verwenden
11:    SetIcon(m_hIcon, FALSE); // Kleines Symbol verwenden
12:
13:    // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
```

```

14:
15: //////////////////////////////////////////////////
16: // EIGENER CODE, ANFANG
17: //////////////////////////////////////////////////
18:
19: // Cursor als Pfeil initialisieren
20: m_bCursor = FALSE;
21:
22: //////////////////////////////////////////////////
23: // EIGENER CODE, ENDE
24: //////////////////////////////////////////////////
25:
26: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
    soll den Fokus erhalten
27: }

```

3. Ändern Sie die Funktion `OnKeyDown` gemäß Listing 3.6, um das Flag `m_bCursor` auf `TRUE` zu setzen, wenn Sie den Cursor wechseln.

Listing 3.6: Die Funktion `OnKeyDown`

```

1: void CMausDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
    einfügen und/oder Standard aufrufen
4:
5: //////////////////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////////////////
8:
9: char lsChar; // Zeichen der gedrückten Taste
10: HCURSOR lhCursor; // Handle auf anzuzeigenden Cursor
11:
12: // Code der gedrückten Taste in Zeichen umwandeln
13: lsChar = char(nChar);
14:
15: // Ist Zeichen ein "A"?
16: if (lsChar == 'A')
17: // Pfeilcursor laden
18: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
19:
20: // Ist Zeichen ein "B"?
21: if (lsChar == 'B')
22: // Balkencursor laden
23: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
24:
25: // Ist Zeichen ein "C"?
26: if (lsChar == 'C')
27: // Sanduhrzeiger laden
28: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);

```



```

29:
30: // Ist Zeichen ein "X"?
31: if (lsChar == 'X')
32: {
33: // Pfeilcursor laden
34: lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
35: // Cursorflag setzen
36: m_bCursor = TRUE;
37: // Bildschirmcursor setzen
38: SetCursor(lhCursor);
39: // Anwendung beenden
40: OnOK();
41: }
42: else
43: {
44: // Cursorflag setzen
45: m_bCursor = TRUE;
46: // Bildschirmcursor setzen
47: SetCursor(lhCursor);
48: }
49:
50: //////////////////////////////////////
51: // EIGENER CODE, ENDE
52: //////////////////////////////////////
53:
54: CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
55: }

```

4. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht WM_SETCURSOR in das Dialogfeldobjekt ein.

5. Nehmen Sie den Code aus Listing 3.7 in die eben erstellte Funktion OnSetCursor auf.

Listing 3.7: Die Funktion OnSetCursor

```

1: BOOL CMAusDlg::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT
message)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
einfügen Âund/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Wenn Cursor gesetzt, TRUE zurückgeben
10: if (m_bCursor)
11: return TRUE;
12: else
13:

```

```

14: //////////////////////////////////////
15: // EIGENER CODE, ENDE
16: //////////////////////////////////////
17:
18: return CDialog::OnSetCursor(pWnd, nHitTest, message);
19: }

```

Die Funktion `OnSetCursor` muß immer `TRUE` zurückgeben oder ansonsten die Funktion der Basisklasse aufrufen. Die Funktion der Basisklasse setzt den Cursor zurück und muß aufgerufen werden, wenn die Anwendung das erste Mal startet. Aus diesem Grund müssen Sie die Variable mit `FALSE` initialisieren, damit die Standardverarbeitung von `OnSetCursor` abläuft, bis der Benutzer eine Taste drückt, um den Cursor zu wechseln. Nachdem der Benutzer den Cursor geändert hat, übergehen Sie die Standardverarbeitung und liefern statt dessen den Wert `TRUE` zurück. Damit kann der Benutzer mit dem Cursor, den er ausgewählt hat, zeichnen. Dazu gehört auch der Sanduhrzeiger, wie es Abbildung 3.7 zeigt.

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

Haben Sie in einer einzigen Funktion alle Aufgaben zusammengefaßt, für die Sie den Sanduhrzeiger anzeigen müssen, und der Sanduhrzeiger muß nach Abschluß dieser Funktion nicht mehr angezeigt werden, gibt es eine einfachere Methode, um den Sanduhrzeiger auf den Bildschirm zu bringen. Deklarieren Sie eine Variable der Klasse `CWaitCursor` am Beginn der Funktion. Daraufhin zeigt Windows automatisch den Sanduhrzeiger an. Sobald das Programm die Funktion verläßt, wird der Cursor wieder in den vorherigen Cursor zurückverwandelt.

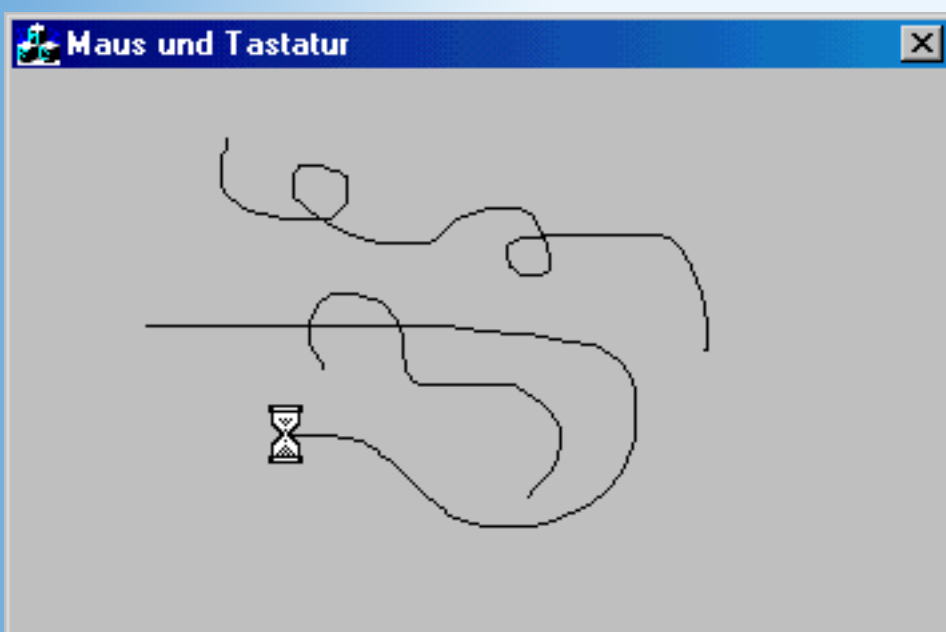


Abbildung 3.7:
Mit dem Sanduhrzeiger zeichnen

Zusammenfassung

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

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

Aufbauend auf diesen Kenntnissen erfahren Sie in der nächsten Lektion, wie man den Windows-Timer einsetzt, um Ereignisse in regelmäßigen Abständen auszulösen. Tag 5 zeigt dann, wie sich mit zusätzlichen Dialogfeldern Rückmeldungen vom Benutzer einholen lassen, so daß man auf der Basis dieser Eingaben das Verhalten der Anwendung steuern kann. Schließlich geht die sechste Lektion darauf ein, wie man Menüs für eine Anwendung erstellt.

Fragen und Antworten

Frage:

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

Antwort:

Wenn man mit einem der Standardbefehle für Gerätekontexte auf den Bildschirm zeichnet, kommt ein sogenannter Stift zum Einsatz, genau wie wenn man mit einem Stift auf Papier zeichnet. Um dickere Linien oder andere Farben darzustellen, muß man einen neuen Stift auswählen. Das läßt sich realisieren, indem man den Code in der Funktion OnMouseMove an der Stelle anpaßt, wo der Gerätekontext geholt wird. Der folgende Code bewirkt, daß die Linie mit einem großen roten Stift gezeichnet wird:

```
// Gerätekontext holen
CClientDC dc(this);

// Einen neuen Stift erzeugen
CPen lpen(PS_SOLID, 16, RGB(255, 0, 0));

// Den neuen Stift verwenden
dc.SelectObject(&lpen);

// Eine Linie vom letzten zum aktuellen Punkt zeichnen
dc.MoveTo(m_iPrevX, m_iPrevY);
dc.LineTo(point.x, point.y);
```

Frage:

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

Antwort:

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

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

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

Workshop

Kontrollfragen

1. Für welche Mausnachrichten kann man Funktionen hinzufügen?
2. Wie kann man ermitteln, ob bei der Nachricht `WM_MOUSEMOVE` die linke Maustaste gedrückt ist?
3. Wie kann man verhindern, daß sich der Mauszeiger in den Standardzeiger zurückverwandelt, nachdem man einen anderen Zeiger festgelegt hat?

Übungen

1. Modifizieren Sie das Zeichenprogramm, so daß beim Drücken der linken Maustaste in Rot und bei der rechten Maustaste in Blau gezeichnet wird.
2. Erweitern Sie die Funktion `OnKeyDown`, um einige der folgenden Standardcursor aufzunehmen:
 - IDC_CROSS
 - IDC_UPARROW
 - IDC_SIZEALL
 - IDC_SIZENWSE
 - IDC_SIZENESW
 - IDC_SIZEWE
 - IDC_SIZENS
 - IDC_NO
 - IDC_APPSTARTING
 - IDC_HELP

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1

Tag 4

Timer

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

Heute lernen Sie, wie ...

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

Funktionsweise von Windows-Timern

Windows-Timer stellen einen Mechanismus bereit, über den man einen oder mehrere Timer mit einer bestimmten Anzahl von Millisekunden auslösen kann. Wenn man einen Timer für ein Intervall von 1000 Millisekunden einrichtet, wird er jede Sekunde ausgelöst. Beim Auslösen eines Timers schickt Windows eine `WM_TIMER`-Nachricht an Ihre Anwendung. Mit dem Klassen-Assistenten können Sie eine Funktion in Ihre Anwendung einbauen, um diese Timer-Nachricht zu behandeln.

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

Beim Starten oder Anhalten eines Timers geben Sie einen ganzzahligen Wert als Timer-ID an. Die Anwendung kann anhand der Timer-ID ermitteln, wann ein Timer ausgelöst hat, sowie Timer starten

und stoppen. Eine bessere Vorstellung von dieser Arbeitsweise bekommen Sie, wenn Sie sich mit der heute zu erstellenden Anwendung beschäftigen.

Die Anwendung mit einer Uhr ausstatten

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

Das Projekt und die Anwendung erstellen

Die heutige Beispielanwendung erstellen Sie in drei Phasen. Zuerst nehmen Sie die Steuerelemente auf, die für die gesamte Anwendung erforderlich sind. In der zweiten Phase fügen Sie den ersten der beiden Timer hinzu. Dieser Timer steuert die Uhr im Dialogfeld der Anwendung. In der dritten Phase kommt der zweite Timer hinzu, den der Benutzer entsprechend seinen Wünschen einstellen, starten und stoppen kann.

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

1. Arbeiten Sie mit den gleichen Einstellungen des Anwendungs-Assistenten wie in den vergangenen drei Tagen, und erstellen Sie ein neues Projekt namens **Timer**. Legen Sie den Titel der Anwendung mit **Timer** fest.
2. Gestalten Sie das Dialogfeld entsprechend Abbildung 4.1 mit den Eigenschaften der Steuerelemente, die in Tabelle 4.1 aufgeführt sind. Denken Sie daran, daß Sie beim Plazieren eines Steuerelements im Fenster mit der rechten Maus klicken können, um das Dialogfeld für die Eigenschaften des Steuerelements über das Kontextmenü zu öffnen.

Tabelle 4.1: Einstellungen der Steuerelementeigenschaften

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Timer-&Intervall:
Eingabefeld	ID	IDC_INTERVAL
Schaltfläche	ID	IDC_STARTTIME
	Titel	Timer &starten
Schaltfläche	ID	IDC_STOPTIMER
	Titel	Timer &anhalten
	Deaktiviert	eingeschaltet
Text	ID	IDC_STATIC
	Titel	Zeit:

Text	ID	IDC_STATICTIME
	Titel	Aktuelle Zeit
Text	ID	IDC_STATIC
	Titel	Zähler:
Text	ID	IDC_STATICCOUNT
	Titel	0
Schaltfläche	ID	IDC_EXIT
	Titel	&Beenden

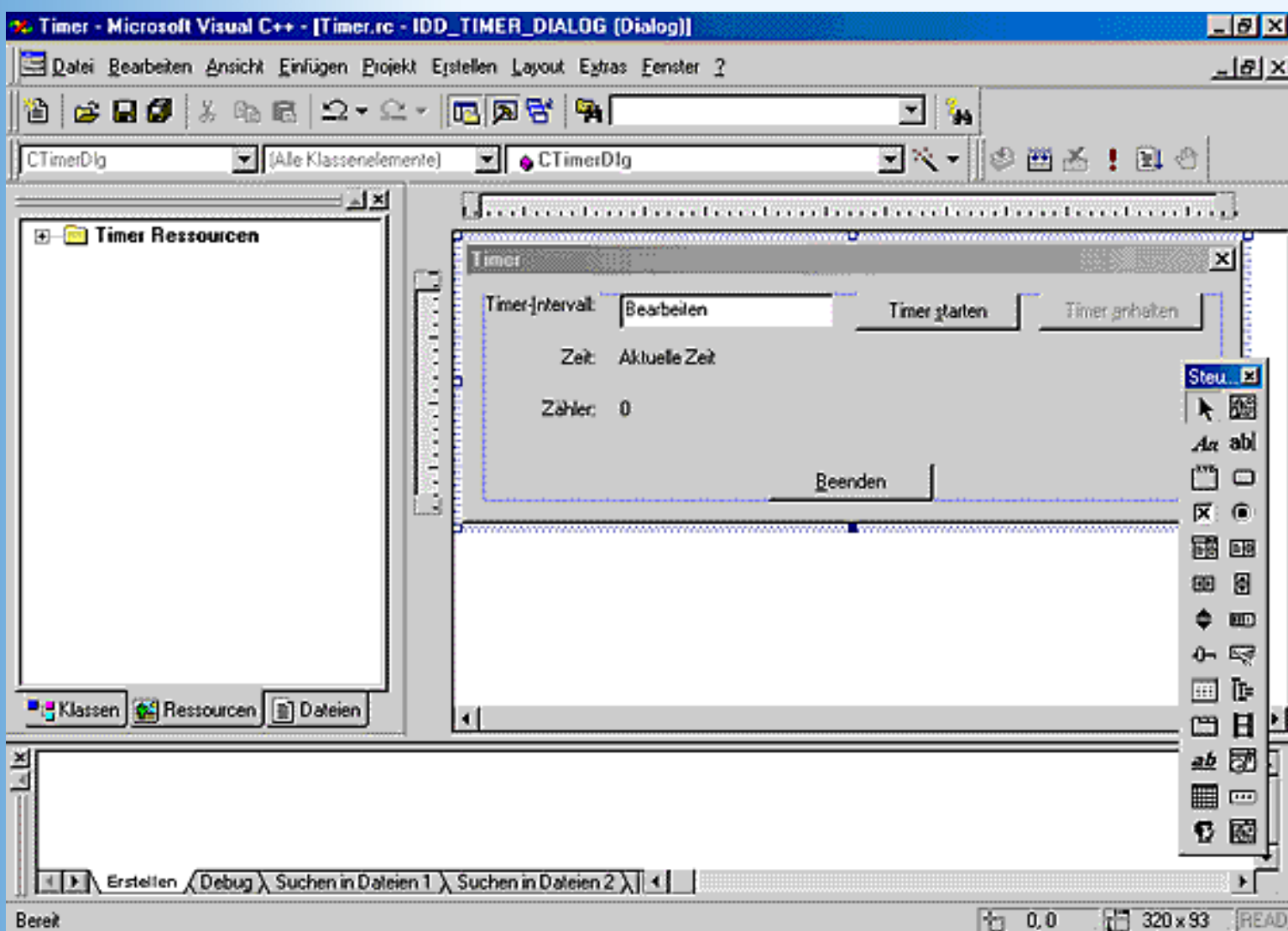


Abbildung 4.1:
Layout des Dialogfelds für die Timer-Anwendung

3. Legen Sie die Tabulator-Reihenfolge fest, wie Sie es am Tag 2 gelernt haben.

4. Fügen Sie für die Schaltfläche **Beenden** den Code zum Schließen der Anwendung wie am Tag 2 hinzu.

Die Timer-IDs hinzufügen

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

1. Klicken Sie im Arbeitsbereich auf der Registerkarte **Ressourcen** mit der rechten Maustaste über dem Ordner **Timer Ressourcen** am Beginn des Ressourcenbaumes. Wählen Sie **Ressourcensymbole** aus dem Kontextmenü, wie es Abbildung 4.2 zeigt.

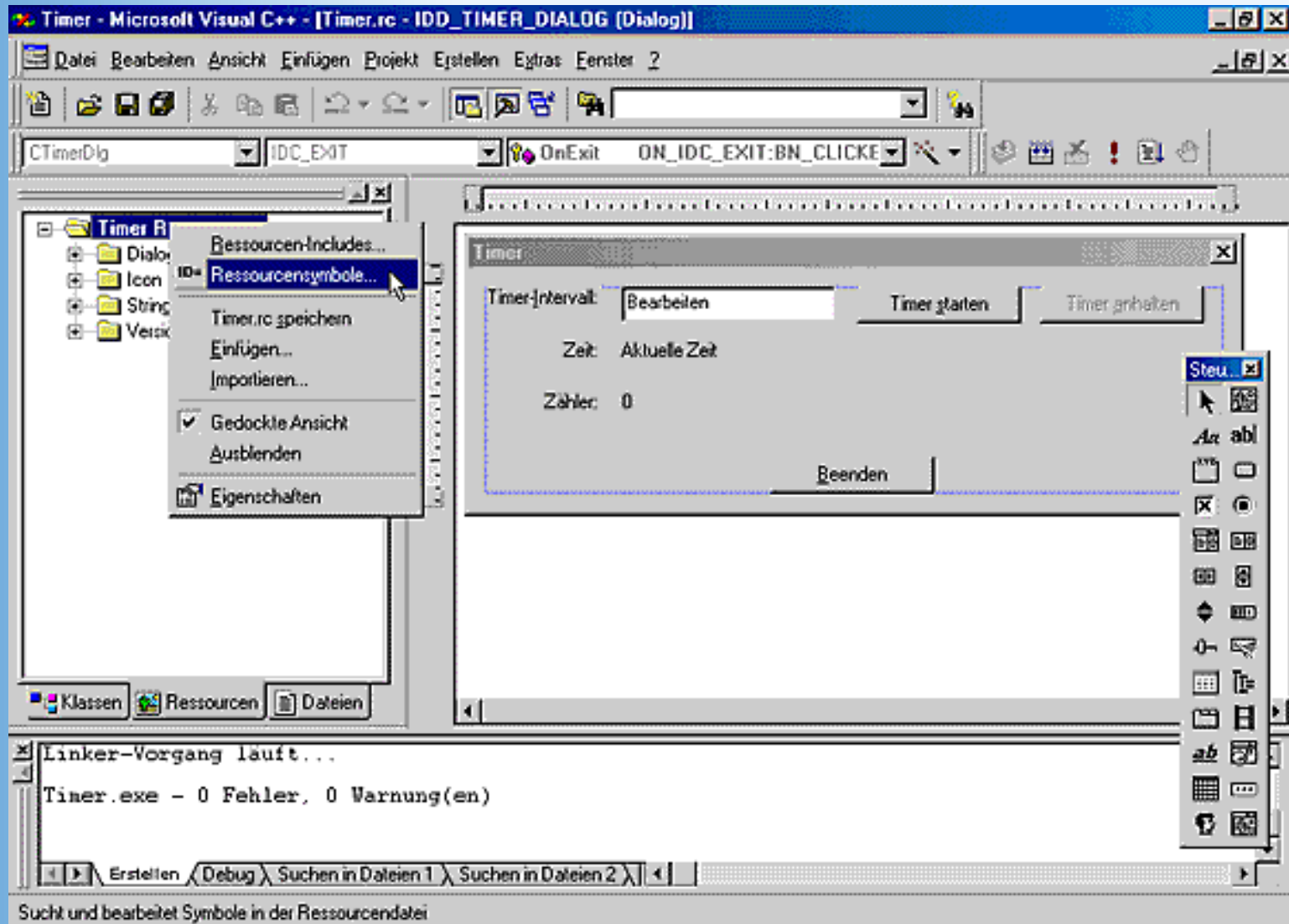


Abbildung 4.2:
Das Kontextmenü Ressourcen

2. Im Dialogfeld **Ressourcensymbole** klicken Sie auf die Schaltfläche **Neu**.
3. Im Dialogfeld **Neues Symbol** geben Sie **ID_CLOCK_TIMER** als Symbolname und **1** als Wert ein, wie es Abbildung 4.3 zeigt.

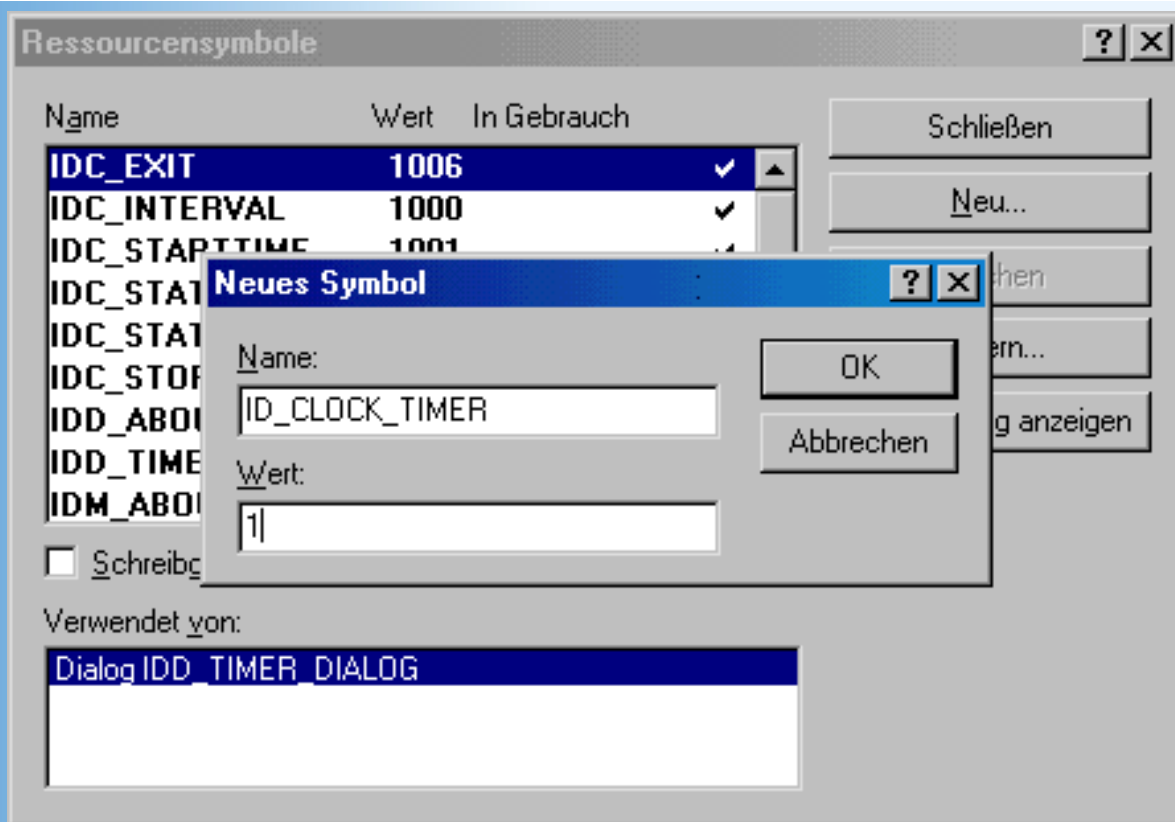


Abbildung 4.3:
Ein neues Ressourcensymbol hinzufügen

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

Den Uhren-Timer starten

Um den Timer für die Uhr zu starten, bearbeiten Sie die Funktion `OnInitDialog`, wie Sie es in den vergangenen beiden Tagen getan haben. Nehmen Sie den neuen Code aus Listing 4.1 in die Funktion auf.

Listing 4.1: Die Funktion `OnInitDialog`

```

1: BOOL CTimerDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:     .
5:     .
6:     .
7:     // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
8:
9:     ///////////////////////////////////
10:    // EIGENER CODE, ANFANG
11:    ///////////////////////////////////
12:
13:    // Timer für Uhr starten
14:    SetTimer(ID_CLOCK_TIMER, 1000, NULL);

```

```

15:
16: //////////////////////////////////////
17: // EIGENER CODE, ENDE
18: //////////////////////////////////////
19:
20: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
    soll den Fokus erhalten
21: }

```

Der Code in diesem Listing startet den Uhren-Timer mit der Funktion `SetTimer`. Das erste an die Funktion `SetTimer` übergebene Argument ist die ID für den Uhren-Timer. Das zweite Argument gibt an, wie oft Sie das Ereignis auslösen möchten. Im vorliegenden Fall wird das Timer-Ereignis alle 1000 Millisekunden, d.h. jede Sekunde, ausgelöst. Im dritten Argument wird die Adresse einer optionalen Callback-Funktion übergeben, die Sie festlegen können, um das Ereignis `WM_TIMER` zu umgehen. Wenn Sie für dieses Argument den Wert `NULL` übergeben, stellt Windows das Ereignis `WM_TIMER` in die Nachrichtenwarteschlange der Anwendung.



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

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

Das Timer-Ereignis der Uhr behandeln

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

1. Nehmen Sie mit Hilfe des Klassen-Assistenten eine Variable für das Steuerelement `IDC_STATICTIME` vom Typ `CString` mit dem Namen `m_sTime` auf.
2. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `WM_TIMER` für das Objekt `CTimerDlg` hinzu.
3. Übernehmen Sie den Code aus Listing 4.2 in die Funktion `OnTimer`.

Listing 4.2: Die Funktion `OnTimer`

```

1: void CTimerDlg::OnTimer(UINT nIDEvent)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
    einfügen und/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG

```

```

7: ///////////////////////////////////
8:
9: // Aktuelle Zeit holen
10: CTime curTime = CTime::GetCurrentTime();
11:
12: // Aktuelle Zeit anzeigen
13: m_sTime.Format("%d:%d:%d", curTime.GetHour(),
14: curTime.GetMinute(),
15: curTime.GetSecond());
16:
17: // Dialogfeld aktualisieren
18: UpdateData(FALSE);
19:
20: ///////////////////////////////////
21: // EIGENER CODE, ENDE
22: ///////////////////////////////////
23:
24: CDialog::OnTimer(nIDEvent);
25: }

```

Der Code in diesem Listing deklariert eine Instanz der Klasse `CTime` und initialisiert sie mit der aktuellen Systemzeit. Als nächstes setzt die Funktion den String `m_sTime` auf die aktuelle Uhrzeit und verwendet dabei die Methode `Format`, um die Zeit in der üblichen Form mit Stunden, Minuten und Sekunden (HH:MM:SS) zu formatieren. Schließlich wird das Dialogfeld mit der aktuellen Uhrzeit aktualisiert. Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollte eine laufende Uhr in der Mitte des Dialogfelds zu sehen sein, wie es Abbildung 4.4 zeigt.

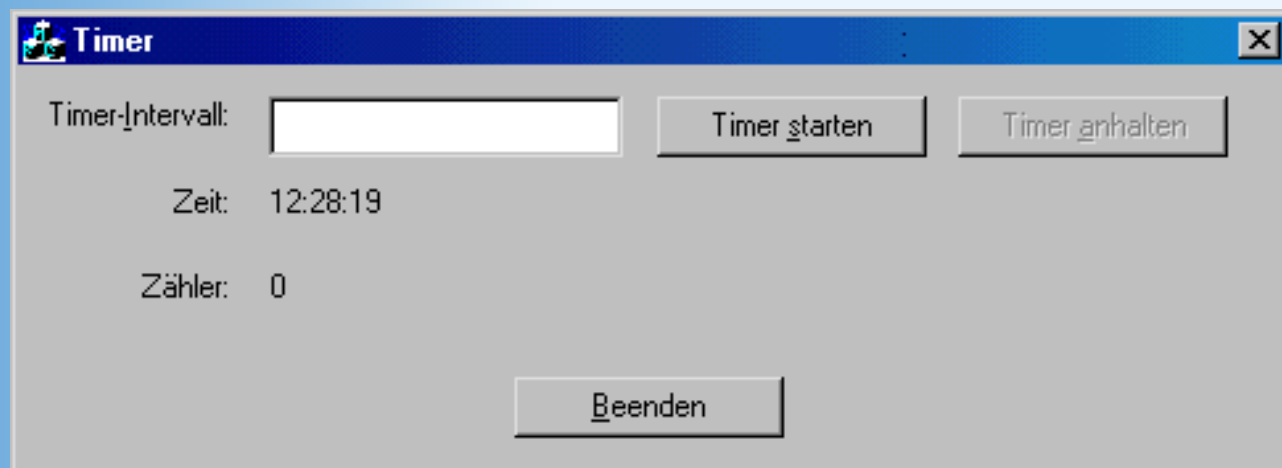


Abbildung 4.4:
Im Dialogfeld der Anwendung ist eine laufende Uhr zu sehen.

Einen zweiten Timer in die Anwendung aufnehmen

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

Die Variablen der Anwendung hinzufügen

Bevor Sie den zweiten Timer in die Anwendung einbauen, sind den Steuerelementen ein paar Variablen zuzuordnen. Für den Uhren-Timer war nur eine einzelne Variable erforderlich, um die Zeitanzeige zu aktualisieren. Jetzt brauchen wir zusätzliche Variablen für die anderen Steuerelemente. In Tabelle 4.2 sind diese Variablen zusammengefaßt.

Tabelle 4.2: Variablen für die Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_STATICCOUNT	m_sCount	Wert	CString
IDC_INTERVAL	m_iInterval	Wert	int
IDC_STARTTIME	m_cStartTime	Control	CButton
IDC_STOPTIMER	m_cStopTime	Control	CButton

Nachdem Sie alle Variablen mit dem Klassen-Assistenten hinzugefügt haben, führen Sie die folgenden Schritte aus:

1. Markieren Sie im Klassen-Assistenten die Variable `m_iInterval`. Tragen Sie in die Eingabefelder unter der Variablenliste als kleinsten Wert 1 und als größten Wert 100000 ein, wie es Abbildung 4.5 zeigt.

MFC-Klassen-Assistent

Nachrichtenzuordnungstabellen | **Member-Variablen** | Automatisierung | ActiveX-Ereignisse | Klasse

Projekt: Timer Klassenname: CTimerDlg

F:\...\Tag04\Timer\TimerDlg.h, F:\...\Timer\TimerDlg.cpp

Steuerelement-IDs:	Typ	Element
IDC_EXIT		
IDC_INTERVAL	int	m_iInterval
IDC_STARTTIME	CButton	m_cStartTime
IDC_STATICCOUNT	CString	m_sCount
IDC_STATICTIME	CString	m_sTime
IDC_STOPTIMER	CButton	m_cStopTime

Beschreibung: int mit Prüfung

Kleinsten Wert: 1

Größter Wert: 100000

Buttons: Klasse hinzufügen..., Variable hinzufügen..., Variable löschen, Spalten aktualisieren, Alles verbinden, OK, Abbrechen

Abbildung 4.5:**Den zu prüfenden Bereich für eine Variable festlegen**

2. Über die Registerkarte **Klassen** des Arbeitsbereichs nehmen Sie eine Member- Variable in die Klasse **CTimerDlg** auf, wie Sie es gestern gelernt haben. Legen Sie den Variablentyp mit **int**, den Namen als **m_iCount** und den Zugriff als **Privat** fest.

3. Mit dem Klassen-Assistenten fügen Sie eine Funktion für die Nachricht **EN_CHANGE** für die Objekt-ID **IDC_INTERVAL** (das Eingabefeld) hinzu. In diese Funktion übernehmen Sie den Code aus Listing 4.3.

Listing 4.3: Die Funktion OnChangeInterval

```

1: void CTimerDlg::OnChangeInterval()
2: {
3: // TODO: Wenn dies ein RICHEDIT-Steuererelement ist, sendet das
   Steuererelement diese
4: // Benachrichtigung nicht, bevor Sie nicht die Funktion
   CDialog::OnInitDialog()
5: // überschreiben und CRichEditCtrl().SetEventMask() aufrufen,
   wobei
6: // eine ODER-Operation mit dem Attribut ENM_CHANGE und der Maske
   erfolgt.
7:
8: // TODO: Fügen Sie hier Ihren Code für die
   Benachrichtigungsbehandlungsroutine des Steuererelements hinzu
9:
10: //////////////////////////////////////
11: // EIGENER CODE, ANFANG
12: //////////////////////////////////////
13:
14: // Variablen aktualisieren
15: UpdateData(TRUE);
16:
17: //////////////////////////////////////
18: // EIGENER CODE, ENDE
19: //////////////////////////////////////
20: }

```

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

Den Timer für den Zähler starten und stoppen

Damit der zweite Timer funktionsfähig wird, müssen Sie ...

- die Variable **m_iInterval** initialisieren,
- den Timer starten, wenn der Benutzer auf die Schaltfläche **IDC_STARTTIME** klickt,

- bei jedem Timer-Ereignis die Variable `m_iCount` inkrementieren und das Dialogfeld aktualisieren,
- den Timer anhalten, wenn der Benutzer auf die Schaltfläche `IDC_STOPTIMER` klickt.

Diese zusätzliche Funktionalität realisieren Sie in folgenden Schritten:

1. Bearbeiten Sie die Funktion `OnInitDialog` entsprechend dem Code von Listing 4.4.

Listing 4.4: Die aktualisierte Funktion `OnInitDialog`

```

1: BOOL CTimerDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:     .
5:     .
6:     .
7:     // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
8:
9:     //////////////////////////////////////
10:    // EIGENER CODE, ANFANG
11:    //////////////////////////////////////
12:
13:    // Zählerintervall initialisieren
14:    m_iInterval = 100;
15:
16:    // Dialogfeld aktualisieren
17:    UpdateData(FALSE);
18:
19:    // Timer für Uhr starten
20:    SetTimer(ID_CLOCK_TIMER, 1000, NULL);
21:
22:    //////////////////////////////////////
23:    // EIGENER CODE, ENDE
24:    //////////////////////////////////////
25:
26:    return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
                // soll den AFokus erhalten
27: }
```

2. Nehmen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `BN_CLICKED` für die Schaltfläche `IDC_STARTTIME` auf, und fügen Sie den Code gemäß Listing 4.5 in die Funktion `OnStarttime` ein.

Listing 4.5: Die Funktion `OnStarttime`

```

1: void CTimerDlg::OnStarttime()
2: {
3:     // TODO: Code für die Behandlungsroutine der Steuerelement-
4:     // Benachrichtigung hier einfügen
5:
6:     //////////////////////////////////////
7:     // EIGENER CODE, ANFANG
```

```

7: ///////////////////////////////////
8:
9: // Variablen aktualisieren
10: UpdateData(TRUE);
11:
12: // Zähler initialisieren
13: m_iCount = 0;
14: // Zähler für Anzeige formatieren
15: m_sCount.Format("%d", m_iCount);
16:
17: // Dialogfeld aktualisieren
18: UpdateData(FALSE);
19: // Timer starten
20: SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
21:
22: ///////////////////////////////////
23: // EIGENER CODE, ENDE
24: ///////////////////////////////////
25: }

```

3. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht BN_CLICKED für die Schaltfläche IDC_STOPTIMER hinzu, und übernehmen Sie in die Funktion OnStoptimer den Code entsprechend Listing 4.6.

Listing 4.6: Die Funktion OnStoptimer

```

1: void CTimerDlg::OnStoptimer()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Timer anhalten
10: KillTimer(ID_COUNT_TIMER);
11:
12: ///////////////////////////////////
13: // EIGENER CODE, ENDE
14: ///////////////////////////////////
15: }

```

4. Aktualisieren Sie die Funktion OnTimer mit dem Code gemäß Listing 4.7.

Listing 4.7: Die aktualisierte Funktion OnTimer

```

1: void CTimerDlg::OnTimer(UINT nIDEvent)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
  einfügen Âund/oder Standard aufrufen
4:

```



```

5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Aktuelle Zeit holen
10: CTime curTime = CTime::GetCurrentTime();
11:
12: // Welcher Timer hat dieses Ereignis ausgelöst?
13: switch (nIDEvent)
14: {
15: // Der Uhren-Timer?
16: case ID_CLOCK_TIMER:
17: // Aktuelle Uhrzeit anzeigen
18: m_sTime.Format("%d:%d:%d", curTime.GetHour(),
19: curTime.GetMinute(),
20: curTime.GetSecond());
21: break;
22: // Der Zähler-Timer?
23: case ID_COUNT_TIMER:
24: // Zähler inkrementieren
25: m_iCount++;
26: // Zähler formatieren und anzeigen
27: m_sCount.Format("%d", m_iCount);
28: break;
29: }
30:
31: // Dialogfeld aktualisieren
32: UpdateData(FALSE);
33:
34: ///////////////////////////////////
35: // EIGENER CODE, ENDE
36: ///////////////////////////////////
37:
38: CDialog::OnTimer(nIDEvent);
39: }

```

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

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

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

Die eigentlich interessanten Dinge passieren in der Funktion `OnTimer`. Hier steht momentan nur der

Code zur Behandlung des Timer-Ereignisses für die Uhr. Um die Funktionalität für den Zähler-Timer hinzuzufügen, ist zu ermitteln, welcher Timer diese Funktion ausgelöst hat. Das einzige Argument an die Funktion `OnTimer` ist aber gerade die Timer-ID. Diese ID läßt sich in einer `switch`-Anweisung testen, um den Timer herauszufinden, der diese Funktion aufgerufen hat, und um zu steuern, welcher Code daraufhin auszuführen ist. Der Code für den Uhren-Timer bleibt gemäß Listing 4.2 erhalten. Der Code für den Zähler-Timer wird an der entsprechenden Stelle in der `switch`-Anweisung untergebracht. Er inkrementiert den Zähler und aktualisiert dann die Variable `m_sCount` auf den neuen Wert.

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie ein Timer-Intervall festlegen und den Timer starten, wie es Abbildung 4.6 zeigt.

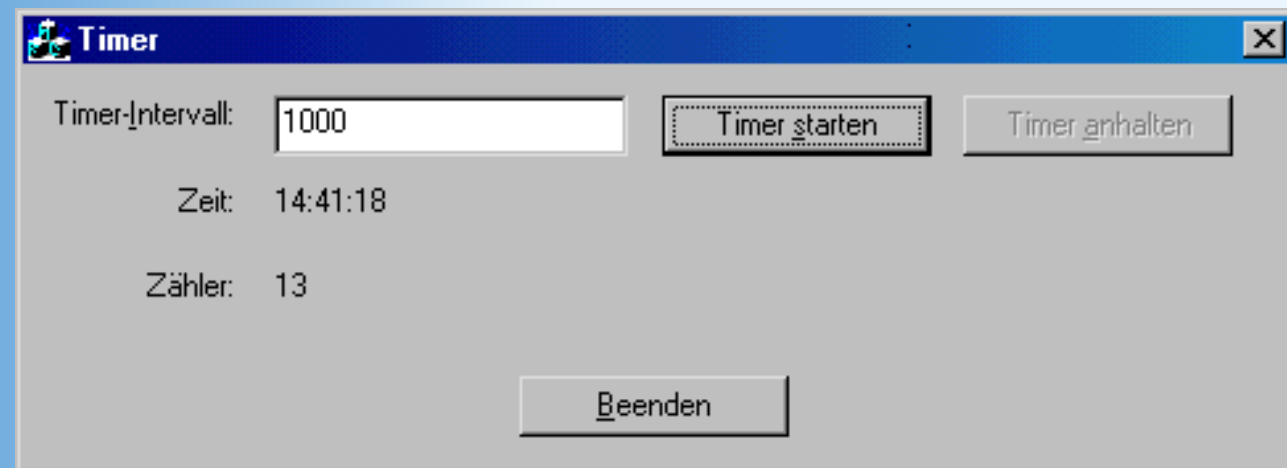


Abbildung 4.6:
Im Dialogfeld der Anwendung ist ein laufender Zähler zu sehen.

Die Schaltfläche Anhalten aktivieren

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

Es ist sinnvoll, nach dem Start des Timers die Schaltfläche **Anhalten** zu aktivieren und die Schaltfläche **Starten** zu deaktivieren. Sobald der Benutzer den Timer angehalten hat, kehren Sie das Ganze um. Das läßt sich auf die gleiche Weise erreichen, wie Sie die Steuerelemente am Tag 2 aktiviert und deaktiviert haben. Diese Lösung kann man auch ein wenig modifizieren.

Beim Hinzufügen der Variablen für die Steuerelemente haben Sie auch Variablen für die Schaltflächen **Starten** und **Anhalten** aufgenommen. Es handelt sich hierbei nicht um normale Variablen, sondern Steuerelementvariablen. Statt nun einen Zeiger auf die Steuerelemente über deren IDs zu ermitteln, können Sie direkt mit den Steuerelementvariablen arbeiten. Überarbeiten Sie dazu die Funktionen `OnStarttime` und `OnStoptimer` gemäß Listing 4.8.

Listing 4.8: Die überarbeiteten Funktionen `OnStarttime` und `OnStoptimer`

```
1: void CTimerDlg::OnStarttime()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
```

```

5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Variablen aktualisieren
10: UpdateData(TRUE);
11:
12: // Zähler initialisieren
13: m_iCount = 0;
14: // Zähler für Anzeige formatieren
15: m_sCount.Format("%d", m_iCount);
16:
17: // Dialogfeld aktualisieren
18: UpdateData(FALSE);
19: // Timer starten
20: SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
21:
22: // Schaltfläche Timer anhalten aktivieren
23: m_cStopTime.EnableWindow(TRUE);
24: // Schaltfläche Timer starten deaktivieren
25: m_cStartTime.EnableWindow(FALSE);
26:
27: ///////////////////////////////////
28: // EIGENER CODE, ENDE
29: ///////////////////////////////////
30: }
31:
32: void CTimerDlg::OnStoptimer()
33: {
34: // TODO: Code für die Behandlungsroutine der Steuerelement-
35: // Benachrichtigung hier einfügen
36: ///////////////////////////////////
37: // EIGENER CODE, ANFANG
38: ///////////////////////////////////
39:
40: // Timer anhalten
41: KillTimer(ID_COUNT_TIMER);
42:
43: // Schaltfläche Timer anhalten deaktivieren
44: m_cStopTime.EnableWindow(FALSE);
45: // Schaltfläche Timer starten aktivieren
46: m_cStartTime.EnableWindow(TRUE);
47:
48: ///////////////////////////////////
49: // EIGENER CODE, ENDE
50: ///////////////////////////////////
51: }

```

Wenn Sie jetzt die Anwendung kompilieren und ausführen, können Sie den Zähler- Timer starten und

anhalten, wie es Abbildung 4.7 zeigt. Damit kann der Benutzer mit dem Timer-Intervall experimentieren - verschiedene Intervalle ausprobieren und die Unterschiede beobachten, wobei die Uhr zum Vergleich weiterläuft.

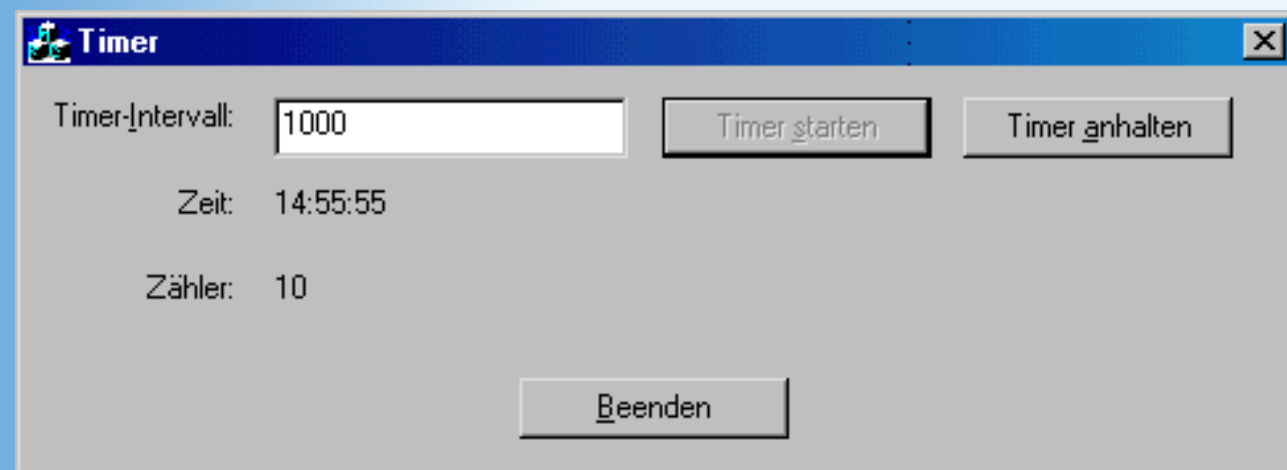


Abbildung 4.7:
Die fertiggestellte Anwendung

Zusammenfassung

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

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

Fragen und Antworten

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

Antwort:
Der verfügbare Bereich beginnt bei 55 Millisekunden und reicht bis 232 - 1 Millisekunden, d.h. knapp 50 Tage.

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

Antwort:
Diese Frage läßt sich nicht allgemeingültig beantworten. Allen Anwendungen steht im Betriebssystem Windows eine begrenzte Anzahl von Timern zur Verfügung. Obwohl diese Zahl mehr als ausreichend sein dürfte, wenn alle Anwendungen zusammen nicht mehr als eine Handvoll Timer verwenden, können diese Ressourcen dennoch knapp werden, wenn eine Anwendung eine größere Anzahl Timer für sich in

Anspruch nimmt. Als Faustregel gilt, daß man nicht mehr als etwa zwei oder drei Timer gleichzeitig betreiben sollte. Falls die Anwendung eine größere Zahl erfordert, empfiehlt es sich, den Entwurf der Anwendung noch einmal kritisch unter die Lupe zu nehmen und so umzubauen, daß sie mit weniger Timern auskommt.

Frage:

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

Antwort:

Ja, die gibt es. Alle Windows-Anwendungen verfügen über eine OnIdle-Funktion, mit der man eine Verarbeitung während der Leerlaufphase auslösen kann. Die Funktion OnIdle kommt am Tag 18 zur Sprache, wenn es um Multitasking geht.

Workshop

Kontrollfragen

1. Auf welche Weise haben Sie die beiden Timer-IDs zu den Ressourcensymbolen hinzugefügt?
2. Welche andere Möglichkeit gibt es, um diese beiden IDs in die Anwendung aufzunehmen?
3. Wie kann man zwei Timer in der Funktion OnTimer auseinanderhalten?
4. Wie viele Timer-Ereignisse empfängt Ihre Anwendung, wenn der Timer für das Intervall von 1 Sekunde eingerichtet wird, die Anwendung 1 Minute beschäftigt ist und somit den Empfang von Timer-Nachrichten unterbindet?

Übung

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

Woche 1**Tag 5****Dialogfelder**

In den meisten Anwendungen gibt es zahlreiche Situationen, wo die Anwendung Informationen vom Benutzer anfordert, beispielsweise wie man die Anwendung konfigurieren möchte oder ob ein Dokument vor dem Verlassen der Anwendung zu speichern ist. In den meisten derartigen Fällen öffnet die Anwendung ein neues Fenster, um die entsprechenden Fragen zu stellen. Man bezeichnet diese Fenster als Dialogfelder.

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

Die bisher erstellten Dialogfelder waren durchweg Anwendungen mit einem einzigen Dialogfenster. Heute lernen Sie, wie man ...

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

Vordefinierte (oder System-) Dialogfelder

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

Meldungsfelder

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

Die Funktion MessageBox

Wie Sie bereits aus den letzten Tagen wissen, kann die Funktion `MessageBox` ein oder zwei Argumente

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



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

Bezüglich der Kombination von Schaltflächen, die im Meldungsfeld erscheinen, sind Ihnen enge Grenzen gesetzt. Eigene Schaltflächenkombinationen lassen sich nicht festlegen. Sollten Sie mit den vordefinierten Schaltflächen oder den zulässigen Kombinationen nicht auskommen, bleibt Ihnen nichts anderes übrig, als ein benutzerdefiniertes Dialogfeld zu erstellen, das dem gewünschten Meldungsfeld entspricht. Tabelle 5.1 listet die verfügbaren Kombinationen von Schaltflächen auf, die man in der Funktion `MessageBox` spezifizieren kann.

Tabelle 5.1: IDs zur Festlegung der Schaltflächenkombination in der Funktion `MessageBox`

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

Um das anzuzeigende Symbol festzulegen, addiert man die ID des Symbols zur ID der Schaltflächenkombination. Die verfügbaren Symbole sind in Tabelle 5.2 aufgeführt. Möchten Sie nur das Symbol oder nur die Schaltflächenkombination festlegen und für das jeweils andere Element den Standardwert übernehmen, geben Sie einfach nur die gewünschte ID an.

Tabelle 5.2: IDs zur Festlegung der Symbole in der Funktion `MessageBox`

ID	Symbol
MB_ICONINFORMATION	Info-Symbol
MB_ICONQUESTION	Fragezeichen-Symbol
MB_ICONSTOP	Stop-Symbol
MB_ICONEXCLAMATION	Ausrufezeichen-Symbol

Wenn man eine Schaltflächenkombination angibt, liegt es nahe, auch den Rückgabewert zu verwenden, damit man ermitteln kann, auf welche Schaltfläche der Benutzer geklickt hat. Der Rückgabewert ist als Integer-Wert definiert. Die verfügbaren Werte sind in Tabelle 5.3 aufgelistet.

Tabelle 5.3: Rückgabewerte der Funktion `MessageBox`

ID	Gewählte Schaltfläche
IDABORT	Beenden
IDRETRY	Wiederholen
IDIGNORE	Ignorieren
IDYES	Ja
IDNO	Nein
IDOK	OK
IDCANCEL	Abbrechen

Eine Dialogfeld-Anwendung erstellen

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

Den ersten Teil der Anwendung erstellen Sie mit folgenden Schritten:

1. Legen Sie mit dem MFC-Anwendungs-Assistenten einen neuen Projektarbeitsbereich an, und nennen Sie das Projekt `Dialoge`.
2. Wählen Sie die gleichen Einstellungen wie in den Anwendungen der letzten Lektionen. Als Titel der Anwendung legen Sie `Dialoge` fest.
3. Gestalten Sie das Hauptdialogfeld der Anwendung entsprechend Abbildung 5.1, und legen Sie die Eigenschaften gemäß Tabelle 5.4 fest.

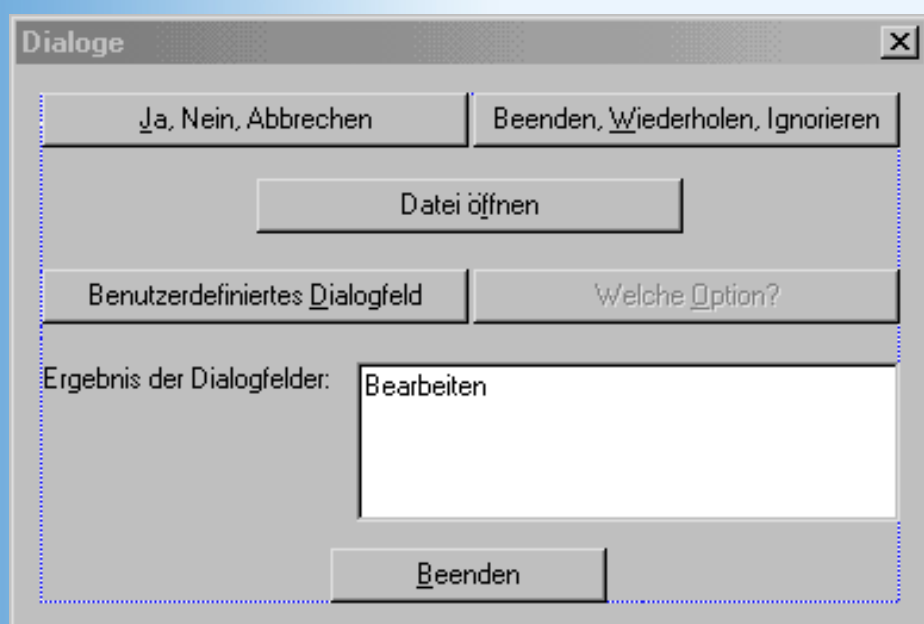


Abbildung 5.1:
Das Layout des Hauptdialogfelds der Anwendung

Tabelle 5.4: Eigenschaften der Steuerelemente für das Hauptdialogfeld

Objekt	Eigenschaft	Einstellung
Schaltfläche	ID	IDC_YESNOCANCEL
	Titel	&Ja, Nein, Abbrechen
Schaltfläche	ID	IDC_ABORTRETRYIGNORE
	Titel	Beenden, &Wiederholen, Ignorieren
Schaltfläche	ID	IDC_FILEOPEN
	Titel	Datei ö&ffnen
Schaltfläche	ID	IDC_BCUSTOMDIALOG
	Titel	Benutzerdefiniertes &Dialogfeld
Schaltfläche	ID	IDC_BWHICHOPTION
	Titel	Welche &Option?
Schaltfläche	ID	IDC_EXIT
	Titel	&Beenden
Text	ID	IDC_STATIC
	Titel	Ergebnis der Dialogfelder:
Eingabefeld	ID	IDC_RESULTS
	Mehrzeilig	Eingeschaltet
	Auto Vert. Bildlauf	Eingeschaltet

4. Verbinden Sie mit Hilfe des Klassen-Assistenten die Variablen mit den in Tabelle 5.5 aufgeführten Steuerelementen.

Tabelle 5.5: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_RESULTS	m_sResults	Wert	CString
IDC_BWHICHOPTION	m_cWhichOption	Control	CButton

5. Weisen Sie wie an den vorherigen Tagen mit Hilfe des Klassen-Assistenten Code an die Schaltfläche **Beenden** zu, um die Anwendung zu schließen.

Die Meldungsfelder codieren

Erstellen Sie mit dem Klassen-Assistenten für die erste Schaltfläche (diejenige mit der Beschriftung Ja, Nein, Abbrechen) eine Funktion für das Klickereignis, wie Sie es von den vergangenen Tagen her kennen. Nehmen Sie in die Funktion für diese Schaltfläche den Code von Listing 5.1 auf.

Listing 5.1: Die Funktion OnYesnocancel

```

1: void CDialogDlg::OnYesnocancel()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
```

ÄBenachrichtigung hier einfügen

```

4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: int iResults; // Variable für Schaltflächenauswahl
10:
11: // Benutzer fragen
12: iResults = MessageBox("Ja, Nein oder Abbrechen drücken",
13: "Dialogfeld Ja, Nein, Abbrechen",
14: MB_YESNOCANCEL | MB_ICONINFORMATION);
15:
16: // Angeklickte Schaltfläche ermitteln
17: // Benutzer die angeklickte Schaltfläche melden
18: switch (iResults)
19: {
20: case IDYES: // Schaltfläche Ja?
21: m_sResults = "Ja! Ja! Ja!";
22: break;
23: case IDNO: // Schaltfläche Nein?
24: m_sResults = "Nein, nein, nein, nein, nein.";
25: break;
26: case IDCANCEL: // Schaltfläche Abbrechen?
27: m_sResults = "Sorry, abgebrochen.";
28: break;
29: }
30:
31: // Dialogfeld aktualisieren
32: UpdateData(FALSE);
33:
34: //////////////////////////////////
35: // EIGENER CODE, ENDE
36: //////////////////////////////////
37: }

```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie verfolgen, wie sich durch Wahl der verschiedenen Schaltflächen im Meldungsfeld der Ablauf der Aktionen in der Anwendung bestimmen lässt. Nehmen Sie mit dem Klassen-Assistenten eine Funktion für die Klickereignisse der Schaltfläche Beenden, Wiederholen, Ignorieren auf, und geben Sie den gleichen Code wie in Listing 5.1 ein. Jetzt verwenden Sie jedoch die Werte `MB_ABORTRETRYIGNORE` und `MB_ICONQUESTION` und ändern die Aufforderungen und Meldungen. Sie können sich nun davon überzeugen, wie sich diese andere Schaltflächenkombination in der gleichen Weise verwenden lässt.

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

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

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie auf die beiden oberen Schaltflächen

klicken und eine Meldung wie in Abbildung 5.2 sehen. Nachdem Sie auf eine der Schaltflächen des Meldungsfelds geklickt haben, erscheint im Eingabefeld des Hauptdialogfelds eine Meldung, die auf die gewählte Schaltfläche hinweist (siehe Abbildung 5.3).

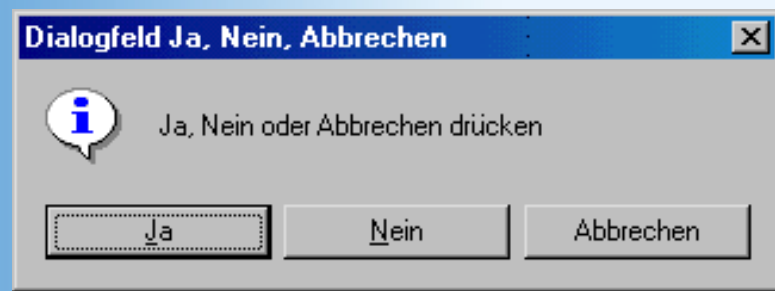


Abbildung 5.2:
Das Meldungsfeld mit drei Auswahlen

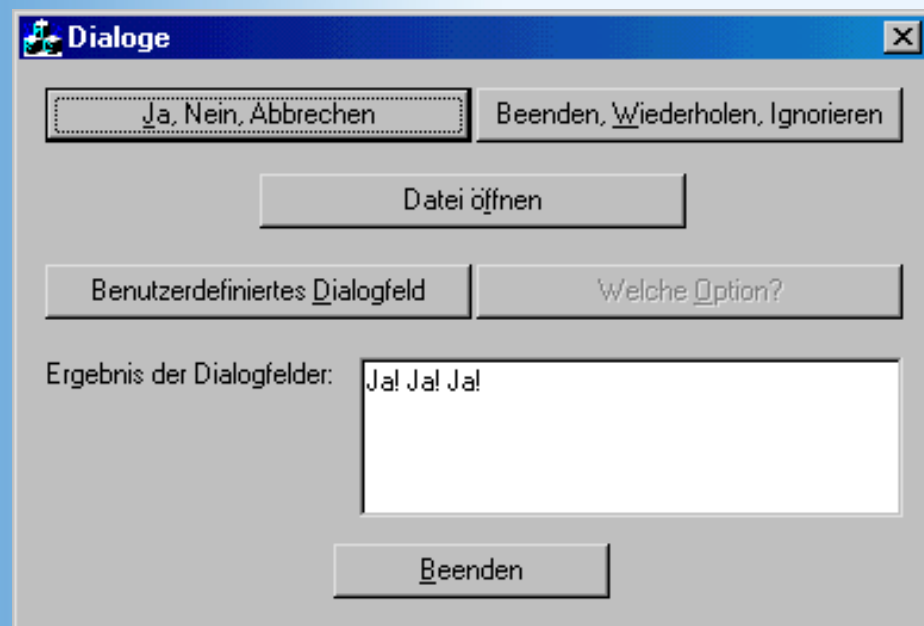


Abbildung 5.3:
Je nach angeklickter Schaltfläche wird eine Meldung angezeigt.

Standarddialogfelder

Standarddialogfelder lassen sich nicht so einfach einsetzen wie die Funktion `MessageBox`, trotzdem bleibt alles noch überschaubar. Die Microsoft Foundation Classes (MFC) bieten mehrere C++-Klassen für Windows-Standarddialogfelder. Tabelle 5.6 bringt eine Übersicht zu diesen Klassen.

Tabelle 5.6: Klassen für Standarddialogfelder

Klasse	Dialogfeldtyp
CFileDialog	Dateiauswahl
CFontDialog	Schriftauswahl
CColorDialog	Farbauswahl
CPageSetupDialog	Seite zum Drucken einrichten
CPrintDialog	Drucken
CFindReplaceDialog	Suchen und Ersetzen

Den in diesen Klassen verkapselten Standarddialogfeldern begegnen Sie auf Schritt und Tritt in den meisten Windows-Anwendungen, wenn Sie zum Beispiel Dateien öffnen, Druckeroptionen einstellen, drucken oder

Begriffe suchen und ersetzen wollen. Neben diesen Auswahlen bieten verschiedene OLE-Standarddialogklassen mehrere Standardfunktionen für OLE- oder ActiveX-Komponenten und Anwendungen.

Alle derartigen Dialogfelder setzt man in der gleichen Weise ein, auch wenn einzelne Eigenschaften und Klassenfunktionen je nach der Funktionalität des Dialogfelds variieren können. Um ein Standarddialogfeld zu verwenden, führen Sie folgende Schritte aus:

1. Deklarieren Sie eine Variable des Klassentyps.
2. Legen Sie alle erforderlichen Eigenschaften fest, bevor Sie das Dialogfeld dem Benutzer anzeigen.
3. Rufen Sie die Methode `DoModal` der Klasse auf, um das Dialogfeld anzuzeigen.
4. Übernehmen Sie den Rückgabewert der Methode `DoModal`, um zu ermitteln, ob der Benutzer auf **OK** oder **Abbrechen** geklickt hat.
5. Wenn der Benutzer auf die Schaltfläche **OK** geklickt hat, lesen Sie alle Eigenschaften, die der Benutzer eventuell im Dialogfeld gesetzt oder geändert hat.

Um die Funktionsweise besser verstehen zu können, fügen Sie der Beispielanwendung die Klasse `CFileDialog` hinzu. Nehmen Sie dazu eine Funktion für das Klickereignis der Schaltfläche Datei öffnen mit Hilfe des Klassen-Assistenten auf. Schreiben Sie in diese Funktion den Code aus Listing 5.2.

Listing 5.2: Die Funktion `OnFileopen`

```

1: void CDialogDlg::OnFileopen()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: CFileDialog m_ldFile(TRUE);
10:
11: // Dialogfeld Öffnen zeigen und Ergebnis auffangen
12: if (m_ldFile.DoModal() == IDOK)
13: {
14: // Gewählten Dateinamen ermitteln
15: m_sResults = m_ldFile.GetFileName();
16: // Dialogfeld aktualisieren
17: UpdateData(FALSE);
18: }
19:
20: //////////////////////////////////
21: // EIGENER CODE, ENDE
22: //////////////////////////////////
23: }
```

Diese Funktion deklariert zuerst eine Instanz der Klasse `CFileDialog`. Dieser Instanz wird `TRUE` als Argument an den Klassenkonstruktor übergeben. Der Klasse ist damit bekannt, daß es sich um ein Dialogfeld Datei öffnen handelt. Bei Übergabe von `FALSE` erscheint ein Dialogfeld Speichern unter. Zwischen beiden Dialogfeldern gibt es keinen funktionellen, sondern nur einen visuellen Unterschied. An den Konstruktor können Sie noch weitere Argumente übergeben und damit die anzuzeigenden Dateierweiterungen, eine Startdatei und deren Standort sowie Filter zur Anzeige ausgewählter Dateien festlegen.

Nachdem Sie die Instanz des Dialogfelds Öffnen erzeugt haben, können Sie deren Funktion `DoModal` aufrufen. Es handelt sich dabei um eine Member-Funktion der Basisklasse `CDialog`. Diese Funktion ist in allen

Dialogfenstern verfügbar. Die Funktion `DoModal` zeigt das Dialogfeld Öffnen wie in Abbildung 5.4 dargestellt an. Aus dem Rückgabewert der Funktion `DoModal` lässt sich ableiten, auf welche Schaltfläche der Benutzer geklickt hat. Die Wahl der Schaltfläche Öffnen liefert den Wert `IDOK` zurück, genau wie bei der Funktion `MessageBox`. Je nach Benutzerauswahl können Sie nun anhand des Rückgabewerts festlegen, wie das Programm fortzusetzen ist.



Ein Dialogfeld kann man in zwei unterschiedlichen Modi anzeigen. Den ersten Modus bezeichnet man als modales Fenster. Solange es angezeigt wird, kann der Benutzer keinerlei Aktionen in der Anwendung ausführen, bis er das Dialogfeld schließt. Das Meldungsfeld ist ein typisches Beispiel eines modalen Fensters. Hier kann der Benutzer erst dann weiterarbeiten, wenn er eine der Schaltflächen im Meldungsfeld betätigt.

Der zweite Modus, in dem sich ein Dialogfeld anzeigen lässt, ist das nichtmodale Fenster. Dieses kann geöffnet sein, während der Benutzer andere Aktionen in der Anwendung ausführt. Typische Beispiele für nichtmodale Fenster sind das Dialogfeld **Suchen** sowie das Dialogfeld **Suchen und Ersetzen** in Microsoft Word. Diese Dialogfelder können geöffnet bleiben, während man weiter an dem Dokument arbeitet, das man durchsucht.

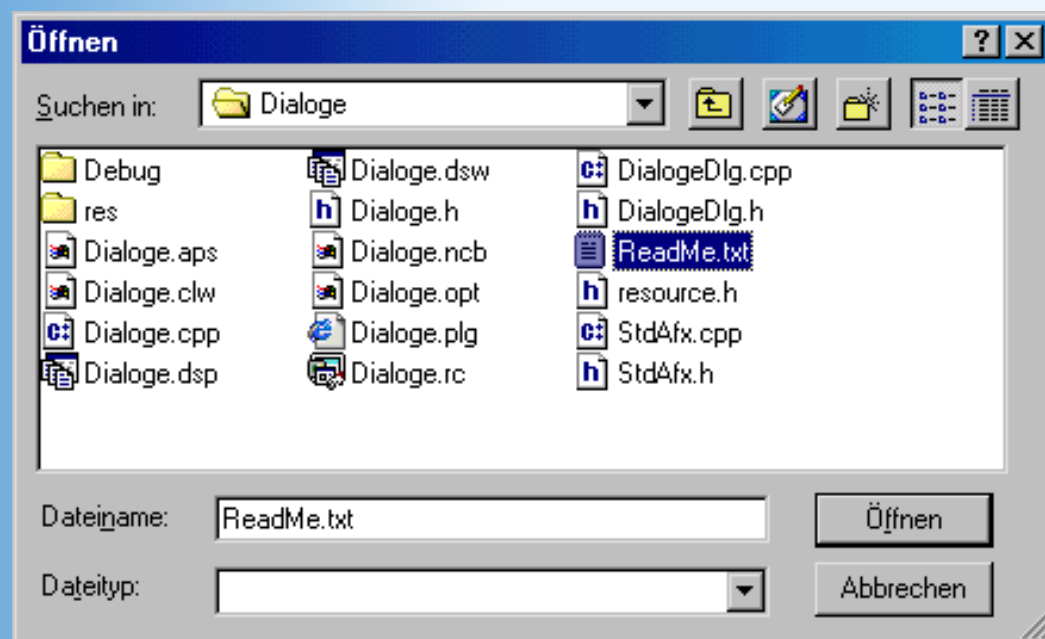


Abbildung 5.4:
Das Dialogfeld Öffnen

Um den Namen der ausgewählten Datei anzuzeigen, setzen Sie die Variable `m_sResults` auf den Rückgabewert der Methode `GetFileName` der Klasse `CFileDialog`. Diese Methode liefert lediglich den Dateinamen ohne Verzeichnispfad oder Laufwerksbezeichnung, wie es Abbildung 5.5 verdeutlicht. Mit anderen Methoden der Klasse kann man auch den Verzeichnispfad (`GetPathName`) und die Dateierweiterung (`GetFileExt`) ermitteln.

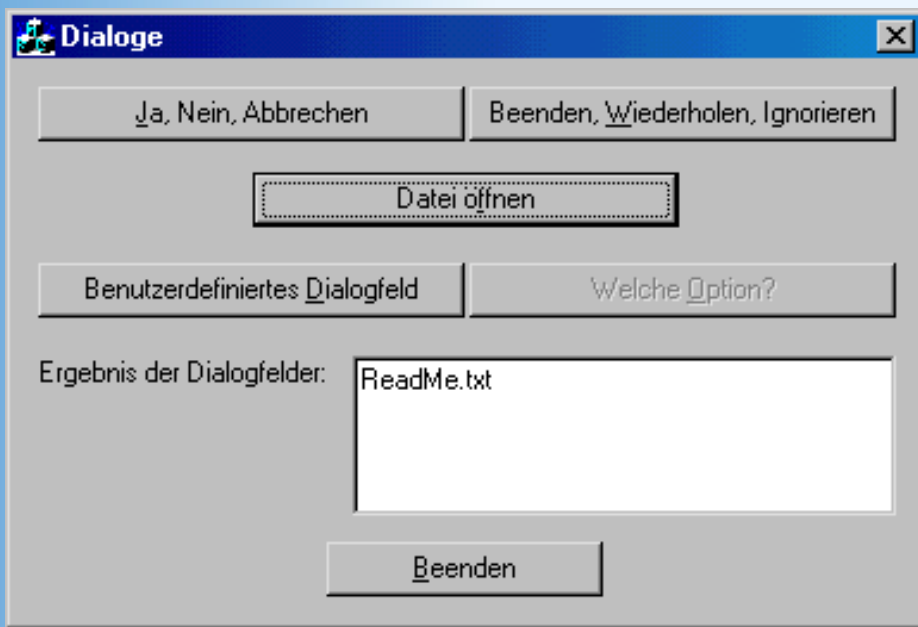


Abbildung 5.5:
Den ausgewählten Dateinamen anzeigen

Eigene Dialogfelder erstellen

Mit Standarddialogfeldern können Sie nun umgehen. Was aber, wenn Sie ein benutzerdefiniertes Dialogfeld für Ihre Anwendung brauchen? Diese Aufgabe ist ziemlich einfach zu lösen, da es sich im wesentlichen um eine Kombination der Schritte handelt, die Sie bereits vom Erstellen des Hauptdialogfelds der Beispielanwendungen und dem Einsatz von Standarddialogfeldern her kennen. Es sind zwar ein paar zusätzliche Schritte erforderlich, die Ihnen aber bald geläufig sein dürften.

Das Dialogfeld erzeugen

Das benutzerdefinierte Dialogfeld für die Beispielanwendung stellt dem Benutzer ein Eingabefeld bereit, in das er Text eingeben, sowie ein Gruppe von Optionsfeldern, aus denen der Benutzer eine Option auswählen kann. Wenn der Benutzer auf die Schaltfläche OK klickt, zeigt die Anwendung den vom Benutzer eingegebenen Text im Anzeigebereich des Hauptdialogfelds der Anwendung an. Über eine weitere Schaltfläche kann sich der Benutzer anzeigen lassen, welches der Optionsfelder ausgewählt ist. Diese Übung zeigt, wie man mit benutzerdefinierten Dialogfeldern Informationen vom Benutzer einholt und wie man die vom Benutzer getroffenen Auswahlen auswertet, nachdem das Dialogfeld geschlossen wird.

Um ein benutzerdefiniertes Dialogfeld für die Anwendung zu erstellen, sind folgende Aufgaben zu erledigen:

- Ein weiteres Dialogfeld zu den Ressourcen der Anwendung hinzufügen.
- Das Layout des Dialogfelds gestalten.
- Die Basisklasse deklarieren, von der das Dialogfeld abgeleitet wird.
- Variablen an die Steuerelemente im Dialogfeld zuweisen.

Danach können Sie das benutzerdefinierte Dialogfeld in Ihrer Anwendung einsetzen. Die genannten Aufgaben führen Sie in folgenden Schritten aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**.
2. Klicken Sie mit der rechten Maustaste auf den Ordner **Dialog**, und wählen Sie aus dem Kontextmenü den Befehl **Dialog einfügen**.
3. Klicken Sie mit der rechten Maustaste auf das neue Dialogfeld in der Baumansicht der Ressourcen, und wählen Sie aus dem Kontextmenü den Befehl **Eigenschaften**.
4. Ändern Sie die Objekt-ID für das neue Dialogfeld in **IDD_MESSAGEDLG**.

5. Wenn Sie das neue Dialogfeld bearbeiten, behalten Sie die Schaltflächen **OK** und **Abbrechen** bei. Verschieben Sie sie aber auf eine andere Position, wie es Abbildung 5.6 zeigt.

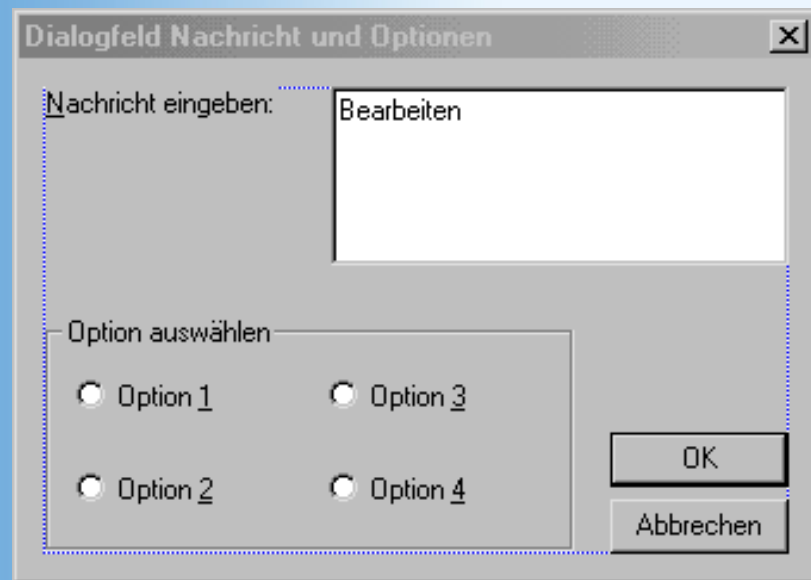


Abbildung 5.6:
Das Layout des benutzerdefinierten Dialogfelds

6. Nehmen Sie den Entwurf für das restliche Dialogfeld vor, und richten Sie die Objekteigenschaften gemäß Tabelle 5.7 ein.

Tabelle 5.7: Eigenschaftseinstellungen der Steuerelemente für das benutzerdefinierte Dialogfeld

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	&Nachricht eingeben:
Eingabefeld	ID	IDC_MESSAGE
	Mehrzeilig	Eingeschaltet
	Auto Vert. Bildlauf	Eingeschaltet
Gruppenfeld	ID	IDC_STATIC
	Titel	Option auswählen
Optionsfeld	ID	IDC_OPTION1
	Titel	Option &1
	Gruppe	eingeschaltet
Optionsfeld	ID	IDC_OPTION2
	Titel	Option &2
Optionsfeld	ID	IDC_OPTION3
	Titel	Option &3
Optionsfeld	ID	IDC_OPTION4
	Titel	Option &4

7. Wenn Sie den Entwurf des Dialogfelds abgeschlossen haben, öffnen Sie den Klassen-Assistenten. Es erscheint das Dialogfeld gemäß Abbildung 5.7.

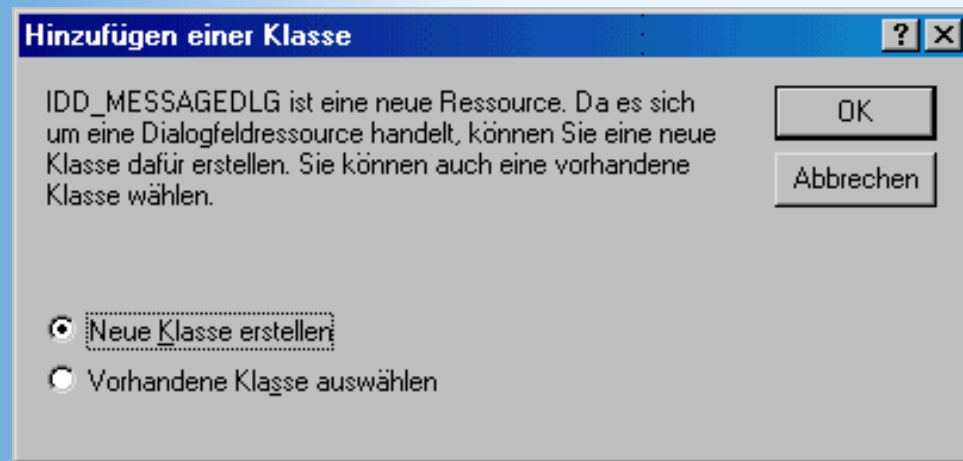


Abbildung 5.7:
Das Dialogfeld Hinzufügen einer Klasse

8. Übernehmen Sie in diesem Dialogfeld die vorgegebene Option *Neue Klasse erstellen*, und klicken Sie auf **OK**. Es erscheint ein weiteres Dialogfeld, in dem Sie den Namen für die neue Klasse spezifizieren und die Basisklasse, von dem die neue Klasse abzuleiten ist, festlegen können.

9. Tragen Sie in das Feld **Name** die Bezeichnung `CMsgDlg` ein, und vergewissern Sie sich, daß im Listenfeld **Basisklasse** der Eintrag `CDialog` ausgewählt ist (siehe Abbildung 5.8).

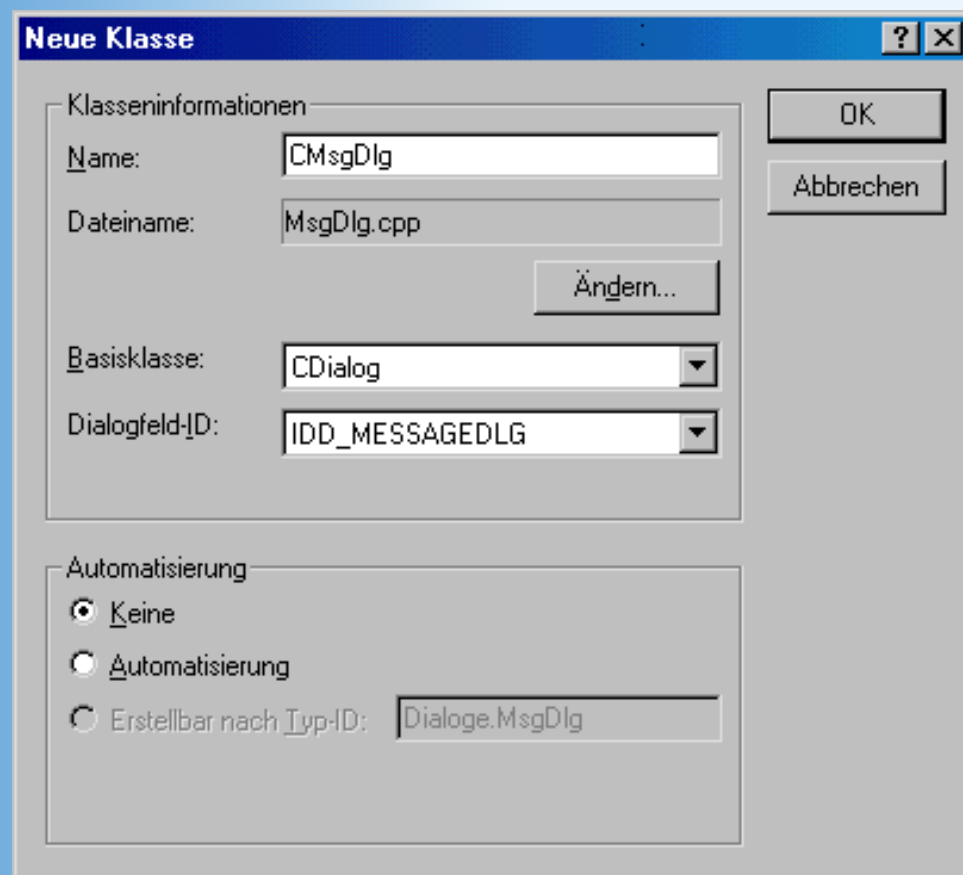


Abbildung 5.8:
Das Dialogfeld Neue Klasse

10. Lassen Sie die anderen Einstellungen im Dialogfeld unverändert, und klicken Sie auf **OK**.

11. Nachdem Sie wieder zum Klassen-Assistenten zurückgekehrt sind, weisen Sie den Steuerelementen im neuen Dialogfeld die Variablen entsprechend Tabelle 5.8 zu.

Tabelle 5.8: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_MESSAGE	m_sMessage	Wert	CString
IDC_OPTION1	m_iOption	Wert	int

Zwei Dinge sind zu beachten, wenn Sie die Steuerelementeigenschaften und Variablen im benutzerdefinierten Dialogfeld konfigurieren. Erstens sollten Sie nur für das erste Optionsfeld die Eigenschaft Gruppe einschalten. Damit wird gekennzeichnet, daß alle sich anschließenden Optionsfelder zu einer einzigen Gruppe gehören, in der nur ein Optionsfeld zu einem bestimmten Zeitpunkt ausgewählt sein kann. Wenn Sie die Eigenschaft Gruppe für alle Optionsfelder einschalten, sind die Optionsfelder voneinander unabhängig, so daß man mehrere Optionen gleichzeitig auswählen kann. Damit entspricht das Verhalten der Optionsfelder dem von Kontrollkästchen, wobei der Hauptunterschied darin besteht, daß der Benutzer die Optionen nicht wieder ausschalten kann. Das widerspricht dem Standardverhalten, daß immer nur eine Option markiert ist. Der andere Unterschied bezieht sich auf das Erscheinungsbild. Optionsfelder weisen runde Auswahlbereiche auf, während Kontrollkästchen quadratisch sind.

Als nächstes fällt auf, daß nur eine einzige Integer-Variable für die Optionsfelder deklariert wurde, und zwar für das Optionsfeld, für das die Eigenschaft Gruppe eingeschaltet ist. Der Wert dieser Variablen hängt davon ab, welches Optionsfeld ausgewählt ist. Handelt es sich um das erste Optionsfeld, hat die Variable den Wert 0, beim zweiten Optionsfeld den Wert 1 usw. Umgekehrt können Sie bestimmen, welches Optionsfeld ausgewählt sein soll, indem Sie den Wert der Variablen festlegen (auf die Nummer des Optionsfelds in der Gruppe minus 1).



In der Programmiersprache C++ beginnen alle Numerierungen mit 0 und nicht mit 1. Die erste Position in einem Array oder in einer Gruppe von Steuerelementen ist demnach die Position 0, die zweite die Position 1, das dritte Element hat den Index 2 usw.

Es sind nun alle Arbeiten abgeschlossen, um das Dialogfeld in der Anwendung einzusetzen. Vielleicht erwarten Sie, daß die Funktion `UpdateData` im Code für das Dialogfeld aufzurufen ist. Da Sie aber die Schaltflächen OK und Abbrechen nicht aus dem Dialogfeld entfernt haben, findet der Aufruf von `UpdateData` automatisch statt, sobald der Benutzer auf OK klickt. Damit brauchen Sie für das zweite Dialogfeld überhaupt keinen Code zu verfassen, sondern nur für das erste.

Das Dialogfeld in der Anwendung einsetzen

Nachdem Sie Ihr benutzerdefiniertes Dialogfeld fertiggestellt haben, können Sie es genauso einsetzen, wie die zu Windows gehörenden Standarddialogfelder. Zuerst müssen Sie eine Instanz der benutzerdefinierten Dialogfeldklasse deklarieren, die den Konstruktor der Klasse aufruft und eine Instanz der Klasse erzeugt. Als nächstes rufen Sie die Methode `DoModal` des Dialogfelds auf und speichern den Rückgabewert der Funktion in einer Variablen. Schließlich werten Sie die Variablen, die Sie den Steuerelementen des Dialogfelds zugeordnet haben, aus.

Die Instanz des Dialogfelds erzeugen

Bevor Sie das benutzerdefinierte Dialogfeld in der Anwendung nutzen können, müssen Sie dem Hauptdialogfeld der Anwendung das benutzerdefinierte Dialogfeld sowie dessen Variablen und Methoden bekanntmachen und festlegen, wie das Hauptdialogfeld mit dem benutzerdefinierten Dialogfeld in Wechselwirkung treten kann. Das läßt sich erreichen, indem Sie die Header-Datei für das benutzerdefinierte Dialogfeld in die Quelldatei für das Hauptdialogfeld der Anwendung einbinden. Führen Sie die folgenden Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Dateien**.
2. Erweitern Sie die Ordner **Dialoge** **Dateien** und danach **Quellcodedateien**.
3. Doppelklicken Sie auf die Datei **Dialogedlg.cpp**. Daraufhin erscheint der Quellcode für das

Hauptdialogfeld der Anwendung im Bearbeitungsbereich des Visual Studios.

4. Gehen Sie an den Beginn des Quellcodes, wo sich die `#include`-Anweisungen befinden, und fügen Sie eine `#include`-Anweisung für die Datei `MsgDlg.h` vor die Datei `DialogeDlg.h` ein, wie es Listing 5.3 zeigt.

Listing 5.3: Die eingebundenen Header-Dateien

```
1: // DialogeDlg.cpp : Implementierungsdatei
2: //
3:
4: #include "stdafx.h"
5: #include "Dialoge.h"
6: #include "MsgDlg.h"
7: #include "DialogeDlg.h"
8:
9: #ifdef _DEBUG
10: #define new DEBUG_NEW
11: #undef THIS_FILE
12: static char THIS_FILE[] = __FILE__;
13: #endif
14:
15:
16: //////////////////////////////////////
17: // CAboutDlg-Dialogfeld für Anwendungsbefehl "Info"
```

Die `#include`-Anweisung für die Datei `MsgDlg.h` muß vor der `#include`-Anweisung für die Datei `DialogeDlg.h` stehen, weil Sie eine Variablendeklaration für das benutzerdefinierte Dialogfeld in die Klasse des Hauptdialogfelds in der Header-Datei des Hauptdialogfelds aufnehmen. Wenn Sie die Header-Datei `MsgDlg.h` nach der Header-Datei für das Hauptdialogfeld einbinden, beschwert sich der Compiler und weigert sich, Ihre Anwendung zu kompilieren, bis Sie die `#include`-Anweisung der Datei `MsgDlg.h` über die `#include`-Anweisung der Datei `DialogeDlg.h` verschieben.



In den Programmiersprachen C und C++ bezeichnet man die `#include`-Anweisung als Compilerdirektive. Sie teilt dem Compiler mit, den Inhalt der angegebenen Datei in den zu kompilierenden Quellcode einzulesen. Man verwendet diese Direktiven, um Deklarationen von Klassen, Strukturen und Funktionen in separate Dateien zu schreiben, die sich in jeden Quellcode aufnehmen lassen, der auf die Informationen in der Header-Datei angewiesen ist. Weitere Informationen zur Funktionsweise der `#include`-Anweisungen und warum man sie verwendet finden Sie im Anhang A.

Nachdem Sie nun dem Hauptdialogfeld der Anwendung das benutzerdefinierte Dialogfeld bekanntgemacht haben, müssen Sie eine Variable Ihres benutzerdefinierten Dialogfelds deklarieren. Führen Sie dazu folgende Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**.
2. Klicken Sie mit der rechten Maustaste auf die Klasse `CDialogeDlg`, um das Kontextmenü zu öffnen.
3. Wählen Sie aus dem Kontextmenü den Befehl **Member-Variable hinzufügen**.
4. Im Dialogfeld **Member-Variable hinzufügen** legen Sie den Variablentyp mit `CMsgDlg`, den Variablennamen mit `m_dMsgDlg` und den Zugriffsstatus als *Privat* fest. Klicken Sie auf **OK**, um die Variable in Ihr Hauptdialogfeld aufzunehmen.

Wenn Sie die Klasse `CDialogeDlg` in der Baumansicht erweitern, ist die Instanz des benutzerdefinierten Dialogfelds als Element der Hauptdialogfeldklasse der Anwendung zu sehen. Das bedeutet, daß Sie nun das benutzerdefinierte Dialogfeld in Ihrer Anwendung nutzen können.

Das Dialogfeld aufrufen und die Variablen lesen

Sie haben nun das benutzerdefinierte Dialogfeld in das Hauptdialogfeld der Anwendung aufgenommen, und zwar als Variable, die immer verfügbar ist, und nicht einfach als lokale Variable, die nur innerhalb einer einzigen Funktion gültig ist (wie bei der Variablen `CFileDialog`). Jetzt können Sie den Code hinzufügen, um das Dialogfeld tatsächlich zu nutzen. Führen Sie dazu die folgenden Schritte aus:

1. Öffnen Sie den Klassen-Assistenten, und fügen Sie eine Funktion für das Klickereignis (`BN_CLICKED`) der Schaltfläche `IDC_BCUSTOMDIALOG` hinzu.
2. Fügen Sie eine Funktion für das Klickereignis der Schaltfläche `IDC_BWHICHOPTION` hinzu.
3. Übernehmen Sie in die Funktion `OnBcustomdialog` den Code aus Listing 5.4.

Listing 5.4: Die Funktion `OnBcustomdialog`

```

1: void CDialogDlg::OnBcustomdialog()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Nachrichtung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Meldungsdialog anzeigen und Ergebnis auffangen
10: if (m_dMsgDlg.DoModal () == IDOK)
11: {
12: // OK gewählt. Nachricht anzeigen, die Benutzer
13: // in Nachrichtendialog eingegeben hat
14: m_sResults = m_dMsgDlg.m_sMessage;
15: // Dialogfeld aktualisieren
16: UpdateData(FALSE);
17: // Schaltfläche welche Option aktivieren
18: m_cWhichOption.EnableWindow(TRUE);
19: }
20:
21: //////////////////////////////////
22: // EIGENER CODE, ENDE
23: //////////////////////////////////
24: }
```

4. Übernehmen Sie in die Funktion `OnBwhichoption` den Code aus Listing 5.5.

Listing 5.5: Die Funktion `OnBwhichoption`

```

1: void CDialogDlg::OnBwhichoption()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Nachrichtung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Gewähltes Optionsfeld ermitteln und mit einer
10: // Meldung die gewählte Option anzeigen.
11: switch(m_dMsgDlg.m_iOption)
12: {
13: case 0: // Erstes Optionsfeld gewählt?
```

```

14: m_sResults = "Die erste Option wurde gewählt.";
15: break;
16: case 1: // Zweites Optionsfeld gewählt?
17: m_sResults = "Die zweite Option wurde gewählt.";
18: break;
19: case 2: // Drittes Optionsfeld gewählt?
20: m_sResults = "Die dritte Option wurde gewählt.";
21: break;
22: case 3: // Viertes Optionsfeld gewählt?
23: m_sResults = "Die vierte Option wurde gewählt.";
24: break;
25: default: // Keine Optionsfelder ausgewählt?
26: m_sResults = "Es wurde keine Option gewählt.";
27: break;
28: }
29:
30: // Dialogfeld aktualisieren
31: UpdateData(FALSE);
32:
33: ///////////////////////////////////
34: // EIGENER CODE, ENDE
35: ///////////////////////////////////
36: }

```

Der Code im ersten Listing ruft die Methode `DoModal` des benutzerdefinierten Dialogfelds auf. Diese Methode zeigt das Dialogfeld an und wartet darauf, daß der Benutzer auf eine der beiden Schaltflächen im Dialogfeld klickt, wie es Abbildung 5.9 zeigt. Wählt der Benutzer die Schaltfläche OK, kopieren Sie die vom Benutzer in das benutzerdefinierte Dialogfeld eingegebene Nachricht in die Variable des Eingabefelds, das die Nachricht für den Benutzer anzeigen soll. Nach der Aktualisierung der Dialogfeldanzeige mit den neuen Variablenwerten, aktivieren Sie die Schaltfläche Welche Option wie in Abbildung 5.10. Entscheidet sich der Benutzer für die Schaltfläche Abbrechen, brauchen Sie gar nichts zu unternehmen. Die Dialogfeldanzeige ändert sich nicht.

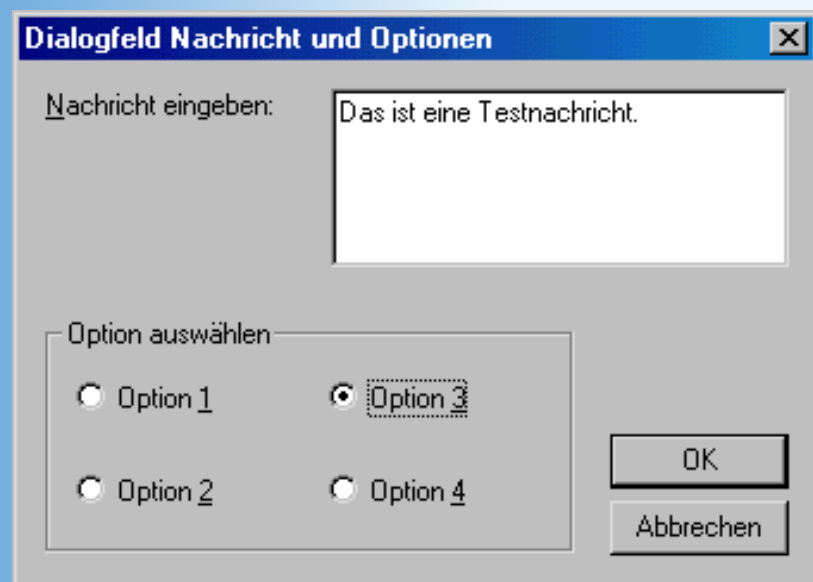


Abbildung 5.9:
Im benutzerdefinierten Dialogfeld kann der Benutzer eine Nachricht eintippen.

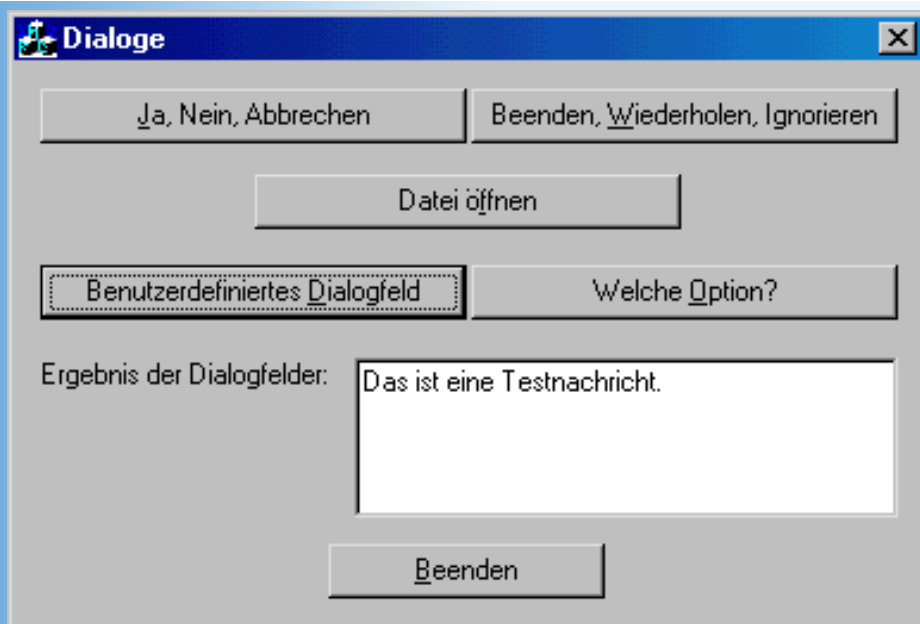


Abbildung 5.10:
Die im benutzerdefinierten Dialogfeld eingegebene Nachricht wird dem Benutzer angezeigt.

Wenn der Benutzer auf die Schaltfläche *Welche Option* klickt, übergeben Sie Variable des Optionsfelds im benutzerdefinierten Dialogfeld an eine `switch`-Anweisung, um eine Meldung auszuwählen, die dem Benutzer das gewählte Optionsfeld anzeigt (siehe Abbildung 5.11). Beachten Sie, daß man in beiden Funktionen direkt auf die Variablen der Steuerelemente im benutzerdefinierten Dialogfeld vom Hauptdialogfeld aus zugreifen kann. Das ist darauf zurückzuführen, daß der Klassen-Assistent automatisch die mit den Steuerelementen verbundenen Variablen als `Public` deklariert und sie somit für einen Zugriff von außerhalb der Dialogfeldklasse zugänglich macht. Dieses Verhalten können Sie ändern, indem Sie den Zugriffsspezifizierer `public:` durch `private:` ersetzen. Nach der Zeile `// {{AFX_DATA`, wo die Deklaration der Variablen steht, dürfen Sie keine Änderungen vornehmen, da die Variablen in einem Makro des MFC- Klassen-Assistenten deklariert werden. Damit können die Assistenten des Visual Studio ohne Mitwirkung des Visual C++-Compilers bei Bedarf die Variablen lokalisieren und manipulieren, wenn Sie Ihre Anwendung kompilieren.

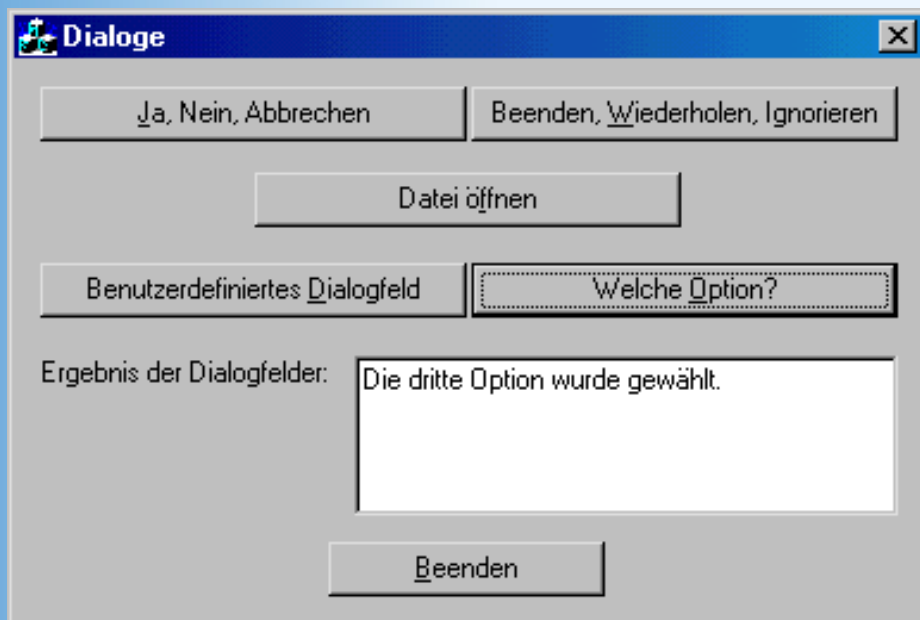


Abbildung 5.11:
Die im benutzerdefinierten Dialogfeld gewählte Option wird dem Benutzer angezeigt.

Zusammenfassung

Heute haben Sie gelernt, wie man zusätzliche Dialogfelder in eine Anwendung einbaut, um die Interaktion des Benutzers mit der Anwendung zu ermöglichen. Es wurde gezeigt, welche Optionen für die einfache `MessageBox`-Funktion verfügbar sind, wie man dem Benutzer verschiedene Schaltflächenkombinationen anbietet und wie man die vom Benutzer getroffene Auswahl bestimmt. Anhand dieser Informationen läßt sich der weitere Ablauf eines Programms festlegen.

Im nächsten Thema ging es um Standarddialogfelder, die in das Betriebssystem Windows integriert sind. Es wurde dargestellt, wie diese Standarddialoge in C++-Klassen der MFC-Klassenbibliothek verkapselt sind. Sie haben gelernt, wie man dem Benutzer die Auswahl von Dateien mit dem Standarddialogfeld Öffnen ermöglicht und wie man ermitteln kann, welche Datei der Benutzer ausgewählt hat.

Schließlich haben Sie erfahren, wie man eigene Dialogfelder entwirft und sie in eine Anwendung einbaut, um Informationen vom Benutzer einzuholen und diese Informationen in der Anwendung auszuwerten.

Fragen und Antworten

Frage:

Für das benutzerdefinierte Dialogfeld wurde kein Code geschrieben. Muß ich meine benutzerdefinierten Dialogfelder immer auf diese Weise erstellen, oder kann ich für die Dialogfelder Code vorsehen?

Antwort:

Die benutzerdefinierten Dialogfelder unterscheiden sich nicht von den Hauptdialogfeldern, die Sie in den bisherigen Anwendungen eingesetzt haben. Wenn Sie das Verhalten des Dialogfelds interaktiv beeinflussen möchten, können Sie soviel Code hinzufügen wie Sie wollen. Im benutzerdefinierten Dialogfeld der heutigen Beispielanwendung war kein Code zu schreiben, da er einfach nicht erforderlich war. Das Dialogfeld hatte lediglich die Funktion `UpdateData` aufzurufen, was aber die Funktion `OnOK` automatisch beim Schließen des Dialogfelds übernimmt. Da Sie weder die Schaltfläche OK noch die Schaltfläche Abbrechen aus dem Dialogfeld gelöscht haben, ist diese Funktionalität bereits vorhanden.

Frage:

Was passiert, wenn ich zwei oder mehr Schaltflächenkombinationen im selben Aufruf der Funktion `MessageBox` angebe?

Antwort:

Nichts. Die Anwendung läßt sich problemlos kompilieren, aber beim Aufruf der Funktion `MessageBox` ereignet sich überhaupt nichts. Das Meldungsfeld wird nicht geöffnet, und der Benutzer hat gar keine Chance, auf die eigentlich vorgesehenen Fragen zu antworten.

Frage:

Wie kann ich das Dialogfeld Öffnen in meine Anwendung einbauen, so daß es bereits mit einem vorgegebenen Verzeichnis geöffnet wird?

Antwort:

Die Klasse `CFileDialog` verfügt über eine öffentliche (public) Eigenschaft namens `m_ofn`. Diese Eigenschaft ist als Struktur realisiert, die mehrere Attribute des Dialogfelds Öffnen enthält. Dazu gehört auch das anfänglich angezeigte Verzeichnis. Die Struktur ist als `OPENFILENAME`-Struktur in Listing 5.6 definiert.

Listing 5.6: Die Struktur `OPENFILENAME`

```
1: typedef struct tagOFN { // ofn
2:   DWORD lStructSize;
3:   HWND hwndOwner;
4:   HINSTANCE hInstance;
5:   LPCTSTR lpstrFilter;
6:   LPTSTR lpstrCustomFilter;
```

```

7: DWORD nMaxCustFilter;
8: DWORD nFilterIndex;
9: LPTSTR lpstrFile;
10: DWORD nMaxFile;
11: LPTSTR lpstrFileName;
12: DWORD nMaxFileName;
13: LPCTSTR lpstrInitialDir;
14: LPCTSTR lpstrTitle;
15: DWORD Flags;
16: WORD nFileOffset;
17: WORD nFileExtension;
18: LPCTSTR lpstrDefExt;
19: DWORD lCustData;
20: LPOFNHOOKPROC lpfnHook;
21: LPCTSTR lpTemplateName;
22: } OPENFILENAME;

```

Diese Attribute können Sie festlegen, bevor Sie die Methode `DoModal` der Dialogfeldklasse aufrufen, um das Verhalten des Dialogfelds Öffnen zu bestimmen. Wenn Sie zum Beispiel wie in Listing 5.7 das Startverzeichnis auf `C:\Temp` setzen, bevor Sie `DoModal` aufrufen, erscheint das Dialogfeld Öffnen mit diesem ausgewählten Verzeichnis.

Listing 5.7: Die überarbeitete Funktion `OnFileopen`

```

1: void CDialogDlg::OnFileopen()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: CFileDialog m_ldFile(TRUE);
10:
11: // Anfangsverzeichnis initialisieren
12: m_ldFile.m_ofn.lpstrInitialDir = "C:\\Temp\\";
13:
14: // Dialogfeld Öffnen anzeigen und Ergebnis auffangen
15: if (m_ldFile.DoModal() == IDOK)
16: {
17: // Gewählten Dateinamen ermitteln
18: m_sResults = m_ldFile.GetFileName();
19: // Dialogfeld aktualisieren
20: UpdateData(FALSE);
21: }
22:
23: //////////////////////////////////////
24: // EIGENER CODE, ENDE
25: //////////////////////////////////////
26: }

```

Workshop

Kontrollfragen

1. Wie lauten die möglichen Rückgabewerte, die Ihre Anwendung aus einem Aufruf der Funktion `MessageBox` erhält, wenn Sie die Schaltflächenkombination `MB_RETRYCANCEL` übergeben?
2. Welche Standarddialogfelder des Betriebssystems Windows sind als MFC-Klassen definiert?
3. Worin liegt der Unterschied zwischen einem modalen und einem nichtmodalen Dialogfeld?
4. Wie läßt sich statt dem in der heutigen Beispielanwendung eingesetzten Dialogfeld **Öffnen** das Dialogfeld **Speichern unter** anzeigen?
5. Warum war es nicht erforderlich, für das benutzerdefinierte Dialogfeld Funktionen oder Code zu erstellen?

Übungen

1. Modifizieren Sie Ihre Anwendung, damit das Verzeichnis zusammen mit dem Dateinamen erscheint. (Hinweis: Die Funktion `GetFileName` liefert den Pfad und den Dateinamen, die im Dialogfeld **Öffnen** ausgewählt wurden.)
2. Nehmen Sie in das benutzerdefinierte Dialogfeld eine Schaltfläche auf, die den Aufruf der Funktion `MessageBox` mit den Auswahlen **Ja** und **Nein** bewirkt. Übergeben Sie das Ergebnis an das Hauptdialogfeld der Anwendung.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](http://www.markt-und-technik.de).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1**Tag 6****Menüs**

Die meisten Windows-Anwendungen lassen sich über Pulldown-Menüs bedienen. Auf diese Weise kann man dem Benutzer eine umfangreiche Palette von Funktionen bereitstellen, ohne daß dafür Schaltflächen im Fenster vorhanden sein müssen. Der wertvolle Platz auf dem Bildschirm läßt sich damit für andere Dinge freihalten.

Heute lernen Sie, wie man ...

- Menüs für eine Visual C++-Anwendung erstellt,
- ein Menü mit dem Hauptdialogfeld einer Anwendung verbindet,
- Funktionen der Anwendung über ein Menü aufruft,
- ein Kontextmenü erzeugt, das sich durch Drücken der rechten Maustaste öffnen läßt,
- Zugriffstasten einrichtet, um die Menübefehle über die Tastatur aufzurufen.

Menüs

Als die ersten Computer-Terminals eingeführt wurden und sich die Benutzer mit Computer-Software auseinanderzusetzen begannen, stellten sogar die Entwickler von großen Mainframe-Systemen fest, daß man dem Benutzer in irgendeiner Form Menüs anbieten sollte, aus denen sich die vom Computer auszuführenden Funktionen auswählen lassen. Diese frühen Menüs waren im Vergleich mit heutigen Standards primitiv und ließen sich nur schwer bedienen. Seit diesen Anfangstagen haben sich die Menüs enorm verbessert. Außerdem haben sich bestimmte Standards für die Gestaltung und Bedienung durchgesetzt, die das Einarbeiten in neue Programme erleichtern.

Die Softwareentwickler, die den Gedanken der grafischen Benutzeroberflächen (Graphical User Interface - GUI) umsetzen, zielen mit einheitlichen Verhaltensmustern der Elemente einer Anwendung darauf ab, Computersysteme und Anwendungen verständlicher und benutzerfreundlicher zu gestalten. Menüs zur Auswahl der Anwendungsfunktionen bilden dabei einen Teil der grafischen Benutzeroberfläche, der sich leichter erlernen läßt, wenn alle in gleicher Weise arbeiten. Im Ergebnis wurden eine Reihe von Menüstilen entwickelt.

Menüstile

Als erste Menüstile wurden Pulldown- und überlappende Menüs standardisiert. Bei diesen Menüs sind die Kategorien in einer Zeile am oberen Rand des Anwendungsfensters angeordnet. Wenn man eine dieser Kategorien auswählt, öffnet sich ein Menü unter der Kategorie. Hier kann man aus einer Anzahl von Menüeinträgen (oder Befehlen) wählen, die verschiedene Funktionen in der Anwendung auslösen.

Eine Variante dieses Menüstils ist das überlappende Menü, beim dem sich ein weiteres Untermenü rechts neben einem Menüeintrag öffnet. Dieses Untermenü ist dem Pulldown-Menü ähnlich und weist eine Reihe von Einträgen auf, die Anwendungsfunktionen auslösen. Dem Menüentwickler sind keine Grenzen gesetzt, wie tief er die überlappenden Menüs verschachtelt. Allerdings wurde schnell klar, daß mehr als zwei überlappende Menüebenen unhandlich sind.

Schließlich wurde ein dritter Menüstil entwickelt, das sogenannte Popup- oder Kontextmenü - ein Menü, das sich mitten in der Anwendung öffnet und frei über den gesamten Arbeitsbereich der Anwendung verschiebbar ist. Die Bezeichnung Kontextmenü rührt daher, weil das spezielle Menü vom markierten Objekt oder Arbeitsbereich abhängig ist, wo sich der Cursor oder Mauszeiger gerade befindet.

Tastenkombinationen - Menüauswahlen aktivieren

Wenn ein Benutzer in einer Anwendung vorrangig mit der Tastatur arbeitet, beispielsweise bei einer Textverarbeitung, ist eine sinkende Produktivität festzustellen, wenn er die Hände von der Tastatur nehmen muß, um einen Menübefehl mit der Maus auszuwählen. Aus diesem Grund sehen die Softwareentwickler bestimmte Tastenkombinationen für verschiedene Menübefehle vor (insbesondere für die am häufigsten genutzten Menüauswahlen). Bei diesen speziellen Tasten und Tastenkombinationen spricht man von Schnelltasten (accelerator keys), Zugriffstasten und Hotkeys.

Die Hotkeys werden in einem Menübefehl durch unterstrichene Buchstaben gekennzeichnet. Wenn man die (Alt)-Taste zusammen mit der Taste des unterstrichenen Buchstabens drückt, wählt man den Menüeintrag aus, dem dieser Buchstabe zugeordnet ist. Auf diese Weise kann man durch die Anwendungsmenüs navigieren, ohne die Hände von der Tastatur nehmen zu müssen.

Für erfahrene Benutzer haben die Anwendungsentwickler Schnelltasten oder Zugriffstasten vorgesehen. Eine Zugriffstaste ist eine einzelne Tastenkombination, über die man eine Anwendungsfunktion direkt auslösen kann, statt sich durch die Anwendungsmenüs zu arbeiten. Erfahrene Benutzer können auf diese Weise zügiger arbeiten, weil sie häufig genutzte Anwendungsfunktionen nicht über den Umweg einer Menüauswahl aufrufen müssen. Damit sich der Benutzer die Zugriffstasten praktisch im Vorübergehen einprägen kann, sind die Tastenkombinationen rechts neben dem betreffenden Menüeintrag aufgeführt.

Standards und Konventionen für Menüs

Es gibt zwar keine eigentlichen Standards, wie Menüs zu entwerfen sind, es haben sich aber eine Reihe von Konventionen durchgesetzt, die sich auf den Entwurf und die Organisation von Menüs beziehen. Diese Konventionen hat Microsoft in *Windows Interface Guidelines for Software Design* für Entwickler von Windows-Anwendungen veröffentlicht. Die Absicht dieser Veröffentlichung ist es, die Entwicklung von einheitlichem Anwendungsverhalten zu erleichtern, um eines der Hauptziele grafischer Benutzeroberflächen zu erreichen. Die Konventionen lauten wie folgt:

- In der obersten Menüleiste sind die Kategorien mit einzelnen Wörtern zu bezeichnen. Eine aus

zwei Wörtern bestehende Kategorie kann der Benutzer leicht für zwei Einzelwortkategorien halten.

- Das Menü Datei ist als erstes Menü von links anzuordnen. Es enthält alle dateiorientierten Funktionen (wie Neu, Öffnen, Speichern, Drucken usw.) sowie die Funktion Beenden. Der Menübefehl Beenden ist am unteren Rand des Menüs zu plazieren und von den übrigen Menüeinträgen durch eine Trennlinie abzusetzen.
- An das Menü Datei schließt sich das Menü Bearbeiten an. Dieses Menü enthält alle Funktionen zur Bearbeitung wie Kopieren, Ausschneiden, Einfügen, Rückgängig, Wiederholen usw.
- Im Menü Ansicht sind die Menüeinträge vorzusehen, mit denen sich das Aussehen des Arbeitsbereichs einer Anwendung steuern läßt.
- Das Menü Fenster kommt in Anwendungen mit mehreren Dokumenten (MDI-Anwendungen) zum Einsatz. Es enthält Befehle, mit denen sich untergeordnete Fenster steuern, das aktuelle Fenster auswählen und das Layout ändern lassen. Dieses Menü ist als vorletztes Menü (d.h. zweites von rechts) in der Menüleiste zu plazieren.
- Das Menü Hilfe ist das letzte (ganz rechts befindliche) Menü in der Menüleiste. Es enthält Menübefehle, um Anweisungen oder Informationen zur Anwendung aufrufen zu können. Wenn die Anwendung einen Copyright-Vermerk oder Unternehmensinformationen anzeigen muß, sollte man sie als letzten Eintrag in diesem Menü unter der Bezeichnung Info über <Anwendungsname> unterbringen.

Menüs entwerfen

Menüs sind in Visual C++-Anwendungen als Ressource definiert. Daher kann man sie im Editor von Visual C++ über die Registerkarte Ressourcen im Arbeitsbereich entwerfen. Wenn Sie eine dialogbasierte Anwendung neu erstellen, ist im Ressourcenzweig noch kein Menüordner vorhanden, was Sie aber ändern können.



Verschiedene Aspekte von Windows-Anwendungen liegen in Form von Ressourcen vor. Dazu gehören das Fensterlayout, Menüs, Symbolleisten, Bilder, Strings, Zugriffstasten usw. Alle diese Merkmale sind in einer sogenannten Ressourcendatei organisiert, auf die der Visual C++-Compiler zurückgreift, um diese Objekte aus deren Definitionen zu erzeugen. Die Ressourcendatei ist eine Textdatei mit der Dateierweiterung `.rc`. Sie enthält eine textuelle Beschreibung aller Objekte, einschließlich der IDs, Beschriftungen, Abmessungen usw.

Manche Ressourcen wie etwa Bilder und Klänge lassen sich nicht in Textform beschreiben, sondern müssen in einem Binärformat gespeichert werden. Diese Ressourcen sind in separaten Dateien abgelegt, wobei der Dateiname und der Standort in die Ressourcendatei eingebunden werden.

Ein Menü erstellen

Ein Menü läßt sich problemlos erstellen. Dazu sind folgende Schritte auszuführen:

1. Die Anwendung erstellen, die den Rahmen für das Menü bildet.
2. Dem Projekt eine Menüressource hinzufügen.

3. Die Menüressource anpassen, um die jeweiligen Menübefehle für die Anwendung einzubinden.
4. Das Menü mit Funktionalität ausstatten, indem man entsprechende Routinen mit den Menübefehlen verbindet.

Die Anwendung erstellen

Für die Beispielanwendung in diesem Kapitel erzeugen Sie eine einfache dialogbasierte Anwendung, die eine einzelne Schaltfläche und ein Menü enthält. Um die Anwendung zu erstellen, führen Sie die folgenden Schritte aus:

1. Erstellen Sie mit dem MFC-Anwendungs-Assistenten eine neue Anwendung. Nennen Sie das Projekt **Menus**.
2. Übernehmen Sie die Standardeinstellungen des Anwendungs-Assistenten in allen Dialogfeldern. Als Titel des Dialogfelds geben Sie **Menus** ein.
3. Wenn der Anwendungs-Assistent das Gerüst erstellt hat, löschen Sie alle Steuerelemente aus dem Dialogfeld.
4. Nehmen Sie in das Dialogfeld eine einzelne Schaltfläche auf. Als Namen legen Sie für diese Schaltfläche **IDC_EXIT** fest und geben als Beschriftung **&Beenden** ein.
5. Fügen Sie mit dem Klassen-Assistenten für diese Schaltfläche eine Funktion hinzu. Schreiben Sie in diese Funktion den Code, der die Funktion **OnOK** aufruft. Wie Sie wissen, bewirkt die Funktion **OnOK** das Schließen der Anwendung.



Wenn Sie nicht mehr genau wissen, wie Sie die Funktion **OnOK** hinzufügen, sehen Sie sich am besten noch einmal das Beispiel im Abschnitt »Die Anwendung schließen« von Tag 2 an.

Ein Menü hinzufügen und anpassen

Nachdem Sie die zugrundeliegende Anwendung erstellt haben, können Sie für die Anwendung ein neues Menü erzeugen. Fügen Sie zuerst eine Menüressource in das Projekt ein. Dabei ruft Visual C++ automatisch den Menü-Editor auf, der Ihnen die Anpassung des Menüs erlaubt. Die folgenden Schritte zeigen, wie Sie ein Menü hinzufügen und anpassen:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**.
2. Markieren Sie den Ordner der Projektressourcen am Beginn der Strukturansicht, in unserem Beispiel **Menus Ressourcen**.
3. Klicken Sie mit der rechten Maustaste, um das Kontextmenü zu öffnen.
4. Wählen Sie aus dem Kontextmenü den Befehl **Einfügen**.
5. Im Dialogfeld **Ressource einfügen** markieren Sie **Menu** in der Liste **Ressourcentypen**, wie es Abbildung 6.1 zeigt. Klicken Sie auf die Schaltfläche **Neu**.

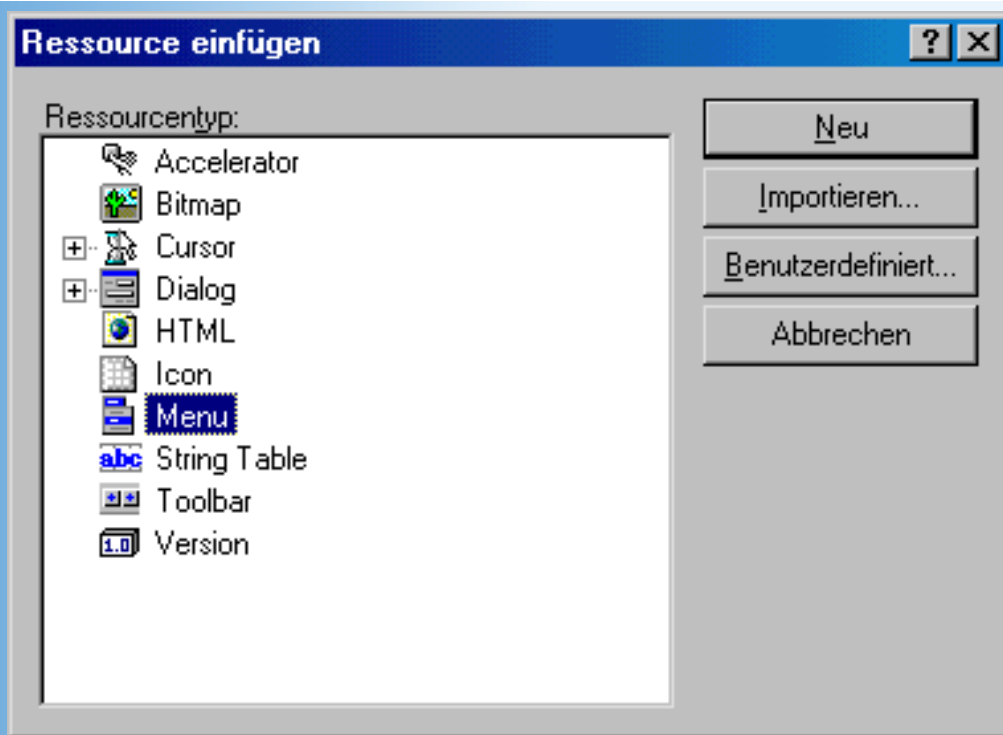


Abbildung 6.1:
Das Dialogfeld Ressource einfügen

6. Der Menü-Editor öffnet den Bearbeitungsbereich des Visual Studios. Der erste Menübefehl ist hervorgehoben (siehe Abbildung 6.2).

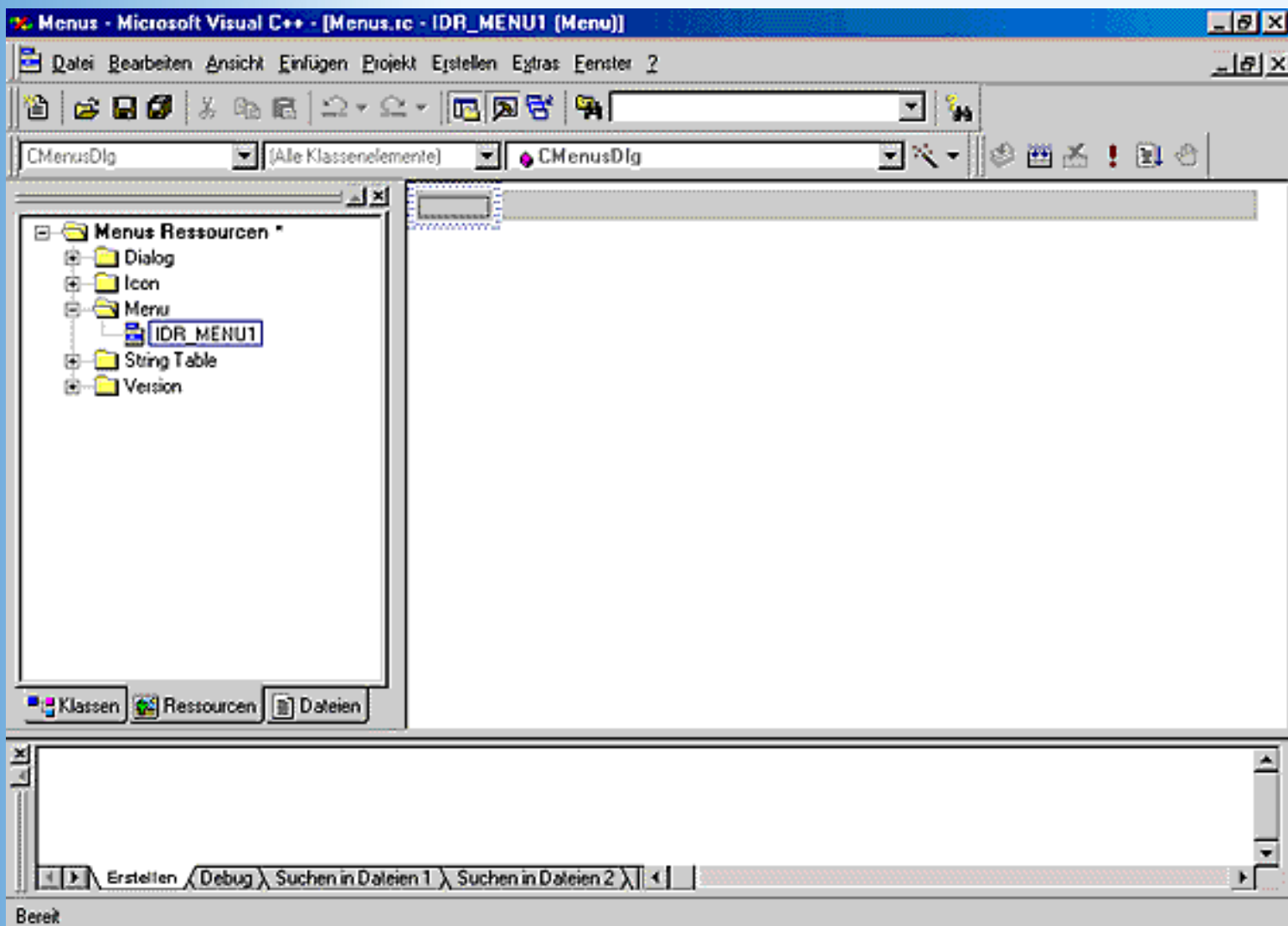


Abbildung 6.2: Ein leeres Menü

Die Menüressource ist nun angelegt, und Sie können sie mit Menübefehlen beleben. Einen Menübefehl fügen Sie in den folgenden Schritten hinzu:

1. Klicken Sie mit der rechten Maustaste auf den hervorgehobenen Bereich, und wählen Sie aus dem Kontextmenü den Befehl **Eigenschaften**.
2. Geben Sie den Titel des Menüs ein, im Beispiel `&Datei`, und schließen Sie dann das Dialogfeld **Eigenschaften**.



Im Dialogfeld **Menübefehl Eigenschaften** legen Sie den Text fest, den der Benutzer in der Menüleiste bei laufender Anwendung sieht. Da das Kontrollkästchen **Popup** eingeschaltet ist (per Vorgabe bei allen Menüeinträgen auf der obersten Ebene der Menüleiste), löst dieses Menüelement keine Funktionen in der Anwendung aus, so daß man auch keine Objekt-ID zuweisen muß.

3. Die erste Position des Dropdown-Menüs ist hervorgehoben. Um den entsprechenden Menübefehl hinzuzufügen, klicken Sie wieder mit der rechten Maustaste auf den hervorgehobenen Bereich und wählen aus dem Kontextmenü den Befehl **Eigenschaften**.
4. Geben Sie eine ID und einen Titel für den Menübefehl ein. Für das Beispiel wählen Sie `IDM_FILE_HELLO` als ID und `&Hello` als Titel. Schließen Sie das Dialogfeld.



Dieses Mal legen Sie im Dialogfeld **Menübefehl Eigenschaften** nicht nur den Text fest, den der Benutzer sieht, wenn er das Menü über die Menüleiste öffnet, sondern auch die Objekt-ID, über die die Zuordnung der Behandlungsroutinen zu den Menüereignissen erfolgt.

Hinweis zur deutschen Übersetzung

Als Objekt-ID können Sie beliebige Bezeichnungen eintragen - zum Beispiel auch `IDM_DATEI_HELLO` statt `IDM_FILE_HELLO`. Wenn Sie Funktionen hinzufügen, die sich auf die Objekt-IDs beziehen, greift Visual C++ auf diese Bezeichnungen zurück, um einen Namen für die Funktion vorzugeben. Bei deutschen Bezeichnern, die keine Umlaute enthalten, läuft alles problemlos ab. Wenn aber wie beim Menübefehl Öffnen Umlaute im Bezeichner enthalten sind, treten Konflikte bei der automatischen Vergabe der Funktionsnamen auf. Da die Standardbefehle für Menüs (wie `IDM_ABOUTBOX`) ohnehin in Englisch vorliegen, verwenden wir in diesem Buch für Objekt-IDs die englischen Bezeichnungen. Auf die Titel von Menübefehlen wirken sich die Umlaute dagegen nicht nachteilig aus und können sogar als Zugriffstasten spezifiziert werden.

Momentan verfügen Sie über ein Menü mit einem einzigen Menübefehl. Sie können nun weitere Menübefehle hinzufügen, indem Sie die obigen Schritte 3 und 4 für jeden hervorgehobenen Bereich wiederholen. In das Menü lassen sich auch Trennlinien einfügen. Es handelt sich dabei um eine waagerechte Linie, mit der man zwei funktionell unterschiedliche Bereiche von Menüauswahlen abgrenzt. Eine Trennlinie fügen Sie folgendermaßen hinzu:

1. Markieren Sie den Menübefehl, wo Sie die Trennlinie platzieren möchten. Im Beispiel sollte die

zweite Menüposition im Dropdown-Menü hervorgehoben sein. Öffnen Sie das Dialogfeld **Menübefehl Eigenschaften** wie im Schritt 3 der obigen Folge. Schalten Sie nun einfach das Kontrollkästchen **Trennlinie** ein, wie es Abbildung 6.3 zeigt, und schließen Sie das Dialogfeld.

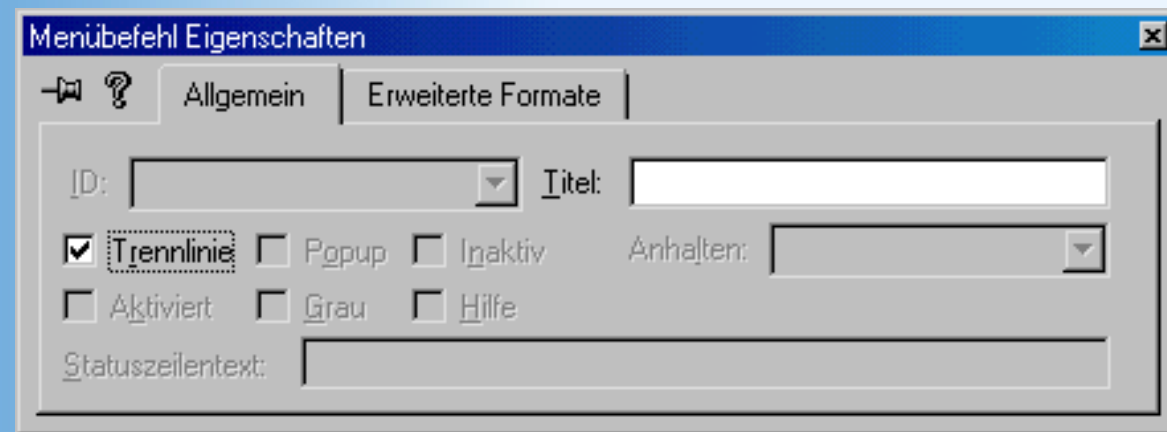


Abbildung 6.3:
Eine Trennlinie festlegen

Um das Beispielprogramm zu vervollständigen, nehmen Sie nach den gleichen Schritten wie oben beschrieben einen Menübefehl Beenden in das Menü Datei auf sowie ein zweites Menü namens Hilfe mit einem Menübefehl Info. Die folgenden Schritte, die der obigen Schrittfolge ähnlich sind, zeigen im Detail das Hinzufügen der zusätzlichen Elemente:

1. Öffnen Sie das Dialogfeld **Menübefehl Eigenschaften** für die dritte Position im Dropdown-Menü, und legen Sie die ID als `IDM_FILE_EXIT` (oder `IDM_DATEI_BEENDEN`, wenn Ihnen das lieber ist) und die Beschriftung mit **&Beenden** fest. Schließen Sie das Dialogfeld.
2. Markieren Sie die zweite Position in der Menüleiste, und öffnen Sie das Dialogfeld **Menübefehl Eigenschaften**. Legen Sie den Titel mit **&Hilfe** fest, und schließen Sie das Dialogfeld.
3. Öffnen Sie das Dialogfeld **Menübefehl Eigenschaften** für die erste Position im Dropdown-Menü des zweiten Eintrags in der Menüleiste. Legen Sie die ID mit `ID_HELP_ABOUT` und den Titel mit **&Info** fest. Schließen Sie das Dialogfeld.

Das Menü ist damit erstellt. Allerdings fehlt noch die Verbindung zur Anwendung.

Das Menü mit dem Dialogfeld verbinden

Sie verfügen nun über ein Menü, das Sie in Ihrer Anwendung einsetzen können. Wenn Sie aber die Anwendung in der jetzigen Entwicklungsphase kompilieren und ausführen, erscheint das Menü nicht. Das Menü ist noch mit dem Dialogfeld zu verbinden. Das erreichen Sie in folgenden Schritten:

1. Öffnen Sie den Dialog-Editor, indem Sie auf das Hauptdialogfeld der Anwendung im Ordner **Dialog** des Arbeitsbereichs doppelklicken. Im Beispiel doppelklicken Sie auf `IDD_MENUS_DIALOG`.
2. Markieren Sie das gesamte Dialogfeld. Achten Sie darauf, daß keine Steuerelemente markiert sind, und öffnen Sie das Dialogfeld **Dialogfeld Eigenschaften**. (Sie öffnen hier die Eigenschaften für das Dialogfeld an sich und nicht für irgendein Steuerelement des Dialogfelds.)
3. Wählen Sie aus der Dropdown-Liste **Menü** das eben erstellte Menü aus, wie es Abbildung 6.4 zeigt.

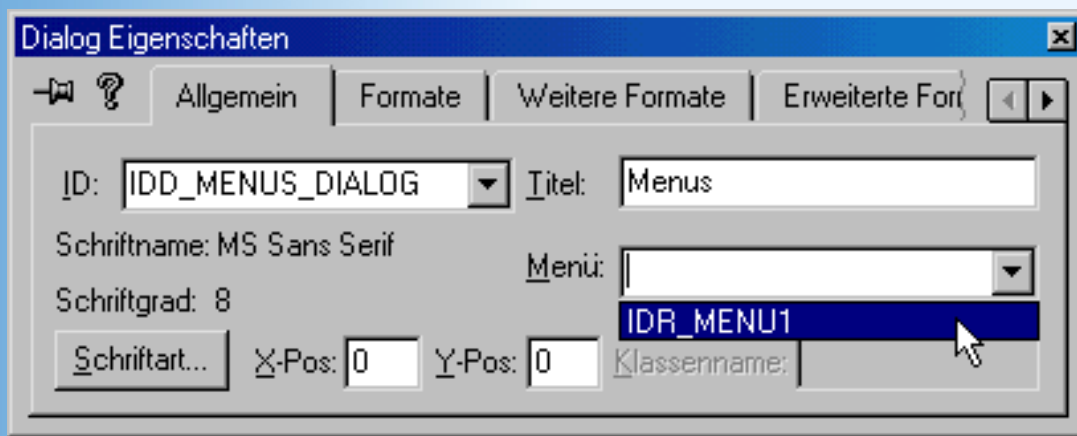


Abbildung 6.4:
Das Menü mit dem Dialogfeld verbinden

Wenn Sie die Anwendung kompilieren und ausführen, ist das Menü mit dem Dialogfeld der Anwendung verbunden, wie es aus Abbildung 6.5 hervorgeht. Wie bei jeder anderen Windows-Anwendung können Sie Menübefehle auswählen. Allerdings gibt es einen kleinen Unterschied. Wenn Sie momentan einen der Menübefehle wählen, passiert überhaupt nichts. Sie müssen noch die Funktionalität für die Menübefehle realisieren.

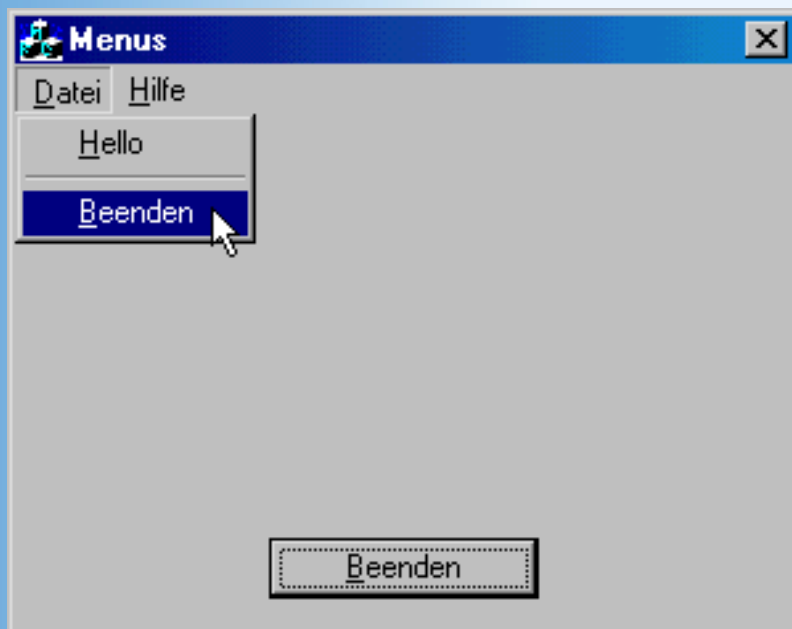


Abbildung 6.5:
Das Menü ist nun Teil des Anwendungsdialogfelds

Menübefehle mit Funktionalität ausstatten

Nunmehr verfügen Sie über ein Menü als Teil der Anwendung. Es wäre nun angebracht, wenn man tatsächlich etwas Sinnvolles damit anstellen könnte. Bevor Ihr Menü irgendeine Aktion bewirken kann, müssen Sie - genau wie überall in Ihren Visual C++-Anwendungen - festlegen, was zu tun ist. Um das Menü der Beispielanwendung mit etwas Funktionalität auszustatten, führen Sie die folgenden Schritte aus:

1. Öffnen Sie den Menü-Editor für Ihr Menü.
2. Öffnen Sie den Klassen-Assistenten über das Menü **Ansicht**.
3. Es erscheint das Dialogfeld **Hinzufügen einer Klasse** (analog zum gestrigen Tag, als Sie ein

zweites Dialogfeld aufgenommen hatten). Behalten Sie die Option *Vorhandene Klasse auswählen* bei, und klicken Sie auf **OK** (siehe Abbildung 6.6).

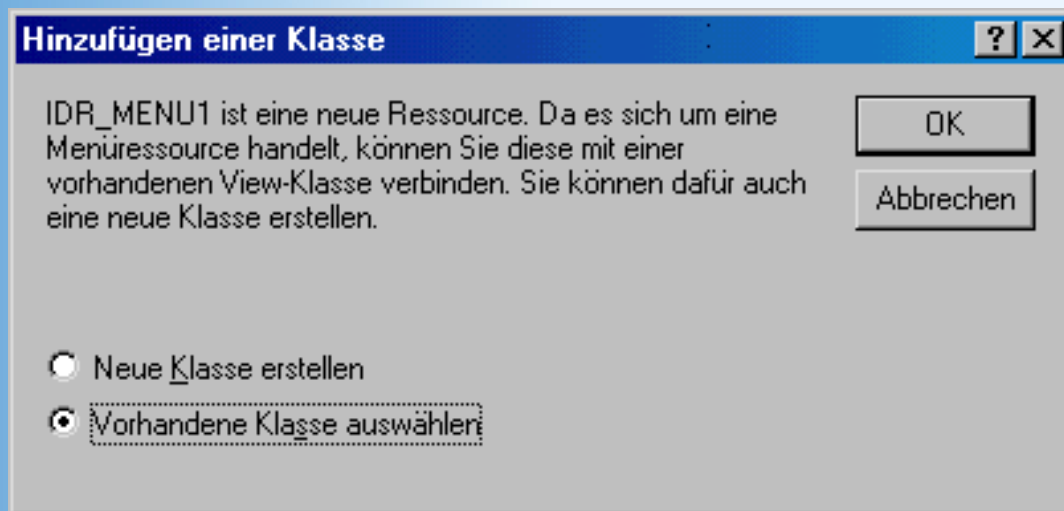


Abbildung 6.6:
Das Dialogfeld Hinzufügen einer Klasse

Als Sie gestern ein zweites Dialogfeld in Ihre Anwendung hinzugefügt haben, mußten Sie eine neue C++-Klasse für dieses Dialogfeld erstellen. Das Menü in der heutigen Lektion verbinden Sie mit einer vorhandenen C++-Klasse für das Dialogfeld, dem das Menü zugeordnet ist.

4. Markieren Sie die C++-Klasse des Hauptdialogfelds in der Klassenliste des Dialogfelds **Klasse auswählen**. Im Beispiel markieren Sie `CMenuDlg`, wie es Abbildung 6.7 zeigt. Damit teilen Sie Visual C++ mit, daß die gesamte Funktionalität, die Sie von den verschiedenen Menübefehlen aufrufen, zur selben Dialogfeldklasse des Fensters, mit dem das Menü verbunden ist, gehört.

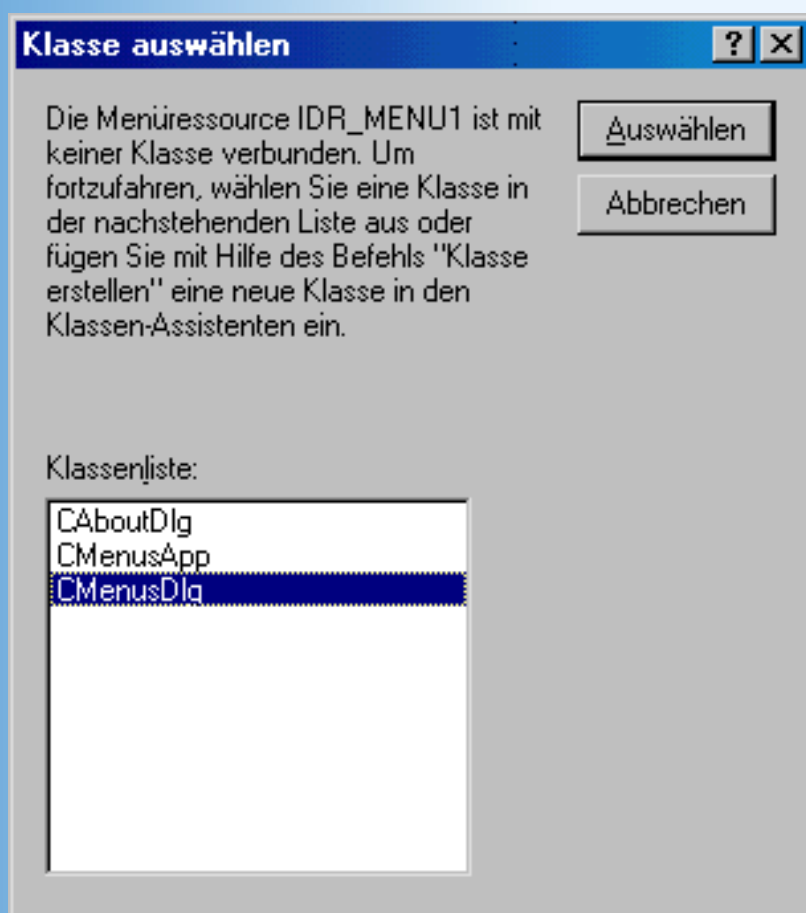


Abbildung 6.7: Das Dialogfeld Klasse auswählen

Für die Menüelemente, mit denen Sie Funktionen in Ihrer Anwendung auslösen möchten, können Sie über den Klassen-Assistenten Behandlungsfunktionen hinzufügen, wie Sie es von den Steuerelementen her kennen, die Sie im Dialogfeld platziert haben.

In die Beispielanwendung nehmen Sie eine Funktion für das Objekt `IDM_FILE_HELLO` (den Menübefehl Hello) für die Nachricht `COMMAND` auf. Nennen Sie die Funktion `OnHello`, und schreiben Sie den Code aus Listing 6.1 in diese Funktion.

Listing 6.1: Die Funktion `OnFileHello`

```
1: void CMenuDlg::OnFileHello()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Meldung anzeigen
10: MessageBox("Hello Leute", "Hello");
11:
12: ///////////////////////////////////
13: // EIGENER CODE, ENDE
14: ///////////////////////////////////
15: }
```



Die Nachricht `COMMAND` erhält das Anwendungsfenster, wenn ein Menübefehl ausgewählt wird. Wenn man eine Funktion für diese Nachricht vorsieht, hat das die gleiche Wirkung, wie eine Funktion zur Auswahl des Menübefehls.

Bereits vorhandene Behandlungsroutinen können Sie von Menübefehlen aufrufen, indem Sie die vorhandene Funktion zum Menüereignis `COMMAND` hinzufügen. Zu diesem Zweck können Sie eine Funktion für die Objekt-ID des Menüs hinzufügen und den Namen der vorhandenen Funktion spezifizieren, statt den vorgeschlagenen Funktionsnamen zu übernehmen.

Um die Funktion `OnExit` für den Menübefehl Beenden erneut zu verwenden, öffnen Sie wieder den Menü-Editor und anschließend den Klassen-Assistenten. Fügen Sie über den Klassen-Assistenten eine Funktion für das Objekt `IDM_FILE_EXIT` für die Nachricht `COMMAND` hinzu. Übernehmen Sie diesmal nicht den vom Klassen-Assistenten vorgeschlagenen Funktionsnamen, sondern geben Sie den Funktionsnamen `OnExit` ein. Damit verbinden Sie automatisch die vorhandene Funktion `OnExit`, die Sie bereits weiter oben für die Schaltfläche Beenden erstellt haben, mit dem Menübefehl Beenden.

Um die Funktionalität des Beispiels abzurunden, fügen Sie noch eine Funktion für das Objekt `ID_HELP_ABOUT` für die Nachricht `COMMAND` hinzu. Schreiben Sie in diese Funktion den Code aus Listing 6.2.

Listing 6.2: Die Funktion `OnHelpAbout`

```
1: void CMenuDlg::OnHelpAbout()
```

```

2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Instanz des Info-Fensters deklarieren
10: CAboutDlg dlgAbout;
11:
12: // Info-Fenster anzeigen
13: dlgAbout.DoModal();
14:
15: //////////////////////////////////
16: // EIGENER CODE, ENDE
17: //////////////////////////////////
18: }

```

Den Menübefehl **Datei / Beenden** haben Sie mit einer vorhandenen Funktion verbunden, die Ihre Anwendung schließt. Für **Datei / Hello** haben Sie eine neue Funktion hinzugefügt, die die Funktion `MessageBox` aufruft, um eine einfache Meldung anzuzeigen. Beim Menü **Hilfe / Info** ist eine weitere Funktion hinzugekommen, die eine Instanz des Dialogfelds Info deklariert und die Methode `DoModal` aufruft.

Wenn Sie die Anwendung kompilieren und ausführen, können Sie sich davon überzeugen, daß alle Menüeinträge funktionieren. Wenn Sie **Hilfe / Info** wählen, wie es Abbildung 6.8 zeigt, erscheint das Dialogfeld Info (siehe Abbildung 6.9). Wählen Sie **Datei / Hello**, kommt die Meldung `Hello Leute` (siehe Abbildung 6.10) auf den Bildschirm. Und mit **Datei / Beenden** läßt sich die Anwendung schließen.

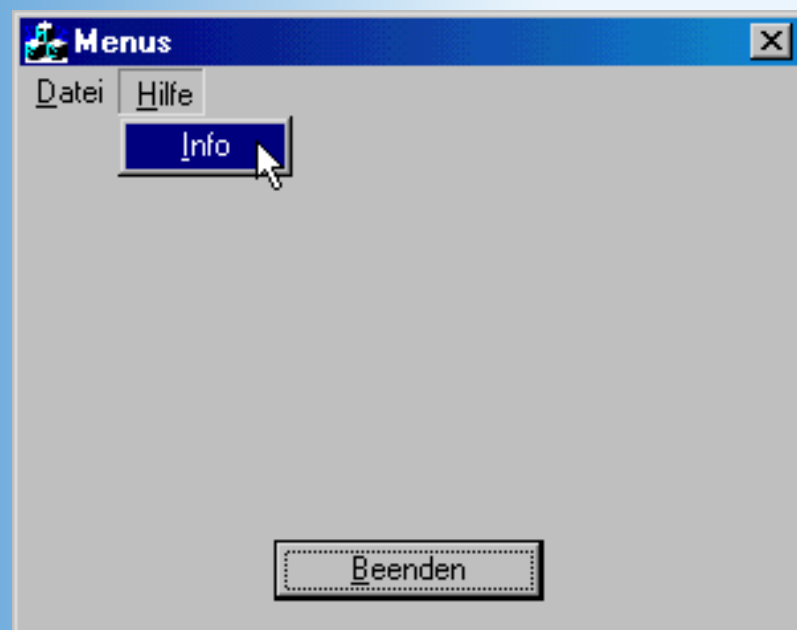


Abbildung 6.8:
Der Menüeintrag Hilfe / Info

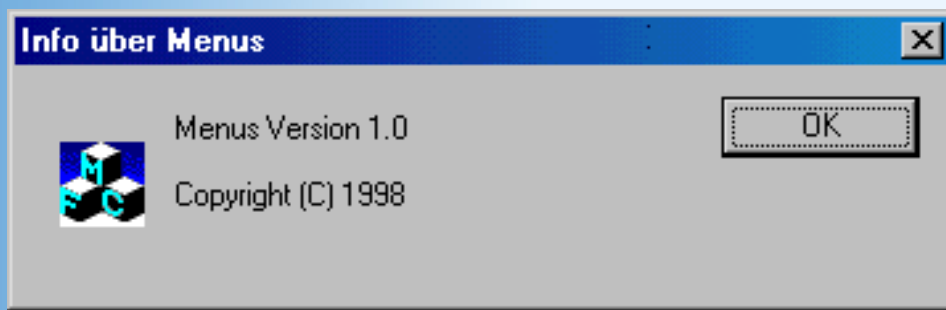


Abbildung 6.9:
Das Dialogfeld Info



Abbildung 6.10:
Das Meldungsfeld Hello Leute

Kontextmenüs erstellen

Die meisten Windows-Anwendungen verfügen über sogenannte Popup- oder Kontextmenüs. Diese ruft man auf, indem man mit der rechten Maustaste auf ein Objekt klickt. Die Bezeichnung Popup (etwa: aufspringen) rührt daher, daß diese Menüs mitten im Anwendungsbereich erscheinen und nicht an eine Menüleiste, den Fensterrahmen oder etwas anderes auf dem Bildschirm (ausgenommen den Mauszeiger) gebunden sind. Man spricht auch von Kontextmenüs, weil der Inhalt dieses Menüs vom Kontext abhängt, in dem es geöffnet wird. Die verfügbaren Befehle im Kontextmenü hängen von der momentanen Auswahl in der Anwendung oder der aktuellen Position des Mauszeigers auf einem bestimmten Objekt ab.

Um ein Kontextmenü in der Anwendung bereitzustellen, gibt es zwei Lösungsansätze. Entweder erstellen Sie ein Menü, das speziell als Kontextmenü vorgesehen ist, oder Sie verwenden eines der Pulldown-Menüs aus dem Hauptmenü, das Sie bereits erstellt haben. Wenn Sie ein Menü speziell als Kontextmenü entwerfen, überspringen Sie die oberste Ebene der Menüleiste, indem Sie ein Leerzeichen oder anderen Text, der nicht auf dem Bildschirm erscheint, als Titel eingeben. Wie das funktioniert, erfahren Sie im Abschnitt »Ein Kontextmenü hinzufügen« von Tag 11, wenn Sie ein benutzerdefiniertes Menü speziell als Kontextmenü erstellen.

Jeder Menübefehl eines Dropdown-Menüs läßt sich seinerseits als Kontextmenü verwenden. Zu diesem Zweck müssen Sie einen Handle auf das Untermenü (das Dropdown-Menü) ermitteln und dann die Funktion `TrackPopupMenu` auf dem Untermenü aufrufen. Die übrige Funktionalität des Kontextmenüs wurde bereits behandelt, als Sie die anderen Menüs erstellt und kodiert haben. Um ein Kontextmenü in die Beispielanwendung einzubauen, führen Sie die folgenden Schritte aus:

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `WM_CONTEXTMENU` in das Dialogfeld der Anwendung ein.



Ein Kontextmenü kann man über zwei verschiedene Nachrichten für Behandlungsfunktionen auslösen. Es liegt zunächst auf der Hand, die Nachricht `WM_RBUTTONDOWN` zu verwenden. Windows sendet diese Nachricht, wenn der Benutzer mit der rechten Maustaste klickt. Weiterhin kann man (und sollte man) die Nachricht `WM_CONTEXTMENU` verwenden, die speziell dafür vorgesehen ist, ein Kontextmenü zu initiieren. Dieses Ereignis entsteht bei einer Reihe von Benutzeraktionen: eine davon ist das Loslassen der rechten Maustaste, eine andere das Drücken der Kontextmenütaste auf einer der neueren Windows-Tastaturen.

2. Nehmen Sie in die Funktion den Code aus Listing 6.3 auf.

Listing 6.3: Die Funktion `OnContextMenu`

```
1: void CMenuDlg::OnContextMenu(CWnd* pWnd, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier
   einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Lokale Variablen deklarieren
10: CMenu *m_lMenu; // Zeiger auf Menü
11: CPoint m_pPoint; // Kopie der Mausposition
12:
13: // Mausposition in lokale Variable kopieren
14: m_pPoint = point;
15: // Position in Bildschirmkoordinaten konvertieren
16: ClientToScreen(&m_pPoint);
17: // Zeiger auf Fenstermenü holen
18: m_lMenu = GetMenu();
19: // Zeiger auf erstes Untermenü holen
20: m_lMenu = m_lMenu->GetSubMenu(0);
21: // Popup-Menü anzeigen
22: m_lMenu->TrackPopupMenu(TPM_CENTERALIGN + TPM_LEFTBUTTON,
23: m_pPoint.x, m_pPoint.y, this, NULL);
24:
25: //////////////////////////////////////
26: // EIGENER CODE, ENDE
27: //////////////////////////////////////
28: }
```

Die Funktion in Listing 6.3 erstellt zunächst eine Kopie der Mausposition. Es handelt sich dabei um eine relative Position im Fensterbereich. Dieser Wert ist in eine absolute Position bezüglich des gesamten Bildschirmbereichs für die Anzeige des Kontextmenüs umzuwandeln. Wenn Sie die Koordinaten des Mauszeigers nicht umwandeln, lässt sich nicht vorhersagen, wo das Kontextmenü erscheint.

Nachdem Sie die absolute Position berechnet haben, ermitteln Sie einen Zeiger auf das Fenstermenü.

Dieser Zeiger sollte immer ein lokaler Zeiger innerhalb der Funktion, wo Sie ihn verwenden, sein, da sich der Ort des Menüs in der laufenden Anwendung ändern kann. Über den Menüzeiger ermitteln Sie als nächstes einen Zeiger auf das erste Dropdown-Menü (die Numerierung der Untermenüs beginnt mit 0, wie bei allen anderen Elementen in C/C++). Nachdem Sie einen Zeiger auf das Untermenü besitzen, können Sie ihn als reguläre Klasseninstanz von `CMenu` behandeln.

Das letzte Teil in diesem Puzzle ist der Aufruf der Elementfunktion `TrackPopupMenu` der Klasse `CMenu`. Diese Funktion übernimmt fünf Argumente und bestimmt damit, wo und wie das Kontextmenü anzuzeigen ist. Das erste Argument ist eine Kombination von zwei Flags. Das erste Flag, `TPM_CENTERALIGN`, zentriert das Kontextmenü bezüglich des Mauszeigers. Statt dessen können Sie auch `TPM_LEFTALIGN` oder `TPM_RIGHTALIGN` verwenden. Diese Flags richten den linken bzw. rechten Rand des Kontextmenüs mit der Mausposition aus. Der zweite Teil dieser Flag-Kombination ist `TPM_LEFTBUTTON`. Dieses Flag bewirkt, daß das Kontextmenü über das Drücken der linken Maustaste aufgerufen wird. Mit `TPM_RIGHTBUTTON` läßt sich das Kontextmenü mit der rechten Maustaste aktivieren.

Das zweite und dritte Argument der Funktion `TrackPopupMenu` legen die Bildschirmposition - nicht die relative Position im Fensterbereich - für das Kontextmenü fest. Das vierte Argument ist ein Zeiger auf das Fenster, das die Nachrichten der Menübefehle erhält. Das letzte Argument beschreibt ein Rechteck, in das der Benutzer klicken kann, ohne das Kontextmenü zu schließen. Übergibt man hier `NULL`, wird das Kontextmenü geschlossen, wenn der Benutzer außerhalb des Kontextmenüs klickt. Mit diesem Code können Sie in Ihrer Anwendung ein Kontextmenü gemäß Abbildung 6.11 realisieren.

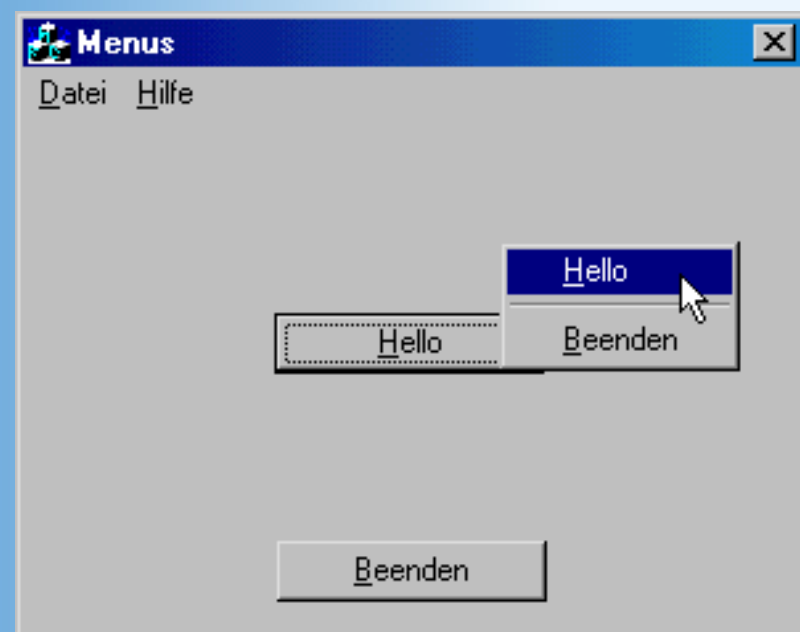


Abbildung 6.11:
Das Kontextmenü in der Praxis

Ein Menü mit Zugriffstasten

Zu den bereits von Anfang an vorhandenen Schnelltagen für die Auswahl von Menübefehlen gehören die Zugriffstasten. Wie bereits weiter vorn in diesem Kapitel erwähnt, sind Zugriffstasten spezielle Tastenkombinationen - gewöhnlich eine Kombination der (Strg)-Taste mit einer anderen Taste - oder Funktionstasten, die innerhalb der gesamten Anwendung eindeutig sind. Jede dieser Tastenkombinationen löst eine bestimmte Menüfunktion aus.

Die Funktionsweise der Zugriffstasten ist mit den Menüs vergleichbar. Es handelt sich ebenfalls um eine Anwendungsressource, die in einer Tabelle auf der Registerkarte Ressourcen des Arbeitsbereichs definiert ist. Jeder Tabelleneintrag verfügt über eine Objekt-ID und den Code für eine Tastenkombination. Nachdem Sie die Zugriffstasten definiert haben, können Sie den Objekt-IDs Funktionalität zuordnen. Einer Zugriffstaste läßt sich auch dieselbe Objekt-ID wie dem korrespondierenden Menüeintrag zuweisen, so daß in der Nachrichtenzuordnungstabelle der Anwendung nur ein Eintrag zu definieren ist.

Nachdem Sie alle Zugriffstasten definiert haben, können Sie die Tastenkombination im Menübefehl angeben, damit der Benutzer über die jeweilige Tastenkombination informiert ist. Hängen Sie ein `\t` am Ende der Beschriftung eines Menübefehls an, und geben Sie anschließend die Tastenkombination an. Die Zeichenfolge `\t` erscheint in der Menüanzeige als Tabulator, der die Beschriftung des Menüs von der Tastenkombination absetzt.

Leider funktionieren Zugriffstasten nicht bei auf Dialogfeldern basierenden Fenstern, so daß sie in der heutigen Anwendung außen vor bleiben müssen. Wie man Zugriffstasten mit Menüs verbindet, lernen Sie in wenigen Tagen bei der Behandlung von SDI- und MDI-Anwendungen.

Zusammenfassung

Schwerpunkt des heutigen Tages waren Menüs in Visual C++-Anwendungen. Sie haben gelernt, wie man mit den Werkzeugen von Visual C++ ein Menü für eine Anwendung erstellt und dann das Menü mit einem Fenster der Anwendung verbindet. Anschließend wurde gezeigt, wie man den verschiedenen Menübefehlen Funktionalität zuordnet. Gegen Ende der heutigen Lektion haben Sie erfahren, wie man einen Teil des Menüs als Popup- oder Kontextmenü verwenden kann. Schließlich wurde erwähnt, wie sich Zugriffstasten in die meisten Anwendungen einbinden lassen.

Fragen und Antworten

Frage:

Muß ich meine Menüs in der gleichen Weise benennen, wie es bei anderen Anwendungen üblich ist? Zum Beispiel verfügen viele Anwendungen über die Menüs Datei und Hilfe. Kann ich meine Menüs irgendwie anders bezeichnen?

Antwort:

Die Menübefehle in der Menüleiste können Sie beliebig benennen. Allerdings gibt es allgemein anerkannte Konventionen, die alle dateiorientierten Funktionen unter einem Menü Datei zusammenfassen und alle hilfebezogenen Funktionen unter einem Menü Hilfe. Wenn Sie ein Menü mit Einträgen wie Broccoli, Bohnen und Möhren haben, werden Sie das Menü wahrscheinlich mit Gemüse bezeichnen, auch wenn man dafür Lebensmittel oder Pflanzen verwenden könnte. Im allgemeinen sollten Sie darauf achten, daß sich die Benutzer leicht in Ihre Anwendung einarbeiten können, so daß die Bezeichnungen der Menüs möglichst die Einträge im Pulldown-Teil des Menüs charakterisieren.

Frage:

Warum kann ich nicht eine einzelne Taste als Zugriffstaste festlegen?

Antwort:

Eine einzelne Taste löst die Nachricht WM_KEY aus und nicht die Menünachrichten. Als die Entwickler von Windows die Arbeitsweise von Zugriffstasten festgelegt haben, gingen sie davon aus, daß einzelne Tasten normalerweise eine Eingabe in die aktive Anwendung darstellen. Wenn einzelne

Tasten als Zugriffstasten erlaubt wären, könnte Windows nicht bestimmen, ob das Zeichen eine Eingabe oder eine Schnell Taste darstellt. Durch die erforderlichen Tastenkombinationen (mit Ausnahme der Funktionstasten) haben die Entwickler sichergestellt, daß Windows diese Entscheidung nicht treffen muß.

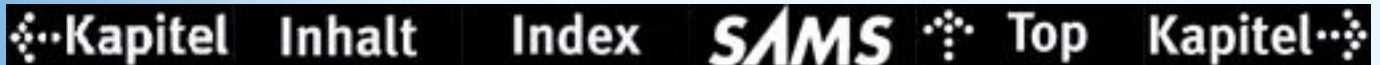
Workshop

Kontrollfragen

1. Welche Nachricht sendet eine Menüauswahl an die Nachrichtenwarteschlange der Anwendung?
2. Wie verbinden Sie ein Menü mit einem Dialogfeld?
3. Welche vorhandene Klasse legen Sie für die Behandlung von Nachrichten für das Menü fest?
4. Durch welche Nachricht sollte man ein Kontextmenü auslösen?

Übungen

1. Nehmen Sie in das Hauptfenster eine Schaltfläche auf, und lassen Sie sie dieselbe Funktion aufrufen wie der Menübefehl **Hello**.
2. Fügen Sie Ihrer Anwendung ein Kontextmenü hinzu, das das Dropdown-Menü **Hilfe** als Kontextmenü verwendet.



Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1**Tag 7****Text und Schriften**

In den meisten Windows-Anwendungen brauchen Sie sich nicht darum zu kümmern, welche Schriften, in welcher Stärke, Höhe usw. zu verwenden sind. Wenn Sie nicht die Schrift explizit festlegen, nimmt Windows eine Standardschrift für die Anwendung an. Wollen Sie eine bestimmte Schrift einsetzen, können Sie die Schrift für ein bestimmtes Dialogfeld über die Eigenschaften des Dialogfelds spezifizieren. Manchmal möchte man aber die in der Anwendung einzusetzende Schrift selbst steuern können oder dem Benutzer die Möglichkeit geben, eine Schrift für eine bestimmte Situation auszuwählen. Für diese Fälle lernen Sie heute, wie man Schriften ändert und auflistet. Insbesondere geht es heute darum, wie man ...

- eine Liste der verfügbaren Schriften erstellt,
- eine zu verwendende Schrift auswählt,
- Schriften dynamisch ändert.

Schriften suchen und verwenden

Bei der Arbeit mit Schriften muß man zunächst wissen, daß nicht jedes System, auf dem die Anwendung läuft, über die gleichen installierten Schriften verfügt. Schriften sind in Dateien festgelegt, die man in Windows-Systemen relativ leicht installieren und wieder entfernen kann. Jeder Computerbenutzer kann sein System mit einer beliebigen Auswahl von Schriften anpassen. Wenn Sie eine Schrift festlegen, die in einem System nicht existiert, wählt Windows entweder die Standardschrift des Systems oder eine möglichst ähnliche alternative Schrift.

Man kann aber die verfügbaren Schriften des Betriebssystems abfragen. Bei dieser Methode können Sie selbst entscheiden oder den Benutzer darüber entscheiden lassen, welche der verfügbaren Schriften zu verwenden sind. Wenn Sie die verfügbaren Schriften abfragen, können Sie auch die aufgelisteten Schriften einschränken oder alle auflisten und dann verschiedene Schriften auf der Basis verschiedener Attribute auswählen.

Die verfügbaren Schriften auflisten

Um eine Liste der verfügbaren Schriften auf einem Computer zu erhalten, rufen Sie die Windows-API-Funktion (API - Application Programming Interface, Anwendungsprogrammierschnittstelle) `EnumFontFamiliesEx` auf. Diese Funktion teilt Windows mit, daß Sie eine Liste der Schriften im System wünschen. Bevor Sie diese Funktion verwenden und erwarten, daß sie Ihnen eine große Liste der verfügbaren Schriften liefert, müssen Sie verstehen, auf welche Weise die Funktion die Liste zurückgibt.

Callback-Funktionen

Eines der Schlüsselargumente der Funktion `EnumFontFamiliesEx` ist die Adresse einer anderen Funktion. Diese zweite Funktion ist eine sogenannte Callback-Funktion (oder Anwendungsrückruf-Routine), die durch das Betriebssystem aufgerufen wird. Bei fast jeder Aufzählungsfunktion im Betriebssystem Windows übergibt

man die Adresse einer Callback-Funktion als Argument, da die Callback-Funktion einmal für jedes Element in der Aufzählungsliste aufgerufen wird. Mit anderen Worten muß man eine Funktion in die Anwendung einbinden, um jede einzelne Schrift des Systems zu empfangen, und dann selbst eine Liste aufbauen.

Wenn man diese Funktion erstellt, um jede Schrift zu empfangen und die Liste zu erstellen, kann man die Callback-Funktion nicht nach Gutdünken selbst festlegen. Alle Callback-Funktionen sind bereits im Windows-API vordefiniert. Man muß einen bestimmten Typ von Callback-Funktion verwenden, um die Liste der Schriften zu empfangen. Der betreffende Funktionstyp lautet `EnumFontFamProc`. Dieser Funktionstyp legt fest, wie Ihre Funktion definiert sein muß, welche Argumente vorhanden sein müssen und welcher Typ des Rückgabewertes zurückzugeben ist. Der Funktionstyp legt nicht fest, wie die Funktion zu benennen ist oder wie sie intern arbeitet. Diese Aspekte liegen vollkommen in Ihrer Hand.

Die Funktion `EnumFontFamiliesEx`

Die Funktion `EnumFontFamiliesEx`, die Sie aufrufen, um die Liste der verfügbaren Schriften abzufragen, übernimmt fünf Argumente. Ein typischer Einsatzfall dieser Funktion sieht folgendermaßen aus:

```
// Eine Gerätekontextvariable erzeugen
CClientDC dc (this);
// Eine LOGFONT-Struktur deklarieren
LOGFONT lLogFont;

// Zeichensatz festlegen
lLogFont.lfCharSet = DEFAULT_CHARSET;
// Alle Schriften spezifizieren
lLogFont.lfFaceName[0] = NULL;
// Muß außer bei Hebräisch oder Arabisch null sein
lLogFont.lfPitchAndFamily = 0;
// Schriftfamilien auflisten
::EnumFontFamiliesEx((HDC) dc, &lLogFont,
(FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
```

Das erste Argument ist ein Gerätekontext, der eine Instanz der Klasse `CClientDC` sein kann. Jede Anwendung, die im Betriebssystem Windows läuft, verfügt über einen Gerätekontext. Dieser stellt dem Betriebssystem eine Reihe notwendiger Informationen - was für die Anwendung verfügbar ist und was nicht - bereit.

Im zweiten Argument übergibt man einen Zeiger auf eine `LOGFONT`-Struktur. Diese enthält die Informationen über die Schriftarten, die man auflisten möchte. In der Struktur kann man angeben, welcher Zeichensatz aufzulisten ist oder ob man alle Schriften einer bestimmten Schriftfamilie haben möchte. Wenn Sie an allen Schriften des Systems interessiert sind, übergeben Sie für dieses Argument den Wert `NULL`.

Das dritte Argument gibt die Adresse der Callback-Funktion an, über die die Liste der Schriften erstellt wird. Die Adresse einer Funktion übergibt man einfach, indem man den Funktionsnamen als Argument verwendet. Der Visual C++-Compiler ersetzt den Funktionsnamen durch die Adresse der Funktion. Allerdings müssen Sie den Typ der Funktion in den Typ derjenigen Callback-Funktion umwandeln, auf die die Funktion zurückgreift.

Das vierte Argument stellt einen `LPARAM`-Wert dar, der an die Callback-Funktion weitergereicht wird. Windows verwendet diesen Parameter nicht, er liefert aber der Callback-Funktion einen Kontext, in dem die Schriftliste zu erstellen ist. Im Beispiel ist der übergebene Wert ein Zeiger auf das Fenster, in dem der Code ausgeführt wird. Auf diese Weise kann die Callback-Funktion diesen Zeiger verwenden, um auf eine beliebige Struktur zuzugreifen, die sie für den Aufbau der Schriftliste benötigt. Der Zeiger kann auch der erste Knoten in einer verketteten Liste von Schriften oder ähnlicher Strukturen sein.

Das fünfte und letzte Argument ist für zukünftige Versionen von Windows reserviert und muß momentan auf 0 gesetzt werden, damit die Anwendung einen Wert übergibt, der kein Fehlverhalten der Funktion bewirkt.

Der Funktionstyp EnumFontFamProc

Wenn Sie Ihre Callback-Funktion erstellen, muß sie als unabhängige Funktion definiert sein und nicht als Element irgendeiner C++-Klasse. Eine typische Funktionsdeklaration von EnumFontFamProc sieht folgendermaßen aus:

```
int CALLBACK EnumFontFamProc(
LPENUMLOGFONT lpelf,
LPNEWTEXTMETRIC lpntm,
DWORD nFontType,
long lParam)
{
// Einen Zeiger auf das Dialogfenster erzeugen
CMyDlg* pWnd = (CMyDlg*) lParam;

// Den Schriftname in das Listenfeld hinzufügen
pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
// 1 zurückgeben, um die Aufzählung fortzusetzen
return 1;
}
```

Das erste Argument an diese Funktion ist ein Zeiger auf eine ENUMLOGFONTEX-Struktur. Diese enthält Informationen über die logischen Attribute der Schrift, einschließlich Schriftname, Stil und Skript. Unter ein und demselben Schriftnamen können mehrere Schriften mit unterschiedlichen Stilen - Standard, Fett, Kursiv und Fett Kursiv - erscheinen.

Das zweite Argument ist ein Zeiger auf eine NEWTEXTMETRICEX-Struktur, die Informationen über die physikalischen Attribute der Schrift wie Höhe, Breite und Zeichenabstand enthält. Es handelt sich dabei um relative Werte, da sie je nach Größe der Schrift zu skalieren sind.

Das dritte Argument ist ein Flag, das den Typ der Schrift spezifiziert. Dieses Flag kann eine Kombination der folgenden Werte darstellen:

- DEVICE_FONTTYPE
- RASTER_FONTTYPE
- TRUETYPE_FONTTYPE

Schließlich gibt das vierte Argument den Wert an, der an die Funktion EnumFontFamiliesEx übergeben wurde. Im Beispiel war das ein Zeiger auf das Dialogfeld, in dem die Liste der Schriften zu erstellen ist. Wenn man diesen Wert als Zeiger auf das Dialogfeld umwandelt, kann die Funktion auf ein Listenfeld zugreifen, um die Schriftnamen hinzuzufügen.

Der Rückgabewert aus dieser Funktion bestimmt, ob sich die Liste der Schriften fortsetzt. Liefert die Funktion 0, beendet das Betriebssystem die Auflistung der verfügbaren Schriften. Wenn die Funktion eine 1 liefert, setzt das Betriebssystem die Liste der verfügbaren Schriften fort.

Eine Schrift verwenden

Um eine bestimmte Schrift in einer Anwendung einzusetzen, ruft man eine Instanz der Klasse CFont auf. Über die Methode CreateFont läßt sich die zu verwendende Schrift zusammen mit deren Größe, Stil und Ausrichtung festlegen. Nachdem Sie eine Schrift erstellt haben, können Sie ein Steuerelement oder Fenster anweisen, diese Schrift zu verwenden, indem Sie die Methode SetFont des jeweiligen Objekts aufrufen. Dieser Ablauf sieht beispielsweise wie folgt aus:

```
CFont m_fFont; // Die zu verwendende Schrift

// Die zu verwendende Schrift erzeugen
```

```
m_fFont.CreateFont(12, 0, 0, 0, FW_NORMAL,
0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH |
FF_DONTCARE, m_sFontName);
```

```
// Die Schrift für den Anzeigebereich festlegen
m_ctlDisplayText.SetFont(&m_fFont);
```



Die im obigen Code verwendete `CFont`-Variable sollte man als Member-Variable der Klasse, in der der Code steht, deklarieren. Im Beispielcode ist die Variable unmittelbar über der Stelle, wo sie verwendet wird, deklariert, um die Art und Weise ihrer Deklaration zu zeigen. Die Variable sollte nicht als lokale Variable in einer Funktion deklariert oder verwendet werden.

Es sieht so einfach aus - lediglich zwei Funktionen aufrufen. Aber an die Funktion `CreateFont` ist eine furchtbar große Zahl von Argumenten zu übergeben. Gerade diese Argumente machen die Methode `CreateFont` zu einer flexiblen Funktion mit umfangreicher Funktionalität. Nachdem man die Schrift erstellt hat, kann man sie ganz einfach einsetzen, indem man die Schrift an die Methode `SetFont` übergibt. Diese Methode ist eine Elementfunktion der Klasse `CWnd` und demzufolge für alle Fenster- und Steuerelementklassen in Visual C++ verfügbar. Das bedeutet, daß man dieses Verfahren für alle visuellen Objekte in einer Visual C++-Anwendung einsetzen kann.

Um die Arbeitsweise der Funktion `CreateFont` zu verstehen, sehen wir uns die einzelnen Argumente an, die an die Funktion zu übergeben sind. Die Funktion ist folgendermaßen definiert:

```
BOOL CreateFont(
int nHeight,
int nWidth,
int nEscapement,
int nOrientation,
int nWeight,
BYTE bItalic,
BYTE bUnderline,
BYTE cStrikeOut,
BYTE nCharSet,
BYTE nOutPrecision,
BYTE nClipPrecision,
BYTE nQuality,
BYTE nPitchAndFamily,
LPCTSTR lpszFaceName);
```

Das erste dieser Argumente, `nHeight`, legt die Höhe der zu verwendenden Schrift fest. Dieser logische Wert wird in einen physikalischen Wert übersetzt. Ist der Wert gleich 0, kommt ein plausibler Standardwert zur Anwendung. Wenn der Wert größer oder kleiner als 0 ist, wird die absolute Höhe in Geräteeinheiten konvertiert. Wichtig ist hierbei vor allem, daß Höhenwerte von 10 und -10 grundsätzlich gleich sind.

Das zweite Argument, `nWidth`, spezifiziert die durchschnittliche Breite der Zeichen in der Schrift. Dieser logische Wert wird in einen physikalischen Wert konvertiert, genau wie es bei der Höhe der Fall ist.

Das dritte Argument, `nEscapement`, bestimmt den Winkel, in dem der Text ausgegeben wird. Dieser Wert ist in Schritten von 0,1 Grad entgegen dem Uhrzeigersinn spezifiziert. Möchte man senkrechten Text von unten nach oben ausgeben, gibt man 900 für dieses Argument an. Der Wert 0 steht für normalen waagrecht von links nach rechts verlaufenden Text.

Das vierte Argument, `nOrientation`, bestimmt den Winkel der einzelnen Zeichen in der Schrift. Das funktioniert genau wie beim vorherigen Argument, steuert aber die zeichenweise Ausgabe und nicht die

zeilenweise Textausgabe. Um Zeichen von oben nach unten auszugeben, setzt man diesen Wert auf 1800. Mit dem Wert 900 werden die Zeichen liegend ausgegeben.

Im fünften Argument, `nWeight`, gibt man die Gewichtung oder die Strichstärke der Schrift an. Dieser Wert umfaßt einen Bereich von 0 bis 1000, wobei 1000 sehr fett bedeutet. Für dieses Argument sind Konstanten definiert, damit man diesen Wert einfach und einheitlich steuern kann. Tabelle 7.1 listet diese Konstanten auf.

Tabelle 7.1: Konstanten für die Gewichtung der Schrift

Konstante	Wert
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_BLACK	900
FW_HEAVY	900

Die tatsächliche Interpretation und Verfügbarkeit dieser Gewichte hängt von der Schrift ab. Einige Schriften haben nur die Gewichte FW_NORMAL, FW_REGULAR und FW_BOLD. Wenn man FW_DONTCARE angibt, wird ein Standardwert wie bei den meisten der übrigen Argumente eingesetzt.

Das sechste Argument, `bItalic`, legt fest, ob die Schrift kursiv auszugeben ist. Es handelt sich um einen Booleschen Wert, wobei 0 für nicht kursive Schrift und jeder andere Wert für eine kursive Schrift steht.

Das siebente Argument, `bUnderline`, ist ebenfalls ein Boolescher Wert und kennzeichnet mit einer 0, daß die Schrift nicht unterstrichen ist, während jeder andere Wert für eine unterstrichene Schrift steht.

Das achte Argument, `cStrikeOut`, gibt an, ob die Zeichen in der Schrift mit einer Linie durch das Zeichen - also durchgestrichen - darzustellen sind. Auch dieser Boolesche Wert bedeutet TRUE - durchgestrichen - bei allen von 0 verschiedenen Werten und FALSE - nicht durchgestrichen - bei 0.

Das neunte Argument, `nCharSet`, legt den Zeichensatz der Schrift fest. Die verfügbaren Konstanten für diesen Wert sind in Tabelle 7.2 aufgeführt.

**Tabelle 7.2:
Zeichensatzkonstanten für
Schriften**

Konstante	Wert
ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
OEM_CHARSET	255

Das System, auf dem Ihre Anwendung läuft, kann andere Zeichensätze verwenden. Darüber hinaus ist der OEM-Zeichensatz systemabhängig. Das verkompliziert die ganze Sache für Systeme unterschiedlicher Hersteller. Wenn Sie einen dieser Zeichensätze verwenden, ist es riskant, die auszugebenden Strings zu manipulieren, so daß es sich empfiehlt, einfach den anzuzeigenden String zu übergeben.

Das zehnte Argument, `nOutPrecision`, legt fest, wie nahe die Ausgabe den angeforderten Werten für Höhe, Breite, Zeichenausrichtung, Textwinkel und Zeichenabstand der Schrift entsprechen muß. Die verfügbaren Werte für dieses Argument lauten:

- `OUT_CHARACTER_PRECIS`
- `OUT_DEFAULT_PRECIS`
- `OUT_DEVICE_PRECIS`
- `OUT_RASTER_PRECIS`
- `OUT_STRING_PRECIS`
- `OUT_STROKE_PRECIS`
- `OUT_TT_PRECIS`

Die Werte `OUT_DEVICE_PRECIS`, `OUT_RASTER_PRECIS` und `OUT_TT_PRECIS` steuern, welche Schrift zu wählen ist, wenn mehrere Schriften mit dem gleichen Namen existieren. Wenn Sie zum Beispiel den Wert `OUT_TT_PRECIS` verwenden und eine Schrift sowohl als TrueType- als auch als Rasterversion spezifizieren, kommt die TrueType- Version zum Einsatz. Praktisch erzwingt `OUT_TT_PRECIS` die Verwendung einer TrueType-Schrift, selbst wenn die angegebene Schrift nicht in einer TrueType-Version vorliegt.

Das elfte Argument, `nClipPrecision`, legt fest, wie die Zeichen, die teilweise außerhalb des Anzeigebereichs liegen, abzuschneiden sind. Die Werte für dieses Argument lauten:

- `CLIP_CHARACTER_PRECIS`
- `CLIP_DEFAULT_PRECIS`
- `CLIP_ENCAPSULATE`
- `CLIP_LH_ANGLES`
- `CLIP_MASK`
- `CLIP_STROKE_PRECIS`
- `CLIP_TT_ALWAYS`

Diese Werte lassen sich mit `OR` verknüpfen, um eine Kombination der Clipping-Verfahren zu spezifizieren.

Das zwölfte Argument, `nQuality`, spezifiziert die Ausgabequalität und gibt an, wie genau das GDI (Graphics Device Interface) versuchen muß, die logischen Attribute in die physikalische Schriftausgabe umzusetzen. Die verfügbaren Werte für dieses Argument lauten:

- `DEFAULT_QUALITY`
- `DRAFT_QUALITY`
- `PROOF_QUALITY`

Im dreizehnten Argument, `nPitchAndFamily`, legt man den Zeichenabstand und die Schriftfamilie fest. Dieser Wert besteht eigentlich aus zwei Werten, die per **OR** verknüpft sind, um einen Kombinationswert zu bilden. Der erste Satz von verfügbaren Werten lautet:

- `DEFAULT_PITCH`
- `VARIABLE_PITCH`
- `FIXED_PITCH`

Diese Werte geben den Zeichenabstand der Schrift an. Die zweite Gruppe von verfügbaren Werten gibt die Schriftfamilie an. Die verfügbaren Werte für diesen Teil des Arguments heißen:

- `FF_DECORATIVE`
- `FF_DONTCARE`
- `FF_MODERN`
- `FF_ROMAN`
- `FF_SCRIPT`
- `FF_SWISS`

Die Schriftfamilie beschreibt auf allgemeine Weise die Erscheinung einer Schrift. Man kann mit dem Wert für die Schriftfamilie eine alternative Schrift wählen, wenn eine bestimmte Schrift nicht im System vorhanden ist.

Das letzte Argument, `lpszFacename`, ist ein normaler C-String, der den Namen der zu verwendenden Schrift angibt. Dieser Schriftname stammt aus der Schriftinformation, die die Callback-Funktion `EnumFontFamProc` zurückgibt.

Schriften verwenden

Heute erstellen Sie eine Anwendung, die dem Benutzer die Möglichkeit gibt, die anzuzeigende Schrift aus einer Liste der verfügbaren Schriften auszuwählen. Der Benutzer gibt den anzuzeigenden Text in der ausgewählten Schrift ein, um sich vom Aussehen der Schrift ein Bild machen zu können.

Das Anwendungsgerüst erstellen

Die heutige Anwendung erstellen Sie mit den folgenden Schritten:

1. Erzeugen Sie mit dem MFC-Anwendungs-Assistenten einen neuen Arbeitsbereich. Nennen Sie das Projekt **Tag7**.
2. Übernehmen Sie die Standardeinstellungen, die Sie bereits in den Projekten der vergangenen Tage verwendet haben. Als Titel der Anwendung geben Sie **Schriften** ein.
3. Entwerfen Sie das Hauptdialogfeld gemäß Abbildung 7.1 mit den Eigenschaften nach Tabelle 7.3.



Abbildung 7.1:
Der Entwurf des Hauptdialogfelds

Tabelle 7.3: Eigenschaftseinstellungen der Steuerelemente

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	&Text eingeben:
Eingabefeld	ID	IDC_ESAMPTEXT
Text	ID	IDC_STATIC
	Titel	Schrift &auswählen
Listenfeld	ID	IDC_LFONTS
Gruppenfeld	ID	IDC_STATIC
	Titel	Schriftbeispiel
Text (innerhalb des Gruppenfeldes; so anpassen, daß Gruppenfeld ausgefüllt wird)	ID	IDC_DISPLAYTEXT
	Titel	Leerer String
Schaltfläche	ID	IDC_EXIT
	Titel	&Beenden

4. Fügen Sie mit dem Klassen-Assistenten die Variablen entsprechend Tabelle 7.4 für die Steuerelemente des Dialogfelds hinzu.

Tabelle 7.4: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ

IDC_DISPLAYTEST	m_ctlDisplayText	Control	CStatic
	m_strDisplayText	Wert	CString
IDC_LFONTS	m_ctlFontList	Control	CListBox
	m_strFontName	Wert	CString
IDC_ESAMPTEXT	m_strSampText	Wert	CString

5. Fügen Sie - wie in den vergangenen Lektionen - der Schaltfläche IDC_EXIT eine Funktion hinzu, um die Anwendung zu schließen.

Eine Schriftliste aufbauen

Damit Sie die Liste der Schriften erstellen können, fügen Sie eine Callback-Funktion hinzu. Damit lassen Sie alle Schriften auflisten und in das Listenfeld des Dialogfelds schreiben. Fügen Sie zu diesem Zweck am Anfang der Header-Datei Tag7Dlg.h die Funktionsdeklaration aus Listing 7.1 ein. Diese Funktion lässt sich nicht mit den in Visual C++ verfügbaren Werkzeugen hinzufügen. Sie müssen die Datei öffnen und die Funktion in eigener Regie eintragen.

Listing 7.1: Die Deklaration der Callback-Funktion in der Header-Datei Tag7Dlg.h

```

1: #if _MSC_VER > 1000
2: #pragma once
3: #endif // _MSC_VER > 1000
4:
5: int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
6: LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam);
7:
8:
9: //////////////////////////////////////
10: // CTag7Dlg Dialogfeld
11:
12: class CTag7Dlg : public CDialog
13: .
14: .
15: .

```

Nachdem Sie die Funktionsdeklaration in die Header-Datei aufgenommen haben, öffnen Sie die Quelldatei Tag7Dlg.cpp, gehen an das Ende der Datei und fügen die Funktionsdefinition aus Listing 7.2 ein.

Listing 7.2: Die Definition der Callback-Funktion in der Quelldatei Tag7Dlg.cpp

```

1: int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
2: LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam)
3: {
4: // Zeiger auf Dialogfenster erzeugen
5: CTag7Dlg* pWnd = (CTag7Dlg*) lParam;
6:
7: // Schriftname in Listenfeld einfügen
8: pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
9: // 1 zurückgeben, um Schriftauflistung fortzusetzen
10: return 1;
11: }

```

Damit haben Sie die Callback-Funktion definiert und müssen nun noch eine Funktion hinzufügen, um die Liste

der Schriften vom Betriebssystem abzurufen. Führen Sie dazu die folgenden Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**.
2. Markieren Sie die Klasse CTag7Dlg, klicken Sie mit der rechten Maustaste, und wählen Sie **Member-Funktion hinzufügen** aus dem Kontextmenü.
3. Legen Sie den Funktionstyp mit **void**, die Funktionsdeklaration als **FillFontList** und den Zugriff als **Privat** fest. Klicken Sie auf **OK**, um das Dialogfeld zu schließen und die Funktion hinzuzufügen.
4. Bearbeiten Sie die Funktionsdefinition gemäß Listing 7.3.

Listing 7.3: Die Funktion FillFontList

```

1: void CTag7Dlg::FillFontList()
2: {
3:   LOGFONT lf;
4:
5:   // LOGFONT-Struktur initialisieren
6:   lf.lfCharSet = DEFAULT_CHARSET;
7:   strcpy(lf.lfFaceName, "");
8:   // Listenfeld leeren
9:   m_ctlFontList.ResetContent();
10:  // Gerätekontextvariable erzeugen
11:  CClientDC dc (this);
12:  // Schriftfamilien aufzählen
13:  ::EnumFontFamiliesEx((HDC) dc, &lf,
14:  (FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
15: }
```

5. Bearbeiten Sie die Funktion **OnInitDialog** gemäß Listing 7.4, um die Funktion **FillFontList** aufzurufen.

Listing 7.4: Die bearbeitete Funktion OnInitDialog

```

1: BOOL CTag7Dlg::OnInitDialog()
2: {
3:   CDialog::OnInitDialog();
4:   .
5:   .
6:   .
7:   // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
8:
9:   ///////////////////////////////////
10:  // EIGENER CODE, ANFANG
11:  ///////////////////////////////////
12:
13:  // Listenfeld für Schriften füllen
14:  FillFontList();
15:
16:  ///////////////////////////////////
17:  // EIGENER CODE, ENDE
18:  ///////////////////////////////////
19:
20:  return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement soll den
  //Fokus erhalten
21: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollte das Listenfeld mit den Namen aller im System verfügbaren Schriften gefüllt sein. Allerdings gibt es einen Aspekt dieser Liste, den Sie in Ihrer Anwendung wahrscheinlich nicht haben möchten. Abbildung 7.2 zeigt viele doppelte Einträge in der Liste der Schriften im Listenfeld. Es wäre schön, wenn man diese Duplikate ausblenden könnte, um nur noch eine Zeile pro Schriftart anzuzeigen.

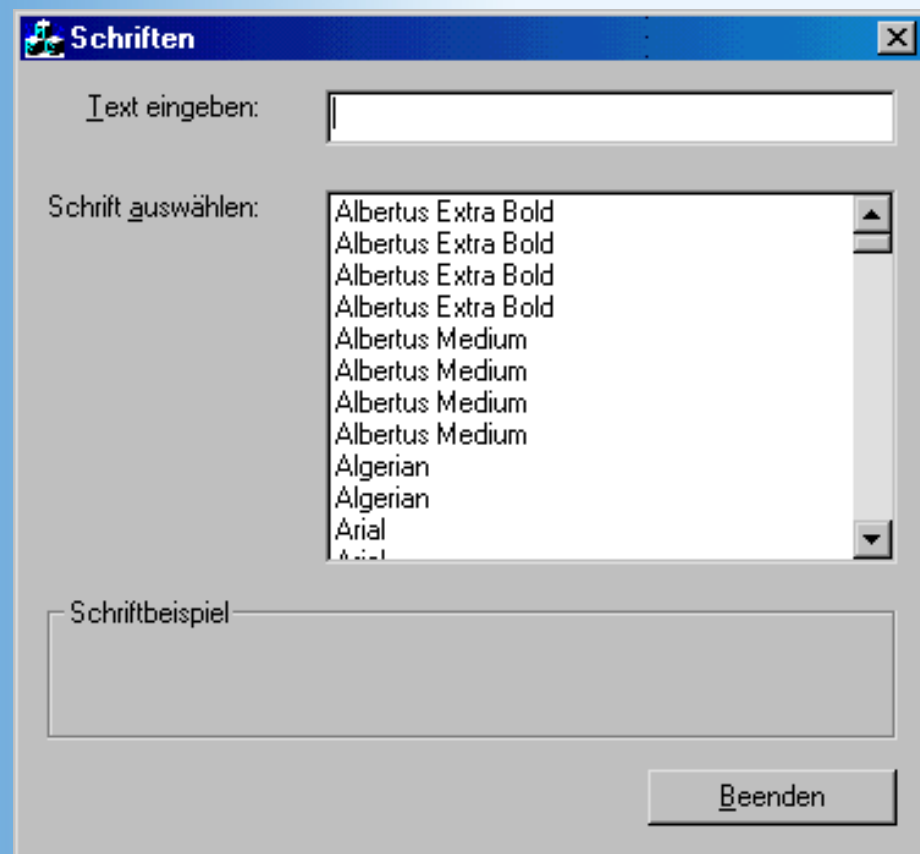


Abbildung 7.2:
Alle Schriften im System auflisten

Es zeigt sich, daß der Funktionsaufruf von `EnumFontFamiliesEx` von Haus aus synchron ist. Das bedeutet, daß die Funktion erst dann zurückkehrt, wenn alle Schriften im System in Aufrufen Ihrer Callback-Funktion aufgelistet sind. Mit etwas Code in der Funktion `FillFontList` lassen sich aber alle doppelten Einträge eliminieren, sobald das Listenfeld gefüllt ist. Dazu modifizieren Sie die Funktion `FillFontList` gemäß Listing 7.5.

Listing 7.5: Die modifizierte Funktion `FillFontList`

```
1: void CTag7Dlg::FillFontList()
2: {
3:     int iCount; // Anzahl der Schriften
4:     int iCurCount; // Aktuelle Schrift
5:     CString strCurFont; // Aktueller Schriftname
6:     CString strPrevFont = ""; // Vorheriger Schriftname
7:     LOGFONT lf;
8:
9:     // LOGFONT-Struktur initialisieren
10:    lf.lfCharSet = DEFAULT_CHARSET;
11:    strcpy(lf.lfFaceName, "");
12:    // Listenfeld leeren
13:    m_ctlFontList.ResetContent();
14:    // Gerätekontextvariable erzeugen
15:    CClientDC dc (this);
16:    // Schriftfamilien aufzählen
```

```

17: ::EnumFontFamiliesEx((HDC) dc, &lf,
18: (FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
19: // Anzahl der Schriften im Listenfeld ermitteln
20: iCount = m_ctlFontList.GetCount();
21: // Schleife vom letzten zum ersten Eintrag im Listenfeld,
22: // um doppelte Einträge zu suchen und zu löschen
23: for (iCurCount = iCount; iCurCount > 0; iCurCount--)
24: {
25: // Aktuellen Schriftname holen
26: m_ctlFontList.GetText((iCurCount - 1), strCurFont);
27: // Mit vorherigem Schriftnamen identisch?
28: if (strCurFont == strPrevFont)
29: {
30: // Wenn ja, dann löschen
31: m_ctlFontList.DeleteString((iCurCount - 1));
32: }
33: // Vorherigen auf aktuellen Schriftname setzen
34: strPrevFont = strCurFont;
35: }
36: }

```

Beachten Sie, daß die `for`-Schleife am Ende der Liste beginnt und rückwärts nach oben arbeitet. Damit kann man den aktuellen Eintrag löschen, ohne sich um den Schleifenzähler kümmern zu müssen. Ansonsten müßte man den Zähler korrigieren, um keine Zeile zu überspringen. Wenn Sie jetzt die Anwendung kompilieren und ausführen, sollten keine doppelten Einträge mehr in der Liste der verfügbaren Schriften zu sehen sein.

Text für Schriftprobe festlegen

Bevor man die Schrift für den Benutzer anzeigen kann, muß man etwas Text in den Anzeigebereich schreiben. In das Eingabefeld im oberen Teil des Dialogfelds gibt der Benutzer den Text ein, der in der ausgewählten Schrift anzuzeigen ist. Führen Sie die folgenden Schritte aus, um die Schriftprobe darzustellen:

1. Übernehmen Sie in die Funktion `OnInitDialog` den Code aus Listing 7.6, um das Eingabefeld zu initialisieren und Text anzuzeigen.

Listing 7.6: Die modifizierte Funktion `OnInitDialog`

```

1: BOOL CTag7Dlg::OnInitDialog()
2: {
3: CDialog::OnInitDialog();
4: .
5: .
6: .
7: // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
8:
9: //////////////////////////////////////
10: // EIGENER CODE, ANFANG
11: //////////////////////////////////////
12:
13: // Listenfeld für Schriften füllen
14: FillFontList();
15:
16: // Einzugebenden Text initialisieren
17: m_strSampText = "Testen";
18: // Text in Feld für Schriftbeispiel kopieren

```

```

19: m_strDisplayText = m_strSampText;
20: // Dialogfeld aktualisieren
21: UpdateData(FALSE);
22:
23: ///////////////////////////////////
24: // EIGENER CODE, ENDE
25: ///////////////////////////////////
26:
27: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement soll den
  Fokus erhalten
28: }

```

2. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `EN_CHANGE` für das Eingabefeld `IDC_ESAMPTTEXT` hinzu.

3. Schreiben Sie in die Funktion den Code aus Listing 7.7.

Listing 7.7: Die Funktion `OnChangeEsamptext`

```

1: void CTag7Dlg::OnChangeEsamptext()
2: {
3: // TODO: Wenn dies ein RICHEDIT-Steuerelement ist, sendet das
  Steuerelement diese
4: // Benachrichtigung nicht, bevor Sie nicht die Funktion
  CDialog::OnInitDialog()
5: // überschreiben und CRichEditCtrl().SetEventMask() aufrufen, wobei
6: // eine ODER-Operation mit dem Attribut ENM_CHANGE und der Maske
  erfolgt.
7:
8: // TODO: Fügen Sie hier Ihren Code für die
  Benachrichtigungsbehandlungsroutine des Steuerelements hinzu
9:
10: ///////////////////////////////////
11: // EIGENER CODE, ANFANG
12: ///////////////////////////////////
13:
14: // Variablen mit Steuerelementen des Dialogfelds aktualisieren
15: UpdateData(TRUE);
16:
17: // Aktuellen Text in Schriftbeispiel kopieren
18: m_strDisplayText = m_strSampText;
19:
20: // Dialogfeld mit Variablen aktualisieren
21: UpdateData(FALSE);
22:
23: ///////////////////////////////////
24: // EIGENER CODE, ENDE
25: ///////////////////////////////////
26: }

```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie in das Eingabefeld Text eingeben können, der gleichzeitig im Anzeigebereich für die Schrift im darunterliegenden Gruppenfeld erscheint.

Anzuzeigende Schrift auswählen

Bevor Sie die Schrift für den Anzeigebereich ändern können, brauchen Sie eine `CFont`-Elementvariable der Dialogfeldklasse, über die Sie die anzuzeigende Schrift festlegen und ändern können. Führen Sie die folgenden Schritte aus, um diese Variable hinzuzufügen:

1. Klicken Sie im Arbeitsbereich auf der Registerkarte **Klassen** mit der rechten Maustaste auf die Klasse `CTag7Dlg`. Wählen Sie aus dem Kontextmenü den Befehl **Member-Variable hinzufügen**.
2. Legen Sie den Typ der Variablen als `CFont`, den Variablennamen mit `m_fSampFont` und den Zugriff als *Privat* fest. Klicken Sie auf **OK**, um das Dialogfeld zu schließen und die Variable hinzuzufügen.

Wenn Sie den Code hinzufügen, um die ausgewählte Schrift zu verwenden, schreiben Sie dafür eine eigene Funktion, die nicht mit einem Steuerelement verbunden ist. Der Grund dafür zeigt sich, wenn Sie weiter an der heutigen Beispielanwendung arbeiten. Um die Funktion für die Anzeige und Verwendung der ausgewählten Schrift hinzuzufügen, führen Sie die folgenden Schritte aus:

1. Klicken Sie im Arbeitsbereich auf der Registerkarte **Klassen** mit der rechten Maustaste auf die Klasse `CTag7Dlg`. Wählen Sie aus dem Kontextmenü den Befehl **Member-Funktion hinzufügen**.
2. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `SetMyFont` und den Zugriffsstatus als *Privat* fest. Klicken Sie auf **OK**, um das Dialogfeld zu schließen und die Funktion hinzuzufügen.
3. Übernehmen Sie in die Funktion den Code aus Listing 7.8.

Listing 7.8: Die Funktion `SetMyFont`

```
1: void CTag7Dlg::SetMyFont()
2: {
3:     CRect rRect; // Rechteck des Anzeigebereichs
4:     int iHeight; // Höhe des Anzeigebereichs
5:
6:     // Wurde eine Schrift ausgewählt?
7:     if (m_strFontName != "")
8:     {
9:         // Abmessungen des Feldes für Schriftbeispiel ermitteln
10:        m_ctlDisplayText.GetWindowRect(&rRect);
11:        // Höhe des Feldes berechnen
12:        iHeight = rRect.top - rRect.bottom;
13:        // Sicherstellen, daß Höhe positiv
14:        if (iHeight < 0)
15:            iHeight = 0 - iHeight;
16:        // Aktuelle Schrift freigeben
17:        m_fSampFont.Detach();
18:        // Zu verwendende Schrift erzeugen
19:        m_fSampFont.CreateFont((iHeight - 5), 0, 0, 0, FW_NORMAL,
20:                               0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
21:                               CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH |
22:                               FF_DONTCARE, m_strFontName);
23:
24:        // Schrift für Anzeigefeld der Schriftprobe festlegen
25:        m_ctlDisplayText.SetFont(&m_fSampFont);
26:    }
27: }
```

4. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `LBN_SELCHANGE` für das Listenfeld `IDC_LFONTS` hinzu, und übernehmen Sie in diese Funktion den Code aus Listing 7.9.

Listing 7.9: Die Funktion OnSelchangeLfonts

```

1: void CTag7Dlg::OnSelchangeLfonts()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Âbenachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Variablen mit Steuerelementen des Dialogfelds aktualisieren
10: UpdateData(TRUE);
11:
12: // Schrift für Beispiel festlegen
13: SetMyFont();
14:
15: //////////////////////////////////
16: // EIGENER CODE, ENDE
17: //////////////////////////////////
18: }

```

Die Funktion `SetMyFont` prüft zuerst, ob eine Schrift ausgewählt wurde. Als nächstes wird der Bereich des statischen Textfelds abgerufen, der für die Anzeige der Schriftprobe vorgesehen ist. Damit läßt sich eine Schrifthöhe festlegen, die etwas kleiner als die Höhe des Bereichs ist, der für die Anzeige der Schrift zur Verfügung steht. Nachdem die Funktion die Höhe des statischen Textfelds berechnet hat und das Ergebnis ein positiver Wert ist, wird die ausgewählte Schrift erzeugt und das statische Textfeld angewiesen, die neu erstellte Schrift zu verwenden.

Die Funktion `OnSelchangeLfonts` kopiert die Steuerelementwerte in die zugeordneten Variablen und ruft dann die Funktion `SetMyFont` auf, um die ausgewählte Schrift zu verwenden. Wenn Sie die Anwendung kompilieren und ausführen, sollten Sie eine Schrift auswählen und das Ergebnis im Anzeigebereich verfolgen können, wie es Abbildung 7.3 zeigt.

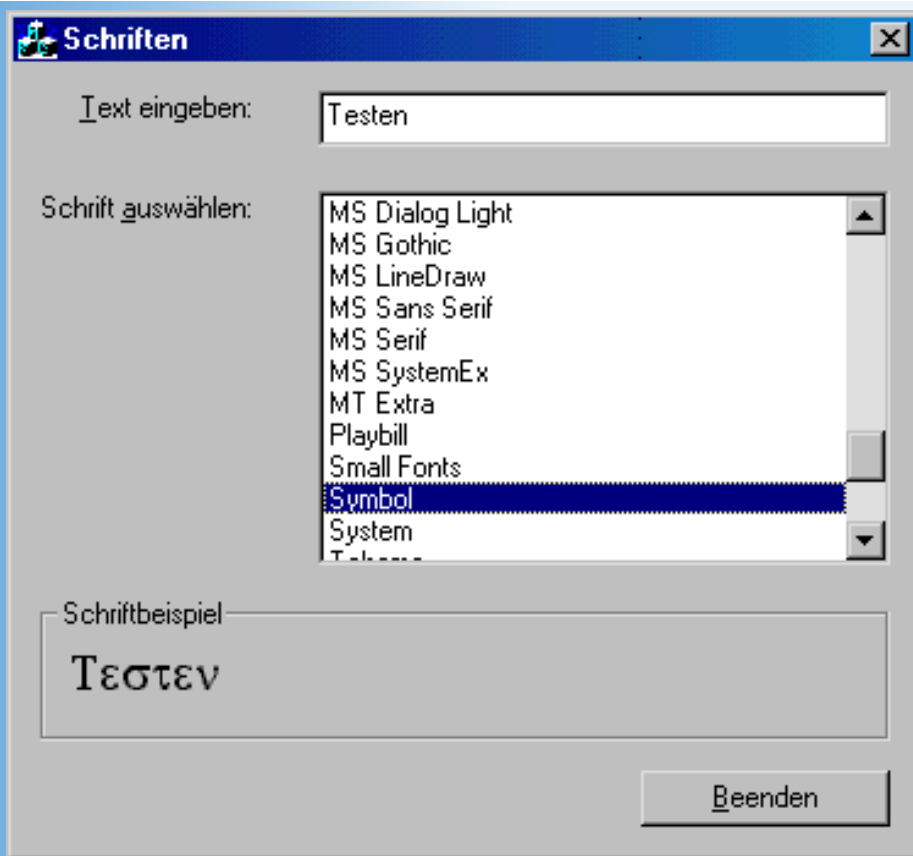


Abbildung 7.3:
Die ausgewählte Schrift anzeigen

Zusammenfassung

Heute haben Sie gelernt, wie man Schriften in Visual C++-Anwendungen verwendet. Dabei haben Sie eine Liste der im System verfügbaren Schriften erstellt und eine Schrift für ein Anzeigeobjekt erzeugt. Weiterhin hat die Lektion gezeigt, wie man Callback-Funktionen erstellt und einsetzt, um eine Liste der Ressourcen vom Betriebssystem Windows zu erhalten. Außerdem haben Sie gelernt, wie man auf Steuerelemente von der Callback-Funktion aus zugreift und dabei einen Fensterzeiger verwendet, den man an die Funktion, die die Ressourcenliste abrufen, übergeben hat.

Fragen und Antworten

Frage:

Der Funktion `CreateFont` muß man eine Vielzahl von Argumenten übergeben. Kann man auch auf eine alternative Funktion zurückgreifen?

Antwort:

Eine derartige Funktion gibt es, obwohl man trotzdem noch alle Informationen festlegt. Die Struktur `LOGFONT` enthält die gleichen Attribute, die man an die Funktion `CreateFont` übergibt. Man kann eine Instanz dieser Struktur deklarieren, die Attribute mit Standardwerten initialisieren und dann diese Struktur an die Funktion `CreateFontIndirect` übergeben. Wenn umfangreiche Schriftänderungen vorzunehmen sind, ist diese Methode zu bevorzugen, weil man dieselbe Instanz der Struktur verwenden kann und nur diejenigen Attribute modifiziert, die sich gegenüber den aktuellen Einstellungen geändert haben. Damit erstellt man dann die verschiedenartigen Schriften.

Um diese alternative Lösung zum Erzeugen einer Schrift zu verwenden, deklariert man eine Instanz der Struktur `LOGFONT` als Element der Dialogfeldklasse und initialisiert dann alle Attribute, bevor man die Funktion `SetMyFont` aufruft. In der Funktion `SetMyFont` nehmen Sie die Modifikationen gemäß Listing 7.10 vor.

Listing 7.10: Die modifizierte Funktion SetMyFont

```

1: void CTag7Dlg::SetMyFont()
2: {
3:
4: // Wurde eine Schrift ausgewählt?
5: if (m_strFontName != "")
6: {
7: // Annahme, daß Schriftgröße bereits in Struktur
8: // m_lLogFont initialisiert wurde. Damit ist nur
9: // noch der Schriftname zu spezifizieren.
10: _tcscpy(m_lLogFont.lfFaceName, m_strFontName);
11: // Zu verwendende Schrift erzeugen
12: m_fSampFont.CreateFontIndirect(&m_lLogFont);
13:
14: // Schrift für Anzeigefeld der Schriftprobe festlegen
15: m_ctlDisplayText.SetFont(&m_fSampFont);
16: }
17: }

```

Frage:

Wie kann ich die Schriften in meiner Liste ausschließlich auf TrueType-Schriften begrenzen?

Antwort:

Testen Sie das an die Callback-Funktion übergebene Argument *nFontType*, um den Schrifttyp zu bestimmen. Wenn Sie zum Beispiel nur TrueType- Schriften in die Schriftliste aufnehmen wollen, modifizieren Sie die Callback-Funktion, um das Argument *nFontType* mit der Konstanten *TRUETYPE_FONTTYPE* zu maskieren, und prüfen dann, ob das Ergebnis gleich *TRUETYPE_FONTTYPE* ist, wie es das folgende Listing zeigt:

```

int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam)
{
// Zeiger auf Dialogfenster erzeugen
CTag7Dlg* pWnd = (CTag7Dlg*) lParam;

// Liste auf TrueType-Schriften begrenzen
if ((nFontType & TRUETYPE_FONTTYPE) == TRUETYPE_FONTTYPE)
{
// Schriftname in Listefeld einfügen
pWnd->m_ctlFontList.AddString(
lpelf->elfLogFont.lfFaceName);
}
// 1 zurückgeben, um Schriftauflistung fortzusetzen
return 1;
}

```

Workshop

Kontrollfragen

1. Wie kann man festlegen, daß der Text unterstrichen wird?
2. Wie läßt sich Text von oben nach unten ausgeben?
3. Wie oft wird die Callback-Funktion *EnumFontFamProc* durch das Betriebssystem aufgerufen?

Übungen

1. Fügen Sie gemäß Abbildung 7.4 ein Kontrollkästchen hinzu, um wählen zu können, ob der eingegebene Text die Schrift repräsentieren soll oder der Schriftname in der ausgewählten Schrift anzuzeigen ist.



Abbildung 7.4:
Ausgewählte Schrift mit Bezeichnung der Schrift anzeigen

2. Fügen Sie gemäß Abbildung 7.5 ein Kontrollkästchen hinzu, um die Textprobe kursiv anzuzeigen.



Abbildung 7.5:

Die ausgewählte Schrift kursiv anzeigen



Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 1 im Rückblick

Die erste Woche liegt hinter Ihnen. Sie haben nun einen ersten Eindruck bekommen, welche Möglichkeiten sich mit Visual C++ beim Erstellen von Anwendungen bieten. Blicken wir noch einmal zurück, was in den ersten sieben Tagen behandelt wurde und was Sie bisher gelernt haben.

Es empfiehlt sich an dieser Stelle, daß Sie ein paar einfache Anwendungen auf eigene Faust entwerfen und erstellen, um Ihre Kenntnisse zu den bisher behandelten Elementen zu vertiefen. Bauen Sie die verschiedensten Steuerelemente in die Anwendungen ein, und fügen Sie weitere Dialogfelder hinzu, damit Sie mit diesen Themen vertraut werden. Erst bei der praktischen Arbeit zeigt sich, ob Sie alle Konzepte verstanden haben. Vielleicht vertiefen Sie sich auch in die MFC-Dokumentation und probieren kompliziertere - und hier nicht behandelte - Funktionen in eigenen Anwendungen aus.

Vor allem verstehen Sie jetzt, wie man Steuerelemente und Dialogfelder in Anwendungen einbindet, um Informationen vom Benutzer einzuholen und ihm anzuzeigen. Das ist ein fundamentaler Teil jeder Windows-Anwendung, da fast jede Anwendung mit dem Benutzer auf irgendeine Weise in Wechselwirkung tritt. Die Standardsteuerelemente können Sie problemlos im Anwendungsfenster unterbringen und sie in die Anwendung einbeziehen. Das gleiche gilt für das Standardmeldungsfeld sowie die von Windows bereitgestellten Dialogfelder. Weiterhin können Sie eigene, benutzerdefinierte Dialogfelder erstellen und in eine geplante Anwendung einbinden. Wiederholen Sie bei Bedarf den Stoff zu Tag 2, um sich mit Steuerelementen vertraut zu machen, und/oder vertiefen Sie Ihre Kenntnisse zu Standard- und benutzerdefinierten Dialogfeldern in der Lektion 5.

Ein weiterer wichtiger Punkt, dem Sie in der Mehrheit Ihrer Anwendungen begegnen, sind die Menüs. Dazu müssen Sie genau wissen, wie man ein gutes Menü entwirft, wie man sicherstellt, daß keine Konflikte mit den Zugriffstasten auftreten, und wie man die Funktionalität der Anwendung mit den Menübefehlen verbindet. Die von Ihnen erstellten Menüs und die darüber ausgelösten Befehle können Sie nun problemlos in Ihre Anwendungen integrieren. Wenn Sie mit diesem Thema nicht 100%ig vertraut sind, wiederholen Sie den Stoff von Tag 6.

In verschiedenen Situationen müssen Sie Aktionen in regelmäßigen Abständen auslösen oder ermitteln, wie lange bestimmte Abläufe dauern. In diesen und anderen Fällen kommen die Timer ins Spiel. Sollten Sie noch Probleme haben, wie man Timer in Anwendungen einbaut, gehen Sie einfach zu Tag 4 zurück.

Das Erscheinungsbild Ihrer Anwendungen läßt sich mit Text und Schriften flexibler gestalten - und Sie können dem Benutzer die Möglichkeit bieten, das Aussehen der Anwendung an die eigenen Vorstellungen anzupassen. Am Tag 7 haben Sie erfahren, wie man die verfügbaren Schriften des Computers, auf dem die Anwendung läuft, untersucht, und falls eine von der Anwendung genutzte Schrift nicht verfügbar ist, eine andere Schrift auswählt, die der geforderten möglichst nahe kommt. Sollten noch Fragen in bezug auf die Infrastruktur von Schriften unter Windows bestehen und wie Sie die Schriften in Ihren Anwendungen einsetzen, wiederholen Sie das Material von Tag 7.

Je nach der Art Ihrer Anwendungen kann es erforderlich sein, die vom Benutzer ausgelösten Maus- und Tastaturereignisse aufzufangen. Wenn Sie etwa eine Zeichenanwendung erstellen, sind diese Informationen von grundlegender Bedeutung. Das gleiche gilt bei einer Anwendung, die Sie mit Drag & Drop-Fähigkeiten ausstatten. Es gibt eine Reihe von Situationen, bei denen diese Funktionalität in Ihren Anwendungen unabdingbar ist. Mit Abschluß des dritten Tages sind Sie damit vertraut, wie man die verschiedenen Mausereignisse auffängt und ermittelt, welche Maustaste beim jeweiligen Ereignis gedrückt wurde. Außerdem können Sie Tastaturereignisse auffangen, wenn die Steuerelemente im Anwendungsfenster nicht selbst diese Tastatureingaben verarbeiten. Wiederholen Sie einfach den Stoff des dritten Tages, falls Ihnen diesbezüglich noch nicht alles geläufig ist.

Schließlich kennen Sie mittlerweile die Entwicklungsumgebung von Visual C++ - das Visual Studio - zur Genüge und wissen, welcher Bereich der Umgebung wofür gedacht ist und wie man die verschiedenen Werkzeuge und Hilfsprogramme beim Erstellen der Anwendung einsetzt. Sie können jetzt problemlos über den Arbeitsbereich durch Ihr Anwendungsprojekt navigieren und die verschiedenen Editoren lokalisieren und starten. Weiterhin können Sie das Symbol, das Ihre Anwendung repräsentiert, finden und selbst gestalten. Insbesondere wissen Sie, wo die einzelnen Elementfunktionen oder Variablen in den Klassen Ihrer Anwendung untergebracht sind.

Nach Abschluß der ersten Woche arbeiten Sie bereits problemlos mit Visual C++. Wenn Sie mit den bisher behandelten Themen einigermaßen vertraut sind, fahren Sie fort und lernen mehr darüber, was sich mit Visual C++ als Ihr Programmierwerkzeug schlechthin alles realisieren läßt. Also auf zur zweiten Woche ...

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2 im Überblick

In der zweiten Woche tauchen Sie in verschiedene kompliziertere Themen ein. Dennoch gehören diese Themen zum Kern beim Erstellen von Windows-Anwendungen. In nahezu jeder Anwendung, die Sie mit Visual C++ erstellen, kommen Ihnen die Kenntnisse der zweiten Woche als auch das bereits in der ersten Woche Gelernte zugute.

Die Woche beginnt mit Tag 8, der sich mit dem Zeichnen von Grafiken in einer Windows-Anwendung beschäftigt. Sie lernen, wie man einfache Linien, Rechtecke und Ellipsen zeichnet. Vor allem aber steht der Gerätekontext im Mittelpunkt. Es wird gezeigt, wie man einen Gerätekontext beim Zeichnen von Grafiken einsetzt, ohne sich um die konkrete Grafikhardware kümmern zu müssen, mit der der Benutzer Ihrer Anwendung arbeitet.

Der neunte Tag demonstriert, wie einfach sich ActiveX-Steuerelemente in eine Anwendung einbinden lassen. Sie erfahren, wie Visual C++ benutzerdefinierte C++- Klassen um die Steuerelemente, die Sie in Ihr Projekt aufnehmen, konstruiert und damit die Interaktion mit einem ActiveX-Steuerelement wie mit jedem anderen C++-Objekt gestattet.

Am Tag 10 lernen Sie, wie man eine grundlegende Anwendung mit einem Dokument - eine sogenannte SDI- (Single Document Interface) Anwendung - erstellt. In diesem Zusammenhang wird die Dokument/Ansicht-Architektur behandelt, die man bei Visual C++ für derartige Anwendungen einsetzt, und Sie erfahren, wie man auf der Basis dieser Architektur eigene Anwendungen erstellt.

In der elften Lektion wenden Sie Ihr Wissen zum Erstellen von SDI-Anwendungen auf Anwendungen mit mehreren Dokumenten - sogenannten MDI- (Multiple Document Interface) Anwendungen - an. Sie sehen, wie man mit der gleichen Dokument/Ansicht-Architektur MDI-Anwendungen erstellt, die zu den gebräuchlichsten Formen der heute verfügbaren Windows-Anwendungen gehören.

Am zwölften Tag erstellen und modifizieren Sie Ihre eigenen Symbolleisten und Statusleisten. Es wird gezeigt, wie man die Schaltflächen einer Symbolleiste mit Menüs in der Anwendung verbindet und zusätzliche Symbolleisten hinzufügt. Weiterhin bringen Sie Ihre eigenen Informationselemente in der Statusleiste, die man in den meisten Windows-Anwendungen am unteren Rand des Fensters findet, unter, und Sie lernen, wie man die Statusleiste aktualisiert, um den jeweiligen Zustand der Anwendung widerzuspiegeln.

Die Dokument/Ansicht-Architektur ist ebenfalls am Tag 13 ein Schwerpunktthema. Hier setzen Sie diese Architektur ein, um die in der Anwendung erzeugten Daten zu speichern und wiederherzustellen. Dabei lernen Sie die Flexibilität dieser Einrichtung kennen und erfahren, wie man die unterschiedlichsten Datentypen in derselben Datei speichern und sie in der Anwendung in der ursprünglichen Form wiederherstellen kann.

Schließlich rundet Tag 14 die Woche mit einer Datenbankanwendung auf der Basis einer ODBC-Datenbank ab. In der Beispielanwendung rufen Sie eine Gruppe von Datensätzen aus der

Datenbank ab und bieten dem Benutzer die Möglichkeit, die Datensätze zu bearbeiten und zu modifizieren. Außerdem lassen sich die Änderungen zurück in die Datenbank schreiben.

Mit Abschluß dieser Woche beherrschen Sie die grundlegenden Aufgaben bei der Anwendungsentwicklung mit Visual C++. An dieser Stelle sollten Sie eine kleine Pause einlegen und etwas experimentieren. Versuchen Sie, die verschiedenen Arten von Anwendungen zu erstellen, Ihre Kenntnisse zu erweitern und herauszufinden, wo Ihre Grenzen liegen, bevor Sie sich der letzten Woche mit komplizierteren Themen zuwenden.

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2

Tag 8

Bilder, Zeichnungen und Bitmaps

Vielleicht ist Ihnen schon aufgefallen, daß viele Anwendungen Zeichnungen und Bilder anzeigen. Das verleiht einer Anwendung einen gewissen Pfiff und Schliff. Bei einigen Anwendungen sind Grafiken ein integraler Bestandteil der gebotenen Funktionalität. Wenn man unter Windows erfolgreich programmieren will, sind gute Kenntnisse unerlässlich, wie man die grafischen Fähigkeiten in einer Anwendung umsetzt. Bisher haben Sie gelernt, wie man Linien zeichnet und wie sich eine Folge von Linien zu einer geschlossenen Zeichnung verbinden läßt. Heute gehen wir über diese Möglichkeiten hinaus. Sie lernen, wie sich kompliziertere Grafikfähigkeiten in eine Anwendung aufnehmen lassen. Insbesondere beschäftigen wir uns heute damit, wie ...

- Windows über einen Gerätekontext Zeichenbefehle in grafische Ausgaben übersetzt,
- man die grafische Ausgabe über verschiedene Abbildungsmodi beeinflussen kann,
- Windows mit Stiften und Pinseln verschiedene Teile der Grafik zeichnet,
- man Bitmaps dynamisch lädt und anzeigt.

Die grafische Geräteschnittstelle (GDI)

Das Betriebssystem Windows stellt eine Reihe von Abstraktionsebenen bereit, die sich auf das Erstellen und den Einsatz von Grafiken in einer Anwendung beziehen. Während der Tage der DOS-Programmierung mußten Sie sich eingehend mit der Ansteuerung der Grafikhardware beschäftigen, um Zeichenoperationen in einer Anwendung zu realisieren. Das erforderte umfangreiches Wissen und Kenntnisse der verschiedenartigsten Grafikkarten, die ein Benutzer in seinen Computer einbauen konnte, wozu noch die verschiedensten Optionen für Monitore und Bildschirmauflösungen kamen. Es gab zwar einige Grafikbibliotheken, die man für die Anwendungsprogrammierung erwerben konnte, aber alles in allem war es ziemlich mühsam, diese Fähigkeit in eine Anwendung einzubauen.

Mit Windows hat Microsoft das Ganze vereinfacht. Zuerst einmal stellt Microsoft ein virtuelles Grafikgerät für alle Windows-Anwendungen bereit. Dieses virtuelle Gerät ändert sich nicht mit der Hardware, sondern bleibt für die gesamte Palette der Grafikhardware, die man eventuell beim Benutzer vorfindet, gleich. Aufgrund dieser Einheitlichkeit lassen sich beliebige Grafikaufgaben in einer Anwendung realisieren, da man sich nicht um die Konvertierung der Ausgaben in Befehle, die die Hardware versteht, kümmern muß.

Gerätekontexte

Bevor man irgendeine Grafik erstellen kann, muß man über den Gerätekontext verfügen, in dem die Grafiken angezeigt werden. Der Gerätekontext enthält Informationen über das System, die Anwendung und das Fenster, in dem die Ausgabe der Zeichnungen und Bilder erfolgt. Das Betriebssystem entnimmt dem Gerätekontext, in welchem Kontext eine Grafik zu zeichnen ist, wie groß der sichtbare Bereich ist, und wo sich der Zeichenbereich momentan auf dem Bildschirm befindet.

Eine Grafik geben Sie immer im Kontext eines Anwendungsfensters aus. Dieses Fenster kann jederzeit als Vollbild, minimiert, teilweise überdeckt oder vollständig unsichtbar sein. Um den jeweiligen Zustand brauchen Sie

sich nicht zu kümmern, weil Sie Ihre Grafiken mit Hilfe des Gerätekontextes in das Fenster zeichnen. Windows verwaltet alle Gerätekontexte und ermittelt daraus, wieviel und welcher Teil der gezeichneten Grafiken tatsächlich für den Benutzer anzuzeigen ist. Praktisch stellt der Gerätekontext, in dem Sie Ihre Grafiken zeichnen, den visuellen Kontext des Fensters dar, in dem Sie die Grafiken ausgeben.

Die meisten Funktionen in bezug auf Zeichnungen und Bilder führt der Gerätekontext mit zwei Ressourcen aus: Stift und Pinsel. Wie ihre Entsprechungen in der realen Welt führen Stifte und Pinsel zwar ähnliche, aber dennoch unterschiedliche Aufgaben aus. Der Gerätekontext verwendet Stifte, um Linien und Figuren zu zeichnen, während Pinsel die Flächen auf dem Bildschirm ausfüllen. Das entspricht völlig der Vorgehensweise, wenn man auf Papier zunächst die Umrisse eines Bildes zeichnet und dann einen Pinsel in die Hand nimmt, um die Flächen zwischen den Linien mit Farben auszufüllen.

Die Gerätekontextklasse

In Visual C++ stellt die MFC-Gerätekontextklasse (CDC) zahlreiche Zeichenfunktionen für Kreise, Quadrate, Linien, Kurven usw. bereit. Diese Funktionen gehören zur Gerätekontextklasse, da sie alle auf Informationen des Gerätekontextes zurückgreifen, um in den Anwendungsfenstern zu zeichnen.

Die Instanz einer Gerätekontextklasse erstellt man mit einem Zeiger auf die Fensterklasse, die man mit dem Gerätekontext verbinden möchte. Damit läßt sich in den Konstruktoren und Destruktoren der Gerätekontextklasse der gesamte Code unterbringen, der sich mit der Zuweisung und Freigabe eines Gerätekontextes befaßt.



Gerätekontextobjekte gehören genau wie alle Arten von Zeichenobjekten zu den Ressourcen im Betriebssystem Windows. Das Betriebssystem verfügt nur über eine begrenzte Zahl dieser Ressourcen. Obwohl die Gesamtzahl der Ressourcen in den neueren Versionen von Windows recht groß ist, kann dennoch ein Mangel an Ressourcen entstehen, wenn eine Anwendung die Ressourcen zwar reserviert, aber nicht wieder korrekt freigibt. Diesen Verlust bezeichnet man als Ressourcenlücke, die analog zu einer Speicherlücke das System des Benutzer blockieren oder lahmlegen kann. Es empfiehlt sich also, die Ressourcen in Funktionen zu erzeugen, wo sie zum Einsatz kommen, und sie sobald wie möglich wieder freizugeben, wenn die Arbeit mit den betreffenden Ressourcen abgeschlossen ist.

Dementsprechend nutzt man Gerätekontexte und deren Zeichenressourcen fast ausschließlich als lokale Variablen innerhalb einer einzigen Funktion. Die einzige echte Ausnahme liegt vor, wenn das Gerätekontextobjekt von Windows erzeugt und in eine ereignisverarbeitende Funktion als Argument übergeben wird.

Die Stiftklasse CPen

Die Stiftklasse CPen haben Sie bereits eingesetzt, um Farbe und Breite für die auf dem Bildschirm zu zeichnenden Zeichen festzulegen. CPen ist das hauptsächliche Ressourcenwerkzeug, mit dem man Linien aller Art auf dem Bildschirm darstellt. Erzeugt man eine Instanz der Klasse CPen, kann man den Typ, die Farbe und die Dicke der Linie spezifizieren. Nachdem der Stift erzeugt ist, läßt er sich als aktuelles Zeichenwerkzeug für den Gerätekontext auswählen und damit für alle Zeichenbefehle an diesen Gerätekontext verwenden. Der folgende Code erzeugt einen neuen Stift und wählt ihn dann als aktuellen Zeichenstift aus:

```
// Den Gerätekontext erzeugen
CDC dc(this);
// Den Stift erzeugen
CPen lPen(PS_SOLID, 1, RGB(0, 0, 0));
// Den Stift als aktuellen Zeichenstift auswählen
dc.SelectObject(&lPen);
```

Es stehen mehrere Stiftstile zur Verfügung, die verschiedene Muster der gezeichneten Linien ergeben. Abbildung 8.1 zeigt die grundlegenden Stile, auf die man mit jeder Farbe in einer Zeichenanwendung zurückgreifen kann.

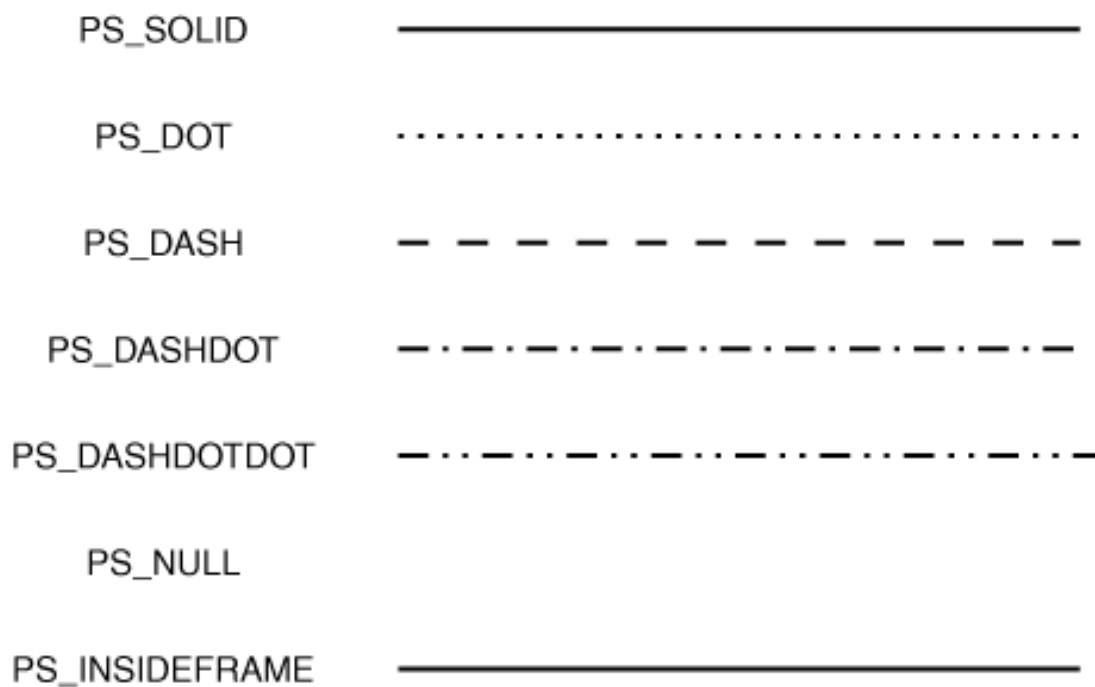


Abbildung 8.1:
Stiftstile in Windows



Alle diese Linienstile ergeben bei einer Stiftdicke größer als 1 durchgehende Linien. Wenn Sie einen anderen Stil als PS_SOLID verwenden möchten, müssen Sie eine Stiftbreite von 1 angeben.

Zusammen mit dem Linienstil, den der Stift zeichnen soll, kann man auch die Breite und Farbe des Stifts festlegen. Die Kombination dieser drei Variablen legt die Erscheinung der resultierenden Linien fest. Die Linienbreite kann Werte ab einschließlich 1 erhalten, obwohl sich bei einer Breite über 32 kaum noch eine vernünftige Linie zeichnen lässt.

Die Farbe ist als RGB-Wert anzugeben, der sich aus drei separaten Werten für die Helligkeit der roten, grünen und blauen Komponenten der Pixel auf dem Computerbildschirm zusammensetzt. Die einzelnen Komponenten können Werte im Bereich zwischen 0 und 255 annehmen. Die Funktion **RGB** faßt die Einzelwerte in einem Format zusammen, das Windows für die Ausgabe benötigt. Einige der gebräuchlicheren Farben sind in Tabelle 8.1 aufgeführt.

**Tabelle 8.1: Gebräuchliche
Windows-Farben**

Farbe	Rot	Grün	Blau
Schwarz	0	0	0
Blau	0	0	255
Dunkelblau	0	0	128
Grün	0	255	0
Dunkelgrün	0	128	0
Cyan	0	255	255
Dunkelcyan	0	128	128
Rot	255	0	0
Dunkelrot	128	0	0

Magenta	255	0	255
Dunkelmagenta	128	0	128
Gelb	255	255	0
Dunkelgelb	128	128	0
Dunkelgrau	128	128	128
Hellgrau	192	192	192
Weiß	255	255	255

Die Pinselklasse CBrush

Mit der Pinselklasse `CBrush` lassen sich Pinsel erzeugen, die definieren, wie Bereiche ausgefüllt werden. Bei einer geschlossenen Figur zeichnet man den Umriss mit dem aktuellen Stift und füllt den Innenbereich mit dem aktuellen Pinsel aus. Pinsel können durchgängige Farben (mit den gleichen RGB-Werten wie die Stifte festzulegen), Linienmuster oder sogar wiederholte Muster, die sich aus einem kleinen Bitmap aufbauen, liefern. Wollen Sie einen Pinsel mit durchgehender Farbe erzeugen, geben Sie dazu die entsprechende Farbe an:

```
CBrush lSolidBrush(RGB(255, 0, 0));
```

Um einen gemusterten Pinsel zu erzeugen, müssen Sie nicht nur die Farbe festlegen, sondern auch das Muster:

```
CBrush lPatternBrush(HS_BSDIAGONAL, RGB(0, 0, 255));
```

Nachdem Sie einen Pinsel erzeugt haben, können Sie ihn mit dem Gerätekontextobjekt auswählen, genau wie es bei Stiften geschieht. Nach der Auswahl eines Pinsels dient er als aktueller Pinsel, wenn man irgendein Objekt zeichnet, das einen Pinsel verwendet.

Wie bei Stiften kann man Pinsel mit einer Reihe von Standardmustern erzeugen, die Abbildung 8.2 zeigt. Neben diesen Mustern gibt es einen zusätzlichen Pinselstil, `HS_BITMAP`, der ein Bitmap als Muster zum Füllen des angegebenen Bereichs verwendet. Das Bitmap ist auf die Größe von 8 mal 8 Pixel begrenzt und damit sogar kleiner, als die normalerweise für Symbolleisten und andere kleine Bilder verwendeten Bitmaps. Gibt man ein größeres Bitmap an, nimmt es nur die 8 mal 8 Punkte ab der oberen linken Ecke ein. Man kann auch einen Bitmap-Pinsel erzeugen, indem man eine Bitmap-Ressource für die Anwendung anlegt und ihr eine Objekt-ID zuweist. Danach können Sie einen Pinsel mit Hilfe des folgenden Codes erzeugen:

```
// Das Bild laden
m_bmpBitmap.LoadBitmap(IDB_MYBITMAP);
// Den Pinsel erzeugen
CBrush lBitmapBrush(&m_bmpBitmap);
```



Wenn Sie Ihre eigenen benutzerdefinierten Muster für einen Pinsel erzeugen möchten, können Sie das Muster als Bitmap der Größe 8 mal 8 Pixel erstellen und den Bitmap-Pinsel verwenden. Damit läßt sich die Zahl der Pinselmuster weit über die begrenzte Anzahl der Standardmuster erweitern.

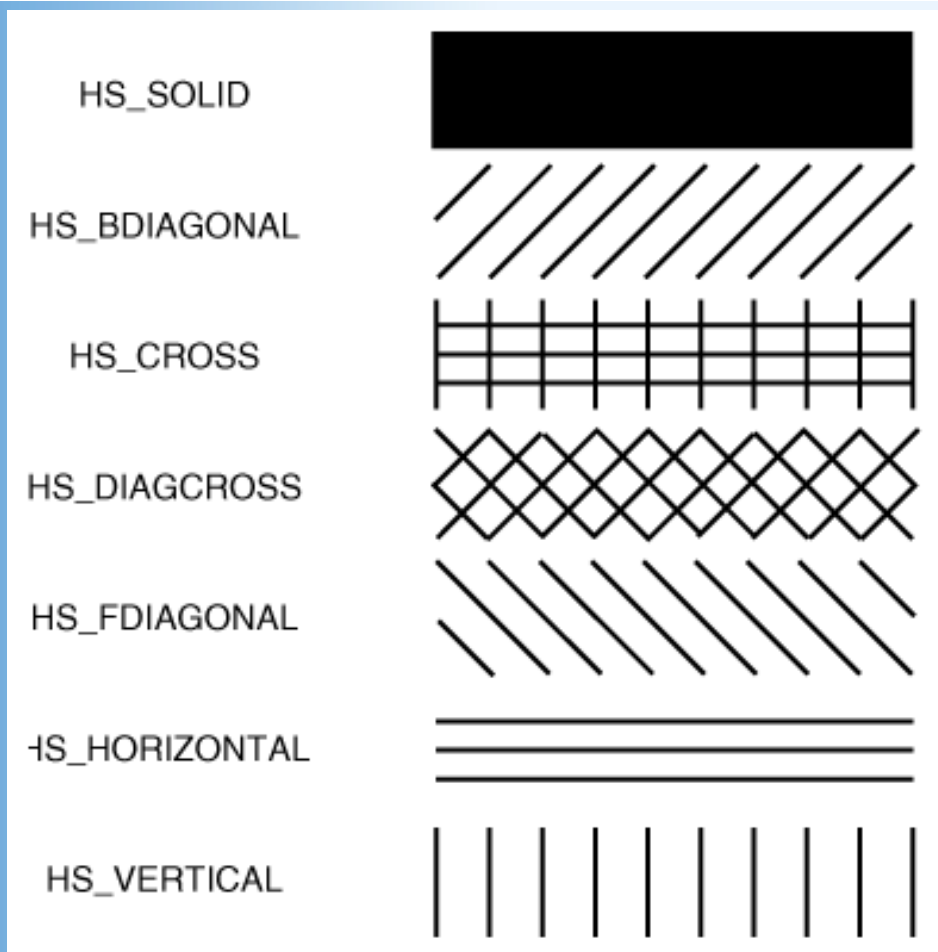


Abbildung 8.2:
Standardpinselmuster

Die Klasse CBitmap

Es gibt verschiedene Möglichkeiten, um Bilder in einer Anwendung anzuzeigen. Feststehende Bitmaps nimmt man als Ressourcen mit zugewiesenen Objekt-IDs in die Anwendung auf und zeigt sie mit statischen Bildsteuerelementen oder ActiveX-Steuerelementen an. Man kann auch die Bitmap-Klasse `CBitmap` einsetzen, um die Anzeige der Bilder weitgehend beeinflussen zu können. Mit Hilfe der Bitmap-Klasse lassen sich Bitmaps dynamisch aus Dateien des Systemlaufwerks laden, die Bilder bei Bedarf in der Größe ändern und sie an den zugewiesenen Platz anpassen.

Wenn man das Bitmap als Ressource hinzufügt, erzeugt man eine Instanz der Klasse `CBitmap` und verwendet die Ressourcen-ID des Bitmap für das zu ladende Bild. Über die API-Funktion `LoadImage` läßt sich ein Bitmap aus einer Datei laden. Nachdem Sie das Bitmap geladen haben, können Sie das Bild über dessen Kennnummer (Handle) mit der Klasse `CBitmap` verbinden, wie es der folgende Beispielcode zeigt:

```
// Die Bitmap-Datei laden
HBITMAP hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
m_sFileName, IMAGE_BITMAP, 0, 0,
LR_LOADFROMFILE | LR_CREATEDIBSECTION);
// Das geladene Bild dem CBitmap-Objekt zuweisen
m_bmpBitmap.Attach(hBitmap);
```

Nachdem Sie das Bitmap in das `CBitmap`-Objekt geladen haben, können Sie einen zweiten Gerätekontext erzeugen und das Bitmap in diesen Gerätekontext selektieren. Wenn Sie den zweiten Gerätekontext erzeugt haben, müssen Sie ihn mit dem ersten Gerätekontext kompatibel machen, bevor das Bitmap in ihn selektiert wird. Da Gerätekontexte durch das Betriebssystem für ein bestimmtes Ausgabegerät (Bildschirm, Drucker usw.) erzeugt werden, müssen Sie sicherstellen, daß der zweite Gerätekontext ebenfalls mit demselben Ausgabegerät wie der erste verbunden ist.

```
// Einen Gerätekontext erzeugen
CDC dcMem;
// Neuen Gerätekontext zum originalen DC kompatibel machen
dcMem.CreateCompatibleDC(dc);
// Bitmap in den neuen DC selektieren
dcMem.SelectObject(&m_bmpBitmap);
```

Wenn Sie das Bitmap in einen kompatiblen Gerätekontext selektieren, können Sie das Bitmap in den normalen Anzeigegerätekontext mit Hilfe der Funktion `BitBlt` kopieren:

```
// Das Bitmap in den Anzeige-DC kopieren
dc->BitBlt(10, 10, bm.bmWidth,
bm.bmHeight, &dcMem, 0, 0,
SRCCOPY);
```

Sie können das Bild auch mittels der Funktion `StretchBlt` kopieren und in der Größe ändern:

```
// Bitmap in den Anzeige-DC kopieren und dabei
// in der Größe ändern
dc->StretchBlt(10, 10, (lRect.Width() - 20),
(lRect.Height() - 20), &dcMem, 0, 0,
bm.bmWidth, bm.bmHeight, SRCCOPY);
```

Mit der Funktion `StretchBlt` läßt sich das Bild an jeden beliebigen Bereich auf dem Bildschirm anpassen.

Abbildungsmodi und Koordinatensysteme

Wenn Sie die Grafikausgabe in ein Fenster vorbereiten, können Sie die Skalierung und den Zielbereich in weiten Grenzen festlegen. Diese Faktoren lassen sich über den Abbildungsmodus und den Zeichenbereich spezifizieren.

Mit dem Abbildungsmodus bestimmen Sie, wie die spezifizierten Koordinaten in Positionen auf dem Bildschirm zu übersetzen sind. Für die einzelnen Abbildungsmodi sind dabei verschiedene Maßeinheiten definiert. Die Abbildungsmodi legen Sie mit der Gerätekontextfunktion `SetMapMode` fest:

```
dc->SetMapMode(MM_ANSIOTROPIC);
```

Die verfügbaren Abbildungsmodi sind in Tabelle 8.2 aufgeführt.

Tabelle 8.2: Abbildungsmodi

Modus	Beschreibung
MM_ANISOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit frei skalierbaren Achsen abgebildet.
MM_HIENGLISH	Eine logische Einheit wird auf 0.001 Zoll abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_HIMETRIC	Eine logische Einheit wird auf 0.01 Millimeter abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_ISOTROPIC	Logische Einheiten werden auf beliebige Einheiten mit gleich skalierten Achsen abgebildet.
MM_LOENGLISH	Eine logische Einheit wird auf 0.01 Zoll abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_LOMETRIC	Eine logische Einheit wird auf 0.1 Millimeter abgebildet. Positive x-Werte gehen nach rechts, positive y-Werte nach oben.
MM_TEXT	Eine logische Einheit wird auf 1 Pixel abgebildet. Positive x gehen nach rechts, positive y nach unten.

MM_TWIPS

Eine logische Einheit wird auf 1/20 Punkt (etwa 1/1440 Zoll, auch als »Twip« bezeichnet) abgebildet. Positive x laufen nach rechts, positive y nach oben.

Wenn Sie die Abbildungsmodi `MM_ANISOTROPIC` oder `MM_ISOTROPIC` verwenden, können Sie den Zielbereich für die Grafik entweder mit der Funktion `SetWindowExt` oder mit `SetViewportExt` festlegen.

Eine Grafikanwendung erstellen

Um einen Einblick von den genannten Möglichkeiten zu bekommen, erstellen Sie heute eine Beispielanwendung, die einen Großteil des bisherigen Stoffes in der Praxis umsetzt. Die Anwendung verfügt über zwei unabhängige Fenster. Das erste enthält eine Anzahl von Optionen, über die man die Figur, das Werkzeug und die Farbe zur Anzeige auswählen kann. Das andere Fenster dient als Leinwand, wo alle ausgewählten Optionen gezeichnet werden. Der Benutzer kann wählen, ob er Linien, Quadrate, Kreise oder ein Bitmap in das zweite Fenster zeichnen will. Außerdem kann er die Farbe spezifizieren und wählen, ob der Stift oder der Pinsel für die Kreise und Quadrate anzuzeigen ist.

Das Anwendungsgerüst erstellen

Wie Sie bisher gelernt haben, erstellen Sie im ersten Schritt das Gerüst einer Anwendung mit der grundlegenden Funktionalität. Es zeigt das erste Dialogfeld der Anwendung an und verfügt über den gesamten Code, der erforderlich ist, um die Anwendung zu starten und zu beenden.

Für die heutige Beispielanwendung beginnen Sie mit dem Gerüst einer dialogfeldbasierenden Anwendung. Dazu legen Sie mit dem Anwendungs-Assistenten ein neues Projekt an, dem Sie einen passenden Namen wie etwa **Grafik** geben. Im ersten Schritt des Anwendungs-Assistenten legen Sie fest, daß Sie eine dialogfeldbasierende Anwendung erstellen wollen. Prinzipiell können Sie dann alle weiteren Standardeinstellungen übernehmen, obwohl die Unterstützung für ActiveX-Steuerelemente nicht erforderlich ist. Außerdem können Sie bei Bedarf einen anderen Titel für das Anwendungsfenster eingeben.

Das Hauptdialogfeld entwerfen

Nachdem Sie sich durch den Anwendungs-Assistenten gearbeitet haben, können Sie an den Entwurf des Hauptdialogfelds gehen. Dieses Fenster enthält drei Gruppen von Optionsfeldern: eine Gruppe zur Auswahl des Zeichenwerkzeugs, die nächste zur Festlegung der Zeichenfigur und das dritte zur Auswahl der Farbe. Neben diesen Gruppen von Optionsfeldern platzieren Sie noch zwei Schaltflächen im Fenster: über die eine Schaltfläche rufen Sie das Dialogfeld **Öffnen** auf, um ein anzuzeigendes Bitmap auszuwählen, und mit der zweiten Schaltfläche schließen Sie die Anwendung.

Die Steuerelemente können Sie gemäß Abbildung 8.3 anordnen. Tabelle 8.3 listet die zugehörigen Eigenschaften der Steuerelemente auf.

Tabelle 8.3: Eigenschaftseinstellungen der Steuerelemente

Objekt	Eigenschaft	Einstellung
Gruppenfeld	ID	IDC_STATIC
	Titel	Werkzeug
Optionsfeld	ID	IDC_RTPEN
	Titel	&Stift
	Gruppe	Eingeschaltet

Optionsfeld	ID	IDC_RTBRUSH
	Titel	&Pinsel
Optionsfeld	ID	IDC_RTBITMAP
	Titel	Bit&map
Gruppenfeld	ID	IDC_STATIC
	Titel	Figur
Optionsfeld	ID	IDC_RSLINE
	Titel	&Linie
	Gruppe	Eingeschaltet
Optionsfeld	ID	IDC_RSCIRCLE
	Titel	&Kreis
Optionsfeld	ID	IDC_RSSQUARE
	Titel	&Quadrat
Gruppenfeld	ID	IDC_STATIC
	Titel	Farbe
Optionsfeld	ID	IDC_RCBLACK
	Titel	Schwar&z
	Gruppe	Eingeschaltet
Optionsfeld	ID	IDC_RCBLUE
	Titel	Bla&u
Optionsfeld	ID	IDC_RCGREEN
	Titel	Gr&ün
Optionsfeld	ID	IDC_RCCYAN
	Titel	&Cyan
Optionsfeld	ID	IDC_RCRED
	Titel	&Rot
Optionsfeld	ID	IDC_RCMAGENTA
	Titel	M&agenta
Optionsfeld	ID	IDC_RCYELLOW
	Titel	&Gelb
Optionsfeld	ID	IDC_RCWHITE
	Titel	&Weiß
Schaltfläche	ID	IDC_BBITMAP
	Titel	B&itmap

Schaltfläche	ID	IDC_BEXIT
	Titel	&Beenden

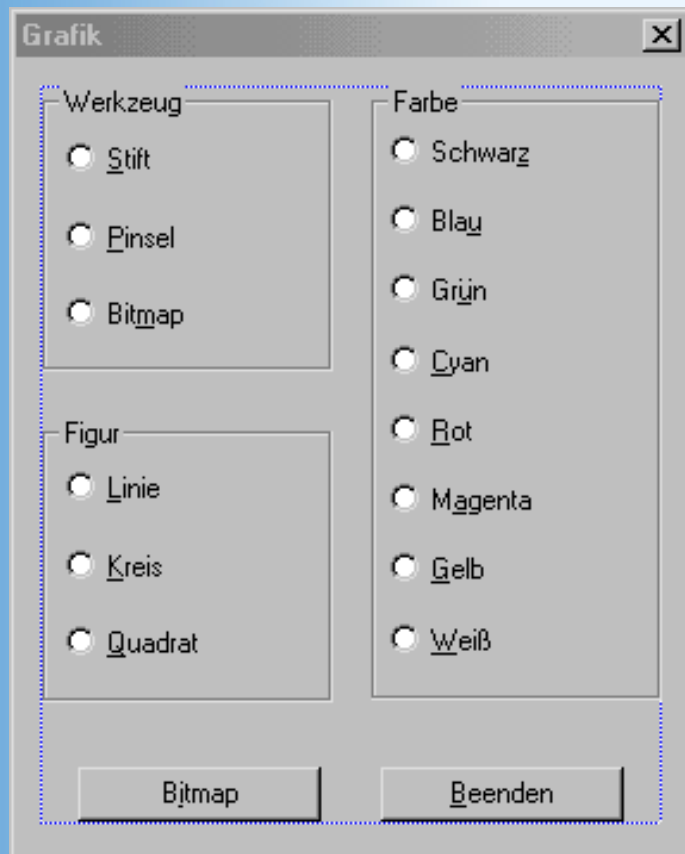


Abbildung 8.3:
Das Layout des Hauptdialogfelds

Nachdem Sie das Hauptdialogfeld entworfen haben, weisen Sie jeder Gruppe der Optionsfelder jeweils eine Variable zu. Öffnen Sie dazu den Klassen-Assistenten, und weisen Sie eine Integer-Variable an jede der drei Objekt-IDs der Optionsfelder zu. Beachten Sie dabei, daß nur die Objekt-IDs für die Optionsfelder, bei denen die Option Gruppe eingeschaltet ist, im Klassen-Assistent erscheinen. Alle darauffolgenden Optionsfelder bekommen dieselbe Variable zugewiesen und erhalten aufeinanderfolgende Werte in der Reihenfolge der Objekt-IDs. Aus diesem Grund ist es wichtig, alle Optionsfelder der einzelnen Gruppen in der Reihenfolge zu erstellen, in der die Werte aufeinanderfolgen sollen.

Um die erforderlichen Variablen an die Gruppen der Optionsfelder der Beispielanwendung zuzuweisen, öffnen Sie den Klassen-Assistenten und fügen die Variablen gemäß Tabelle 8.4 für die Objekte des Dialogfelds hinzu.

Tabelle 8.4: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_RTPEN	m_iTool	Wert	int
IDC_RSLINE	m_iShape	Wert	int
IDC_RCBLACK	m_iColor	Wert	int

Da der Klassen-Assistent geöffnet ist, können Sie gleich zur ersten Registerkarte wechseln und eine Behandlungsroutine für die Schaltfläche **Beenden** hinzufügen. Im Code für die Schaltfläche rufen Sie wie gewohnt die Funktion `OnOK` auf. Jetzt können Sie Ihre Anwendung kompilieren und ausführen. Überzeugen Sie sich davon, daß Sie alle Gruppen der Optionsfelder korrekt definiert haben, daß Sie nur jeweils ein Optionsfeld in einer Gruppe auswählen können und daß Sie eine Option in der einen Gruppe wählen können, ohne eine der anderen Gruppen zu beeinflussen.

Das zweite Dialogfeld hinzufügen

Nachdem Sie das Hauptdialogfeld erstellt haben, fügen Sie das zweite Fenster hinzu, auf dem Sie in der Art einer Leinwand Ihre Grafiken zeichnen. Es handelt sich um ein nichtmodales Dialogfeld, das die ganze Zeit, während die Anwendung läuft, geöffnet bleibt. In dieses Dialogfeld nehmen Sie keine Steuerelemente auf und stellen damit eine vollkommen leere Leinwand zum Zeichnen bereit.

Um das zweite Dialogfeld zu erstellen, gehen Sie im Arbeitsbereich auf die Registerkarte Ressourcen. Klicken Sie im Ressourcenbaum mit der rechten Maustaste auf den Ordner `Dialog`. Wählen Sie aus dem Kontextmenü den Befehl **Dialog einfügen**. Wenn das neue Dialogfeld im Dialog-Editor geöffnet wird, entfernen Sie alle Steuerelemente aus dem Fenster. Anschließend öffnen Sie das Dialogfeld **Eigenschaften** für das Fenster und schalten das Kontrollkästchen Systemmenü auf der zweiten Registerkarte des Eigenschaftsdialogfelds aus. Damit verhindert man, daß der Benutzer das Dialogfeld schließt, ohne die Anwendung zu beenden. Weiterhin geben Sie diesem Dialogfeld eine Objekt-ID, die seine Funktion beschreibt, beispielsweise `IDD_PAINT_DLG`.

Haben Sie den Entwurf des zweiten Dialogfelds abgeschlossen, erstellen Sie für dieses Fenster mit dem Klassen-Assistenten eine neue Klasse. Wenn Sie den Klassen-Assistenten öffnen, erscheint eine Frage, ob Sie eine neue Klasse für das zweite Dialogfeld erstellen möchten. Übernehmen Sie die Standardeinstellung, und klicken Sie auf die Schaltfläche OK. Im nächsten Dialogfeld geben Sie den Namen der neuen Klasse ein, beispielsweise `CPaintDlg`. Stellen Sie sicher, daß die Basisklasse auf `CDialog` gesetzt ist. Nachdem Sie in diesem Dialogfeld auf OK geklickt und die neue Klasse erstellt haben, können Sie den Klassen-Assistenten schließen.



Das neue Dialogfeld muß markiert sein, wenn Sie den Klassen-Assistenten öffnen. Wenn das Dialogfeld nicht markiert ist und Sie zu einem anderen Objekt oder sogar zu Code in Ihrer Anwendung gewechselt haben, weiß der Klassen-Assistent nicht, daß Sie eine Klasse für das zweite Dialogfeld in Ihrer Anwendung benötigen.

Für das erste Dialogfeld fügen Sie nun Code hinzu, um das zweite Dialogfeld zu öffnen. Das erreichen Sie mit zwei Codezeilen in der Funktion `OnInitDialog` in der Klasse des ersten Fensters. Zuerst erstellen Sie das Dialogfeld mit der Methode `Create` der Klasse `CDialog`. Diese Funktion übernimmt zwei Argumente: die Objekt-ID des Dialogfelds und einen Zeiger auf das übergeordnete Fenster, bei dem es sich um das Hauptdialogfeld handelt. Die zweite Funktion ist `ShowWindow`, der Sie den Wert `SW_SHOW` als einziges Argument übergeben. Diese Funktion zeigt das zweite Dialogfeld neben dem ersten an. Schließlich kommen noch einige Zeilen hinzu, um die Variablen zu initialisieren. Die Funktion `OnInitDialog` sollte dann Listing 8.1 entsprechen.

Listing 8.1: Die Funktion `OnInitDialog`

```
1: BOOL CGrafikDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:     .
6:     .
7:     .
8:
9: 27:
10: 28: // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
11: 29:
12: 30: //////////////////////////////////////
13: 31: // EIGENER CODE, ANFANG
14: 32: //////////////////////////////////////
15: 33:
16: 34: // Variablen initialisieren und Dialogfeld aktualisieren
17: 35: m_iColor = 0;
18: 36: m_iShape = 0;
```

```

37: m_iTool = 0;
38: UpdateData(FALSE);
39:
40: // Das zweite Dialogfeld erzeugen
41: m_dlgPaint.Create(IDD_PAINT_DLG, this);
42: // Zweites Dialogfeld anzeigen
43: m_dlgPaint.ShowWindow(SW_SHOW);
44:
45: //////////////////////////////////////
46: // EIGENER CODE, ENDE
47: //////////////////////////////////////
48:
49: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement soll den
  Fokus erhalten
50: }

```

Bevor Sie die Anwendung kompilieren und ausführen können, müssen Sie die Header-Datei für die zweite Dialogfeldklasse in den Quellcode des ersten Dialogfelds einbinden. Weiterhin ist die zweite Dialogfeldklasse als Variable in die erste aufzunehmen. Zu diesem Zweck müssen Sie lediglich eine Member-Variable in die erste Dialogfeldklasse hinzufügen, wobei Sie als Variablentyp den Klassentyp - in diesem Fall `CPaintDlg` - spezifizieren, den in Listing 8.1 verwendeten Variablennamen `m_dlgPaint` angeben und den Variablenzugriff als Privat festlegen. Um die Header-Datei in das erste Dialogfeld einzubinden, gehen Sie an den Beginn des Quellcodes für das erste Dialogfeld und fügen eine `#include`-Anweisung wie in Listing 8.2 hinzu.

Listing 8.2: Die `#include`-Anweisung des Hauptdialogs

```

1: // GrafikDlg.cpp : Implementierungsdatei
2: //
3:
4: #include "stdafx.h"
5: #include "Grafik.h"
6: #include "PaintDlg.h"
7: #include "GrafikDlg.h"
8:

```

Umgekehrt müssen Sie die Header-Datei für das Hauptdialogfeld in den Quellcode für das zweite Dialogfeld einbinden. Bearbeiten Sie die Datei `PaintDlg.cpp`, so daß die `#include`-Anweisungen denen in Listing 8.2 entsprechen.

Wenn Sie Ihre Anwendung kompilieren und ausführen, sollte das zweite Dialogfeld zusammen mit dem ersten geöffnet werden. Schließt man das erste Dialogfeld und beendet damit die Anwendung, wird das zweite Dialogfeld ebenfalls geschlossen, selbst wenn Sie bisher keinerlei Code hinzugefügt haben, der das bewirkt. Das zweite Dialogfeld ist ein untergeordnetes Fenster zum ersten Dialogfeld. Wenn Sie das zweite Dialogfeld - in Zeile 41 des Listings - erstellen, übergeben Sie einen Zeiger auf das erste Dialogfeld als übergeordnetes Fenster für das zweite Fenster. Damit wird eine Beziehung zwischen übergeordnetem und untergeordnetem Fenster eingerichtet. Wird das übergeordnete Fenster geschlossen, schließt auch das untergeordnete. Es handelt sich hier um die gleiche Beziehung, die das erste Dialogfeld mit allen darauf platzierten Steuerelementen hat. Jedes dieser Steuerelemente ist ein untergeordnetes Fenster des Dialogfelds. Praktisch haben Sie das zweite Dialogfeld zu einem weiteren Steuerelement des ersten Dialogfelds gemacht.

Grafikfunktionen realisieren

Da alle Variablen für die Optionsfelder als öffentlich (`public`) deklariert sind, kann sie das zweite Dialogfeld sehen und referenzieren. Die gesamte Funktionalität zum Zeichnen läßt sich in der zweiten Dialogfeldklasse unterbringen. Allerdings müssen Sie einen Teil der Funktionen im ersten Dialogfeld realisieren, um die Variablen zu synchronisieren und das zweite Dialogfeld anzuweisen, die Grafiken zu zeichnen. Das Ganze ist aber einfacher, als Sie vielleicht annehmen.

Sobald ein Fenster neu zu zeichnen ist (etwa weil es hinter einem anderen Fenster verborgen war und nun in den Vordergrund kommt, oder weil es minimiert war, oder weil es außerhalb des sichtbaren Bereichs gelegen hat), löst das Betriebssystem die Funktion `OnPaint` des Dialogfelds aus. Die gesamte Funktionalität zum Zeichnen der Grafiken können Sie in dieser Funktion unterbringen und die angezeigten Grafiken dauerhaft machen.

Es ist nun klar, wo der Code für die Anzeige der Grafiken hingehört. Wie kann man aber das zweite Dialogfeld veranlassen, seine `OnPaint`-Funktion aufzurufen, wenn der Benutzer eine der Auswahlen im ersten Dialogfeld verändert? Nun, man kann das zweite Dialogfeld verbergen und dann wieder anzeigen, aber das dürfte dem Benutzer etwas seltsam vorkommen. Eigentlich »überzeugt« eine einzige Funktion das zweite Fenster, daß es sein gesamtes Dialogfeld neu zu zeichnen hat. Diese Funktion, `Invalidate`, erfordert keine Argumente und ist eine Elementfunktion der Klasse `CWnd`, so daß man sie für jedes Fenster oder Steuerelement einsetzen kann. Die Funktion `Invalidate` teilt dem Fenster - und dem Betriebssystem - mit, daß der Anzeigebereich des Fensters nicht mehr gültig und neu zu zeichnen ist. Man kann die Funktion `OnPaint` im zweiten Dialogfeld nach Belieben aufrufen, ohne daß man auf irgendwelche ausgefallenen Tricks oder Hacks zurückgreifen müßte.

Wie Sie mittlerweile wissen, können alle Optionsfelder die gleiche Funktionalität für ihre Klickereignisse verwenden. Somit genügt eine einzige Behandlungsroutine für das Klickereignis aller Optionsfelder. In dieser Behandlungsfunktion synchronisieren Sie die Klassenvariablen mit den Steuerelementen des Dialogfelds durch Aufruf der Funktion `UpdateData` und weisen dann das zweite Dialogfeld mit Hilfe seiner Funktion `Invalidate` an, sich selbst neu zu zeichnen. Mit dem Code in Listing 8.3 erstellen Sie eine einzige Behandlungsroutine, die diese beiden Dinge erledigt.

Listing 8.3: Die Funktion `OnRSelection`

```
1: void CGrafikDlg::OnRSelection()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÄBenachrichtigung hier einfügen
4:
5: // Daten synchronisieren
6: UpdateData(TRUE);
7: // Zweites Dialogfeld neu zeichnen
8: m_dlgPaint.Invalidate();
9: }
```

Linien zeichnen

Wenn Sie die Anwendung jetzt kompilieren und ausführen, zeichnet sich das zweite Dialogfeld neu, wenn Sie ein anderes Optionsfeld im Hauptdialogfeld wählen. Davon bemerken Sie allerdings gar nichts. Sie haben zwar das Neuzeichnen initiiert, aber dem zweiten Dialogfeld noch nicht mitgeteilt, was zu zeichnen ist. Das erledigen wir nun im nächsten Schritt für die Beispielanwendung.

Am einfachsten lassen sich Linien in verschiedenen Stilen im zweiten Dialogfeld zeichnen, weil Sie bereits etwas Erfahrung damit haben. Für die einzelnen Stiftstile ist jeweils ein Stift zu erzeugen, der die momentan ausgewählte Farbe verwendet. Nachdem Sie alle Stifte erstellt haben, gehen Sie in einer Schleife durch die einzelnen Stifte, selektieren sie nacheinander und zeichnen mit dem jeweiligen Stift eine Linie über das Dialogfeld. Bevor Sie diese Schleife starten können, müssen Sie ein paar Berechnungen anstellen, um die Lage der Linien - die Anfangs- und Endpunkte - im Dialogfeld zu bestimmen.

Zunächst erstellen Sie eine Farbtabelle, die je einen Eintrag für die Optionen des Gruppenfelds Farben im ersten Dialogfeld hat. Dazu fügen Sie dem zweiten Dialogfeld (`CPaintDlg`) eine Elementvariable hinzu. Legen Sie den Variablentyp als `static const COLORREF`, den Namen als `m_crColors[8]` und den Zugriff als `Public` fest. Öffnen Sie den Quellcode der Datei für die zweite Dialogfeldklasse, und fügen Sie die Farbtabelle aus Listing 8.4 am Beginn der Datei vor dem Konstruktor und Destruktor der Klasse ein.

Listing 8.4: Die Farbtabelle

```
1: const COLORREF CPaintDlg::m_crColors[8] = {
```

```

2: RGB( 0, 0, 0), // Schwarz
3: RGB( 0, 0, 255), // Blau
4: RGB( 0, 255, 0), // Grün
5: RGB( 0, 255, 255), // Cyan
6: RGB( 255, 0, 0), // Rot
7: RGB( 255, 0, 255), // Magenta
8: RGB( 255, 255, 0), // Gelb
9: RGB( 255, 255, 255) // Weiß
10: };
11:
////////////////////////////////////
12: // Dialogfeld CPaintDlg
.
.
.

```

Die Farbtabelle befindet sich nun am richtigen Platz, und Sie können eine neue Funktion hinzufügen, um die Linien zu zeichnen. Um die Funktion `OnPaint` nicht übermäßig kompliziert und unüberschaubar zu machen, ist es sinnvoller, nur einen kleinen Teil des Codes in dieser Funktion unterzubringen. Mit diesem Code ermittelt man lediglich, was im zweiten Dialogfeld zu zeichnen ist, und ruft dann spezialisierte Funktionen auf, die die verschiedenartigen Figuren zeichnen. In diesem Sinne fügen Sie eine neue Member-Funktion für die zweite Dialogfeldklasse zum Zeichnen von Linien hinzu. Legen Sie den Typ der Funktion als `void`, die Deklaration als `DrawLine(CPaintDC *pdc, int iColor)` und den Zugriff als Privat fest. Schreiben Sie den Code aus Listing 8.5 in diese Funktion.

Listing 8.5: Die Funktion `DrawLine`

```

1: void CPaintDlg::DrawLine(CPaintDC *pdc, int iColor)
2: {
3: // Stifte deklarieren und erzeugen
4: CPen lSolidPen (PS_SOLID, 1, m_crColors[iColor]);
5: CPen lDotPen (PS_DOT, 1, m_crColors[iColor]);
6: CPen lDashPen (PS_DASH, 1, m_crColors[iColor]);
7: CPen lDashDotPen (PS_DASHDOT, 1, m_crColors[iColor]);
8: CPen lDashDotDotPen (PS_DASHDOTDOT, 1, m_crColors[iColor]);
9: CPen lNullPen (PS_NULL, 1, m_crColors[iColor]);
10: CPen lInsidePen (PS_INSIDEFRAME, 1, m_crColors[iColor]);
11:
12: // Zeichenbereich ermitteln
13: CRect lRect;
14: GetClientRect(lRect);
15: lRect.NormalizeRect();
16:
17: // Abstand zwischen den Linien berechnen
18: CPoint pStart;
19: CPoint pEnd;
20: int liDist = lRect.Height() / 8;
21: CPen *lOldPen;
22: // Anfangspunkte festlegen
23: pStart.y = lRect.top;
24: pStart.x = lRect.left;
25: pEnd.y = pStart.y;
26: pEnd.x = lRect.right;
27: int i;
28: // Schleife durch die verschiedenen Stifte
29: for (i = 0; i < 7; i++)

```



```

30: {
31: // Bei welchem Stift sind wir?
32: switch (i)
33: {
34: case 0: // durchgehend
35: lOldPen = pdc->SelectObject(&lSolidPen);
36: break;
37: case 1: // Punkte
38: pdc->SelectObject(&lDotPen);
39: break;
40: case 2: // Striche
41: pdc->SelectObject(&lDashPen);
42: break;
43: case 3: // Strichpunkt
44: pdc->SelectObject(&lDashDotPen);
45: break;
46: case 4: // Strich Punkt Punkt
47: pdc->SelectObject(&lDashDotDotPen);
48: break;
49: case 5: // Unsichtbar
50: pdc->SelectObject(&lNullPen);
51: break;
52: case 6: // Innenseite
53: pdc->SelectObject(&lInsidePen);
54: break;
55: }
56: // Nach unten zur nächsten Position
57: pStart.y = pStart.y + liDist;
58: pEnd.y = pStart.y;
59: // Linie zeichnen
60: pdc->MoveTo(pStart);
61: pdc->LineTo(pEnd);
62: }
63: // Originalstift selektieren
64: pdc->SelectObject(lOldPen);
65: }

```

Jetzt brauchen Sie noch die Funktion `OnPaint`, die die Funktion `DrawLine` im Bedarfsfall aufruft. Fügen Sie die Funktion `OnPaint` über den Klassen-Assistenten als Behandlungsfunktion für die Nachricht `WM_PAINT` hinzu. Es fällt auf, daß der generierte Code für diese Funktion eine `CPaintDC`-Variable statt der normalen `CDC`-Klasse erstellt hat. Die Klasse `CPaintDC` ist von der Gerätekontextklasse `CDC` abgeleitet. Sie ruft automatisch die API-Funktionen `BeginPaint` und `EndPaint` auf, die alle Windows-Anwendungen vor dem Zeichnen während der Verarbeitung der `WM_PAINT`-Nachricht aufrufen müssen. Man kann sie genau wie ein normales Gerätekontextobjekt behandeln und auch die gleichen Funktionen aufrufen.

In der Funktion `OnPaint` ist ein Zeiger auf das übergeordnete Fenster zu ermitteln, damit man die Werte der Variablen überprüfen kann, die mit den Gruppen von Optionsfeldern für die Festlegung von Farbe, Werkzeug und der im zweiten Dialogfeld zu zeichnenden Figur verbunden sind. Aus diesen Informationen leiten Sie ab, ob die Funktion `DrawLine` oder eine andere Funktion, die Sie noch schreiben, aufzurufen ist.

Um diese Funktionalität in die Anwendung aufzunehmen, fügen Sie eine Behandlungsroutine für die Nachricht `WM_PAINT` in die zweite Dialogfeldklasse ein und schreiben den Code aus Listing 8.6 in diese Funktion.

Listing 8.6: Die Funktion `OnPaint`

```

1: void CPaintDlg::OnPaint()
2: {

```

```

3: CPaintDC dc(this); // device context for painting
4:
5: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
6:
7: // Zeiger auf übergeordnetes Fenster ermitteln
8: CGrafikDlg *pWnd = (CGrafikDlg*)GetParent();
9: // Ist der Zeiger gültig?
10: if (pWnd)
11: {
12: // Bitmap als Werkzeug gewählt?
13: if (pWnd->m_iTool == 2)
14: {
15: }
16: else // Nein, Figur zeichnen
17: {
18: // Wird Linie gezeichnet?
19: if (pWnd->m_iShape == 0)
20: DrawLine(&dc, pWnd->m_iColor);
21: }
22: }
23: // Kein Aufruf von CDialog::OnPaint() für Zeichnungsnachrichten
24:}

```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie Linien über das zweite Dialogfeld zeichnen können, wie es Abbildung 8.4 demonstriert.

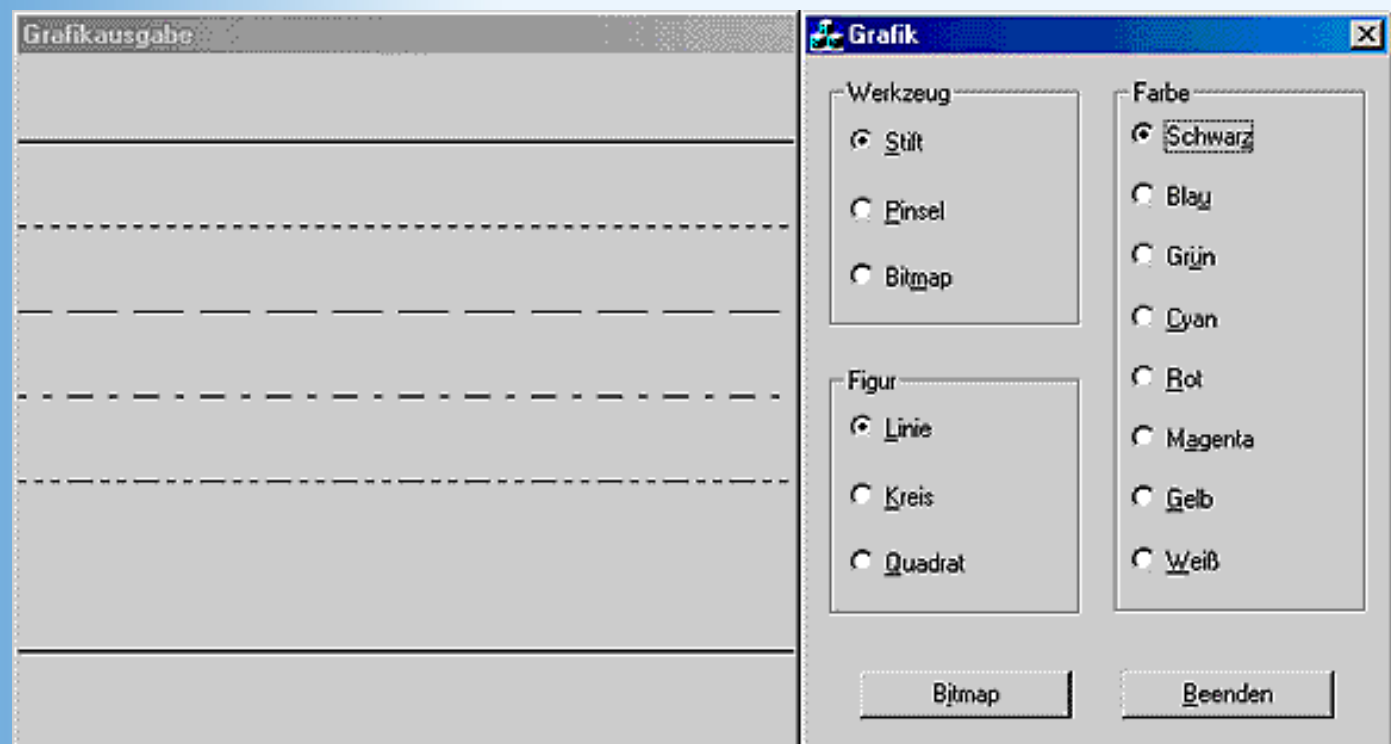


Abbildung 8.4:
Linien im zweiten Dialogfeld zeichnen

Kreise und Quadrate zeichnen

Die grundlegenden Strukturen sind nun realisiert, und Sie wissen, wie man die Ausgaben im zweiten Dialogfeld nach Belieben ändern kann. Jetzt können Sie den Code in das zweite Dialogfeld aufnehmen, um die Kreise und Quadrate zu zeichnen. Für diese Figuren kommen die Gerätekontextfunktionen `Ellipse` und `Rectangle` zum Einsatz. Diese Funktionen verwenden den momentan ausgewählten Stift und Pinsel, um die Figuren an der angegebenen Position zu zeichnen. Bei beiden Funktionen kann man ein `CRect`-Objekt übergeben, um das

Rechteck zu definieren, in dem die spezifizierte Figur darzustellen ist. Die Funktion `Rectangle` füllt den gesamten angegebenen Bereich, während die Funktion `Ellipse` einen Kreis oder eine Ellipse zeichnet, wobei die Mittelpunkte der Rechteckseiten vom Umfang der Ellipse berührt werden.

Da diese Funktionen sowohl den Stift als auch den Pinsel verwenden, müssen Sie einen unsichtbaren Stift bzw. einen unsichtbaren Pinsel erzeugen und selektieren, um dem Benutzer zu ermöglichen, entweder den Stift oder den Pinsel zu wählen. Beim Stift können Sie zu diesem Zweck auf den Null-Stift zurückgreifen, während Sie für den Pinsel einen durchgehenden Pinsel in der Hintergrundfarbe des Fensters (Hellgrau) erzeugen müssen.

Die Positionen für die Figuren berechnen Sie nach einem anderen Verfahren als für die Linien. Bei den Linien konnte man einfach die Höhe des Fensters durch 8 teilen und dann eine Linie in jedem Abschnitt vom linken Rand zum rechten ziehen. Bei Ellipsen und Rechtecken teilen Sie das Dialogfenster in 8 gleich große Rechtecke. Am einfachsten läßt sich das erreichen, indem man zwei Reihen mit je vier Figuren erzeugt. Zwischen jeder Figur bleibt etwas Platz, damit der Benutzer die verschiedenen Stifte für die Umrisse der einzelnen Figuren erkennen kann.

Um diese Funktionalität in die Beispielanwendung einzubauen, fügen Sie für die zweite Dialogfeldklasse eine neue Funktion hinzu. Legen Sie den Funktionstyp als `void`, die Deklaration als `DrawRegion(CPaintDC *pdc, int iColor, int iTool, int iShape)` und den Zugriff als `Privat` fest. In die Funktion schreiben Sie den Code aus Listing 8.7.

Listing 8.7: Die Funktion `DrawRegion`

```
1: void CPaintDlg::DrawRegion(CPaintDC *pdc, int iColor, int iTool, int
iShape)
2: {
3: // Stifte deklarieren und erzeugen
4: CPen lSolidPen (PS_SOLID, 1, m_crColors[iColor]);
5: CPen lDotPen (PS_DOT, 1, m_crColors[iColor]);
6: CPen lDashPen (PS_DASH, 1, m_crColors[iColor]);
7: CPen lDashDotPen (PS_DASHDOT, 1, m_crColors[iColor]);
8: CPen lDashDotDotPen (PS_DASHDOTDOT, 1, m_crColors[iColor]);
9: CPen lNullPen (PS_NULL, 1, m_crColors[iColor]);
10: CPen lInsidePen (PS_INSIDEFRAME, 1, m_crColors[iColor]);
11:
12: // Pinsel deklarieren und erzeugen
13: CBrush lSolidBrush(m_crColors[iColor]);
14: CBrush lBDiagBrush(HS_BDIAGONAL, m_crColors[iColor]);
15: CBrush lCrossBrush(HS_CROSS, m_crColors[iColor]);
16: CBrush lDiagCrossBrush(HS_DIAGCROSS, m_crColors[iColor]);
17: CBrush lFDiagBrush(HS_FDIAGONAL, m_crColors[iColor]);
18: CBrush lHorizBrush(HS_HORIZONTAL, m_crColors[iColor]);
19: CBrush lVertBrush(HS_VERTICAL, m_crColors[iColor]);
20: CBrush lNullBrush(RGB(192, 192, 192));
21:
22: // Größe der Zeichenbereiche berechnen
23: CRect lRect;
24: GetClientRect(lRect);
25: lRect.NormalizeRect();
26: int liVert = lRect.Height() / 2;
27: int liHeight = liVert - 10;
28: int liHorz = lRect.Width() / 4;
29: int liWidth = liHorz - 10;
30: CRect lDrawRect;
31: CPen *lOldPen;
32: CBrush *lOldBrush;
33: int i;
```

```

34: // Schleife durch alle Pinsel und Stifte
35: for (i = 0; i < 7; i++)
36: {
37: switch (i)
38: {
39: case 0: // Durchgehend
40: // Position für diese Figur bestimmen.
41: // Die erste Reihe beginnen
42: lDrawRect.top = lRect.top + 5;
43: lDrawRect.left = lRect.left + 5;
44: lDrawRect.bottom = lDrawRect.top + liHeight;
45: lDrawRect.right = lDrawRect.left + liWidth;
46: // Passenden Stift und Pinsel auswählen
47: lOldPen = pdc->SelectObject(&lSolidPen);
48: lOldBrush = pdc->SelectObject(&lSolidBrush);
49: break;
50: case 1: // Punkt - Diagonal von rechts oben nach links unten
51: // Position für diese Figur bestimmen.
52: lDrawRect.left = lDrawRect.left + liHorz;
53: lDrawRect.right = lDrawRect.left + liWidth;
54: // Passenden Stift und Pinsel auswählen
55: pdc->SelectObject(&lDotPen);
56: pdc->SelectObject(&lBDiagBrush);
57: break;
58: case 2: // Strich - Pinsel kreuzweise
59: // Position für diese Figur bestimmen.
60: lDrawRect.left = lDrawRect.left + liHorz;
61: lDrawRect.right = lDrawRect.left + liWidth;
62: // Passenden Stift und Pinsel auswählen
63: pdc->SelectObject(&lDashPen);
64: pdc->SelectObject(&lCrossBrush);
65: break;
66: case 3: // Strichpunkt - Diagonal kreuzweise
67: // Position für diese Figur bestimmen.
68: lDrawRect.left = lDrawRect.left + liHorz;
69: lDrawRect.right = lDrawRect.left + liWidth;
70: // Passenden Stift und Pinsel auswählen
71: pdc->SelectObject(&lDashDotPen);
72: pdc->SelectObject(&lDiagCrossBrush);
73: break;
74: case 4: // Strich Punkt Punkt - Diagonal von links oben nach rechts
       unten
75: // Position für diese Figur bestimmen.
76: // Zweite Reihe beginnen
77: lDrawRect.top = lDrawRect.top + liVert;
78: lDrawRect.left = lRect.left + 5;
79: lDrawRect.bottom = lDrawRect.top + liHeight;
80: lDrawRect.right = lDrawRect.left + liWidth;
81: // Passenden Stift und Pinsel auswählen
82: pdc->SelectObject(&lDashDotDotPen);
83: pdc->SelectObject(&lFDiagBrush);
84: break;
85: case 5: // Null - Horizontal
86: // Position für diese Figur bestimmen.
87: lDrawRect.left = lDrawRect.left + liHorz;

```

```

88: lDrawRect.right = lDrawRect.left + liWidth;
89: // Passenden Stift und Pinsel auswählen
90: pdc->SelectObject(&lNullPen);
91: pdc->SelectObject(&lHorizBrush);
92: break;
93: case 6: // Innenseite - Vertikal
94: // Position für diese Figur bestimmen.
95: lDrawRect.left = lDrawRect.left + liHorz;
96: lDrawRect.right = lDrawRect.left + liWidth;
97: // Passenden Stift und Pinsel auswählen
98: pdc->SelectObject(&lInsidePen);
99: pdc->SelectObject(&lVertBrush);
100: break;
101: }
102: // Welches Werkzeug wird verwendet?
103: if (iTool == 0)
104: pdc->SelectObject(lNullBrush);
105: else
106: pdc->SelectObject(lNullPen);
107: // Welche Figur wird gezeichnet?
108: if (iShape == 1)
109: pdc->Ellipse(lDrawRect);
110: else
111: pdc->Rectangle(lDrawRect);
112: }
113: // Auf ursprünglichen Stift und Pinsel zurücksetzen
114: pdc->SelectObject(lOldBrush);
115: pdc->SelectObject(lOldPen);
116:}

```

Nachdem Sie nun die Kreise und Quadrate im zweiten Dialogfeld zeichnen können, müssen Sie noch diese Funktion aufrufen, wenn der Benutzer die Figuren entweder mit einem Stift oder einem Pinsel ausgewählt hat. Dazu fügen Sie die Zeilen 21 und 22 von Listing 8.8 in die Funktion `OnPaint` ein.

Listing 8.8: Die modifizierte Funktion `OnPaint`

```

1: void CPaintDlg::OnPaint()
2: {
3: CPaintDC dc(this); // device context for painting
4:
5: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
6:
7: // Zeiger auf übergeordnetes Fenster ermitteln
8: CGrafikDlg *pWnd = (CGrafikDlg*)GetParent();
9: // Ist der Zeiger gültig?
10: if (pWnd)
11: {
12: // Bitmap als Werkzeug gewählt?
13: if (pWnd->m_iTool == 2)
14: {
15: }
16: else // Nein, Figur zeichnen
17: {
18: // Wird Linie gezeichnet?
19: if (m_iShape == 0)
20: DrawLine(&dc, pWnd->m_iColor);

```



```

21: else // Ellipse oder Rechteck zeichnen
22: DrawRegion(&dc, pWnd->m_iColor, pWnd->m_iTool,
&pWnd->m_iShape);
23: }
24: }
25: // Kein Aufruf von CDialog::OnPaint() für Zeichnungsnachrichten
26:}

```

Jetzt können Sie die Anwendung kompilieren und ausführen und sollten nicht nur Linien, sondern auch Quadrate und Kreise anzeigen können, wobei Sie auch zwischen der Anzeige der Umrisse und der ausgefüllten Figuren ohne Randlinien wählen können, wie es Abbildung 8.5 darstellt.

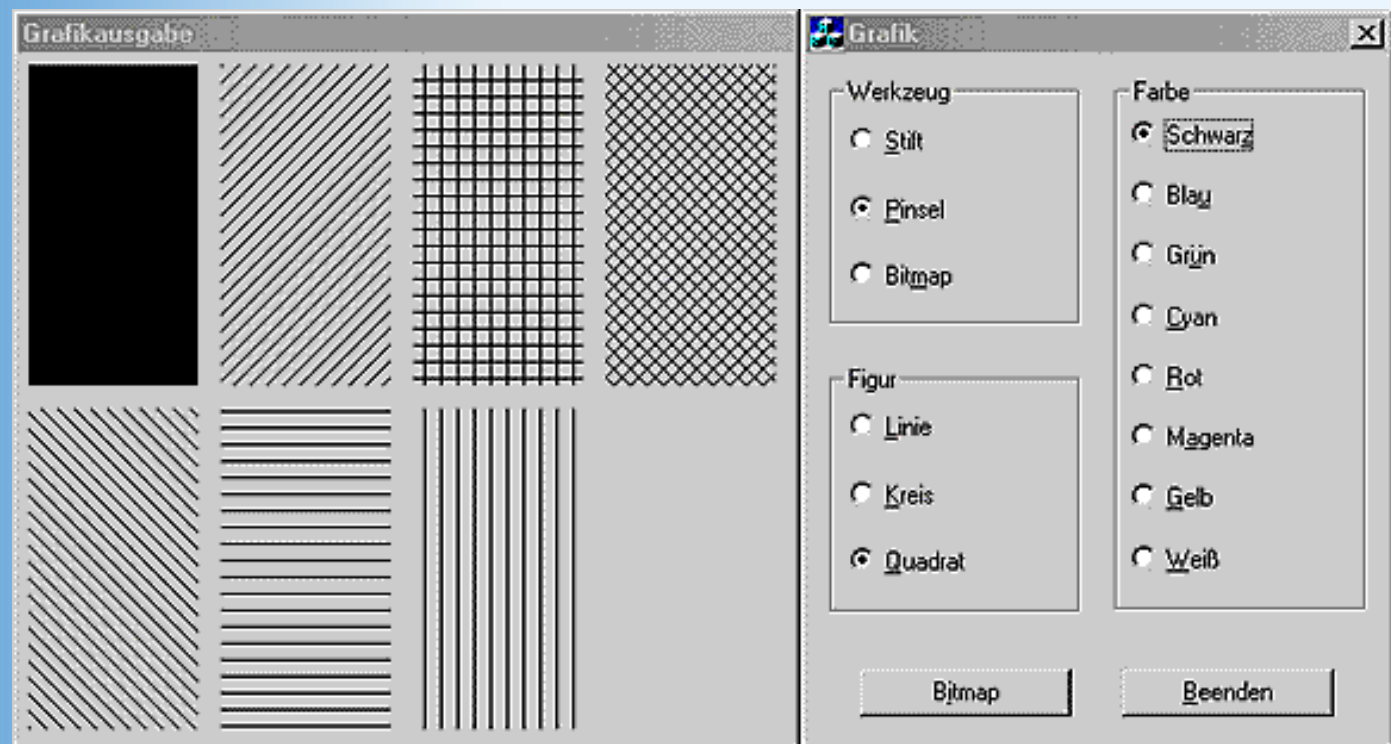


Abbildung 8.5:
Rechtecke im zweiten Dialogfeld zeichnen

Bitmaps laden

Mit der Anwendung lassen sich jetzt verschiedene Grafiken im zweiten Dialogfeld zeichnen. Es ist noch die Funktionalität zu ergänzen, um Bitmaps zu laden und anzuzeigen. Die Bitmaps lassen sich in einfacher Weise zu den Ressourcen in der Anwendung hinzufügen, mit eigenen Objekt-IDs versehen und dann mit der Funktion `LoadBitmap` und dem Makro `MAKEINTRESOURCE` in ein Objekt der Klasse `CBitmap` laden. Allerdings ist das nicht sehr nützlich, wenn man eigene Anwendungen erstellt. Sinnvoll ist es eigentlich nur, wenn man Bitmaps aus Dateien vom Systemlaufwerk laden kann. Das Bitmap laden Sie mit der API-Funktion `LoadImage` in den Speicher und verbinden dann das geladene Bild mit dem `CBitmap`-Objekt.

In Ihrer Anwendung können Sie zu diesem Zweck eine Funktion mit der Schaltfläche `Bitmap` auf dem ersten Dialogfeld verbinden. Die Schaltfläche ruft das Dialogfeld Öffnen auf, in dem der Benutzer ein anzuzeigendes Bitmap auswählen kann. Für das Dialogfeld Öffnen erstellt man einen Filter, um aus den verfügbaren Dateien nur die Bitmaps aufzulisten, die sich im zweiten Dialogfeld anzeigen lassen. Nachdem der Benutzer ein Bitmap ausgewählt hat, ermitteln Sie die Namen für die Datei und den Pfad aus dem Dialogfeld und laden das Bitmap mit der Funktion `LoadImage`. Mit einem gültigen Handle auf das Bitmap, das in den Speicher geladen wurde, löschen Sie das aktuelle Bitmap aus dem `CBitmap`-Objekt. Wurde ein Bitmap in das `CBitmap`-Objekt geladen, lösen Sie das `CBitmap`-Objekt von dem jetzt gelöschten Bild. Nachdem Sie sichergestellt haben, daß im `CBitmap`-Objekt noch kein Bild geladen ist, verbinden Sie das eben in den Speicher geladene Bitmap mit Hilfe der Funktion `Attach`. Jetzt machen Sie das zweite Dialogfeld mit der Funktion `Invalidate` ungültig, so daß ein noch angezeigtes altes Bitmap durch das neuere ersetzt wird.

Um diese Funktionalität zu unterstützen, fügen Sie gemäß Tabelle 8.5 in die erste Dialogfeldklasse eine Stringvariable für den Namen des Bitmaps und eine CBitmap-Variable für das Bitmap selbst hinzu.

Tabelle 8.5: Bitmap-Variablen

Name	Typ	Zugriff
m_sBitmap	CString	Public
m_bmpBitmap	CBitmap	Public

Anschließend fügen Sie mit dem Klassen-Assistenten eine Behandlungsroutine für das Klickereignis der Schaltfläche Bitmap hinzu. In die Funktion schreiben Sie den Code aus Listing 8.9.

Listing 8.9: Die Funktion OnBbitmap

```

1: void CGrafikDlg::OnBbitmap()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: // Filter für Dialogfeld Öffnen erstellen
6: static char BASED_CODE szFilter[] = "Bitmap-Dateien (*.bmp)|*.bmp||";
7: // Dialogfeld Öffnen erzeugen
8: CFileDialog m_ldFile(TRUE, ".bmp", m_sBitmap,
9: OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, szFilter);
10:
11: // Dialogfeld Öffnen anzeigen und Ergebnis übernehmen
12: if (m_ldFile.DoModal() == IDOK)
13: {
14: // Gewählten Dateinamen ermitteln
15: m_sBitmap = m_ldFile.GetPathName();
16: // Gewählte Bitmap-Datei laden
17: HBITMAP hBitmap = (HBITMAP) ::LoadImage(AfxGetInstanceHandle(),
18: m_sBitmap, IMAGE_BITMAP, 0, 0,
19: LR_LOADFROMFILE | LR_CREATEDIBSECTION);
20:
21: // Ist Handle für das geladene Bild gültig?
22: if (hBitmap)
23: {
24: // Aktuelles Bitmap löschen
25: if (m_bmpBitmap.DeleteObject())
26: // War Bitmap vorhanden, lösen
27: m_bmpBitmap.Detach();
28: // Aktuell geladenes Bitmap mit Bitmap-Objekt verbinden
29: m_bmpBitmap.Attach(hBitmap);
30: }
31: // Zweites Dialogfeld ungültig machen
32: m_dlgPaint.Invalidate();
33: }
34: }

```

Bitmaps anzeigen

Nachdem Sie nun Bitmaps in den Arbeitsspeicher laden können, müssen Sie die Bitmaps noch anzeigen. Dazu kopieren Sie das Bitmap mit Hilfe der Funktion `GetBitmap` aus dem `CBitmap`-Objekt in eine `BITMAP`-Struktur. Die Funktion `GetBitmap` ermittelt die Breite und Höhe des Bitmaps. Als nächstes erzeugen

Sie einen neuen Gerätekontext, der mit dem Bildschirmgerätekontext kompatibel ist. Selektieren Sie das Bitmap in den neuen Gerätekontext, und kopieren Sie es dann von diesem zweiten Gerätekontext in den originalen Gerätekontext, wobei Sie während des Kopiervorgangs die Größe mit Hilfe der Funktion `StretchBlt` anpassen.

Um diese Funktionalität in der Beispielanwendung zu realisieren, fügen Sie eine neue Member-Funktion in die zweite Dialogfeldklasse ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `ShowBitmap(CPaintDC *pdc, CWnd *pWnd)` und den Zugriff als Privat fest. In die Funktion übernehmen Sie den Code aus Listing 8.10.



Beachten Sie, daß der übergebene Fensterzeiger als Zeiger auf ein `CWnd`- Objekt deklariert ist und nicht als Klassentyp Ihres Hauptdialogfelds. Um ihn als Zeiger auf den Klassentyp des ersten Dialogfelds zu deklarieren, müssen Sie die Klasse für das erste Dialogfeld vor der Klassendeklaration für das zweite Dialogfeld deklarieren. Unterdessen erfordert das erste Dialogfeld, daß die zweite Dialogklasse zuerst deklariert wird. Das hat Einfluß auf die Reihenfolge, in der die Include-Dateien im Quellcode am Beginn jeder Datei eingebunden werden. Man kann nicht beide Klassen vor der jeweils anderen deklarieren, eine muß die erste sein. Obwohl man das Problem umgehen kann, indem man einen Platzhalter für die zweite Klasse vor der Deklaration der ersten Klasse deklariert, ist es in diesem Fall einfacher, den Typ des Zeigers als Zeiger auf die erste Dialogfeldklasse umzuwandeln. Im Anhang A finden Sie weitere Hinweise dazu, wie man einen Platzhalter für die zweite Klasse deklariert.

Listing 8.10: Die Funktion `ShowBitmap`

```
1: void CPaintDlg::ShowBitmap(CPaintDC *pdc, CWnd *pWnd)
2: {
3: // Zeiger in einen Zeiger auf die Dialogklasse des Hauptfensters
   Umwandeln
4: CGrafikDlg *lpWnd = (CGrafikDlg*)pWnd;
5: BITMAP bm;
6: // Geladenes Bitmap holen
7: lpWnd->m_bmpBitmap.GetBitmap(&bm);
8: CDC dcMem;
9: // Gerätekontext erzeugen, in den Bitmap geladen wird
10: dcMem.CreateCompatibleDC(pdc);
11: // Bitmap in den kompatiblen Gerätekontext selektieren
12: CBitmap* pOldBitmap = (CBitmap*)dcMem.SelectObject(lpWnd->m_bmpBitmap);
13: CRect lRect;
14: // Anzeigebereich verfügbar machen
15: GetClientRect(lRect);
16: lRect.NormalizeRect();
17: // Bitmap in Dialogfeld kopieren und in Größe anpassen
18: pdc->StretchBlt(10, 10, (lRect.Width() - 20),
19: (lRect.Height() - 20), &dcMem, 0, 0,
20: bm.bmWidth, bm.bmHeight, SRCCOPY);
21: }
```

Jetzt können Sie das momentan ausgewählte Bitmap im Dialogfeld anzeigen. Es ist noch die Funktion `OnPaint` im zweiten Dialogfeld anzupassen, um die Funktion `ShowBitmap` aufzurufen. Ob ein Bitmap spezifiziert wurde, läßt sich anhand der Variablen `m_sBitmap` im ersten Dialogfeld testen. Wenn der String leer ist, gibt es kein Bitmap, das anzuzeigen wäre. Ist der String nicht leer, rufen Sie die Funktion `ShowBitmap` auf. Um dieses letzte Stück der Funktionalität in Ihrer Anwendung zu realisieren, fügen Sie die Zeilen 15 bis 18 aus Listing 8.11 in die Funktion `OnPaint` ein.

Listing 8.11: Die modifizierte Funktion `OnPaint`

```
1: void CPaintDlg::OnPaint()
2: {
3: CPaintDC dc(this); // device context for painting
```

```

4:
5: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
6:
7: // Zeiger auf übergeordnetes Fenster ermitteln
8: CGrafikDlg *pWnd = (CGrafikDlg*)GetParent();
9: // Ist der Zeiger gültig?
10: if (pWnd)
11: {
12: // Bitmap als Werkzeug gewählt?
13: if (pWnd->m_iTool == 2)
14: {
15: // Ist ein Bitmap ausgewählt und geladen?
16: if (pWnd->m_sBitmap != "")
17: // Bitmap anzeigen
18: ShowBitmap(&dc, pWnd);
19: }
20: else // Nein, Figur zeichnen
21: {
22: // Wird Linie gezeichnet?
23: if (m_iShape == 0)
24: DrawLine(&dc, pWnd->m_iColor);
25: else // Ellipse oder Rechteck zeichnen
26: DrawRegion(&dc, pWnd->m_iColor, pWnd->m_iTool,
27: pWnd->m_iShape);
28: }
29: }
30: // Kein Aufruf von CDialog::OnPaint() für Zeichnungsnachrichten
31:}

```

Jetzt sollten Sie in der Lage sein, ein Bitmap von Ihrem System auszuwählen und es im zweiten Dialogfeld anzuzeigen (siehe Abbildung 8.6).

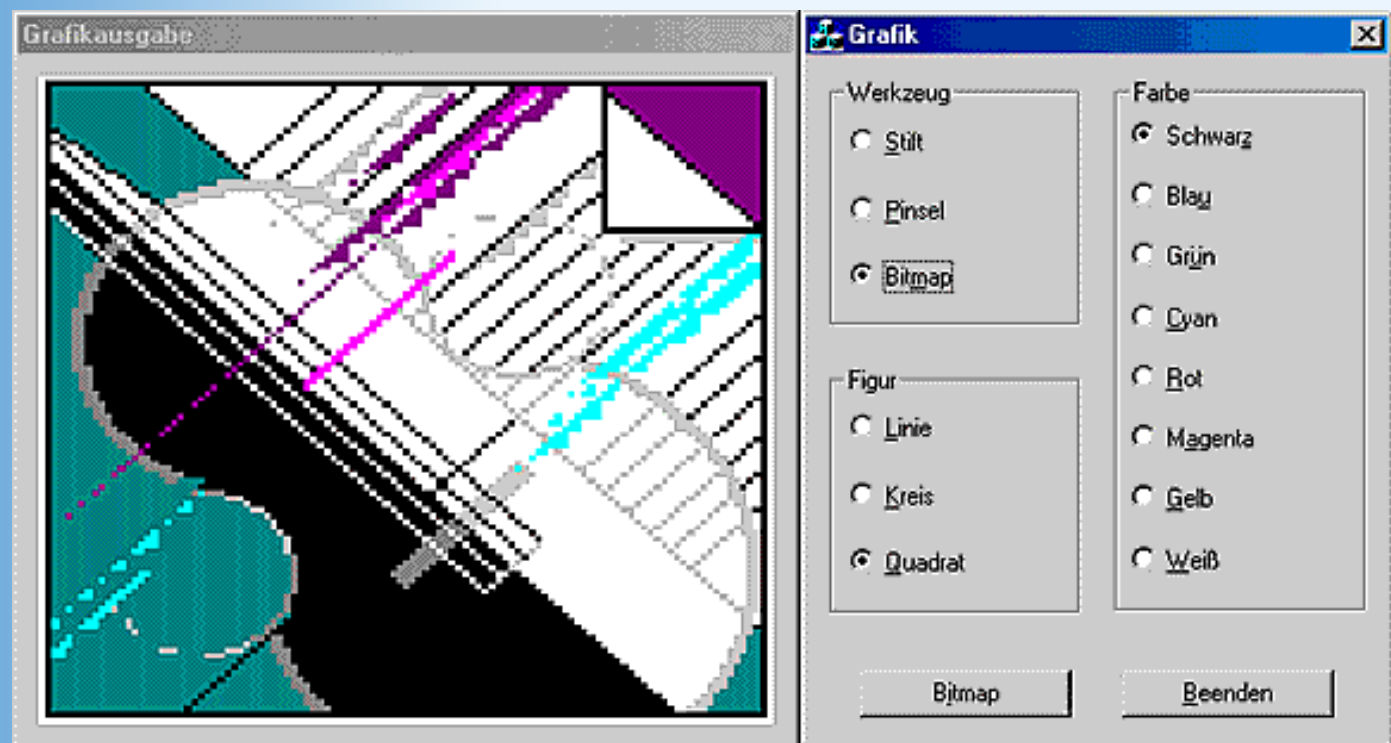


Abbildung 8.6:
Ein Bitmap im zweiten Dialogfeld anzeigen

Zusammenfassung

Was für ein Tag, um die Woche zu beginnen! Heute haben Sie eine Menge gelernt. Vor allem ging es darum, wie Windows die Gerätekontextobjekte verwendet, damit Sie Grafiken immer in der gleichen Weise zeichnen können, ohne sich um die konkrete Hardware kümmern zu müssen, die der Benutzer auf seinem Computer installiert hat. Am Beispiel der grundlegenden GDI-Objekte Stift und Pinsel wurde gezeigt, wie man diese Objekte zum Zeichnen von Figuren in Fenstern und Dialogfeldern einsetzt. Weiterhin haben Sie erfahren, wie man Bitmaps vom Systemlaufwerk lädt und sie auf dem Bildschirm für den Benutzer anzeigt. Zum Zeichnen von Figuren mit Stiften und Pinseln stehen verschiedene Stile zur Verfügung. Außerdem haben Sie gelernt, wie man Farben für Stifte und Pinsel festlegt, so daß man steuern kann, wie sich Bilder dem Benutzer präsentieren.

Fragen und Antworten

Frage:

Warum muß ich sowohl einen Stift als auch einen Pinsel spezifizieren, wenn ich eigentlich nur den einen oder den anderen verwenden möchte?

Antwort:

Wenn Sie ein Objekt zeichnen, das ausgefüllt ist, zeichnen Sie immer mit beiden Werkzeugen. Der Stift ist für den Umriß verantwortlich, während der Pinsel den Innenbereich ausfüllt. Man kann nicht nur den einen oder den anderen verwenden, sondern immer nur beide. Wenn man nur den einen oder anderen anzeigen möchte, sind spezielle Schritte zu unternehmen.

Frage:

Warum werden alle Stiftstile durchgängig, wenn ich die Breite des Stifts über 1 erhöhe?

Antwort:

Bei einem breiteren Stift erhöht man auch die Größe der Punkte, die zum Zeichnen verwendet werden. Wie Sie noch aus Lektion 3 wissen, erhalten Sie nur verstreute Punkte, wenn Sie die von der Maus ausgelösten Ereignisse zum Zeichnen von Punkten einzeln auffangen. Sobald Sie die Größe der Punkte für die zu zeichnende Linie erhöhen, werden die Lücken zwischen den Punkten von beiden Seiten aufgefüllt, so daß eine durchgängige Linie entsteht.

Workshop

Kontrollfragen

1. Welche drei Werte faßt man zu einem Farbenwert zusammen?
2. Mit welchem Instrument zeichnet man in ein Fenster, ohne daß man die vom Benutzer eingesetzte Grafikkarte kennen muß?
3. Welche Größe kann man für ein Bitmap verwenden, um einen Pinsel daraus zu erstellen?
4. Welche Nachricht sendet Windows an ein Fenster, um es anzuweisen, sich selbst neu zu zeichnen?
5. Wie kann man erreichen, daß sich ein Fenster selbst neu zeichnet?

Übungen

1. Machen Sie das zweite Dialogfeld in der Größe veränderbar, und stellen Sie sicher, daß sich die im Fenster vorhandenen Zeichnungen ebenfalls in der Größe anpassen, wenn man die Größe des Dialogfelds verändert.
2. Nehmen Sie einen Bitmap-Pinsel in die Gruppe der Pinsel auf, um Rechtecke und Ellipsen zu zeichnen.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2**Tag 9****ActiveX-Steuerelemente**

Auf dem heutigen Markt für Produkte der Anwendungsentwicklung gibt es Tausende vorgefertigter Komponenten, die man in eine Anwendung einbauen und damit die Funktionalität der Anwendung im Handumdrehen erweitern kann. Ursprünglich war es die Domäne der Visual Basic-Programmierer, jetzt aber kann man in nahezu jeder Programmiersprache - einschließlich Visual C++ - auf fertige ActiveX-Steuerelemente zurückgreifen. Heute lernen Sie, wie man ActiveX-Steuerelemente in Visual C++-Anwendungen einbaut und die vorhandene Funktionalität nutzt.

Insbesondere geht es heute um folgende Themen:

- Was sind ActiveX-Steuerelemente, und wie arbeiten sie?
- Wie kann man ActiveX-Steuerelemente in den Projekt-Arbeitsbereich einfügen?
- Wie kann man ActiveX-Steuerelemente in einer Visual C++-Anwendung einsetzen?
- Wie ruft man die verschiedenen Methoden auf, die zu einem ActiveX-Steuerelement gehören?
- Wie behandelt man Ereignisse, die vom ActiveX-Steuerelement ausgelöst werden?

Was sind ActiveX-Steuerelemente?

Ein ActiveX-Steuerelement ist eine Softwarekomponente, die sich in die verschiedensten Programme einbauen lässt und die man wie einen nativen Teil des Programms verwenden kann. Das Konzept ist mit den Komponenten einer Stereoanlage vergleichbar. Wenn man ein neues Kassettendeck kauft, kann man es einfach mit der bereits vorhandenen Anlage verbinden und deren Funktionalität erweitern, ohne die bisherigen Funktionen zu beeinträchtigen. ActiveX-Steuerelemente bringen die gleichen Möglichkeiten der Zusammenarbeit von Komponenten in Softwareanwendungen.

ActiveX-Steuerelemente firmierten ursprünglich unter der Bezeichnung OLE 2.0. Die von Microsoft entwickelte Technologie OLE 2.0 diente dazu, zwei oder mehr Anwendungen zu kombinieren und wie eine einzige Anwendung erscheinen zu lassen (oder zumindest eine Umschaltung zwischen den verschiedenartigen Anwendungen innerhalb derselben Benutzeroberfläche zu erreichen). Dieser Gedanke war eine Erweiterung der ursprünglichen OLE-Technologie (OLE - Object Linking and Embedding, Objekte verknüpfen und einbetten), die es lediglich erlaubte, die mit verschiedenen Anwendungen erzeugten Dokumente zu einem Dokument zusammenzufassen. Beim Aufpolieren der OLE-Technologien für eine Arbeit in einer verteilten Umgebung (wie etwa dem Internet) hat sich Microsoft dafür entschieden, auch einen neuen Namen zu verwenden. Somit war ActiveX geboren.

ActiveX und die IDispatch-Schnittstelle

Die ActiveX-Technologie setzt auf der COM-Technologie von Microsoft auf (COM - Component Object Model) und stützt sich auf deren Oberfläche und das Interaktionsmodell, um die ActiveX-Steuerelemente nahtlos zu integrieren. Die COM-Technologie definiert, wie ActiveX-Objekte konstruiert sind und wie ihre Schnittstellen auszusehen haben. Die auf COM aufsetzende Schicht legt fest, welche Schnittstellen die verschiedenartigen Objekte unterstützen sollten und wie unterschiedliche Objekttypen miteinander in Wechselwirkung treten.



Die COM-Technologie von Microsoft definiert, wie Komponenten über die Schnittstellen miteinander interagieren können. Eine Schnittstelle verhält sich wie ein Funktionsaufruf an eine ActiveX-Komponente. COM legt dabei fest, wie dieser Funktionsaufruf aufgebaut sein muß, wie der Aufruf zu erfolgen hat und welche unterstützende Funktionalität den Funktionsaufruf begleiten muß.

Über Schnittstellen wie **IUnknown**, die in jedem COM-Objekt erforderlich sind, fragt man die Komponente ab, um herauszufinden, welche anderen Schnittstellen die Komponente bietet. Jede Schnittstelle unterstützt einen speziellen Satz von Funktionen - eine Schnittstelle behandelt etwa die visuelle Erscheinung des Steuerelements, eine andere steuert, wie die Erscheinung des Steuerelements mit der umgebenden Anwendung in Wechselwirkung tritt, eine weitere löst Ereignisse in der umgebenden Anwendung aus, usw.

Zu den Schlüsseltechnologien der ActiveX-Steuerelemente gehört die Automatisierung. Damit kann eine Anwendung, die in eine andere Anwendung eingebettet ist, sich selbst aktivieren, ihren Teil der Benutzeroberfläche oder des Dokuments steuern, Änderungen am Dokument vornehmen und sich dann wieder schließen, wenn der Benutzer zu einem anderen Teil der Anwendung geht, der nicht von der eingebetteten Anwendung gesteuert wird.

Diese Vorgänge laufen beispielsweise ab, wenn man ein Excel-Tabellenblatt in ein Word-Dokument einbettet. Ein Klick auf das Tabellenblatt aktiviert Excel. Das Tabellenblatt läßt sich dann mit Excel bearbeiten, selbst wenn man weiterhin in Word arbeitet. Sobald die Änderungen am Tabellenblatt abgeschlossen sind, schließt sich Excel automatisch, und man kann in Word weiterarbeiten.

Ein Schlüssel zur Arbeitsweise der Automatisierung ist eine spezielle Schnittstelle namens **IDispatch** (auch als **Dispinterface** bezeichnet). Die **IDispatch**-Schnittstelle besteht aus einem Zeiger auf eine Tabelle verfügbarer Methoden, die sich im ActiveX-Steuerelement oder der eingebetteten Anwendung ausführen lassen. Diesen Methoden sind ID-Nummern - sogenannte **DISPIDs** - zugeordnet, die ebenfalls in eine Tabelle geladen werden. In dieser Tabelle kann man dann nach der ID für eine bestimmte Methode nachschlagen. Nachdem man die **DISPID** für eine bestimmte Methode kennt, kann man diese Methode über die Methode **Invoke** der **IDispatch**-Schnittstelle aufrufen. Dabei übergibt man die **DISPID**, um die auszuführende Methode zu kennzeichnen. Abbildung 9.1 zeigt, wie die **IDispatch**-Schnittstelle die Methode **Invoke** verwendet, um Methoden im ActiveX-Objekt auszuführen.

IDispatch::Invoke(DISPID)

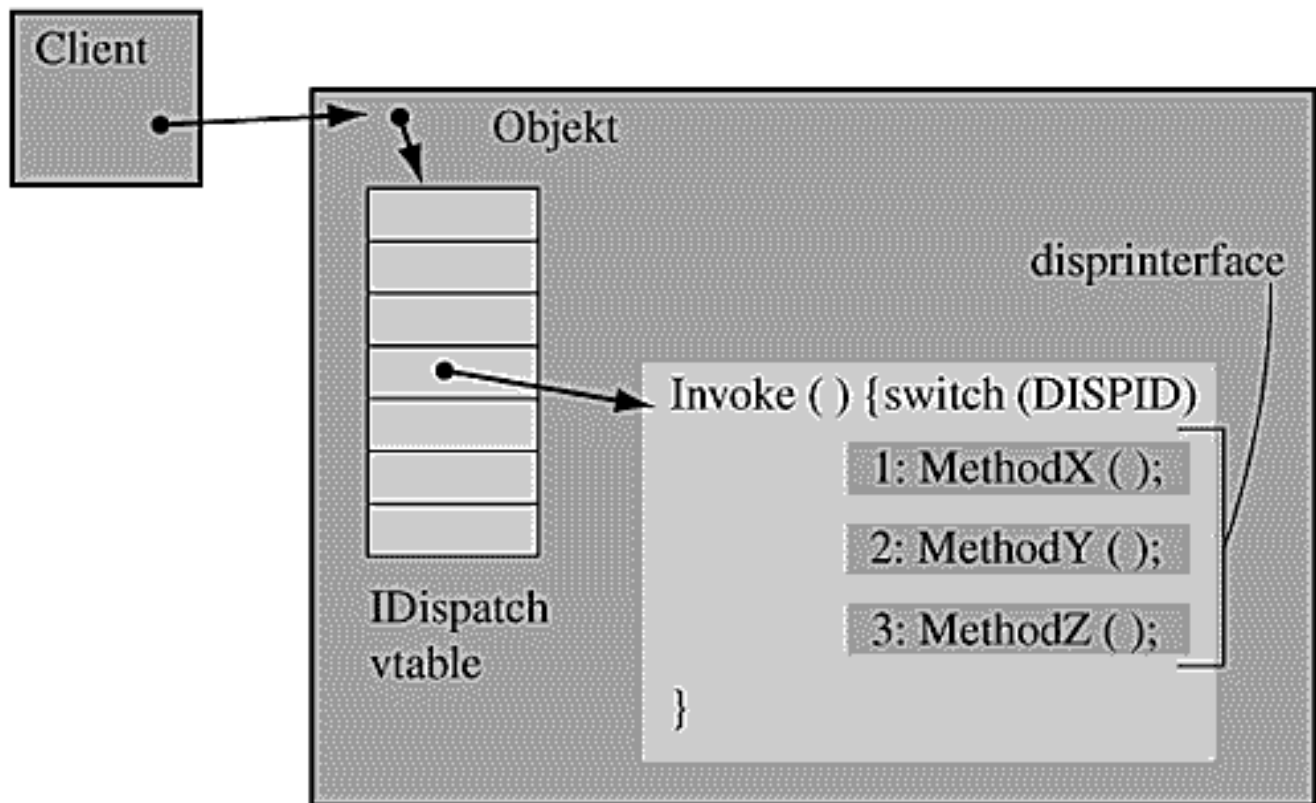


Abbildung 9.1:
Die ActiveX-Schnittstelle IDispatch

ActiveX-Container und -Server

Um ein ActiveX-Objekt in ein anderes einzubetten, muß man das eingebettete Objekt als ActiveX-Server und das Objekt, das das erste Objekt enthält, als ActiveX-Container implementieren. Ein ActiveX-Objekt, das man in ein anderes einbetten kann, ist ein ActiveX-Server, ob es sich dabei nun um eine vollständige Anwendung oder nur ein kleines ActiveX-Steuerelement handelt. Jedes ActiveX-Objekt, das andere ActiveX- Objekte in sich aufnehmen kann, ist ein ActiveX-Container.



Verwechseln Sie nicht die Begriffe *Container* und *Server* mit dem Begriff *Client* in Abbildung 9.1. Der Client ist das Objekt, das die **IDispatch**- Schnittstelle des anderen Objekts aufruft. Wie Sie in Kürze sehen werden, rufen sowohl der Container als auch der Server die **IDispatch**-Schnittstellen des »Partners« auf und machen ihn damit zu ihrem Client.

Diese beiden Typen von ActiveX-Objekten schließen sich nicht gegenseitig aus. Ein ActiveX-Server kann auch ActiveX-Container sein. Ein Beispiel für dieses Konzept liefert der Internet Explorer von Microsoft. Der Internet Explorer ist als ActiveX-Server implementiert, der im Gerüst eines ActiveX-Containers (das ebenfalls Word, Excel, PowerPoint oder eine beliebige andere ActiveX-Server-Anwendung aufnehmen kann) läuft. Zur selben Zeit, da Internet Explorer als ActiveX-Server innerhalb der Browser- Umgebung läuft, kann er andere ActiveX-Steuerelemente

enthalten.

ActiveX-Steuerelemente sind eine spezielle Instanz eines ActiveX-Servers. Manche ActiveX-Server sind ebenfalls Anwendungen, die eigenständig laufen können. ActiveX-Steuerelemente können nicht eigenständig laufen und müssen in einen ActiveX- Container eingebettet werden. Durch den Einsatz von ActiveX-Komponenten in Ihrer Visual C++-Anwendung machen Sie Ihre Anwendung automatisch zu einem ActiveX- Container.

Der größte Teil der Interaktion zwischen dem ActiveX-Container und einem ActiveX- Steuerelement findet über die `IDispatch`-Schnittstellen statt. Eine dieser `IDispatch`- Schnittstellen gehört zum Steuerelement und wird durch den Container verwendet, um Aufrufe der verschiedenen Methoden auszuführen, die das Steuerelement dem Container verfügbar macht.

Der Container stellt dem Steuerelement zwei `IDispatch`-Schnittstellen bereit. Das Steuerelement verwendet die erste dieser `IDispatch`-Schnittstellen, um Ereignisse in der Container-Anwendung auszulösen. Die zweite Schnittstelle dient dazu, die Eigenschaften des Steuerelements festzulegen, wie es Abbildung 9.2 darstellt. Die meisten Eigenschaften eines ActiveX-Steuerelements werden eigentlich vom Container bereitgestellt, aber durch das Steuerelement verwaltet. Wenn man eine Eigenschaft für das Steuerelement setzt, ruft der Container eine Methode im Steuerelement auf, um es anzuweisen, die Eigenschaften aus dem Container zu lesen. Größtenteils sind diese Abläufe für den Programmierer transparent, da Visual C++ eine Reihe von C++-Klassen um die Schnittstellen des ActiveX-Steuerelements aufbaut. Diese C++-Klassen legen Methoden frei. Genau über diese Methoden treten Sie in Wechselwirkung, und nicht durch direkte Aufrufe der `IDispatch`-Schnittstelle des Steuerelements.

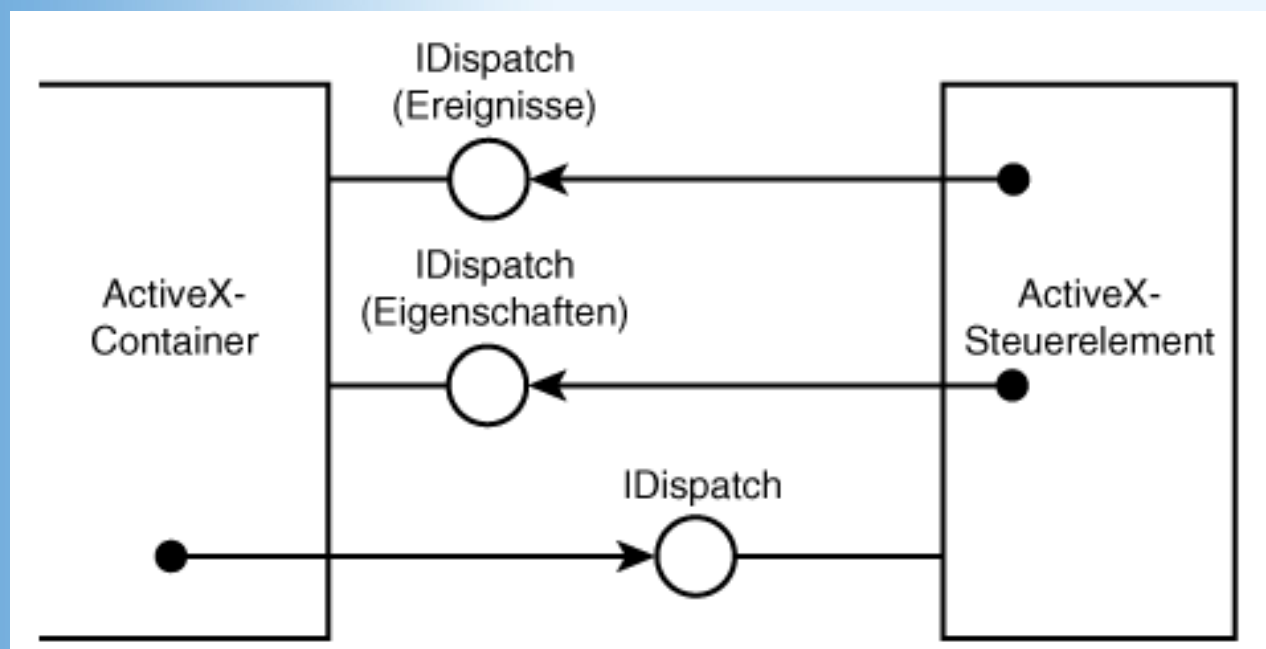


Abbildung 9.2:
ActiveX-Container und Steuerelemente treten hauptsächlich über wenige `IDispatch`-Schnittstellen in Wechselwirkung.

Ein ActiveX-Steuerelement in das Projekt aufnehmen

Wenn man sich die Arbeitsweise von ActiveX-Steuerelementen ansieht, scheint es schwieriger zu sein als es in Wirklichkeit ist, Steuerelemente in Anwendungen einzubauen. Mit Visual C++ ist es schon

einfach, ActiveX-Steuerelemente in eine Anwendung zu integrieren, und noch einfacher, sie zu verwenden. Bevor Sie ein ActiveX-Steuerelement in Ihre Anwendung aufnehmen, erzeugen Sie zunächst das Anwendungsgerüst, in das Sie ein ActiveX-Steuerelement einbauen:

1. Legen Sie mit dem MFC-Anwendungs-Assistenten ein neues Projekt namens **ActiveX** an.
2. Übernehmen Sie die Standardeinstellungen wie in den vergangenen Lektionen, lassen Sie aber im zweiten Dialogfeld des Assistenten das Kontrollkästchen für ActiveX-Steuerelemente eingeschaltet. Als Titel der Anwendung können Sie beispielsweise **ActiveX-Steuerelemente** eintragen.
3. Nachdem Sie das Anwendungsgerüst erstellt haben, entfernen Sie alle Steuerelemente aus dem Dialogfeld und fügen eine Schaltfläche hinzu.
4. Setzen Sie die ID der Schaltfläche auf **IDC_EXIT** und den Titel auf **&Beenden**.
5. Mit dem Klassen-Assistenten fügen Sie nun eine Funktion für das Ereignis **BN_CLICKED** der Schaltfläche hinzu.
6. In die eben erstellte Funktion tragen Sie den bereits aus vergangenen Lektionen bekannten Aufruf von **OnOK** ein, um die Anwendung schließen zu können.

Das Steuerelement registrieren

Bevor Sie ein ActiveX-Steuerelement in Ihr Dialogfeld aufnehmen, müssen Sie das Steuerelement sowohl bei Windows als auch bei Visual C++ registrieren. Die Registrierung für Windows kann auf zwei Wegen erfolgen. Der erste besteht darin, daß man die Installationsroutine, die zum ActiveX-Steuerelement gehört, ausführt. Wenn es keine Installationsroutine gibt, müssen Sie das Steuerelement manuell registrieren. Führen Sie in diesem Fall die folgenden Schritte aus:

1. Öffnen Sie die MS-DOS-Eingabeaufforderung.
2. Wechseln Sie in das Verzeichnis, wo sich das ActiveX-Steuerelement in Ihrem System befindet.
3. Führen Sie den Befehl **regsvr32** aus. Geben Sie als Argument der Befehlszeile den Namen des ActiveX-Steuerelements an. Wenn Sie zum Beispiel ein Steuerelement namens **MYCTL.OCX** registrieren, das sich im Verzeichnis **Windows\System** befindet, führen Sie die folgenden Befehle an der MS-DOS-Eingabeaufforderung aus:

```
C:\WINDOWS>CD system
```

```
C:\WINDOWS\SYSTEM>regsvr32 MYCTL.OCX
```



Es empfiehlt sich, die zum Steuerelement gehörende Installationsroutine auszuführen, da die manuelle Registrierung das Steuerelement möglicherweise nicht für die Programmentwicklung aktiviert. Steuerelemente können für die Entwicklung und die Nutzung lizenziert sein. Wenn ein Steuerelement lediglich für die Nutzung lizenziert ist, können Sie es nicht in Ihren Visual C++-Anwendungen einsetzen. Dieser Mechanismus schützt Steuerelemententwickler, die ihre Produkte gegen eine entsprechende Gebühr an die Anwendungsentwickler verkaufen. Derartige Steuerelemente sind zwar auf dem jeweiligen System installiert, lassen sich aber nicht mit einer anderen Anwendung nutzen.



COM- und ActiveX-Objekte speichern eine Menge Informationen in der Registrierungsdatenbank von Windows. Wenn eine Anwendung auf ein ActiveX-Objekt zurückgreift, bezieht sich das Betriebssystem auf diese Informationen in der Windows-Registrierung und ermittelt, ob die Anwendung das Objekt in der angeforderten Weise verwenden darf. Mit dem Dienstprogramm `regsvr32.exe` werden zwar die meisten Informationen über das Steuerelement in die Systemregistrierung eingetragen, dennoch können zusätzliche Angaben in der Registrierung erforderlich sein, damit sich das Steuerelement in der vorgesehenen Weise nutzen lässt.

Nachdem Sie nun das gewünschte ActiveX-Steuerelement im Betriebssystem angemeldet haben, müssen Sie es noch bei Visual C++ registrieren und in Ihr Projekt einfügen. Führen Sie dazu folgende Schritte aus:

1. Wählen Sie **Projekt / Dem Projekt hinzufügen / Komponenten und Steuerelemente** aus dem Menü von Visual C++.
2. Im Dialogfeld **Sammlung der Komponenten und Steuerelemente** gehen Sie zum Ordner **Registered ActiveX Controls**, wie es Abbildung 9.3 zeigt.

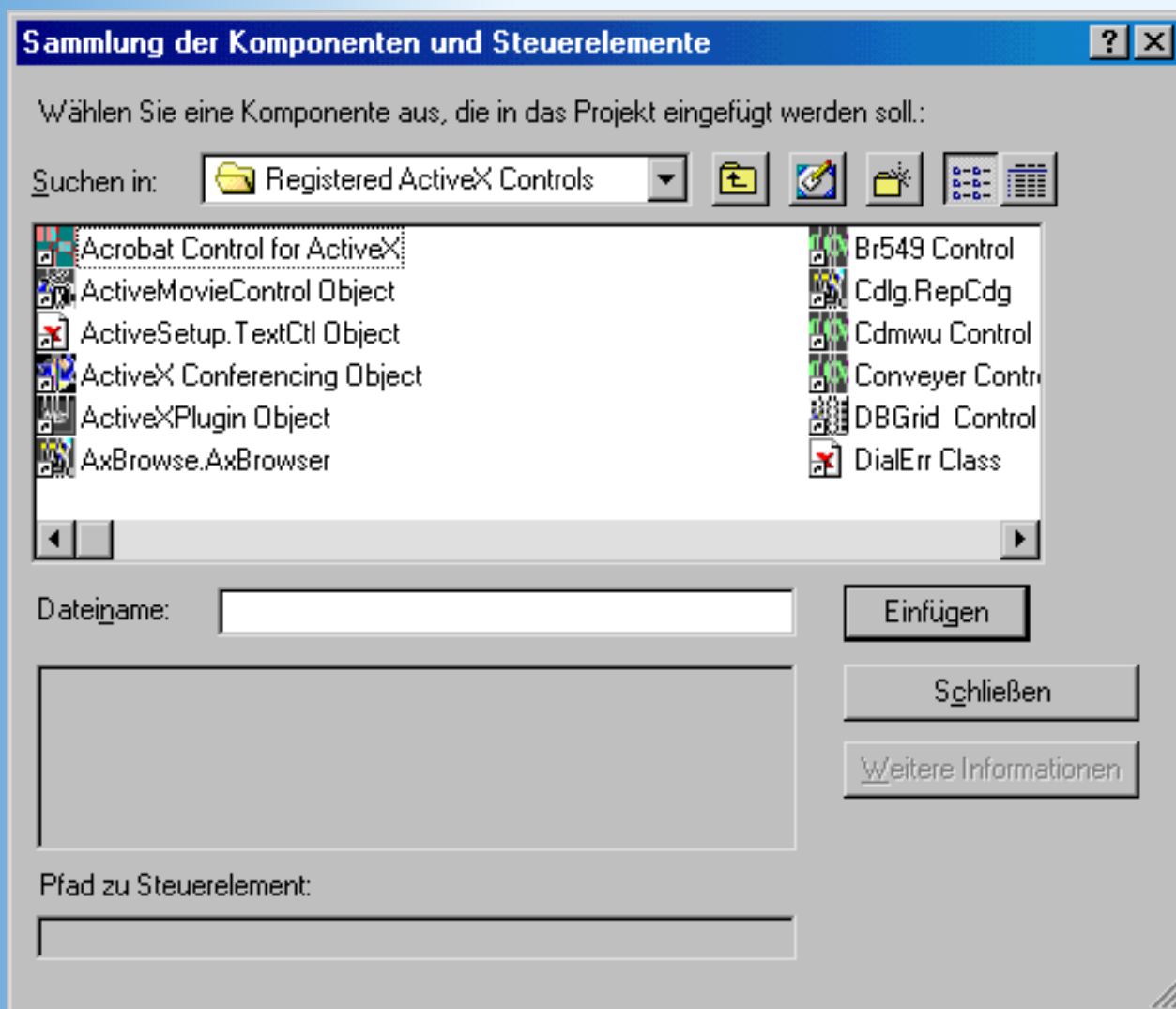


Abbildung 9.3:
Die ActiveX-Steuerelemente, die sich in das Projekt aufnehmen lassen

3. Markieren Sie das Steuerelement, das Sie registrieren wollen, beispielsweise das Steuerelement **Microsoft FlexGrid Control**, und klicken Sie auf die Schaltfläche **Einfügen**.

4. Klicken Sie im Meldungsfeld mit der Frage, ob Sie diese Komponente einfügen wollen, auf **OK**.

5. Im Dialogfeld **Klassen bestätigen** klicken Sie auf **OK**, um die angegebenen C++-Klassen hinzuzufügen (siehe Abbildung 9.4).

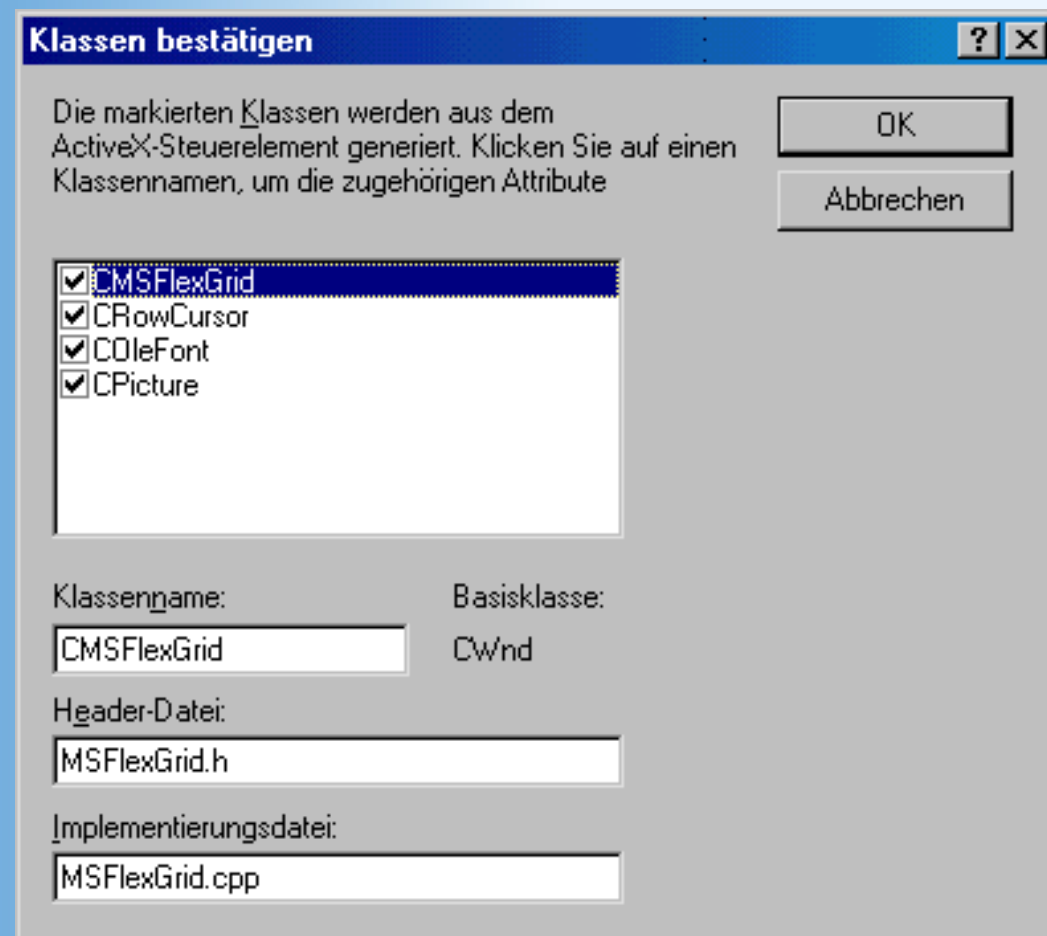


Abbildung 9.4:
Visual C++ gibt darüber Auskunft, welche Klassen in das Projekt eingebunden werden.

6. Klicken Sie im Dialogfeld **Sammlung der Komponenten und Steuerelemente** auf die Schaltfläche **Schliessen**, um die Aufnahme der Steuerelemente in Ihr Projekt fertigzustellen.

7. Das Steuerelement FlexGrid sollte nun in der Steuerelementpalette für Ihr Dialogfeld enthalten sein, wie es Abbildung 9.5 zeigt.

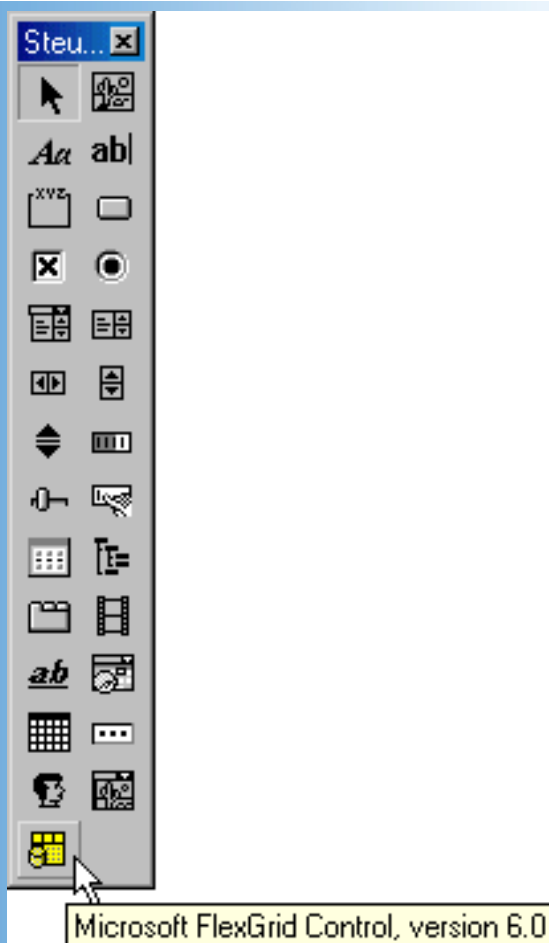


Abbildung 9.5:
Das ActiveX-Steuerelement FlexGrid wird in die Steuerelementpalette hinzugefügt und läßt sich in Dialogfeldern verwenden.

Im Arbeitsbereich sehen Sie auf der Registerkarte Klassen die vier Klassen, die Visual C++ in Ihr Projekt eingefügt hat. Wenn Sie die Klassenbäume erweitern, erscheinen die zahlreichen Methoden für diese Klassen. Visual C++ hat diese Klassen und Methoden nach einer Untersuchung des gerade hinzugefügten ActiveX-Steuerelements angelegt und Klassenmethoden erzeugt, um die jeweiligen Methoden der `IDispatch`-Schnittstelle des Steuerelements aufzurufen.



Wenn Sie mit älteren ActiveX-Steuerelementen in Ihren Visual C++-Anwendungen arbeiten, ist Visual C++ eventuell nicht in der Lage, die Klassen und Methoden zu erzeugen, um die Funktionalität des Steuerelements abzukapseln. Die Informationen im Steuerelement, die Visual C++ die notwendigen Angaben liefern, um diese Klassen und Methoden erstellen zu können, gehören zu den neueren Erweiterungen der ActiveX-Spezifikation. Letztendlich bringen ältere Steuerelemente diese Informationen nicht mit, wodurch sich ihr Einsatz mit Visual C++ schwieriger gestaltet.

Das Steuerelement in das Dialogfeld aufnehmen

Nachdem Sie nun das Steuerelement FlexGrid in Ihr Projekt eingebunden haben, können Sie es in Ihr Dialogfeld genau wie jedes andere Steuerelement aufnehmen. Legen Sie die Eigenschaften des Steuerelements gemäß Tabelle 9.1 fest.

Tabelle 9.1: Einstellungen der Steuerelementeigenschaften

Objekt	Eigenschaft	Einstellung
FlexGrid-Steuerelement	ID	IDC_MSFGGRID
	Rows (Zeilen)	20
	Cols (Spalten)	4
	MergeCells	2 - Zeilen beschränkt
	Format (Formatstring)	<Gebiet <Produkt <Verkäufer > Umsatz (Leerzeichen je nach gewünschter Spaltenbreite hinzufügen.)

Nachdem Sie das Steuerelement in Ihr Dialogfeld aufgenommen haben, enthält das Eigenschaftsdialogfeld eine zusätzliche Registerkarte, die alle Eigenschaften des Steuerelements enthält (siehe Abbildung 9.6). Die Eigenschaften des Steuerelements können Sie über diese Registerkarte festlegen oder wie gewohnt über die anderen Registerkarten arbeiten.

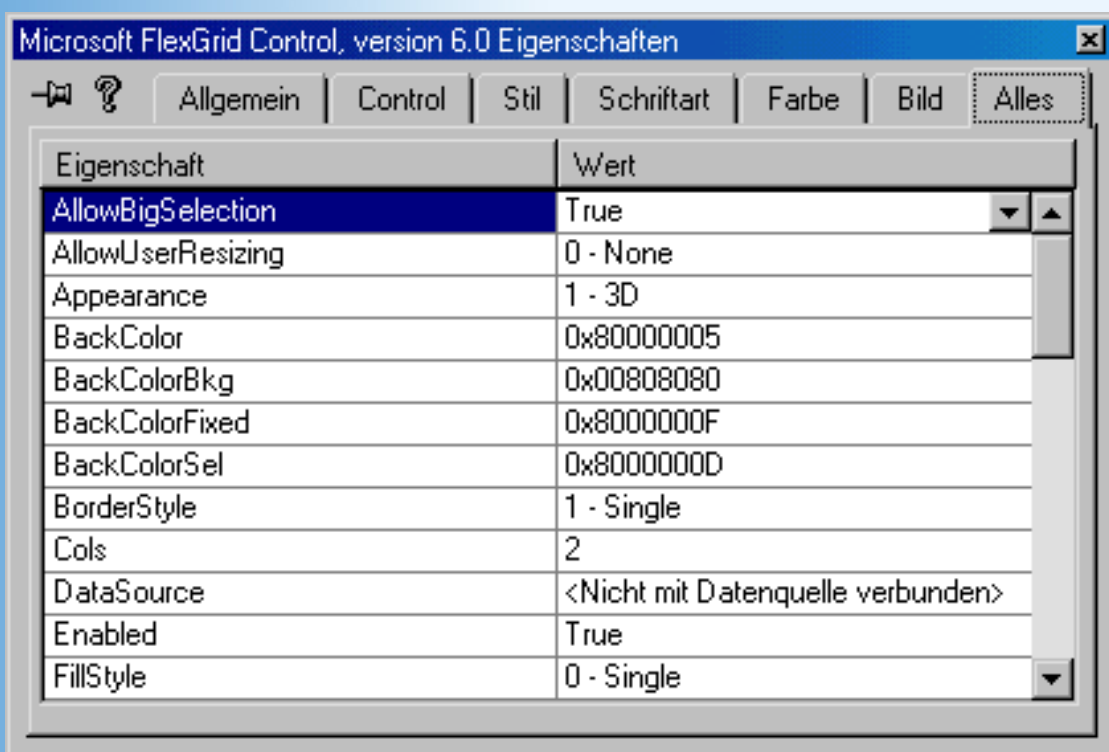


Abbildung 9.6:
Die Registerkarte Eigenschaften eines ActiveX-Steuerelements enthält alle Eigenschaften des Steuerelements.

Wenn Sie alle Eigenschaften des Steuerelements eingestellt haben, fügen Sie eine Variable für das Steuerelement hinzu, damit Sie es per Code ansprechen können. Öffnen Sie dazu die Registerkarte Member-Variablen hinzufügen im Klassen-Assistenten. Da Sie eine Variable für ein Steuerelement hinzufügen, sind nur Variablen vom Typ `Control` verfügbar, so daß Sie nur den Variablennamen bereitstellen müssen. Benennen Sie die Variable für dieses Beispiel mit `m_ctlFGrid`.

ActiveX-Steuerelemente in einer Anwendung einsetzen

Nachdem Visual C++ alle Klassen generiert hat, um das ActiveX-Steuerelement zu verkapseln, können Sie wie bei jedem Standardsteuerelement die Methoden aufrufen und auf die Steuerelementereignisse reagieren. Als erstes verwenden Sie die Methoden des Steuerelements, um Informationen über das Steuerelement einzuholen und Daten innerhalb des Steuerelements zu modifizieren. Dann lernen Sie, wie man mit Visual C++ auf Steuerelementereignisse reagiert.

Mit dem Steuerelement in Wechselwirkung treten

Die heutige Beispielanwendung generiert eine Reihe von Produktumsätzen über fünf Verkaufsgebiete mit fünf Verkäufern. Man kann durch die Daten blättern, die nach Gebiet und Produkt sortiert sind, und vergleichen, wie jeder Verkäufer die einzelnen Produkte an den Mann gebracht hat.

Für dieses Projekt erstellen Sie ein Array mit Werten, die in die Zellen der Tabelle geladen werden. Die Anwendung sortiert dann die Tabelle in aufsteigender Reihenfolge und greift dabei auf die internen Sortierfähigkeiten des Steuerelements FlexGrid zurück.

Daten in das Steuerelement laden

Als erstes erstellen Sie eine Funktion, um Daten in das Steuerelement FlexGrid zu laden. Nehmen Sie in die Klasse `CActiveXDlg` eine neue Funktion auf. Dazu klicken Sie im Arbeitsbereich auf der Registerkarte Klassen mit der rechten Maustaste auf die Klasse `CActiveXDlg` und wählen den Befehl Member-Funktion hinzufügen aus dem Kontextmenü. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `LoadData` und den Zugriff als Privat fest. Klicken Sie auf die Schaltfläche OK, und fügen Sie den Code aus Listing 9.1 in die Funktion ein.

Listing 9.1: Die Funktion LoadData

```
1: void CActiveXDlg::LoadData()
2: {
3:     int liCount; // Zeilenzähler für Tabelle
4:     CString lsAmount; // Umsatz
5:
6:     // Zufallszahlengenerator initialisieren
7:     srand((unsigned)time(NULL));
8:     // Array im Steuerelement erzeugen
9:     for (liCount = m_ctlFGrid.GetFixedRows();
10:         liCount < m_ctlFGrid.GetRows(); liCount++)
11:     {
12:         // Werte für erste Spalte (Gebiet) generieren
13:         m_ctlFGrid.SetTextArray(GenID(liCount, 0), RandomStringValue(0));
14:         // Werte für zweite Spalte (Produkt) generieren
15:         m_ctlFGrid.SetTextArray(GenID(liCount, 1), RandomStringValue(1));
16:         // Werte für dritte Spalte (Verkäufer) generieren
17:         m_ctlFGrid.SetTextArray(GenID(liCount, 2), RandomStringValue(2));
18:         // Werte für vierte Spalte (Umsatz) generieren
19:         lsAmount.Format("%5d.00", rand());
```



```

20: // Vierte Spalte füllen
21: m_ctlFGrid.SetTextArray(GenID(liCount, 3), lsAmount);
22: }
23:
24: // Aufeinanderfolgende, zusammengehörige Zeilen in diesen Spalten
   Vereinigen
25: m_ctlFGrid.SetMergeCol(0, TRUE);
26: m_ctlFGrid.SetMergeCol(1, TRUE);
27: m_ctlFGrid.SetMergeCol(2, TRUE);
28:
29: // Tabelle sortieren
30: DoSort();
31: }

```

Diese Funktion initialisiert zuerst den Zufallszahlengenerator. Als nächstes durchläuft sie eine Schleife über alle Zeilen des Steuerelements und schreibt Daten in die einzelnen Zellen. Die Gesamtzahl der Zeilen im Steuerelement ermitteln Sie durch Aufruf der Methode `GetRows`, die Anzahl der Kopfzeilen durch Aufruf der Methode `GetFixedRows`. Über die Methode `SetTextArray` lassen sich Daten in das Steuerelement einfügen. Die Methode `SetTextArray` übernimmt die Zellen-ID als erstes Argument und den Zelleninhalt als zweites. Beide Argumente generieren Sie über Funktionen, die Sie in Kürze erstellen.

Nachdem die Daten in den Tabellenzellen eingetragen sind, rufen Sie die Methode `SetMergeCol` auf, um dem Steuerelement mitzuteilen, daß es die Zellen in den ersten drei Spalten zusammenfassen kann, falls angrenzende Zeilen den gleichen Wert enthalten. Schließlich sortieren Sie das Steuerelement mit einer weiteren Funktion, die noch zu erstellen ist.

Die ID einer Zelle berechnen

Die Zellen im Steuerelement `FlexGrid` sind fortlaufend von links nach rechts und von oben nach unten numeriert. Im Beispiel hat das Steuerelement in der ersten Zeile, die die Überschriften enthält (und die bereits ausgefüllt sind), die Zellen 0 bis 3, die zweite Reihe die Zellen 4 bis 7 usw. Die ID einer Zelle läßt sich demnach berechnen, indem man die Anzahl der Spalten mit der aktuellen Zeilennummer multipliziert und dazu die Spaltennummer addiert. Besteht das Steuerelement zum Beispiel aus vier Spalten, und die Zelle liegt in der dritten Spalte und vierten Zeile, berechnet sich die Zellen-ID zu $(4 * 3) + 2 = 14$. (Beachten Sie dabei, daß die Zählung der Spalten und Zeilen mit 0 beginnt. Die Nummer der dritten Spalte ist demnach 2 und die Nummer der vierten Zeile gleich 3.)

Nachdem Sie nun wissen, wie man die Zellen-ID berechnet, implementieren Sie diese Formel in einer Funktion. Fügen Sie eine neue Funktion in die Klasse `CActiveXDlg` nach der gleichen Methode wie die Funktion `LoadData` hinzu. Für die Funktion legen Sie als Typ `int` und als Deklaration `GenID(int m_iRow, int m_iCol)` fest. In die Funktion schreiben Sie den Code gemäß Listing 9.2.

Listing 9.2: Die Funktion `GenID`

```

1: int CActiveXDlg::GenID(int m_iRow, int m_iCol)
2: {
3: // Anzahl der Spalten ermitteln
4: int liCols = m_ctlFGrid.GetCols();
5:
6: // ID auf Basis der Spaltenzahl, der aktuellen Spalte

```

```

7: // und der aktuellen Zeile berechnen
8: return (m_iCol + liCols * m_iRow);
9: }

```

Zufallsdaten erzeugen

Die ersten drei Spalten der Tabelle wollen wir in der Beispielanwendung mit zufälligen Daten füllen. In der ersten Spalte sollen die Namen der Gebiete stehen. Die zweite Spalte erhält die Namen der Produkte, und die dritte Spalte führt die Verkäufer auf. In einer `switch`-Anweisung bestimmt man die Spalte, für die Daten zu generieren sind. Im entsprechenden `case`-Zweig generiert man dann eine zufällige Zahl, unterzieht diese einer Modulo-Division und wertet das Ergebnis in einer weiteren `switch`-Anweisung aus, so daß man unter einer begrenzten Menge von Datenstrings wählen kann.

Um diese Funktionalität zu implementieren, nehmen Sie in die Klasse `CActiveXDlg` eine weitere Funktion auf. Für den Typ legen Sie `CString` und als Deklaration `RandomStringValue(int m_iColumn)` fest. In die Funktion übernehmen Sie den Code aus Listing 9.3

Listing 9.3: Die Funktion `RandomStringValue`

```

1: CString CActiveXDlg::RandomStringValue(int m_iColumn)
2: {
3:   CString lsStr; // Rückgabestring
4:   int liCase; // Zufallszahl als ID
5:
6:   // Für welche Spalte werden Daten generiert?
7:   switch (m_iColumn)
8:   {
9:   case 0: // Die erste Spalte (Gebiet)
10:    // Zufallszahl zwischen 0 und 4 erzeugen
11:    liCase = (rand() % 5);
12:    // Welcher Wert wurde erzeugt?
13:    switch (liCase)
14:    {
15:    case 0:
16:    // 0 - Gebiet Nordwest
17:    lsStr = "Nordwest";
18:    break;
19:    case 1:
20:    // 1 - Gebiet Südwest
21:    lsStr = "Südwest";
22:    break;
23:    case 2:
24:    // 2 - Gebiet Mitte
25:    lsStr = "Mitte";
26:    break;
27:    case 3:
28:    // 3 - Gebiet Nordost
29:    lsStr = "Nordost";
30:    break;
31:    default:
32:    // 4 - Gebiet Südost

```

```
33: lsStr = "Südost";
34: break;
35: }
36: break;
37: case 1: // Die zweite Spalte (Produkt)
38: // Zufallszahl zwischen 0 und 4 erzeugen
39: liCase = (rand() % 5);
40: // Welcher Wert wurde erzeugt?
41: switch (liCase)
42: {
43: case 0:
44: // 0 - Dusch zu zweit
45: lsStr = "Dusch zu zweit";
46: break;
47: case 1:
48: // 1 - April Spezial-Dragees
49: lsStr = "April Spezial-Dragees";
50: break;
51: case 2:
52: // 2 - Nachelle No. 6
53: lsStr = "Nachelle No. 6";
54: break;
55: case 3:
56: // 3 - Eau-De-Vie
57: lsStr = "Eau-De-Vie";
58: break;
59: default:
60: // 4 - Haut&Haar
61: lsStr = "Haut&Haar";
62: break;
63: }
64: break;
65: case 2: // Die dritte Spalte (Verkäufer)
66: // Zufallszahl zwischen 0 und 3 erzeugen
67: liCase = (rand() % 4);
68: // Welcher Wert wurde erzeugt?
69: switch (liCase)
70: {
71: case 0:
72: // 0 - Samson
73: lsStr = "Samson";
74: break;
75: case 1:
76: // 1 - Harry
77: lsStr = "Harry";
78: break;
79: case 2:
80: // 2 - Susi
81: lsStr = "Susi";
```

```

82: break;
83: default:
84: // 3 - Linda
85: lsStr = "Linda";
86: break;
87: }
88: break;
89: }
90: // Generierten String zurückgeben
91: return lsStr;
92: }

```

Das Steuerelement sortieren

Um die Tabelle des Steuerelements FlexGrid zu sortieren, muß man alle Spalten auswählen und dann die Sortierung mit aufsteigend festlegen. Um diese Funktionalität zu implementieren, fügen Sie in die Klasse CActiveXDlg eine weitere Funktion ein. Legen Sie den Typ als `void` und die Definition mit `DoSort` fest. In die Funktion übernehmen Sie den Code aus Listing 9.4.

Listing 9.4: Die Funktion DoSort

```

1: void CActiveXDlg::DoSort()
2: {
3: // Aktuelle Spalte auf 0 setzen
4: m_ctlFGrid.SetCol(0);
5: // Spaltenauswahl für alle Spalten festlegen
6: m_ctlFGrid.SetColSel((m_ctlFGrid.GetCols() - 1));
7: // Aufsteigend sortieren
8: m_ctlFGrid.SetSort(1);
9: }

```

Die Funktion `DoSort` setzt die aktuelle Spalte mit Hilfe der Methode `SetCol` auf die erste Spalte (d.h. Spalte 0). Als nächstes markiert sie mit der Methode `SetColSel` alle Spalten von der ersten bis zur letzten, so daß alle Spalten im Steuerelement ausgewählt sind. Schließlich ruft die Funktion die Methode `SetSort` auf, um das Steuerelement anzuweisen, die Spalten in aufsteigender Reihenfolge zu sortieren. Die Sortierreihenfolge wird durch Übergabe einer 1 an die Methode festgelegt.

Die Funktionalität zum Füllen des Steuerelements mit Daten ist damit vollständig realisiert. In der Funktion `OnInitDialog` ist nun noch die Funktion `LoadData` aufzurufen, um die Daten zu laden, bevor das Steuerelement für den Benutzer sichtbar ist. Bearbeiten Sie dazu die Funktion `OnInitDialog` gemäß Listing 9.5.

Listing 9.5: Die Funktion OnInitDialog

```

1: BOOL CActiveXDlg::OnInitDialog()
2: {
3: CDialog::OnInitDialog();
4: .
5: .
6: .
7: // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
8:

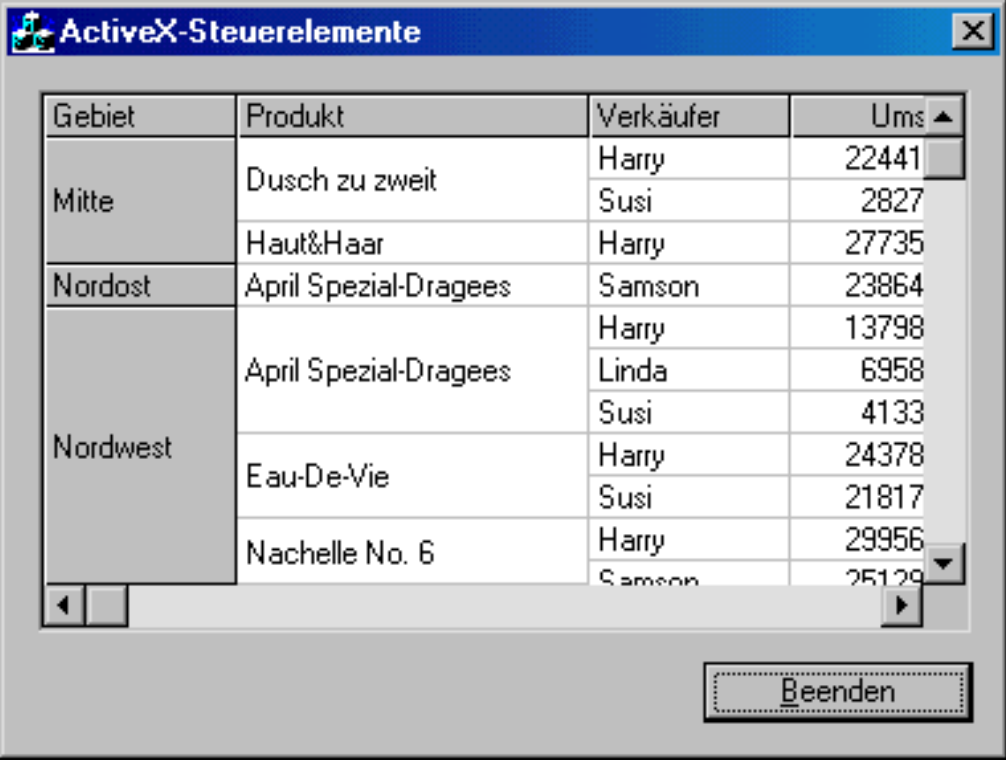
```

```

9: ////////////////////////////////////
10: // EIGENER CODE, ANFANG
11: ////////////////////////////////////
12:
13: // Daten in Grid-Steuerelement laden
14: LoadData();
15:
16: ////////////////////////////////////
17: // EIGENER CODE, ENDE
18: ////////////////////////////////////
19:
20: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
soll den ÂFokus erhalten
21: }

```

Wenn Sie die Anwendung in der jetzigen Entwicklungsphase kompilieren und ausführen, werden die Daten geladen und sortiert, wie es Abbildung 9.7 zeigt.



Gebiet	Produkt	Verkäufer	Ums ▲
Mitte	Dusch zu zweit	Harry	22441
		Susi	2827
	Haut&Haar	Harry	27735
Nordost	April Spezial-Dragees	Samson	23864
Nordwest	April Spezial-Dragees	Harry	13798
		Linda	6958
		Susi	4133
	Eau-De-Vie	Harry	24378
		Susi	21817
	Nachelle No. 6	Harry	29956
		Samson	25129

Abbildung 9.7:
FlexGrid mit Daten gefüllt

Auf Steuerelementereignisse reagieren

Das Steuerelement FlexGrid reagiert momentan noch nicht auf Eingaben, die Sie vielleicht in die Tabelle aufnehmen möchten. Wenn Sie auf eine der Zellen klicken und versuchen, den Wert zu ändern, passiert überhaupt nichts. Es ist also ein Steuerelementereignis hinzuzufügen, um die Eingabe zu behandeln. ActiveX-Steuerelemente machen verschiedene Ereignisse für die Nutzung in Visual C++-Anwendungen verfügbar. Mit dem Klassen-Assistenten können Sie sich die verfügbaren Ereignisse ansehen und bestimmen, welche Ereignisse für die gewünschte Funktionalität erforderlich sind und welche man ignorieren kann. Die meisten ActiveX-Steuerelemente verbinden mit den verfügbaren Ereignissen keine Standardfunktionalität, sondern erwarten, daß der Programmierer dem

Steuerelement sagt, was bei welchem Ereignis zu tun ist.

In der Beispielanwendung fügen wir zwei Steuerelementereignisse hinzu, um das Klicken und die Bewegung der Maus aufzufangen. Wenn der Benutzer auf einen Spaltenkopf klickt und ihn an eine andere Position zieht, soll sich die Anordnung der Spalten entsprechend ändern. Um diese Funktionalität zu implementieren, sind zwei Steuerelementereignisse aufzufangen, wenn die Maustaste gedrückt und wenn sie wieder losgelassen wird. Für das erste Ereignis müssen Sie prüfen, ob der Benutzer einen Spaltenkopf angeklickt hat. Wenn ja, ist die ausgewählte Spalte gemeint. Beim zweiten Ereignis ist die ausgewählte Spalte zu der Spalte zu verschieben, auf der die Maustaste losgelassen wurde.

Um diese Funktionalität zu erreichen, erzeugen Sie eine neue Klassenvariable, in der Sie die Nummer der angeklickten Spalte zwischen zwei Ereignissen verwalten. Fügen Sie in die Klasse `CActiveXDlg` eine neue Variable ein, genauso wie Sie weiter oben die Funktionen hinzugefügt haben. Legen Sie den Variablentyp als `int`, den Namen als `m_iMouseCol` und den Zugriff als Privat fest.

Die ausgewählte Spalte auffangen

Um das Klickereignis für das Steuerelement abzufangen, führen Sie die folgenden Schritte aus:

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion zur Behandlung der Nachricht `MouseDown` für das Objekt `IDC_MSFGGRID` hinzu.
2. In die Funktion übernehmen Sie den Code aus Listing 9.6.

Listing 9.6: Die Funktion `OnMouseDownMsfggrid`

```
1: void CActiveXDlg::OnMouseDownMsfggrid(short Button, short Shift,
long x, long y)
2: {
3: // TODO: Code für die Behandlungsroutine der
Steuerelement-Benachrichtigung Ähier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Hat Benutzer auf eine Datenzeile und nicht die
10: // Kopfzeile geklickt?
11: if (m_ctlFGrid.GetMouseRow() != 0)
12: {
13: // Wenn ja, dann Spaltenvariable auf 0 setzen
14: // und beenden
15: m_iMouseCol = 0;
16: return;
17: }
18: // Angeklickte Spalte sichern
19: m_iMouseCol = m_ctlFGrid.GetMouseCol();
20:
21: //////////////////////////////////////
22: // EIGENER CODE, ENDE
23: //////////////////////////////////////
```

24: }

Die Funktion prüft mit der Methode `GetMouseRow` die angeklickte Zeile. Wenn es sich nicht um die erste Zeile handelt, tragen Sie eine 0 in die Spaltenvariable ein und verlassen die Funktion. Andernfalls wird durch Aufruf der Methode `GetMouseCol` die angeklickte Spalte ermittelt. Die zurückgegebene Spaltennummer kommt in die Variable `m_iMouseCol`, die Sie gerade in die Klasse aufgenommen haben.

Spalte zum Zielort verschieben

Nachdem Sie nun die Nummer der ausgewählten Spalte ermittelt haben, müssen Sie die Spalte ermitteln, wo der Benutzer die Maustaste losläßt. Um das Ereignis beim Loslassen der Maus für das Steuerelement aufzufangen, führen Sie die folgenden Schritte aus:

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `MouseUp` des Objekts `IDC_MSFGGRID` hinzu.
2. Schreiben Sie in die Funktion den Code aus Listing 9.7.

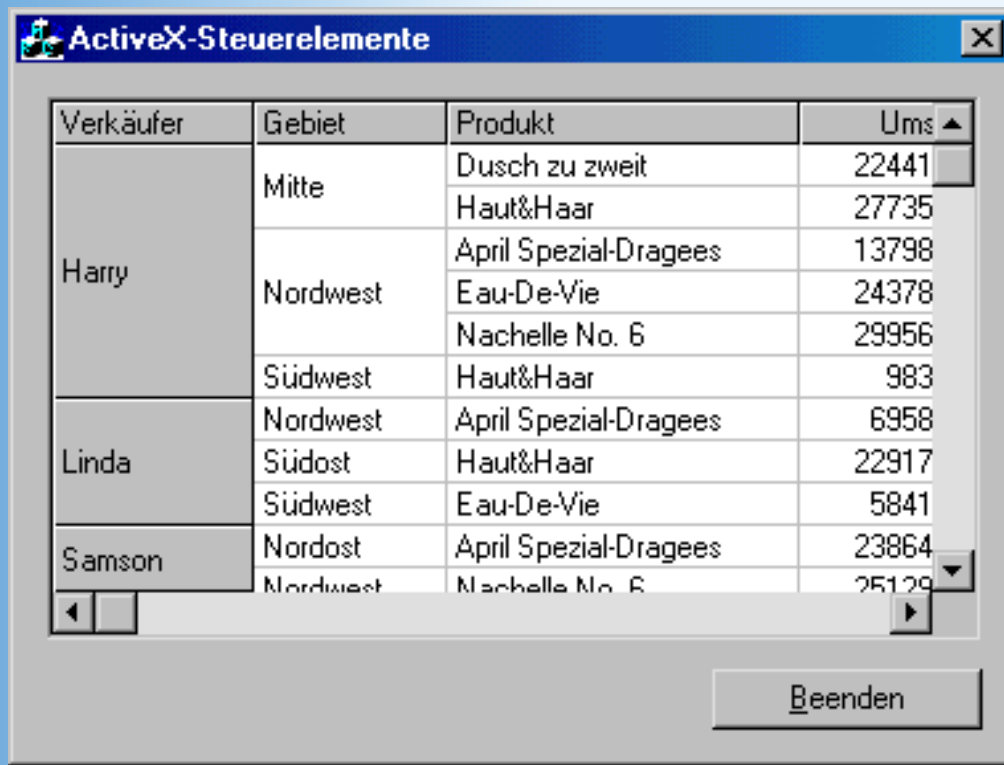
Listing 9.7: Die Funktion `OnMouseUpMsfgrid`

```

1: void CActiveXDlg::OnMouseUpMsfgrid(short Button, short Shift, long
x, long y)
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Wenn gewählte Spalte gleich erste Spalte,
10: // dann ist nichts zu tun.
11: if (m_iMouseCol == 0)
12: return;
13: // Neuzeichnen des Steuerelements abschalten
14: m_ctlFGrid.SetRedraw(FALSE);
15: // Gewählte Spaltenposition ändern
16: m_ctlFGrid.SetColPosition(m_iMouseCol, m_ctlFGrid.GetMouseCol());
17: // Tabelle neu sortieren
18: DoSort();
19: // Neuzeichnen wieder einschalten
20: m_ctlFGrid.SetRedraw(TRUE);
21:
22: ///////////////////////////////////
23: // EIGENER CODE, ENDE
24: ///////////////////////////////////
25: }
```

Die Funktion prüft zuerst, ob eine ausgewählte Spalte zu verschieben ist. Wenn nicht, wird die Funktion ohne weitere Aktionen verlassen. Ist eine Spalte markiert, schaltet die Funktion das Neuzeichnen des Steuerelements mit der Methode `SetRedraw` ab, damit die Verschiebung für den

Benutzer nicht sichtbar ist. Als nächstes verschiebt die Funktion mit der Methode `SetColPosition` die ausgewählte Spalte zu der Spalte, wo der Benutzer die Maustaste losgelassen hat. Anschließend erfolgt der Aufruf der Funktion `DoSort`, um die Tabelle neu zu sortieren. Schließlich wird das Neuzeichnen wieder aktiviert, um das Steuerelement zu aktualisieren und dem Benutzer die verschobene Spalte anzuzeigen. Wenn Sie die Anwendung kompilieren und ausführen, sollten Sie nun in der Lage sein, die Spalten per Drag & Drop der Spaltenköpfe zu verschieben, wie es Abbildung 9.8 zeigt.



Verkäufer	Gebiet	Produkt	Ums ▲
Harry	Mitte	Dusch zu zweit	22441
		Haut&Haar	27735
	Nordwest	April Spezial-Dragees	13798
		Eau-De-Vie	24378
		Nachelle No. 6	29956
Linda	Südwest	Haut&Haar	983
	Nordwest	April Spezial-Dragees	6958
	Südost	Haut&Haar	22917
Samson	Südwest	Eau-De-Vie	5841
	Nordost	April Spezial-Dragees	23864
	Nordwest	Nachelle No. 6	25129

Abbildung 9.8:
FlexGrid mit neu angeordneten Spalten

Zusammenfassung

Heute haben Sie gelernt, wie man ActiveX-Steuerelemente in Visual C++-Anwendungen einsetzt, um in einfacher Weise die Funktionalität der Anwendung zu erweitern. Zuerst wurde die grundsätzliche Arbeitsweise von ActiveX-Steuerelementen erläutert und gezeigt, wie sie mit der Container-Anwendung in Wechselwirkung treten. Weiterhin haben Sie gelernt, wie man ein ActiveX-Steuerelement in das Entwicklungsprojekt einbindet, so daß man das Steuerelement in der Anwendung nutzen kann. Es wurde dargestellt, wie Visual C++ verschiedene C++-Klassen erzeugt, um die hinzugefügten ActiveX-Steuerelemente abzukapseln, und wie man mit dem Steuerelement über die freigelegten Methoden dieser generierten C++-Klassen interagiert. Schließlich ging es darum, wie man Ereignisse auffängt, die das ActiveX-Steuerelement auslöst, so daß man in der Anwendung auf die Ereignisse reagieren kann.

Fragen und Antworten

Frage:
Wie kann ich ermitteln, welche Methoden verfügbar sind, wenn ich mit einem ActiveX-Steuerelement arbeite?

Antwort:

Untersuchen Sie die C++-Klassen, die Visual C++ erstellt, um das Steuerelement abzukapseln. Dadurch erhalten Sie einen Überblick über die verfügbare Funktionalität. Wenn Sie die Dokumentation zum Steuerelement besitzen, können Sie sie mit den C++-Klassen vergleichen, um herauszufinden, welche Klassenmethode welche Methode des Steuerelements aufruft. Untersuchen Sie auch die Liste der Ereignisse für das Steuerelement im Klassen-Assistenten. Hieraus können Sie ersehen, welche Ereignisse für das Steuerelement verfügbar sind.

Frage:

Wie kann ich die auf meinem Computer für andere Anwendungen installierten ActiveX-Steuerelemente in meinen Visual C++-Anwendungen nutzen?

Antwort:

Das hängt davon ab, wie die Steuerelemente lizenziert sind und welche Anwendung die Steuerelemente installiert hat. Wurden die Steuerelemente durch ein anderes Entwicklungswerkzeug installiert, besitzen Sie wahrscheinlich auch die Entwicklungslizenz für das Steuerelement. In diesem Fall sollten Sie es in Ihren Visual C++-Anwendungen einsetzen können. Wenn die Steuerelemente von einer Endbenutzer-Anwendung installiert wurden, wie zum Beispiel Word oder Quicken, werden Sie wahrscheinlich nur über die Laufzeitlizenz für das Steuerelement verfügen. Wenn Sie diese Steuerelemente in eigenen Anwendungen einsetzen möchten, müssen Sie sich mit dem Hersteller des Steuerelements in Verbindung setzen, um eine Entwicklungslizenz für das Steuerelement zu erwerben.

Frage:

Da mir das Steuerelement FlexGrid nicht erlaubt, Daten direkt in das Steuerelement einzugeben, wie kann ich dann meinen Benutzern die Möglichkeit bieten, Daten in das Steuerelement genau wie in ein Tabellenblatt einzugeben?

Antwort:

Um diese Funktionalität für das Steuerelement FlexGrid zu implementieren, müssen Sie ein unverankertes Eingabefeld in Ihrem Fenster bereitstellen. Im Code ermitteln Sie, welche Zelle der Benutzer bearbeiten möchte, und setzen dann das Eingabefeld vor diese Zelle. Durch diese Anordnung erhält der Benutzer das Gefühl, als würde er die Daten direkt in die Zelle eintragen. Bei einer anderen Lösung könnte man ein zusätzliches Feld zur Dateneingabe außerhalb der Tabelle vorsehen, das etwa wie die Bearbeitungsleiste in Excel funktioniert. Hier gibt der Benutzer seine Daten ein. Man kann die Zellen, durch die der Benutzer navigiert, hervorheben, um ihm eine Rückmeldung für seine Aktionen zu bieten.

Workshop

Kontrollfragen

1. Wie ruft der ActiveX-Container die Methoden in einem ActiveX-Steuerelement auf?
2. Wie löst ein ActiveX-Steuerelement Ereignisse in der Container-Anwendung aus?
3. Welche Option im Anwendungs-Assistenten muß aktiviert sein, damit ActiveX- Steuerelemente in einer Visual C++-Anwendung wie gewünscht funktionieren?
4. Wie erleichtert Visual C++ die Arbeit mit ActiveX-Steuerelementen?
5. Warum kann es schwierig sein, in Visual C++ mit älteren ActiveX-Steuerelementen zu arbeiten?

Übung

Modifizieren Sie die Anwendung, so daß der Benutzer auf einen Spaltenkopf doppelklicken kann, um die Spalte als erste Spalte der Tabelle anzuordnen.

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2**Tag 10****SDI-Anwendungen**

Heute erstellen Sie mit Visual C++ eine andere Art von Anwendungen als in den vergangenen Lektionen - SDI-Anwendungen oder Einzeldokumentanwendungen (SDI - Single Document Interface). Eine SDI-Anwendung ist eine dokumentbezogene Anwendung, die nur mit einem Dokument zu einem bestimmten Zeitpunkt und nur mit einem Typ von Dokument arbeiten kann.

Typische Beispiele für SDI-Anwendungen sind der Windows-Editor, WordPad und Paint. Diese Anwendungen können nur eine Art von Aufgabe realisieren und nur an einer Aufgabe zu einem bestimmten Zeitpunkt arbeiten. WordPad verhält sich wie eine SDI-Version der Textverarbeitung Word. Man kann damit viele Aufgaben wie in Word erledigen. Während es in Word aber möglich ist, an mehreren Dokumenten zur gleichen Zeit zu arbeiten, schränkt Sie WordPad auf nur ein einziges Dokument zu einem bestimmten Zeitpunkt ein.

Insbesondere lernen Sie heute, wie ...

- Visual C++ die Dokument/Ansicht-Architektur für das Erstellen von SDI-Anwendungen verwendet.
- man das Gerüst einer SDI-Anwendung erstellt.
- man die Daten von der visuellen Repräsentation der Daten trennt.
- man die Daten in eigenen C++-Klassen abkapselt.
- man die Interaktion zwischen den Daten und den Menüs realisiert.

Die Dokument/Ansicht-Architektur

Bei einer SDI-Anwendung erstellt der Anwendungs-Assistent mehr Klassen als bei einer dialogfeldbasierenden Anwendung. Die Klassen realisieren jeweils bestimmte Funktionsweisen einer SDI-Anwendung. Wenn man einmal von der Dialogfeldklasse `About` absieht, besteht eine SDI-Anwendung aus vier speziellen Klassen:

- die von `CWinApp` abgeleitete Klasse
- die von `CFrameView` abgeleitete Klasse
- die von `CDocument` abgeleitete Klasse
- die von `CView` abgeleitete Klasse

Die Klasse `CWinApp` erzeugt alle anderen Komponenten in der Anwendung. Diese Klasse empfängt alle Ereignisnachrichten und übergibt dann die Nachrichten an die Klassen `CFrameView` und `CView`.

Die Klasse `CFrameView` bildet den Fensterrahmen. Die Klasse nimmt die Menüs, Symbolleisten, Bildlaufleisten und alle anderen sichtbaren Objekte auf, die mit dem Rahmen verbunden sind. Die Klasse bestimmt, wieviel vom Dokument zu einem bestimmten Zeitpunkt zu sehen ist. Bei SDI-Anwendungen brauchen Sie an den beiden ersten Klassen nur wenige (oder überhaupt keine) Programmänderungen vorzunehmen.

Die Klasse `CDocument` nimmt Ihr Dokument auf. In dieser Klasse erstellen Sie die erforderlichen

Datenstrukturen, um die Daten des Dokuments zu speichern und zu verwalten. Die Klasse empfängt die Eingaben von der Klasse `CView` und übergibt Anzeigeinformationen an die Klasse `CView`. Außerdem ist die Klasse für das Speichern und Abrufen der Dokumentdaten in bzw. aus Dateien verantwortlich.

Die Klasse `CView` ist die Klasse, die die visuelle Repräsentation des Dokuments für den Benutzer anzeigt. Die Klasse reicht Eingabeinformationen an die Klasse `CDocument` weiter und empfängt Anzeigeinformationen aus der Klasse `CDocument`. Das meiste, was Sie in dieser Klasse realisieren, bezieht sich auf die Ausgabe des Dokuments für den Benutzer und die Behandlung der Benutzereingaben. Die Klasse `CView` hat mehrere abgeleitete Klassen, die man als Basisklassen für die Ansichtsklassen verwenden kann. Diese abgeleiteten Klassen sind in Tabelle 10.1 aufgeführt.

Tabelle 10.1: Von `CView` abgeleitete Klassen

Klasse	Beschreibung
<code>CEditView</code>	Liefert die Funktionalität eines Eingabefelds. Mit dieser Klasse lassen sich einfache Texteditoren implementieren.
<code>CFormView</code>	Die Basisklasse für Ansichten, die Steuerelemente enthalten. Mit dieser Klasse lassen sich formularbasierte Dokumente in Anwendungen bereitstellen.
<code>CHtmlView</code>	Liefert die Funktionalität eines Webbrowsers. Diese Ansicht behandelt direkt die URL-Navigation, Hyperlinks usw. Verwaltet eine Verlaufsliste für das Durchsuchen in Rückwärts- und Vorwärtsrichtung.
<code>CListView</code>	Stellt die Funktionalität von Listen in der Dokument/Ansicht-Architektur bereit.
<code>CRichEditView</code>	Realisiert Zeichen- und Absatzformatierungen. Mit dieser Klasse lassen sich Textverarbeitungen implementieren.
<code>CScrollView</code>	Stellt Bildlauffähigkeiten für eine <code>CView</code> -Klasse bereit.
<code>CTreeView</code>	Stellt die Funktionalität zur Steuerung von Bäumen in der Dokument/Ansicht-Architektur bereit.

Alle vier Klassen arbeiten zusammen, um die vollständige Funktionalität einer SDI-Anwendung bereitzustellen, wie es Abbildung 10.1 zeigt. Auf der Grundlage dieser Architektur lassen sich relativ einfach leistungsfähige dokumentbezogene Anwendungen aufbauen.



**Abbildung 10.1:
Die Dokument/Ansicht-Architektur**



Lassen Sie sich vom Begriff *Dokument* nicht täuschen. Es bedeutet nicht, daß Sie nur Anwendungen wie Textverarbeitungen und Tabellenkalkulationen erstellen können. Im weiteren Sinne bezeichnet man mit Dokument die Daten, die Ihre Anwendung verarbeitet, während sich die Ansicht auf die visuelle Repräsentation der Daten bezieht. Beispielsweise könnte man die Anwendung Solitär als Dokument/Ansicht-Anwendung erstellen, wobei das Dokument die Karten und deren Lage im Spielfeld darstellt. In diesem Fall handelt es sich bei der Ansicht um die Anzeige der Karten, wobei jede Karte dort erscheint, wo es das Dokument festgelegt hat.

Eine SDI-Anwendung erstellen

Um eine Vorstellung von der Arbeitsweise der Dokument/Ansicht-Architektur in einer realen Anwendung zu bekommen, erstellen Sie heute eine neue Version der Zeichenanwendung aus Lektion 3. In der neuen Version bleibt die Zeichnung des Benutzers dauerhaft erhalten. Das bedeutet, daß die Zeichnung nicht gelöscht wird, wenn ein anderes Fenster über der Anwendung liegt. Außerdem kann man jetzt die Zeichnungen speichern und wiederherstellen.

Das Anwendungsgerüst erstellen

Das Gerüst für die heutige Anwendung erstellen Sie in folgenden Schritten:

1. Legen Sie mit dem Anwendungs-Assistenten ein neues Projekt namens **Tag10** an.
2. Im ersten Dialogfeld des Anwendungs-Assistenten wählen Sie die Option *Einzelnes Dokument (SDI)*.
3. Im zweiten Schritt des Assistenten übernehmen Sie die Standardwerte.
4. Im dritten Dialogfeld des Assistenten schalten Sie das Kontrollkästchen für die Unterstützung von ActiveX-Steuerelementen aus.
5. Behalten Sie die Standardwerte im vierten Dialogfeld des Anwendungs-Assistenten bei, und klicken Sie auf die Schaltfläche **Weitere Optionen**.
6. Im Dialogfeld **Weitere Optionen** geben Sie eine aus drei Buchstaben bestehende Erweiterung für die Dateien ein, die Ihre Anwendung generiert (beispielsweise **dhc** oder **dvp**). Klicken Sie auf die Schaltfläche **Schliessen**, um das Dialogfeld zu schließen, und klicken Sie dann auf **Weiter**, um zum nächsten Schritt des Assistenten zu gelangen.
7. Übernehmen Sie wieder die Standardwerte im fünften Dialogfeld des Anwendungs-Assistenten.
8. Im sechsten und letzten Schritt des Assistenten können Sie die Basisklasse wählen, von der Ihre Ansichtsklasse abgeleitet wird. Übernehmen Sie den Vorgabewert **CView**, und klicken Sie auf **Fertigstellen**. Der Anwendungs-Assistent generiert nun das Gerüst der Anwendung.

Eine Linienklasse erzeugen

Als erstes beschäftigen wir uns mit der Frage, wie man die Daten in der Dokumentklasse darstellt. Bei jeder Zeichenanwendung hat man es mit einer Reihe von Linien zu tun. Jede Linie besteht aus einem Anfangspunkt und einem Endpunkt. Man könnte nun annehmen, daß man für die Datendarstellung eine Folge von Punkten verwendet. In diesem Fall müssen Sie auch spezielle Anpassungen vornehmen, wo die Folge von Linien zwischen den Punkten endet und die nächste beginnt. Es ist sinnvoller, die Zeichnung als Folge von Linien zu repräsentieren. Damit kann man jede einzelne Linie, die im Fenster gezeichnet wird, speichern, ohne sich darum kümmern zu müssen, wo eine Gruppe von zusammenhängenden Punkten endet und die nächste beginnt.

Leider stellt die MFC-Bibliothek keine Klasse für Linienobjekte bereit, obwohl es eine Klasse für Punktoobjekte (**CPoint**) gibt. Es bleibt Ihnen nichts weiter übrig, als ihre eigene Linienklasse in folgenden Schritten zu erzeugen:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**, und markieren Sie das oberste Objekt im Baum (**Tag10 Klassen**). Klicken Sie mit der rechten Maustaste, und wählen Sie den Befehl **Neue Klasse** aus dem Kontextmenü.
2. Im Dialogfeld **Neue Klasse** wählen Sie als Klassentyp *Allgemeine Klasse*. Geben Sie als Klassennamen **CLine** ein, und klicken Sie auf die erste Zeile im Listenfeld **Basisklasse**. Geben Sie **CObject** als Basisklasse ein, und belassen Sie den Zugriff als **public**, wie es Abbildung 10.2 zeigt.

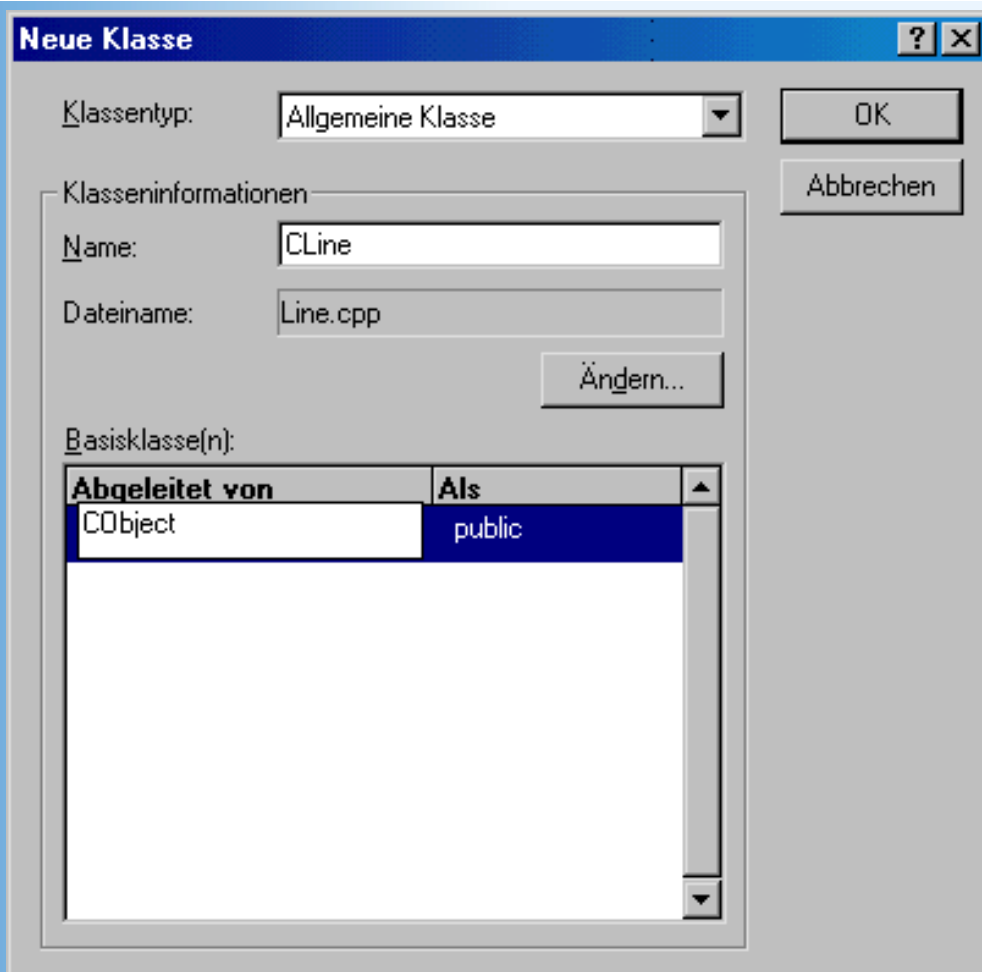


Abbildung 10.2:
Der Assistent Neue Klasse

3. Wenn Sie auf **OK** klicken, um die Klasse **CLine** hinzuzufügen, erhalten Sie den Hinweis, daß der Klassen-Assistent die passende Header-Datei zur Ableitung der Klasse **CLine** von der Klasse **CObject** nicht finden kann (siehe Abbildung 10.3). Klicken Sie auf **OK**, um dieses Meldungsfeld zu schließen.

Die passende Header-Klasse ist bereits in die Dateien der Klasse **CLine** eingebunden. Solange sich der Compiler beschwert, daß er die Definition für das **CObject**-Objekt nicht finden kann, kümmern Sie sich nicht um diese Meldung. Wenn Sie allerdings eine Basisklasse verwenden, die etwas tiefer in der MFC-Klassenhierarchie angesiedelt ist, sollten Sie diese Meldung ernst nehmen und die passende Header-Datei zu den `#include`-Anweisungen in der Quelldatei der Klasse hinzufügen.

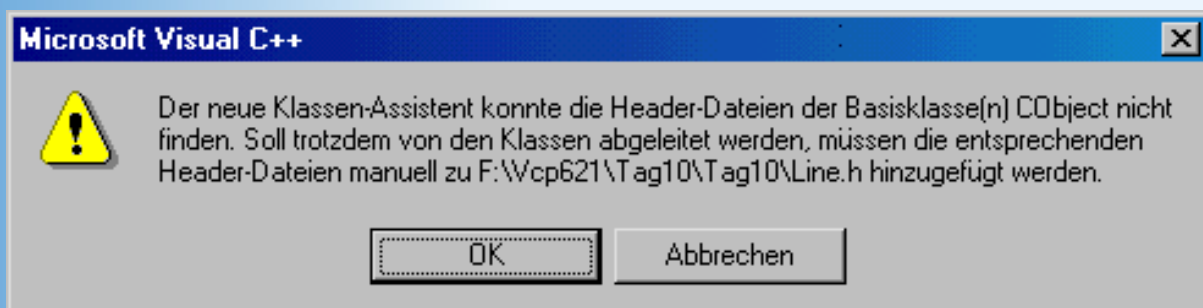


Abbildung 10.3:
Warnung, daß Header-Dateien der Basisklassen nicht gefunden werden konnten

Die Klasse CLine konstruieren

Momentan muß Ihre **CLine**-Klasse nur zwei Datenelemente aufnehmen, und zwar die beiden Endpunkte der Linie, die sie repräsentiert. Fügen Sie diese beiden Datenelemente hinzu, und nehmen Sie auch einen

Klassenkonstruktor auf, der beide Werte setzt, wenn die Instanz der Klasse erzeugt wird. Führen Sie dazu folgende Schritte aus:

1. Markieren Sie im Arbeitsbereich auf der Registerkarte **Klassen** die Klasse **CLine**.
2. Klicken Sie mit der rechten Maustaste auf die Klasse **CLine**, und wählen Sie den Befehl **Member-Variable hinzufügen** aus dem Kontextmenü.
3. Geben Sie **CPoint** als Variablentyp und **m_ptFrom** als Variablenname ein. Als Zugriff wählen Sie die Option *Privat*. Klicken Sie auf **OK**, um die Variable hinzuzufügen.
4. Wiederholen Sie die Schritte 2 und 3, nennen Sie die Variable diesmal **m_ptTo**.
5. Klicken Sie mit der rechten Maustaste auf die Klasse **CLine**, und wählen Sie **Member-Funktion hinzufügen** aus dem Kontextmenü.
6. Lassen Sie den Funktionstyp leer, und geben Sie als Funktionsdeklaration **CLine(CPoint ptFrom, CPoint ptTo)** ein. Klicken Sie auf **OK**, um die Funktion hinzuzufügen.
7. In die neue Funktion übernehmen Sie den Code aus Listing 10.1.

Listing 10.1: Der Konstruktor von CLine

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo)
2: {
3: // Anfangs- und Endpunkte initialisieren
4: m_ptFrom = ptFrom;
5: m_ptTo = ptTo;
6: }
```

Dieser Objektkonstruktor initialisiert die Anfangs- und Endpunkte mit den Punkten, die man an den Konstruktor übergibt.

Die Klasse CLine zeichnen

Nach den Regeln des objektorientierten Entwurfs sollte Ihre Klasse **CLine** in der Lage sein, sich selbst zu zeichnen. Wenn die Ansichtsklasse die Linie anzeigen soll, kann sie dann mit einer Nachricht das Linienobjekt anweisen, sich selbst neu zu zeichnen. Diese Funktionalität realisieren Sie in folgenden Schritten:

1. Wählen Sie **Member-Funktion hinzufügen** aus dem Kontextmenü, um eine neue Funktion in die Klasse **CLine** einzufügen.
2. Legen Sie den Funktionstyp als **void** und die Funktionsdeklaration als **Draw(CDC *pDC)** fest.
3. Schreiben Sie den Code aus Listing 10.2 in die gerade erstellte Funktion **Draw**.

Listing 10.2: Die Funktion Draw der Klasse CLine

```
1: void CLine::Draw(CDC * pDC)
2: {
3: // Linie zeichnen
4: pDC->MoveTo(m_ptFrom);
5: pDC->LineTo(m_ptTo);
6: }
```

Diese Funktion ist fast identisch mit der, die Sie letzte Woche erstellt haben. Es handelt sich um eine einfache Funktion, die eine Bewegung zum ersten Punkt im Gerätekontext ausführt und dann eine Linie zum zweiten Punkt im Gerätekontext zeichnet.

Die Funktionalität des Dokuments implementieren

Nachdem Sie nun über ein Objekt verfügen, mit dem sich die vom Benutzer ausgeführten Zeichnungen darstellen lassen, können Sie diese `CLine`-Objekte im Dokumentobjekt in einem einfachen dynamischen Array speichern. Für dieses Array fügen Sie eine `CObArray`-Elementvariable in die Dokumentklasse ein.

Die Klasse `CObArray` ist eine Objektarrayklasse, die sich selbst dynamisch in der Größe an die Anzahl der eingefügten Elemente anpaßt. Die Klasse kann beliebige Objekte aufnehmen, die von der Klasse `CObject` abgeleitet sind. Die Anzahl der Objekte ist lediglich durch die Größe des Hauptspeichers begrenzt. Die MFC-Bibliothek enthält weitere dynamische Arrayklassen wie `CStringArray`, `CByteArray`, `CWordArray`, `CDWordArray` und `CPtrArray`. Diese Klassen unterscheiden sich durch den Objekttyp, den sie aufnehmen können.

Fügen Sie das `CObArray` in `CTag10Doc` mit dem Assistenten Member-Variable hinzufügen ein, und benennen Sie das Objekt mit `m_oaLines`.

Linien hinzufügen

Als erstes müssen Sie der Dokumentklasse beibringen, wie man neue Linien hinzufügt. Dazu sind lediglich die Anfangs- und Endpunkte zu holen, ein neues Linienobjekt anzulegen und dann das Objekt in das Objektarray einzufügen. Fügen Sie eine neue Member-Funktion in die Klasse `CTag10Doc` ein. Legen Sie den Funktionstyp als `CLine*`, die Deklaration mit `AddLine(CPoint ptFrom, CPoint ptTo)` und den Zugriffsstatus als `Public` fest. Schreiben Sie den Code aus Listing 10.3 in die Funktion.

Listing 10.3: Die Funktion `AddLine` der Klasse `CTag10Doc`

```
1: CLine * CTag10Doc::AddLine(CPoint ptFrom, CPoint ptTo)
2: {
3: // Ein neues CLine-Objekt erzeugen
4: CLine *pLine = new CLine(ptFrom, ptTo);
5: try
6: {
7: // Die neue Linie in das Objektarray einfügen
8: m_oaLines.Add(pLine);
9: // Dokument als bearbeitet markieren
10: SetModifiedFlag();
11: }
12: // Ist Speicherausnahme aufgetreten?
13: catch (CMemoryException* perr)
14: {
15: // Meldung für Benutzer, schlechte Neuigkeiten
16: // mitteilen.
17: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
18: // Wurde Linienobjekt erzeugt?
19: if (pLine)
20: {
21: // Objekt löschen
22: delete pLine;
23: pLine = NULL;
24: }
25: // Ausnahmeobjekt löschen
26: perr->Delete();
27: }
28: return pLine;
29: }
```

Auf den ersten Blick handelt es sich um eine unkomplizierte Funktion. Als erstes legt sie eine neue Instanz von `CLine` an und übergibt an den Konstruktor die Anfangs- und Endpunkte als Argumente. Unmittelbar darauf erscheint die folgende interessante Konstruktion, die den Programmablauf steuert:

```
try
{
.
.
.
}
catch (...)
{
.
.
.
}
```

Was soll das bedeuten? Diese Konstruktion ist ein Beispiel der strukturierten Ausnahmebehandlung. Bestimmte Codeabschnitte könnten aufgrund eines unvorhergesehenen Umstandes scheitern, beispielsweise wegen Speichermangels oder unzureichendem Platz auf dem Datenträger. Den problematischen Codeabschnitt können Sie in einen `try`-Block einschließen. Diesem sollten sich immer ein oder mehrere `catch`-Blöcke anschließen. Wenn ein Problem während der Codeausführung im `try`-Block auftritt, springt das Programm unmittelbar zu den `catch`-Blöcken. Jeder `catch`-Block legt fest, welchen Ausnahmetyp er behandelt (im speziellen Fall der Funktion `AddLine` behandelt er lediglich Ausnahmen infolge Speichermangels). Gibt es für das aufgetretene Problem einen entsprechenden `catch`-Block, wird dieser Block ausgeführt und bietet der Anwendung die Möglichkeit, Gegenmaßnahmen einzuleiten. Wenn kein `catch`-Block für den aufgetretenen Problemtyp vorhanden ist, springt das Programm zu einer Standardbehandlungsroutine für Ausnahmen - die wahrscheinlich Ihre Anwendung herunterfährt. Weitere Informationen zur strukturierten Ausnahmebehandlung finden Sie im Anhang A.

Innerhalb des `try`-Blocks fügen Sie die neue `CLine`-Instanz in das Array der Linienobjekte ein. Als nächstes rufen Sie die Funktion `SetModifiedFlag` auf, um das Dokument als bearbeitet (noch nicht gespeichert) zu markieren. Wenn der Benutzer die Anwendung schließt oder ein anderes Dokument öffnen, ohne zuerst die aktuelle Zeichnung zu speichern, fordert ihn die Anwendung auf, die aktuelle Zeichnung zu speichern (mit dem bekannten Meldungsfeld Ja, Nein, Abbrechen).

Im `catch`-Block informieren Sie den Benutzer, daß das System keinen Speicher mehr zur Verfügung hat, und löschen dann das `CLine`-Objekt und das Ausnahmeobjekt.

Schließlich gibt die Funktion das `CLine`-Objekt an die aufrufende Routine zurück. Damit kann das Ansichtsbjekt das Linienobjekt veranlassen, sich selbst zu zeichnen.

Die Linienanzahl ermitteln

Als nächstes Element nehmen Sie in die Dokumentklasse eine Funktion auf, die die Anzahl der Linien im Dokument zurückgibt. Diese Funktionalität ist erforderlich, da das Ansichtsbjekt in einer Schleife das Linienarray durchgehen muß, um jedes Linienobjekt anzuweisen, sich selbst zu zeichnen. Das Ansichtsbjekt muß die Gesamtzahl der Linien im Dokument bestimmen und jede gewünschte Linie aus dem Dokument zurückgeben können.

Um die Anzahl der Linien im Dokument zurückzugeben, ermittelt man die Anzahl der Linien im Objektarray, so daß man einfach den Rückgabewert der Methode `GetSize` der Klasse `CObArray` zurückgeben kann. Fügen Sie dazu eine neue Elementfunktion in die Klasse `CTag10Doc` ein. Legen Sie den Typ als `int` und die Deklaration als `GetLineCount` mit Zugriffsstatus `Public` fest. In die Funktion übernehmen Sie den Code aus Listing 10.4.

Listing 10.4: Die Funktion GetLineCount der Klasse CTag10Doc

```

1: int CTag10Doc::GetLineCount()
2: {
3: // Größe des Arrays zurückgeben
4: return m_oaLines.GetSize();
5: }

```

Eine bestimmte Linie abrufen

Schließlich brauchen Sie noch eine Funktion, die eine bestimmte Linie aus dem Dokument liefert. Dazu gibt man einfach das Objekt an der angegebenen Position im Objektarray zurück. Fügen Sie eine neue Elementfunktion in die Klasse CTag10Doc ein. Legen Sie den Typ als CLine*, die Deklaration als GetLine(int nIndex) und den Zugriff mit Public fest. In die Funktion schreiben Sie den Code aus Listing 10.5.

Listing 10.5: Die Funktion GetLine der Klasse CTag10Doc

```

1: CLine* CTag10Doc::GetLine(int nIndex)
2: {
3: // Zeiger auf Linienobjekt für die angegebene
4: // Position im Objektarray zurückgeben
5: return (CLine*)m_oaLines[nIndex];
6: }

```



Beachten Sie, daß das zurückzugebende Objekt in den Typ eines CLine-Objekts umzuwandeln ist. Da die Klasse CObArray ein Array von CObjects ist, stellt jedes vom Array gelieferte Element eine Instanz von CObject und nicht eine Instanz eines CLine-Objekts dar.

Die Zeichnung anzeigen

Die Dokumentklasse verfügt nun über Fähigkeit, die Zeichnung zu speichern. Als nächstes sind im Ansichtsbjekt die Benutzereingaben zu lesen und dann anzuzeigen. Die Mausereignisse zur Übernahme der Benutzereingaben sind fast identisch zu denen, die Sie letzte Woche erstellt haben. Der zweite Teil der Funktionalität betrifft die Ausgabe der Zeichnung. Dazu erweitern Sie eine bereits vorhandene Funktion in der Ansichtsbjektklasse.

Bevor Sie diese Funktionen hinzufügen, brauchen Sie noch eine Elementvariable in der Klasse CTag10View, um die vorherige Position des Mauszeigers festzuhalten (genau wie in der Anwendung der letzten Woche). Fügen Sie über den Arbeitsbereich eine Elementvariable in die Klasse CTag10View ein. Legen Sie den Typ als CPoint, den Namen als m_ptPrevPos und den Zugriff als Privat fest.

Die Mausereignisse hinzufügen

Die Benutzereingaben fangen Sie mit Mausereignissen ab. Öffnen Sie dazu den Klassen-Assistenten, und fügen Sie der Klasse CTag10View Funktionen für die Nachrichten WM_LBUTTONDOWN, WM_LBUTTONUP und WM_MOUSEMOVE hinzu. Den Code für diese Funktionen finden Sie in Listing 10.6.

Listing 10.6: Die Mausfunktionen der Klasse CTag10View

```

1: void CTag10View::OnLButtonDown(UINT nFlags, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
  Äund/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG

```

```

7: ///////////////////////////////////
8:
9: // Maus auffangen, damit sie keine andere Anwendung
10: // fangen kann, wenn sie den Fensterbereich verläßt
11: SetCapture();
12: // Punkte sichern
13: m_ptPrevPos = point;
14:
15: ///////////////////////////////////
16: // EIGENER CODE, ENDE
17: ///////////////////////////////////
18:
19: CView::OnLButtonDown(nFlags, point);
20: }
21:
22: void CTag10View::OnLButtonUp(UINT nFlags, CPoint point)
23: {
24: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
    Âund/oder Standard aufrufen
25:
26: ///////////////////////////////////
27: // EIGENER CODE, ANFANG
28: ///////////////////////////////////
29:
30: // Wurde Maus aufgefangen?
31: if (GetCapture() == this)
32: // Wenn ja, freigeben, damit sie anderen
33: // Anwendungen zur Verfügung steht.
34: ReleaseCapture();
35:
36: ///////////////////////////////////
37: // EIGENER CODE, ENDE
38: ///////////////////////////////////
39:
40: CView::OnLButtonUp(nFlags, point);
41: }
42:
43: void CTag10View::OnMouseMove(UINT nFlags, CPoint point)
44: {
45: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
    Âund/oder Standard aufrufen
46:
47: ///////////////////////////////////
48: // EIGENER CODE, ANFANG
49: ///////////////////////////////////
50:
51: // Prüfen, ob linke Maustaste gedrückt
52: if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
53: {
54: // Wurde Maus aufgefangen?
55: if (GetCapture() == this)
56: {
57: // Gerätekontext holen

```



```

58: CClientDC dc(this);
59:
60: // Die Linie ins Dokument aufnehmen
61: CLine *pLine = GetDocument()->AddLine(m_ptPrevPos, point);
62:
63: // Aktuellen Linienzug zeichnen
64: pLine->Draw(&dc);
65:
66: // Aktuellen Punkt als vorherigen speichern
67: m_ptPrevPos = point;
68: }
69: }
70:
71: //////////////////////////////////////
72: // EIGENER CODE, ENDE
73: //////////////////////////////////////
74:
75: CView::OnMouseMove(nFlags, point);
76: }

```

Die Funktion `OnLButtonDown` ruft zuerst die Funktion `SetCapture` auf. Diese Funktion »fängt« die Maus und verhindert damit, daß irgendwelche anderen Anwendungen irgendwelche Mausereignisse erhalten, selbst wenn die Maus den Fensterbereich der Anwendung verläßt. Damit kann der Benutzer die Maus aus dem Anwendungsfenster herausziehen und dann wieder in das Fenster zurückführen, ohne die Zeichnung unterbrechen zu müssen. Alle Mausnachrichten gelangen zu dieser Anwendung, bis die Maus in der Funktion `OnLButtonUp` durch Aufruf der Funktion `ReleaseCapture` freigegeben wird. In der Zeit dazwischen können Sie ermitteln, ob die Anwendung die Maus gefangen hat, indem Sie die Funktion `GetCapture` in einer `if`-Anweisung einsetzen und den Rückgabewert der Funktion mit `this` vergleichen. Wenn die Maus gefangen wurde, führen Sie den übrigen Code in diesen Funktionen aus, andernfalls finden keine weiteren Aktionen statt.

Die Funktion `OnMouseMove` erzeugt zuerst den Gerätekontext und realisiert dann mehrere Dinge in einer einzigen Codezeile:

```
CLine *pLine = GetDocument()->AddLine(m_ptPrevPos, point);
```

Diese Zeile erzeugt einen neuen Zeiger auf eine Instanz der Klasse `CLine`. Als nächstes wird die Funktion `GetDocument` aufgerufen, die einen Zeiger auf das Dokumentobjekt zurückgibt. Mit diesem Zeiger ruft die Funktion `OnMouseMove` die Funktion `AddLine` der Dokumentklasse auf und übergibt die vorherigen und aktuellen Punkte als Argumente. Mit dem Rückgabewert der Funktion `AddLine` wird der Objektzeiger von `CLine` initialisiert. Der `CLine`-Zeiger läßt sich nun verwenden, um die Funktion `Draw` des Linienobjekts aufzurufen.



Ein Zeiger ist die Adresse eines Objekts. Mit Zeigern lassen sich Objekte in einem Programm effizienter übergeben. Wenn man einen Zeiger auf ein Objekt statt das Objekt selbst übergibt, verhält sich das genauso, wie wenn man sagt »Die Fernbedienung liegt auf dem Sofa zwischen dem zweiten und dritten Kissen neben dem herausgefallenen Kleingeld«, statt jemandem die Fernbedienung direkt in die Hand zu drücken. Gibt man die eigentliche Fernbedienung jemand anderem, muß man - um im Programmierjargon zu sprechen - eine genaue Kopie der Fernbedienung erstellen und die Kopie an die andere Person aushändigen. Offensichtlich ist es einfacher, jemandem nur die Stelle zu sagen, wo sich die Fernbedienung finden läßt, anstatt eine genaue Kopie der Fernbedienung zu bauen.

Die Notation `->` bedeutet, daß der Zugriff auf die Funktionen oder Eigenschaften eines Objekts über einen Zeiger erfolgt. Mit einem Punkt `.` kennzeichnet man dagegen den Zugriff über das Objekt

Die Zeichnung darstellen

In der Ansichtsklasse wird die Funktion `OnDraw` immer dann aufgerufen, wenn das für den Benutzer ausgegebene Bild neu zu zeichnen ist. Es kann zum Beispiel sein, daß ein anderes Fenster vor dem Anwendungsfenster gelegen hat, das Fenster vom minimierten Zustand gerade wiederhergestellt worden ist oder ein neues Dokument aus einer Datei geladen wurde. Warum die Ansicht neu zu zeichnen ist, spielt keine Rolle. Als Anwendungsentwickler müssen Sie sich nur darum kümmern, den Code in die Funktion `OnDraw` aufzunehmen, um das Dokument wiederzugeben, für dessen Erstellung Ihre Anwendung vorgesehen ist.

Gehen Sie zur Funktion `OnDraw` in der Klasse `CTag10View`, und fügen Sie den Code aus Listing 10.7 ein.

Listing 10.7: Die Funktion `OnDraw` der Klasse `CTag10View`

```

1: void CTag10View::OnDraw(CDC* pDC)
2: {
3:     CTag10Doc* pDoc = GetDocument();
4:     ASSERT_VALID(pDoc);
5:
6:     // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen Daten
    Ähinzufügen
7:
8:     ///////////////////////////////////
9:     // EIGENER CODE, ANFANG
10:    ///////////////////////////////////
11:
12:    // Anzahl der Linien im Dokument ermitteln
13:    int liCount = pDoc->GetLineCount();
14:
15:    // Gibt es Linien im Dokument?
16:    if (liCount)
17:    {
18:        int liPos;
19:        CLine *lptLine;
20:
21:        // Schleife durch die Linien des Dokuments
22:        for (liPos = 0; liPos < liCount; liPos++)
23:        {
24:            // Anfangs- und Endpunkte für alle Zeilen holen
25:            lptLine = pDoc->GetLine(liPos);
26:            // Linie zeichnen
27:            lptLine->Draw(pDC);
28:        }
29:    }
30:
31:    ///////////////////////////////////
32:    // EIGENER CODE, ENDE
33:    ///////////////////////////////////
34: }
```

Die Funktion ermittelt zuerst, wie viele Linien im Dokument zu zeichnen sind. Gibt es keine Linien, ist auch nichts zu tun. Sind im Dokument Linien gespeichert, geht die Funktion in einer `for`-Schleife durch die Linien, holt jede Linie aus dem Dokument und ruft dann die Funktion `Draw` des Linienobjekts auf.

Bevor Sie Ihre Anwendung kompilieren und ausführen können, müssen Sie noch die Header-Datei für die Klasse `CLine` in die Quellcodedatei für die Dokument- und Ansichtsklassen einbinden. Fügen Sie dazu in beide Dateien (`Tag10Doc.cpp` und `Tag10View.cpp`) die Datei `Line.h` als Include-Datei hinzu, wie es Listing 10.8 zeigt.

Listing 10.8: Die Include-Anweisungen von `CTag10Doc`

```
1: #include "stdafx.h"
2: #include "Tag10.h"
3: #include "MainFrm.h"
4: #include "Line.h"
5: #include "Tag10Doc.h"
```

Jetzt sollten Sie Ihre Anwendung kompilieren und ausführen können. Es ist nun möglich, Figuren zu zeichnen, wie es Abbildung 10.4 zeigt. Wenn Sie das Fenster minimieren und dann wiederherstellen, oder wenn Sie das Fenster einer anderen Anwendung vor das Fenster Ihrer Anwendung legen, sollte Ihre Zeichnung wieder erscheinen, sobald das Fenster Ihrer Anwendung wieder sichtbar ist (im Gegensatz zur Anwendung, die Sie vor einer Woche erstellt haben).

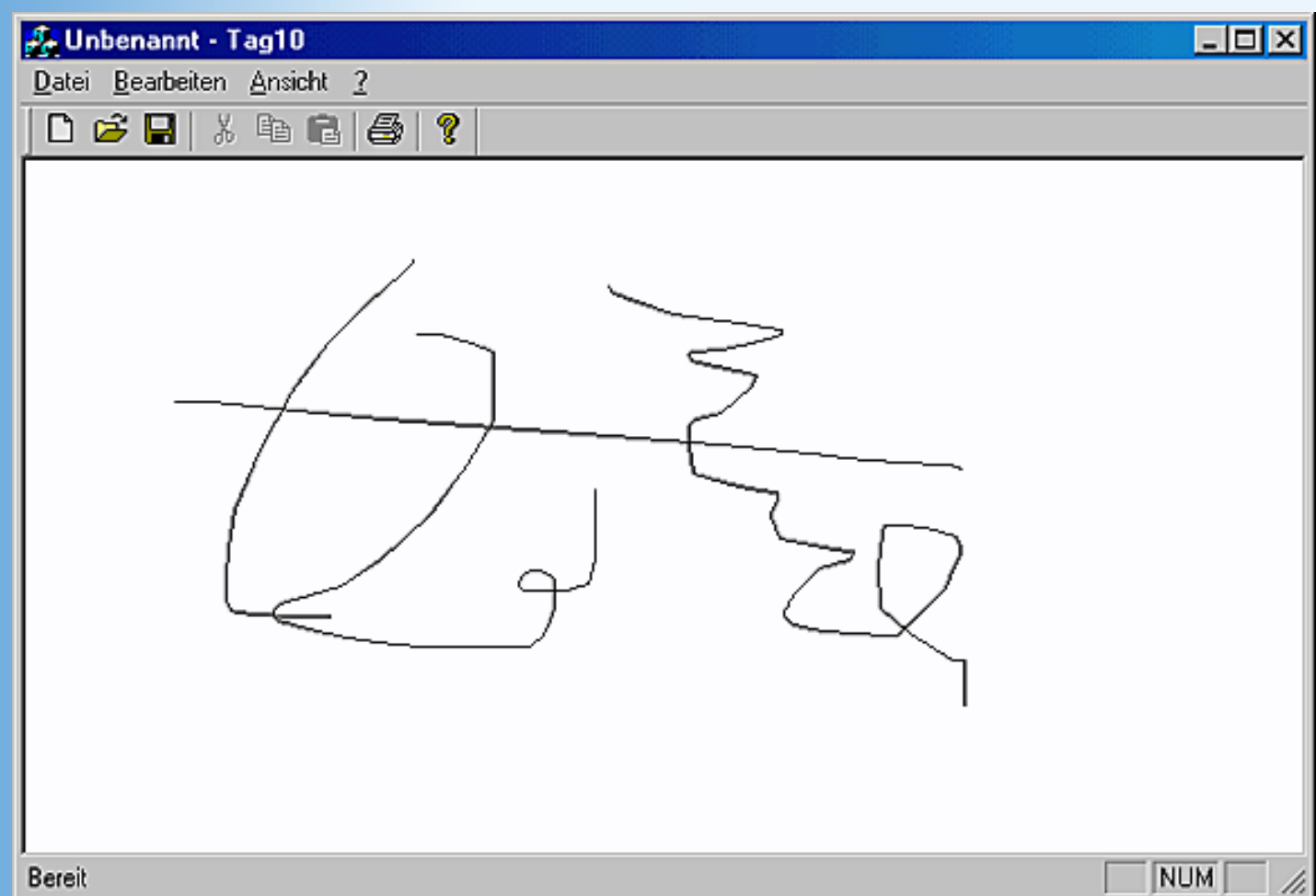


Abbildung 10.4:
Mit der Anwendung zeichnen

Die Zeichnung speichern und laden

Nachdem Ihre Gemälde nun nicht mehr verschwinden, sobald Sie sie aus den Augen lassen, wäre es schön, wenn man sie auch für die Ewigkeit festhalten könnte. Im Menü `Datei` Ihrer Anwendung sind bereits die Menübefehle `Öffnen`, `Speichern` und `Speichern unter` vorhanden, die aber momentan noch nichts bewirken. Die Menübefehle zum Drucken funktionieren zwar schon, aber die Befehle zum Speichern und Laden einer Zeichnung noch nicht. Nicht einmal beim Menübefehl `Neu` passiert etwas. Machen wir uns also an die Arbeit.

Die aktuelle Zeichnung löschen

Wenn Sie die Klasse `Ctag10Doc` untersuchen, fällt Ihnen sicherlich die Funktion `OnNewDocument` auf, in der Sie den Code einbauen könnten, um die aktuelle Zeichnung zu löschen. Weit gefehlt! Diese Funktion ist dafür vorgesehen, alle Klasseneinstellungen zu initialisieren, um mit der Arbeit an einer neuen Zeichnung zu beginnen, und nicht um eine vorhandene Zeichnung zu löschen. Statt dessen müssen Sie den Klassen-Assistenten öffnen und eine Funktion für die Nachricht `DeleteContents` hinzufügen. Diese Nachricht ist dafür vorgesehen, den aktuellen Inhalt der Dokumentklasse zu löschen. Nehmen Sie in die neue Funktion den Code aus Listing 10.9 auf.

Listing 10.9: Die Funktion `DeleteContents` der Klasse `Ctag10Doc`

```
1: void Ctag10Doc::DeleteContents()
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Anzahl der Linien im Objektarray ermitteln
10: int liCount = m_oaLines.GetSize();
11: int liPos;
12:
13: // Gibt es Objekte im Array?
14: if (liCount)
15: {
16: // Schleife durch Array, dabei jedes Objekt löschen
17: for (liPos = 0; liPos < liCount; liPos++)
18: delete m_oaLines[liPos];
19: // Array zurücksetzen
20: m_oaLines.RemoveAll();
21: }
22:
23: //////////////////////////////////////
24: // EIGENER CODE, ENDE
25: //////////////////////////////////////
26:
27: CDocument::DeleteContents();
28: }
```

Die Funktion geht in einer Schleife durch das Objektarray und löscht alle Linienobjekte im Array. Nachdem alle Linien gelöscht sind, setzt die Funktion das Array durch Aufruf der Methode `RemoveAll` zurück. Wenn Sie die Anwendung kompilieren und ausführen, können Sie den Befehl `Datei / Neu` wählen, und wenn Sie sich nicht dafür entscheiden, die aktuelle Zeichnung zu speichern, wird der Fensterinhalt gelöscht.

Die Zeichnung speichern und wiederherstellen

Die Funktionalität zum Speichern und Wiederherstellen der Zeichnungen lässt sich relativ einfach implementieren, ist aber vielleicht nicht ohne weiteres verständlich. Momentan soll das nicht weiter stören, in drei Tagen führen Sie sich eine ganze Lektion zu Gemüte, die sich mit dem Speichern und Wiederherstellen von Dateien - der sogenannten Serialisierung - beschäftigt. Suchen Sie zunächst die Funktion `Serialize` in der Klasse `Ctag10Doc`. Die Funktion sieht folgendermaßen aus:

```
1: void Ctag10Doc::Serialize(CArchive& ar)
```

```

2: {
3: if (ar.IsStoring())
4: {
5: // ZU ERLEDIGEN: Hier Code zum Speichern einfügen
6: }
7: else
8: {
9: // ZU ERLEDIGEN: Hier Code zum Laden einfügen
10: }
11: }

```

Löschen Sie den gesamten Inhalt der Funktion, und nehmen Sie dafür den Code aus Listing 10.10 auf.

Listing 10.10: Die Funktion Serialize der Klasse CTag10Doc

```

1: void CTag10Doc::Serialize(CArchive& ar)
2: {
3: //////////////////////////////////////
4: // EIGENER CODE, ANFANG
5: //////////////////////////////////////
6:
7: // Serialisierung an Objektarray weitergeben
8: m_oaLines.Serialize(ar);
9:
10: //////////////////////////////////////
11: // EIGENER CODE, ENDE
12: //////////////////////////////////////
13: }

```

Die Funktion stützt sich auf die Funktionalität der Klasse `CObArray`. Dieses Objektarray reicht sein Array von Objekten nach unten weiter und ruft dabei die Funktion `Serialize` auf jedem Objekt auf. Das bedeutet, daß Sie eine Funktion `Serialize` in die Klasse `CLine` aufnehmen müssen. Legen Sie den Funktionstyp als `void` und die Deklaration mit `Serialize(CArchive& ar)` fest. In die Funktion schreiben Sie den Code aus Listing 10.11.

Listing 10.11: Die Funktion Serialize der Klasse CLine

```

1: void CLine::Serialize(CArchive &ar)
2: {
3: CObject::Serialize(ar);
4:
5: if (ar.IsStoring())
6: ar << m_ptFrom << m_ptTo;
7: else
8: ar >> m_ptFrom >> m_ptTo;
9: }

```

Die Funktion weist grundsätzlich den gleichen Programmfluß auf, wie die ursprüngliche Version der `Serialize`-Funktion in der Klasse `CTag10Doc`. Die Funktion greift auf die Funktionalität der E/A-Streams von C++ zurück, um die Inhalte zu speichern und wiederherzustellen.

Wenn Sie die Anwendung zum jetzigen Zeitpunkt kompilieren und ausführen, erwarten Sie sicherlich, daß die Funktionen zum Speichern und Öffnen arbeiten. Leider tun sie das aber (noch) nicht. Wenn Sie die Anwendung ausführen und versuchen, eine Zeichnung zu speichern, weist Sie ein Meldungsfeld darauf hin, daß die Anwendung die Datei nicht speichern konnte (siehe Abbildung 10.5).

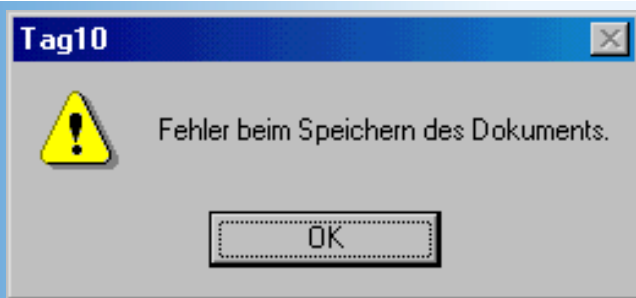


Abbildung 10.5:
Die Zeichnungen lassen sich momentan noch nicht speichern.

Der Grund für das Scheitern liegt darin, daß man Visual C++ mitteilen muß, daß eine Klasse serialisierbar sein soll. Zu diesem Zweck fügen Sie eine Zeile in die Header-Datei und eine Zeile in die Quellcodedatei der Klasse CLine ein. Öffnen Sie die Header- Datei der Klasse CLine (Line.h), und schreiben Sie die Zeile DECLARE_SERIAL gemäß Listing 10.12 unmittelbar nach der ersten Zeile der Klassendefinition.

Listing 10.12: Die Header-Datei Line.h für Serialisierung angepaßt

```
1: class CLine : public CObject
2: {
3: DECLARE_SERIAL (CLine)
4: public:
5: void Serialize(CArchive& ar);
6: void Draw(CDC *pDC);
7: CLine(CPoint ptFrom, CPoint ptTo);
```

Als nächstes öffnen Sie die Quellcodedatei von CLine und fügen die Zeile IMPLEMENT_SERIAL gemäß Listing 10.13 unmittelbar vor die Konstruktorfunktionen der Klasse ein.

Listing 10.13: Die Quellcodedatei Line.cpp für Serialisierung angepaßt

```
1: // Line.cpp: Implementierung der Klasse CLine.
2: //
3: //////////////////////////////////////
4:
5: #include "stdafx.h"
6: #include "Tag10.h"
7: #include "Line.h"
8:
9: #ifdef _DEBUG
10: #undef THIS_FILE
11: static char THIS_FILE[]=__FILE__;
12: #define new DEBUG_NEW
13: #endif
14:
15: IMPLEMENT_SERIAL (CLine, CObject, 1)
16: //////////////////////////////////////
17: // Konstruktion/Destruktion
18: //////////////////////////////////////
19:
20: CLine::CLine()
21: {
22:
23: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie Ihr Selbstporträt zeichnen und für die

Ewigkeit speichern können (siehe Abbildung 10.6).

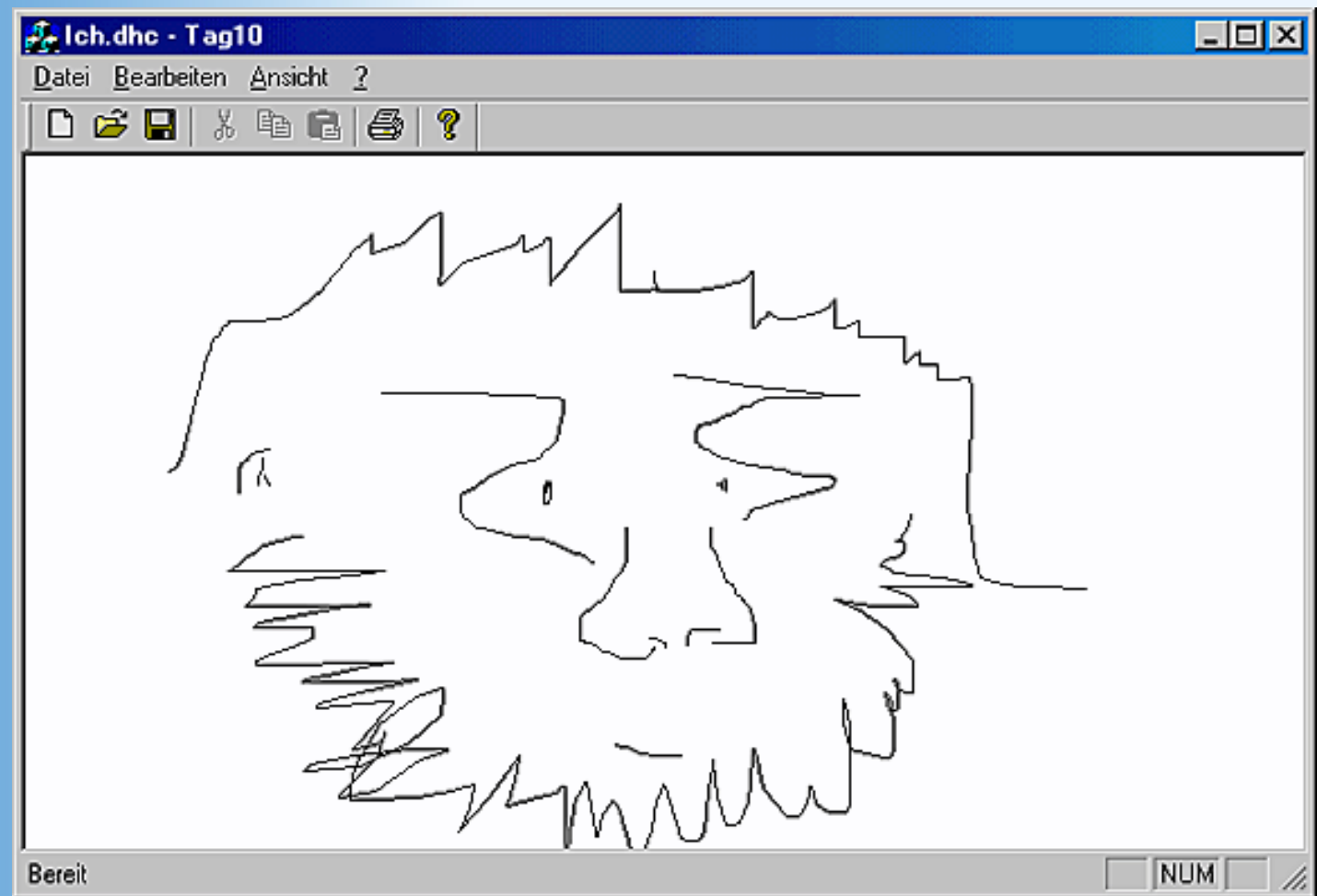


Abbildung 10.6:
Ein Selbstporträt

Interaktion mit Menüs

Nachdem Sie nun über ein funktionsfähiges Zeichenprogramm verfügen, wäre es schön, wenn der Benutzer die Farbe auswählen könnte, mit der er zeichnen will. Dazu sind Änderungen an der Klasse `CLine` erforderlich, um die Farbe mit der Linie zu verbinden, sowie an der Klasse `CTag10Doc`, um die momentan ausgewählte Farbe zu verwalten. Schließlich ist noch ein Pulldown-Menü vorzusehen, aus dem sich die gewünschte Farbe auswählen läßt.

Farben in die Klasse `CLine` aufnehmen

Die Änderungen in der Klasse `CLine` sind recht unkompliziert. Als erstes fügen Sie eine weitere Member-Variable hinzu, um die Farbe jeder Linie zu speichern. Als nächstes modifizieren Sie den Klassenkonstruktor, um die Liste der Attribute durch die übergebene Farbe zu erweitern. Drittens ist die Funktion `Draw` zu modifizieren, damit diese die angegebene Farbe verwendet. Und schließlich müssen Sie noch an der Funktion `Serialize` arbeiten, um die Farbinformation zusammen mit den Punktkoordinaten speichern und wiederherstellen zu können. Führen Sie dazu folgende Schritte aus:

1. Markieren Sie im Arbeitsbereich auf der Registerkarte **Klassen** die Klasse `CLine`. Klicken Sie mit der rechten Maustaste, und wählen Sie den Befehl **Member-Variable hinzufügen** aus dem Kontextmenü.
2. Legen Sie den Variablentyp als `COLORREF`, den Namen als `m_crColor` und den Zugriff als *Privat* fest. Klicken Sie auf **OK**, um die Variable hinzuzufügen.
3. Klicken Sie mit der rechten Maustaste in der Baumansicht der Registerkarte **Klassen** auf den Konstruktor

von `CLine`. Aus dem Kontextmenü wählen Sie den Befehl **Gehe zu Deklaration**.

4. Fügen Sie `COLORREF crColor` als drittes Argument in die Konstruktordeklaration ein.

5. Klicken Sie mit der rechten Maustaste in der Baumansicht der Registerkarte **Klassen** auf den Konstruktor von `CLine`. Aus dem Kontextmenü wählen Sie den Befehl **Gehe zu Definition**.

6. Modifizieren Sie den Konstruktor gemäß Listing 10.14, um das dritte Argument hinzuzufügen und das Element `m_crColor` auf das neue Argument zu setzen.

Listing 10.14: Der modifizierte Konstruktor von `CLine`

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor)
2: {
3: // Anfangs- und Endpunkte initialisieren
4: m_ptFrom = ptFrom;
5: m_ptTo = ptTo;
6: m_crColor = crColor;
7: }
```

7. Gehen Sie nach unten zur Funktion `Draw`, und nehmen Sie die Änderungen gemäß Listing 10.15 vor.

Listing 10.15: Die modifizierte Funktion `Draw`

```
1: void CLine::Draw(CDC * pDC)
2: {
3: // Stift erzeugen
4: CPen lpen (PS_SOLID, 1, m_crColor);
5:
6: // Neuen Stift als Zeichenobjekt festlegen
7: CPen* pOldPen = pDC->SelectObject(&lpen);
8: // Linie zeichnen
9: pDC->MoveTo(m_ptFrom);
10: pDC->LineTo(m_ptTo);
11: // Vorherigen Stift zurücksetzen
12: pDC->SelectObject(pOldPen);
13: }
```

8. Blättern Sie weiter nach unten zur Funktion `Serialize`, und ändern Sie die Funktion entsprechend Listing 10.16 ab.

Listing 10.16: Die modifizierte Funktion `Serialize`

```
1: void CLine::Serialize(CArchive &ar)
2: {
3: CObject::Serialize(ar);
4:
5: if (ar.IsStoring())
6: ar << m_ptFrom << m_ptTo << (DWORD) m_crColor;
7: else
8: ar >> m_ptFrom >> m_ptTo >> (DWORD) m_crColor;
9: }
```

Bei diesen Schritten fällt eigentlich nur auf, daß man den Rückgabewert der Funktion `SelectObject` übernimmt, wenn man den Zeichenstift für die Linien spezifiziert. Diesen Schritt haben Sie letzte Woche nicht ausgeführt. Der Rückgabewert der Funktion `SelectObject` ist der Stift, der benutzt wurde, bevor Sie ihn geändert haben. Damit läßt sich der ursprüngliche Stift im Gerätekontext wiederherstellen, wenn der Benutzer die Zeichnung mit dem neuen Stift beendet hat.

Farbige Dokumente

Die an der Klasse `CTag10Doc` vorzunehmenden Änderungen sind nur etwas umfangreicher als die an der Klasse `CLine`. Es ist eine Elementvariable für die aktuelle Farbe hinzuzufügen sowie eine Farbtabelle, um die Farb-IDs in RGB-Werte zu konvertieren. In der Funktion `OnNewDocument` ist die Variable für die aktuelle Farbe zu initialisieren. Dann müssen Sie die Funktion `AddLine` modifizieren, um die aktuelle Farbe in den Konstruktor von `CLine` hinzuzufügen. Schließlich nehmen Sie noch eine Funktion auf, um die aktuelle Farbe zurückzugeben. Das ist momentan alles, bis Sie die Behandlungsroutinen für die Menübefehle in Angriff nehmen. Führen Sie zunächst die folgenden Schritte aus:

1. Markieren Sie im Arbeitsbereich auf der Registerkarte **Klassen** die Klasse `CTag10Doc`. Klicken Sie mit der rechten Maustaste, und wählen Sie den Befehl **Member-Variable hinzufügen** aus dem Kontextmenü.
2. Legen Sie den Variablentyp als `UINT`, den Namen mit `m_nColor` und den Zugriff als *Privat* fest. Klicken Sie auf **OK**, um die Variable hinzuzufügen.
3. Wiederholen Sie Schritt 1.
4. Legen Sie den Variablentyp als `static const COLORREF`, den Namen als `m_crColors[8]` und den Zugriff als *Public* fest.
5. Öffnen Sie den Quellcode von `CTag10Doc` (Datei `Tag10Doc.cpp`), und fügen Sie die Farbtabelle gemäß Listing 10.17 ein.

Listing 10.17: Die Spezifikation der Farbtabelle

```

1: //}}AFX_MSG_MAP
2: END_MESSAGE_MAP( )
3:
4: const COLORREF CTag10Doc::m_crColors[8] = {
5: RGB( 0, 0, 0), // Schwarz
6: RGB( 0, 0, 255), // Blau
7: RGB( 0, 255, 0), // Grün
8: RGB( 0, 255, 255), // Cyan
9: RGB( 255, 0, 0), // Rot
10: RGB( 255, 0, 255), // Magenta
11: RGB( 255, 255, 0), // Gelb
12: RGB( 255, 255, 255) // Weiß
13: };
14:
15:
16: //////////////////////////////////////
17: // CTag10Doc Konstruktion/Destruktion
18: CTag10Doc::CTag10Doc( )
19: .
20: .
21: .
22: }
```

6. Blättern Sie nach unten zur Funktion `OnNewDocument`, und nehmen Sie die Änderungen gemäß Listing 10.18 vor.

Listing 10.18: Die modifizierte Funktion `OnNewDocument`

```

1: BOOL CTag10Doc::OnNewDocument( )
2: {
3: if (!CDocument::OnNewDocument())
```



```

4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: //////////////////////////////////////
10: // EIGENER CODE, ANFANG
11: //////////////////////////////////////
12:
13: // Farbe mit Schwarz initialisieren
14: m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;
15:
16: //////////////////////////////////////
17: // EIGENER CODE, ENDE
18: //////////////////////////////////////
19:
20: return TRUE;
21: }

```

7. Gehen Sie weiter zur Funktion `AddLine`, und passen Sie die Funktion entsprechend Listing 10.19 an.

Listing 10.19: Die modifizierte Funktion `AddLine`

```

1: CLine* CTag10Doc::AddLine(CPoint ptFrom, CPoint ptTo)
2: {
3: // Ein neues CLine-Objekt erzeugen
4: CLine *pLine = new CLine(ptFrom, ptTo, m_crColors[m_nColor]);
5: try
6: {
7: // Die neue Linie in das Objektarray einfügen
8: m_oaLines.Add(pLine);
9: // Dokument als bearbeitet markieren
10: SetModifiedFlag();
11: }
12: // Ist Speicherausnahme aufgetreten?
13: catch (CMemoryException* perr)
14: {
15: // Meldung für Benutzer, schlechte Neuigkeiten
16: // mitteilen
17: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
18: // Wurde Linienobjekt erzeugt?
19: if (pLine)
20: {
21: // Objekt löschen
22: delete pLine;
23: pLine = NULL;
24: }
25: // Ausnahmeobjekt löschen
26: perr->Delete();
27: }
28: return pLine;
29: }

```

8. Fügen Sie der Klasse `CTag10doc` eine neue Member-Funktion hinzu. Legen Sie den Funktionstyp als `UINT`, die Deklaration mit `GetColor` und den Zugriff als *Public* fest.

9. In die Funktion `GetColor` schreiben Sie den Code aus Listing 10.20.

Listing 10.20: Die Funktion `GetColor`

```
1: UINT CTag10Doc::GetColor()
2: {
3: // Aktuelle Farbe zurückgeben
4: return ID_COLOR_BLACK + m_nColor;
5: }
```

Die Funktionen `OnNewDocument` und `GetColor` führen eine Subtraktion bzw. Addition der Farbe mit dem Wert `ID_COLOR_BLACK` durch. Der Wert `ID_COLOR_BLACK` ist die kleinste ID des durchnummerierten Farbmenüs, wenn Sie die Menübefehle hinzufügen. Diese Berechnungen verwalten die Variable als Zahl zwischen 0 und 7, während diese Funktionen beim Umgang mit den Menüs wieder Vergleiche mit den eigentlichen Menü-IDs zulassen.

Das Menü modifizieren

Nun kommt der interessantere Teil. In das Hauptmenü fügen Sie ein neues Pulldown- Menü hinzu. In dieses Menü nehmen Sie Befehle für alle Farben der Farbtabelle auf. Für die Menübefehle müssen Sie noch Behandlungsroutinen erstellen, um den Befehl für die momentan ausgewählte Farbe mit einem Kontrollhäkchen zu versehen. Das Ganze realisieren Sie in folgenden Schritten:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**. Erweitern Sie den Baum, so daß der Inhalt des Ordners **Menu** zu sehen ist. Doppelklicken Sie auf die Menüressource.
2. Ziehen Sie das leere Menükästchen (am rechten Ende der Menüleiste) nach links, und legen Sie es vor dem Menüeintrag **Ansicht** ab.
3. Öffnen Sie das Dialogfeld **Eigenschaften** für den leeren Menüeintrag. Legen Sie den Titel mit **&Farben** fest. Schließen Sie den Eigenschaftsdialog.
4. Fügen Sie die Untermenüeinträge in das Menü **Farbe** hinzu. Erstellen Sie die Untermenüeinträge in der Reihenfolge von Tabelle 10.2, und verwenden Sie die dort angegebenen Eigenschaften. Das fertige Menü sollte dann Abbildung 10.7 entsprechen.

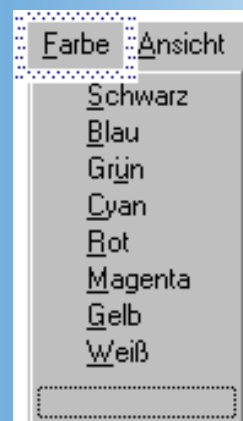


Abbildung 10.7:
Das Menü Farbe in der Entwurfsansicht

Tabelle 10.2: Eigenschaftseinstellungen des Menüs

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	ID_COLOR_BLACK
	Titel	&Schwarz

Menübefehl	ID	ID_COLOR_BLUE
	Titel	&Blau
Menübefehl	ID	ID_COLOR_GREEN
	Titel	Gr&ün
Menübefehl	ID	ID_COLOR_CYAN
	Titel	&Cyan
Menübefehl	ID	ID_COLOR_RED
	Titel	&Rot
Menübefehl	ID	ID_COLOR_MAGENTA
	Titel	&Magenta
Menübefehl	ID	ID_COLOR_YELLOW
	Titel	&Gelb
Menübefehl	ID	ID_COLOR_WHITE
	Titel	&Weiß

- Öffnen Sie den Klassen-Assistenten. Markieren Sie CTag10Doc im Kombinationsfeld **Klassenname**.
- Fügen Sie für die beiden Nachrichten COMMAND und UPDATE_COMMAND_UI Funktionen für alle Befehle des Farbmenüs hinzu.
- Nachdem Sie die Funktion für den letzten Menübefehl hinzugefügt haben, klicken Sie auf **Code bearbeiten**.
- Bearbeiten Sie die Funktionen des Menüs **Schwarz** gemäß Listing 10.21.

Listing 10.21: Die Funktionen des Menüs Schwarz

```

1: void CTag10Doc::OnColorBlack()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Aktuelle Farbe auf Schwarz setzen
10: m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;
11:
12: //////////////////////////////////////
13: // EIGENER CODE, ENDE
14: //////////////////////////////////////
15: }
16:
17: void CTag10Doc::OnUpdateColorBlack(CCmdUI* pCmdUI)
18: {
19: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
  Benutzeroberfläche hier einfügen
20:

```

```

21: ///////////////////////////////////
22: // EIGENER CODE, ANFANG
23: ///////////////////////////////////
24:
25: // Prüfen, ob Menübefehl Schwarz mit Kontrollhäkchen zu versehen ist
26: pCmdUI->SetCheck(GetColor() == ID_COLOR_BLACK ? 1 : 0);
27:
28: ///////////////////////////////////
29: // EIGENER CODE, ENDE
30: ///////////////////////////////////
31: }

```

9. Bearbeiten Sie die Funktionen des Menüs **Blau** gemäß Listing 10.22. Die restlichen Menüfunktionen behandeln Sie in der gleichen Weise, wobei Sie für ID_COLOR_BLUE die jeweilige Menü-ID einsetzen.

Listing 10.22: Die Funktionen des Menüs Blau

```

1: void CTag10Doc::OnColorBlue()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: // Aktuelle Farbe auf Blau setzen
10: m_nColor = ID_COLOR_BLUE - ID_COLOR_BLACK;
11:
12: ///////////////////////////////////
13: // EIGENER CODE, ENDE
14: ///////////////////////////////////
15: }
16:
17: void CTag10Doc::OnUpdateColorBlue(CCmdUI* pCmdUI)
18: {
19: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
    Benutzeroberfläche hier einfügen
20:
21: ///////////////////////////////////
22: // EIGENER CODE, ANFANG
23: ///////////////////////////////////
24:
25: // Prüfen, ob Menübefehl Blau mit Kontrollhäkchen zu versehen ist
26: pCmdUI->SetCheck(GetColor() == ID_COLOR_BLUE ? 1 : 0);
27:
28: ///////////////////////////////////
29: // EIGENER CODE, ENDE
30: ///////////////////////////////////
31: }

```

In der ersten der beiden Menüfunktionen, der COMMAND-Funktion, wird die Variable der aktuellen Farbe auf die neue Farbe gesetzt. Wenn Sie die Menübefehle in der richtigen Reihenfolge hinzugefügt haben, weisen sie fortlaufende ID-Nummern auf, die mit ID_COLOR_BLACK beginnen. Zieht man den Wert ID_COLOR_BLACK von der Menü-ID ab, erhält man dann die Position in der Farbtabelle für die ausgewählte Farbe. Beispielsweise hat Schwarz die Position 0 in der Farbtabelle. ID_COLOR_BLACK -

`ID_COLOR_BLACK = 0`. Blau steht auf der Position 1 in der Farbtabelle. Da `ID_COLOR_BLUE` um 1 größer als `ID_COLOR_BLACK` sein sollte, liefert `ID_COLOR_BLUE - ID_COLOR_BLACK` das Ergebnis 1.

Die zweite Funktion, die Funktion `UPDATE_COMMAND_UI`, erfordert eine nähere Erläuterung. Das Ereignis `UPDATE_COMMAND_UI` wird für jeden Menübefehl aufgerufen, unmittelbar bevor dieser angezeigt wird. Mit dieser Behandlungsfunktion können Sie in Abhängigkeit von der aktuellen Farbe neben dem betreffenden Menübefehl ein Kontrollhäkchen setzen oder dieses entfernen. Mit diesem Ereignis lassen sich auch Menübefehle aktivieren oder deaktivieren oder bei Bedarf andere Modifikationen realisieren. Der Code in dieser Funktion

```
pCmdUI->SetCheck(GetColor() == ID_COLOR_BLUE ? 1 : 0);
```

bewirkt mehreres. Erstens ist das als einziges Argument übergebene `pCmdUI`-Objekt ein Zeiger auf ein Menüobjekt. Die Funktion `SetCheck` kann das Kontrollhäkchen vor den Menübefehl setzen oder entfernen, je nachdem, ob das übergebene Argument 1 oder 0 ist (1 setzt das Kontrollhäkchen, 0 entfernt es). Der Argumentabschnitt für die Funktion `SetCheck` stellt eine Konstruktion zur Programmsteuerung dar, die Ihnen vielleicht etwas verwirrend vorkommt, wenn Sie noch nicht allzuviel Zeit in die Programmierung mit C/C++ investiert haben. Die erste Hälfte

```
GetColor() == ID_COLOR_BLUE
```

ist eine einfache Boolesche Bedingungsanweisung, die entweder `TRUE` (wahr) oder `FALSE` (falsch) liefert. Der sich an diese Bedingungsanweisung anschließende Teil

```
? 1 : 0
```

stellt grundsätzlich eine `if . . . else`-Anweisung dar, allerdings in Kurzform. Wenn die Bedingungsanweisung `TRUE` liefert, dann ist der Wert gleich 1, und wenn die Anweisung `FALSE` ergibt, dann gilt der Wert 0. Damit läßt sich auf geschickte Weise eine `if . . . else`-Steuerstruktur im Argument an eine andere Funktion übergeben.

Nachdem Sie die Anwendung kompiliert und gestartet haben, sollten Sie die Zeichenfarbe ändern können. Wenn Sie das Farbmenü öffnen, sollte die aktuelle Zeichenfarbe mit einem Kontrollhäkchen versehen sein, wie es Abbildung 10.8 zeigt.



Abbildung 10.8:
Die aktuelle Farbe im Menü festlegen.

Zusammenfassung

Was für ein Tag! Diese Lektion war mit Informationen voll gepackt. Zuerst wurde gezeigt, was eine SDI-Anwendung ist und welche bekannten Standardanwendungen in diesem Stil aufgebaut sind. Als nächstes haben Sie die Dokument/Ansicht-Architektur kennengelernt, die Visual C++ für SDI-Anwendungen verwendet. Sie haben eine eigene einfache Klasse erstellt, die sich für eine Zeichenanwendung eignet. Dann haben Sie eine Zeichenanwendung erstellt, mit der Sie Bilder zeichnen und verwalten konnten. Es wurde gezeigt, wie man Dokumente in der Dokument/Ansicht-Architektur speichert und wiederherstellt. Weiterhin wurde die Klasse `CObArray` behandelt und wie man sie einsetzen kann, um ein dynamisches Objektarray für das Speichern verschiedenartiger Klassen zu erzeugen. Schließlich haben Sie gelernt, wie man Kontrollhäkchen in Menübefehlen von MFC-Anwendungen setzt und entfernt.

Fragen und Antworten

Frage:

Gibt es eine Möglichkeit, die Anzahl der Funktionen für die Nachrichten `COMMAND` und `UPDATE_COMMAND_UI` für die Menüs zu verringern?

Antwort:

Ja. Man kann alle Ereignisse des Farbmenüs an dieselbe Funktion senden. In dieser Funktion läßt sich der Wert `nID` (der als Argument übergeben wird) untersuchen und mit den Menü-IDs vergleichen, um zu ermitteln, welcher Menübefehl die Funktion aufgerufen hat. Letztendlich kann man die `COMMAND`-Funktion für die Farbmenüs wie folgt formulieren:

```
void CTag10Doc::OnColorCommand(UINT nID)
```



```

{
// TODO: Code für Befehlsbehandlungsroutine hier einfügen

////////////////////////
// EIGENER CODE, ANFANG
////////////////////////

// Aktuelle Farbe setzen
m_nColor = nID - ID_COLOR_BLACK;

////////////////////////
// EIGENER CODE, ENDE
////////////////////////
}

```

Bei den UPDATE_COMMAND_UI-Funktionen kann man analog vorgehen, allerdings mit einem kleinen Unterschied. Bei diesen Funktionen kann man den Wert pCmdUI->m_nID untersuchen, um zu bestimmen, für welchen Menübefehl die Funktion aufgerufen wurde. Damit sieht die Funktion UPDATE_COMMAND_UI folgendermaßen aus:

```

void CTag10Doc::OnUpdateColor(CCmdUI* pCmdUI)
{
// TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
Benutzeroberfläche hier einfügen

////////////////////////
// EIGENER CODE, ANFANG
////////////////////////

// Prüfen, ob Menübefehl mit Kontrollhäkchen zu versehen ist
pCmdUI->SetCheck(GetColor() == pCmdUI->m_nID ? 1 : 0);

////////////////////////
// EIGENER CODE, ENDE
////////////////////////
}

```

Frage:

Worin unterscheiden sich SDI- und MDI-Anwendungen?

Antwort:

Während SDI-Anwendungen nur eine Aufgabe ausführen können, lassen sich bei MDI-Anwendungen (MDI - Multiple Document Interface) mehrere Dokumente gleichzeitig öffnen. Darüber hinaus müssen in einer MDI-Anwendung die Dokumente nicht alle vom gleichen Typ sein. Mehr über MDI-Anwendungen lernen Sie morgen.

Workshop

Kontrollfragen

1. Wofür steht SDI?
2. Welche Funktionalität ist in der Ansichtsklasse realisiert?
3. Welche Funktion wird aufgerufen, um das Dokument neu zu zeichnen, wenn das Fenster hinter einem

anderen Fenster verborgen war?

4. Wo bringt man den Code unter, um das aktuelle Dokument zu löschen, bevor man mit einem neuen Dokument beginnt?

5. Welchen Zweck erfüllt die Dokumentklasse?

Übung

Fügen Sie ein weiteres Pulldown-Menü hinzu, um die Breite des Zeichenstifts zu steuern. Verwenden Sie die Einstellungen gemäß Tabelle 10.3:

**Tabelle 10.3: Menü
für Auswahl der
Stiftbreite**

Menüeintrag	Breite
Sehr dünn	1
Dünn	8
Mittel	16
Dick	24
Sehr dick	32



Das zweite Argument des Stiftkonstruktors gibt die Breite an.

Woche 2

Tag 11

MDI-Anwendungen

Heute lernen Sie, wie man mit Visual C++ MDI-Anwendungen erstellt. MDI steht für Multiple Document Interface, zu deutsch Mehrfachdokumentschnittstelle. Damit lassen sich Anwendungen erstellen, bei denen der Benutzer an mehreren Dokumenten gleichzeitig arbeiten und zwischen den Fenstern der Anwendung umschalten kann. In diesem Kapitel erfahren Sie, ...

- worin der Unterschied zwischen SDI- und MDI-Anwendungen besteht.
- wie man eine MDI-Anwendung erstellt.
- wie man die Nachrichten von mehreren Menübefehlen an eine einzige Behandlungsfunktion sendet.
- wie sich ein Kontextmenü in einer Anwendung nach der Dokument/Ansicht-Architektur realisieren läßt.

Was sind MDI-Anwendungen?

In bezug auf die Codierung mit Visual C++ gibt es kaum einen Unterschied zwischen SDI- und MDI-Anwendungen. Erst wenn man tiefer in die beiden Anwendungsstile einsteigt, treten die Unterschiede zutage. Eine SDI-Anwendung erlaubt dem Benutzer lediglich, an einem Dokument zu einem bestimmten Zeitpunkt zu arbeiten und läßt dabei nur einen einzigen Dokumenttyp zu. MDI-Anwendungen erlauben dagegen dem Benutzer nicht nur, an mehreren Dokumenten gleichzeitig zu arbeiten, sondern bieten auch die Möglichkeit, mehrere Arten von Dokumenten zu behandeln.

Eine MDI-Anwendung ist in einer Art Fenster-in-Fenster aufgebaut, wobei ein Rahmenfenster ein oder mehrere untergeordnete Fenster umschließt. Diesen Anwendungsstil findet man in vielen bekannten Softwarepaketen wie zum Beispiel Word und Excel.

Die Architektur einer MDI-Anwendung entspricht weitgehend der einer SDI-Anwendung. Praktisch besteht der einzige Unterschied bei einer einfachen MDI-Anwendung nur darin, daß der Anwendungs-Assistent neben den üblichen Klassen noch eine zweite Rahmenklasse erzeugt, wie es Abbildung 11.1 zeigt. Der Abbildung können Sie auch entnehmen, daß die Herangehensweise bei der Entwicklung von MDI-Anwendungen der für SDI-Anwendungen sehr ähnlich ist.

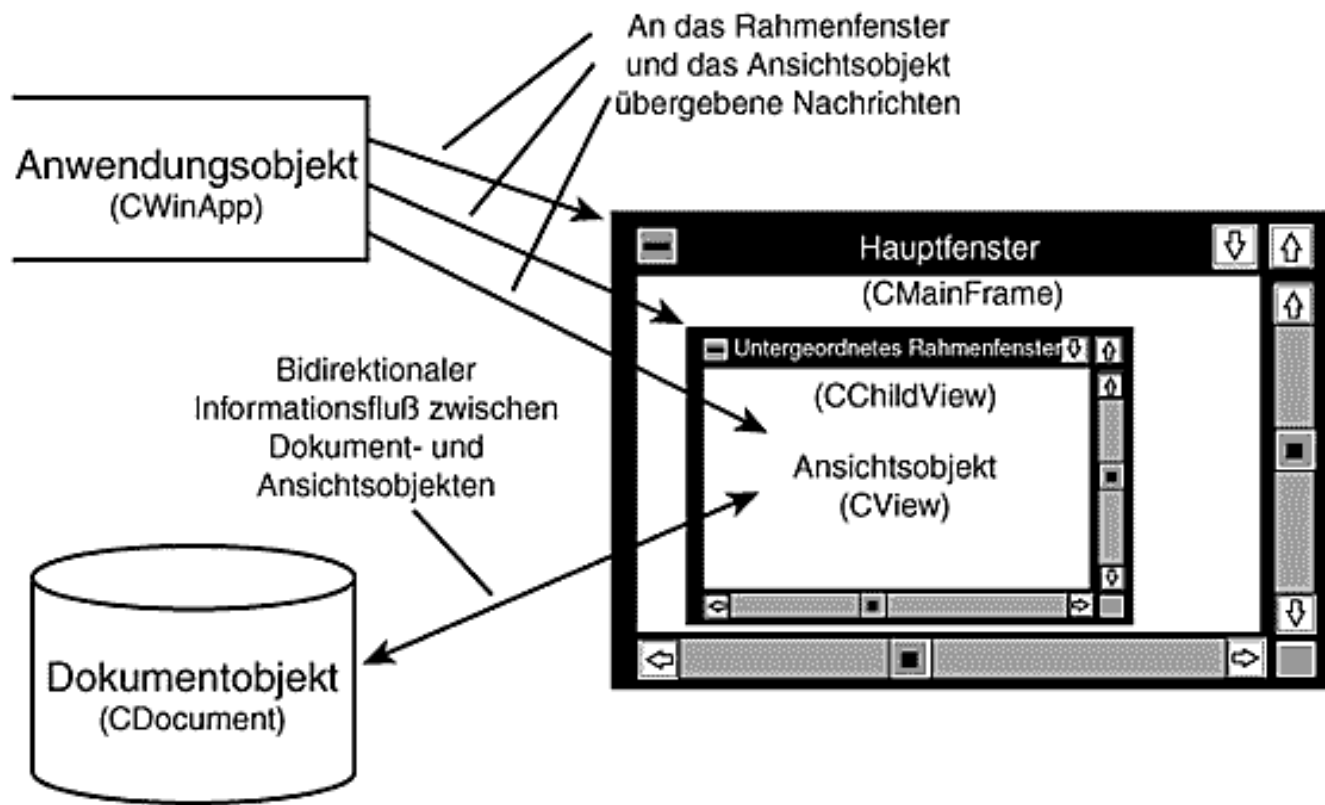


Abbildung 11.1:
Die Dokument/Ansicht-Architektur einer MDI-Anwendung

Für eine MDI-Anwendung erstellen Sie lediglich eine Klasse mehr als bei einer SDI- Anwendung. Zu diesen Klassen gehören:

- die von CWinApp abgeleitete Klasse
- die von CMDIFrameWnd abgeleitete Klasse
- die von CMDIChildWnd abgeleitete Klasse
- die von CDocument abgeleitete Klasse
- die von CView abgeleitete Klasse

Die beiden MDI-spezifischen Klassen CMDIFrameWnd (die Klasse CMainFrame in Ihrem Projekt) und CMDIChildWnd (die Klasse CChildFrame in Ihrem Projekt) sind die beiden einzigen Klassen, die sich von den bisher erstellten SDI-Anwendungen unterscheiden.

Die von CMDIFrameWnd abgeleitete Klasse CMainFrame stellt den Hauptrahmen der Anwendung dar und liefert einen abgeschlossenen Bereich auf dem Desktop, in dem die gesamte Interaktion der Anwendung stattfindet. Mit dem Rahmenfenster sind das Menü und die Symbolleisten verbunden.

Die von CMDIChildWnd abgeleitete Klasse CChildFrame bildet den Rahmen, der die CView-Klassen aufnimmt. Dieser Rahmen leitet die Nachrichten und Ereignisse an die Ansichtsklasse zur Verarbeitung oder Anzeige weiter.

Gewissermaßen ist die Funktionalität der Rahmenklasse einer SDI-Anwendung in die beiden MDI-spezifischen Klassen einer MDI-Anwendung aufgeteilt worden. Gleichzeitig gibt es eine zusätzliche Unterstützung für die Ausführung mehrerer untergeordneter Rahmen in deren eigenen Dokument/Ansicht-Klasseninstanzen.

Ein MDI-Zeichenprogramm erstellen

Damit Sie sich selbst davon überzeugen können, wie ähnlich sich die Dokument/Ansicht-Architekturen von SDI- und MDI-Anwendungen sind, erstellen Sie heute die gleiche Zeichenanwendung wie gestern - nur eben als MDI-Anwendung.

Das Anwendungsgerüst erstellen

Das Gerüst für die heutige Beispielanwendung erstellen Sie in folgenden Schritten:

1. Legen Sie mit dem Anwendungs-Assistenten ein neues Projekt namens **Tag11** an.
2. Im ersten Dialogfeld des Anwendungs-Assistenten wählen Sie die Option *Mehrere Dokumente (MDI)*, wie es Abbildung 11.2 zeigt.

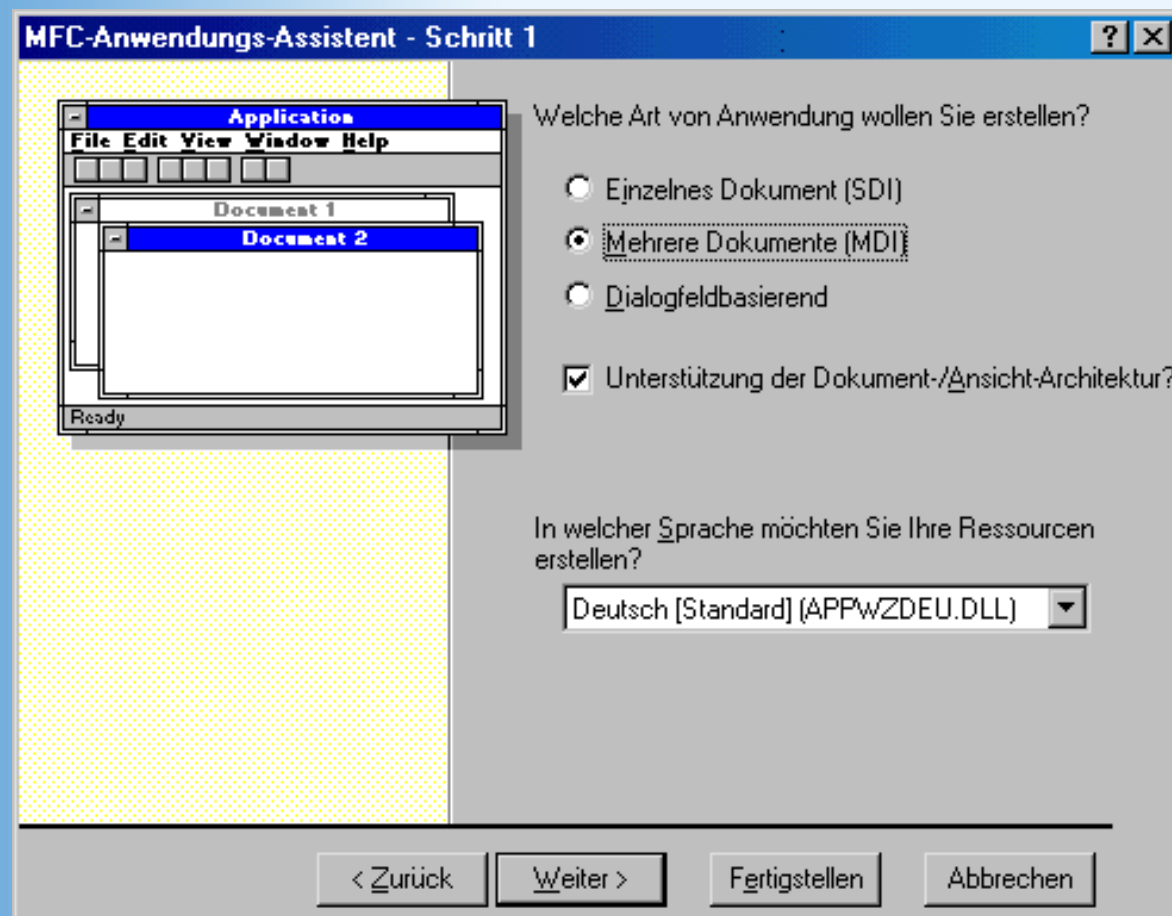


Abbildung 11.2:
Eine MDI-Anwendung festlegen

3. Übernehmen Sie die Voreinstellungen im zweiten Schritt des Assistenten.
4. Im dritten Dialogfeld des Anwendungs-Assistenten schalten Sie die Unterstützung für ActiveX-Steuerelemente aus.
5. Im vierten Dialogfeld lassen Sie die Voreinstellungen unverändert und klicken auf die Schaltfläche **Weitere Optionen**.
6. Im Dialogfeld **Weitere Optionen** geben Sie eine aus drei Buchstaben bestehende Erweiterung für die Dateien ein, die Ihre Anwendung generiert (beispielsweise **dhc** oder **dvp**). Klicken Sie auf die Schaltfläche **Schliessen**, um das Dialogfeld zu schließen, und dann auf **Weiter**, um zum nächsten Schritt des Anwendungs-Assistenten zu gelangen.
7. Übernehmen Sie die Standardeinstellungen im fünften Schritt des Assistenten.
8. Im sechsten und letzten Dialogfeld des Anwendungs-Assistenten übernehmen Sie **CView** als Basisklasse und klicken auf **Fertigstellen**. Der Anwendungs-Assistent generiert nun das Gerüst der Anwendung.

Die Zeichenfunktionalität realisieren

Da Sie die gleiche Anwendung wie gestern, allerdings jetzt als MDI-Version, erstellen, nehmen Sie auch die gleiche Funktionalität in die Anwendung auf, die Sie bereits aus der vergangenen Lektion kennen. Um Zeit zu sparen und noch einmal die Ähnlichkeit der beiden Anwendungsarchitekturen zu betonen, führen Sie die gleichen Schritte wie gestern aus, um die Klasse `CLine` zu erstellen und die Funktionalität - diesmal in die Klassen `Ctag11Doc` und `Ctag11View` - hinzuzufügen. Bauen Sie auch die Unterstützung für die Auswahl der Farben und der Stiftbreite (wie in der Übung zu Tag 10) in die Klassen `Ctag11Doc` und `CLine` ein, nehmen Sie aber noch keine Behandlungsfunktionen für Menüereignisse auf, und lassen Sie auch noch das Farbmenü außen vor. Bei diesem Entwicklungsstand haben Sie nun eine Anwendung vor sich, bei der Sie mehrere Zeichnungen öffnen können, wobei aber nur die Zeichenfarbe Schwarz zur Verfügung steht.



Da die Menüs noch nicht erstellt sind und die Initialisierung der Farben auf die IDs des Farbmenüs zurückgreift, sollten Sie vorerst die Initialisierung der Farbe mit `0` fest codieren, damit sich die Anwendung kompilieren läßt. Nachdem Sie das Farbmenü hinzugefügt haben, stehen auch die Menü-IDs zur Verfügung, so daß Sie wieder die IDs in Ihrem Code verwenden können. In der Zwischenzeit ändern Sie in der Funktion `OnNewDocument` der Klasse `Ctag11Doc` die Zeile

```
m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;
```

in

```
m_nColor = 0;
```

In der Funktion `GetColor` ist eine äquivalente Änderung erforderlich, da auch diese Funktion eine der IDs des Farbmenüs verwendet.

Die Funktionalität zur Menübehandlung realisieren

Nachdem nun die gesamte Funktionalität in die Anwendung eingebunden ist, möchten Sie sicherlich das Farbmenü hinzufügen, damit sich alle Farben für die Zeichnungen verwenden lassen. Wenn Sie den Baum auf der Registerkarte Ressourcen erweitern und in den Ordner `MENU` sehen, finden Sie nicht nur ein, sondern zwei definierte Menüs. In welches nehmen Sie nun das Farbmenü auf?

Das Menü `IDR_MAINFRAME` ist verfügbar, wenn keine untergeordneten Fenster geöffnet sind. Wenn Sie Ihre Anwendung ausführen und alle untergeordneten Fenster schließen, wechselt die Menüleiste, und es verschwinden alle Menüs, die sich auf untergeordnete Fenster beziehen. Sobald Sie ein anderes Dokument - entweder ein neues oder ein vorhandenes - öffnen, ändert sich die Menüleiste erneut und zeigt alle Menüs an, die für die Dokumente maßgeblich sind.

Das Menü `IDR_TAG11TYPE` erscheint, wenn ein untergeordnetes Fenster geöffnet ist. Dieses Menü enthält alle Funktionen, die sich auf Dokumente beziehen. Demzufolge fügen Sie das Farbmenü in dieses Menü ein. Dabei gehen Sie genauso vor, wie am gestrigen Tag, und verwenden auch die gleichen Menüeigenschaften.



Für das Menü **Farbe** müssen Sie jetzt eine andere Zugriffstaste festlegen, da der - gestern verwendete - Buchstabe `F` in der heutigen MDI-Anwendung bereits für das Menü **Fenster** vergeben ist.

Nachdem Sie alle Menübefehle hinzugefügt haben, sind noch die Behandlungsroutinen für die Menüereignisse zu realisieren. Heute implementieren wir die Behandlungsroutinen nach einem anderen Ansatz als gestern. Der Abschnitt »Fragen und Antworten« am Ende des gestrigen Kapitels hat die Frage nach einer einzigen Behandlungsroutine für alle Befehle des Farbmenüs aufgeworfen. Heute setzen Sie diesen Gedanken in die Tat um. Leider versteht der Klassen-Assistent nicht, wie sich mehrere Menünachrichten an dieselbe Funktion weiterleiten lassen, so daß Sie das in eigener Regie nach den folgenden Schritten realisieren müssen:

1. Öffnen Sie die Header-Datei `Tag11Doc.h`.

2. Gehen Sie in der Header-Datei nach unten, bis Sie zum geschützten (`protected`) Abschnitt mit der Deklaration der Nachrichtenzuordnungstabelle `AFX_MSG` gelangen (suchen Sie nach `//{{AFX_MSG(CTag11Doc))`).

3. Fügen Sie die Funktionsdeklarationen aus Listing 11.1 vor der gesuchten Zeile hinzu. (Der Suchstring kennzeichnet den Beginn der vom Klassen-Assistenten verwalteten Nachrichtenzuordnungstabelle. Alles, was Sie zwischen diese Markierung und die Endemarkierung `//}}AFX_MSG` schreiben, betrachtet der Klassen-Assistent als sein »Eigentum« und wird es möglicherweise entfernen oder verstümmeln.)

Listing 11.1: Die Deklarationen der Behandlungsroutine in `Tag11Doc.h`

```
.
.
.
1: #ifdef _DEBUG
2: virtual void AssertValid() const;
3: virtual void Dump(CDumpContext& dc) const;
4: #endif
5:
6: protected:
7:
8: // Generierte Message-Map-Funktionen
9: protected:
10: afx_msg void OnColorCommand(UINT nID);
11: afx_msg void OnUpdateColorUI(CCmdUI* pCmdUI);
12: //{{AFX_MSG(CTag11Doc)
13: // HINWEIS - An dieser Stelle werden Member-Funktionen vom Klassen-
  Assistanten eingefügt und entfernt.
14: // Innerhalb dieser generierten Quelltextabschnitte NICHTS ÄVERÄNDERN!
15: //}}AFX_MSG
16: DECLARE_MESSAGE_MAP()
17: private:
18: UINT m_nColor;
19: CObArray m_oaLines;
20: };
```

4. Öffnen Sie die Quellcodedatei `Tag11Doc.cpp`.

5. Suchen Sie nach der Zeile `BEGIN_MESSAGE_MAP`, und fügen Sie unmittelbar danach die Zeilen aus Listing 11.2 ein. Es ist wichtig, daß dieser Code zwischen der Zeile `BEGIN_MESSAGE_MAP` und der Zeile `//{{AFX_MSG_MAP` steht. Wenn Sie diese Befehle zwischen die Zeilen `//{{AFX_MSG_MAP` und `//}}AFX_MSG_MAP` schreiben, entfernt oder verstümmelt sie der Klassen-Assistent.

Listing 11.2: Die Einträge der Nachrichtenzuordnungstabelle in `Tag11Doc.cpp`

```
1:
  //////////////////////////////////////
2: // CTag11Doc
3:
4: IMPLEMENT_DYNCREATE(CTag11Doc, CDocument)
5:
6: BEGIN_MESSAGE_MAP(CTag11Doc, CDocument)
7: ON_COMMAND_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE, OnColorCommand)
8: ON_UPDATE_COMMAND_UI_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE,
  OnUpdateColorUI)
9: //{{AFX_MSG_MAP(CTag11Doc)
```

```

10: // HINWEIS - Hier werden Mapping-Makros vom Klassen-Assistenten
    Äingefügt und entfernt.
11: // Innerhalb dieser generierten Quelltextabschnitte NICHTS ÄVERÄNDERN!
12: //}}AFX_MSG_MAP
13: END_MESSAGE_MAP()
14:
15: const COLORREF CTag11Doc::m_crColors[8] = {
16: RGB( 0, 0, 0), // Schwarz
17: RGB( 0, 0, 255), // Blau
18: .
19: .
20: .

```

6. Gehen Sie ans Ende der Datei, und fügen Sie die beiden Behandlungsroutinen gemäß Listing 11.3 hinzu.

Listing 11.3: Die Behandlungsroutinen für das Farbmenü

```

1: void CTag11Doc::OnColorCommand(UINT nID)
2: {
3: // Aktuelle Farbe setzen
4: m_nColor = nID - ID_COLOR_BLACK;
5: }
6:
7: void CTag11Doc::OnUpdateColorUI(CCmdUI* pCmdUI)
8: {
9: // Prüfen, ob Menübefehl mit Kontrollhäkchen zu versehen ist
10: pCmdUI->SetCheck(GetColor() == pCmdUI->m_nID ? 1 : 0);
11: }

```

In Listing 11.1 sind die beiden hinzugefügten Funktionsdeklarationen als Behandlungsroutinen für Nachrichten durch die Deklaration des Funktionstyps `afx_msg` spezifiziert. Diese Art der Funktionsdeklarationen muß mit geschütztem Zugriff versehen sein. Ansonsten sind diese Funktionen praktisch identisch mit jeder anderen Funktionsdeklaration von Klassenelementen.

In Listing 11.2 handelt es sich bei den beiden Einträgen `ON_COMMAND_RANGE` und `ON_UPDATE_COMMAND_UI_RANGE` in der Nachrichtenzuordnungstabelle um Standardeinträge, die aber der Klassen-Assistent weder unterstützt noch versteht. Wenn Sie sich die Einträge der Nachrichtenzuordnungstabelle in den Anwendungen der vergangenen Tage ansehen, finden Sie die Einträge `ON_COMMAND` und `ON_UPDATE_COMMAND_UI`. Diese Makros haben zwei Argumente, die Nachrichten-ID und der Name der Behandlungsfunktion, die für die Nachricht aufzurufen ist. Die neuen Einträge in der Nachrichtenzuordnungstabelle funktionieren genauso, weisen aber zwei Argumente für Nachrichten-IDs statt nur ein Argument auf. Die beiden ID-Argumente markieren Anfang und Ende eines Bereichs von Nachrichten-IDs, der an die spezifizierte Funktion zu übergeben ist. Die beiden Nachrichten-IDs beziehen sich auf den ersten und letzten Menübefehl, die Sie im Farbmenü erstellt haben.



Die Nachrichtenzuordnungstabelle ist ein von Visual C++ und MFC verwendeter Mechanismus, um in einfacher Weise Ereignisnachrichten und die jeweils aufzurufenden Behandlungsfunktionen zu spezifizieren. Die Befehle in der Nachrichtenzuordnungstabelle konvertiert der Visual C++-Compiler in eine schnelle und effiziente Tabelle für den Aufruf der passenden Behandlungsroutine, wenn die Anwendung eine Nachricht erhält. Immer, wenn Sie eine Funktion über den Klassen-Assistenten aufnehmen, fügen Sie nicht nur die Funktion in den Code hinzu, sondern auch einen Eintrag in die Nachrichtenzuordnungstabelle für diese Klasse.

Wenn Sie den Eintrag `ON_COMMAND_RANGE` der Nachrichtenzuordnungstabelle verwenden, wird die Nachrichten-ID automatisch als Argument an die Behandlungsfunktion übergeben. Dadurch ist es möglich, die Funktion entsprechend Listing 11.3 zu erstellen, um die Nachrichten der Farbauswahl zu behandeln. Wenn Sie die

Anwendung zum jetzigen Zeitpunkt kompilieren und ausführen, sollte die Farbauswahl genau wie in der gestrigen Anwendung funktionieren (siehe Abbildung 11.3).

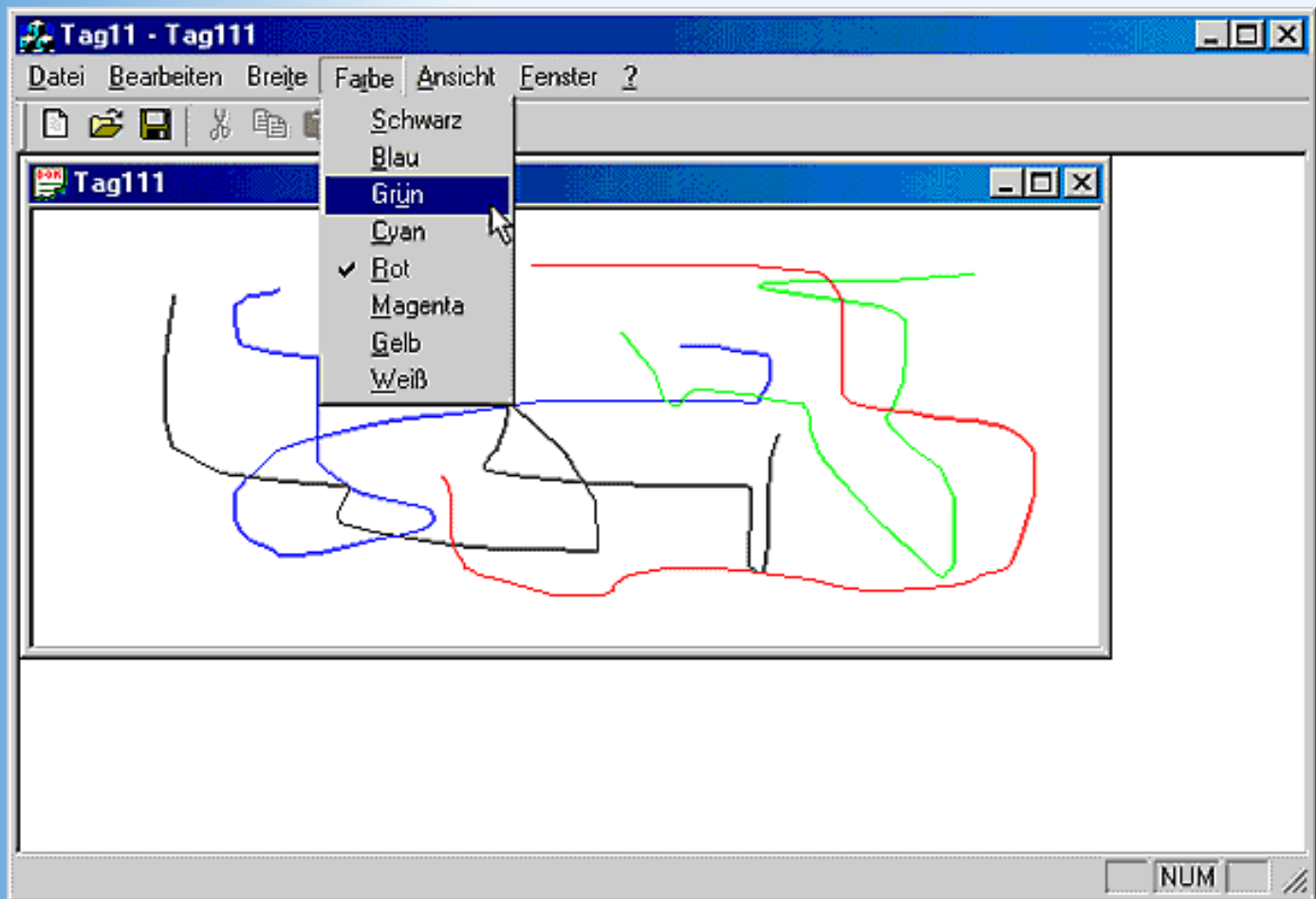


Abbildung 11.3:
Die MDI-Anwendung ausführen

Ein Kontextmenü hinzufügen

In den meisten Windows-Anwendungen kann man mit der rechten Maustaste klicken, und es erscheint ein sogenanntes Kontextmenü oder Popup-Menü. In Lektion 6 haben Sie bereits ein einfaches Kontextmenü erstellt. Allerdings gibt es einen bestimmten Mechanismus für das Erstellen und Verwenden dieser Kontextmenüs, wenn Windows annimmt, daß das Menü geöffnet werden sollte. Nach diesem Verfahren können Sie Kontextmenüs hinzufügen, die sich genau wie bei jeder anderen Windows-Anwendung verhalten (und wenn Microsoft den Auslösemechanismus von Kontextmenüs bei einer neueren Version von Windows ändert, verhält sich Ihre Anwendung weiterhin dem Windows-Standard entsprechend).

Windows schreibt in die Ereigniswarteschlange die Nachricht `WM_CONTEXTMENU`, wenn der Benutzer die rechte Maustaste losläßt oder die Taste für das Kontextmenü (bei neueren Windows-Tastaturen) betätigt. Wenn Sie eine Behandlungsroutine für die Nachricht `WM_CONTEXTMENU` vorsehen, können Sie ein Kontextmenü anzeigen und darauf vertrauen, daß es genau zum richtigen Zeitpunkt erscheint.

Um die Anwendung mit einem Kontextmenü auszustatten, erstellen Sie ein neues Menü, das als Kontextmenü dient. Führen Sie dazu die folgenden Schritte aus:

1. Klicken Sie im Arbeitsbereich auf der Registerkarte **Ressourcen** mit der rechten Maustaste auf den Ordner **Menu**.
2. Wählen Sie den Befehl **Menu einfügen** aus dem Kontextmenü.
3. Markieren Sie (immer noch im Arbeitsbereich) das neue Menü, öffnen Sie das zugehörige Eigenschaftsdialogfeld, und nennen Sie das Menü `IDR_CONTEXTMENU`.

4. Im Menü-Editor geben Sie als Titel für die Hauptmenüebene ein einzelnes Leerzeichen an. Das bewirkt, daß Visual C++ den ersten Menübefehl in den Dropdown- Bereich des Menüs hinzufügt.

5. Legen Sie für den ersten Menübefehl den Titel **Breite** fest, und schalten Sie das Kontrollkästchen **Popup** ein. (Daraufhin wird das Kombinationsfeld **ID** deaktiviert, der Titel des gerade modifizierten Menübefehls zeigt einen Pfeil, und rechts daneben erscheint ein weiterer Menübefehl.)

6. Fügen Sie zu diesem Zeitpunkt noch keine Menübefehle in das überlappende Menü **Breite** hinzu (das heben wir uns für eine Übung am Ende des Kapitels auf). Markieren Sie statt dessen den Menübefehl unterhalb des Befehls **Breite**, und öffnen Sie dessen Eigenschaftsdialogfeld. Legen Sie die Beschriftung mit **Fa&rben** fest, und schalten Sie das Kontrollkästchen **Popup** ein.

7. Nehmen Sie in das überlappende Farbménü die Menübefehle für die Farben auf, wie Sie es bereits für das Menü **IDR_TAG11TYPE** vorgenommen haben. Verwenden Sie auch die gleichen Eigenschaftseinstellungen. Wenn Sie die ID lieber suchen, statt per Hand einzutippen, können Sie sie auch aus der Dropdown-Liste **ID** auswählen. Wenn Sie damit fertig sind, sollte das Menü wie in Abbildung 11.4 aussehen.

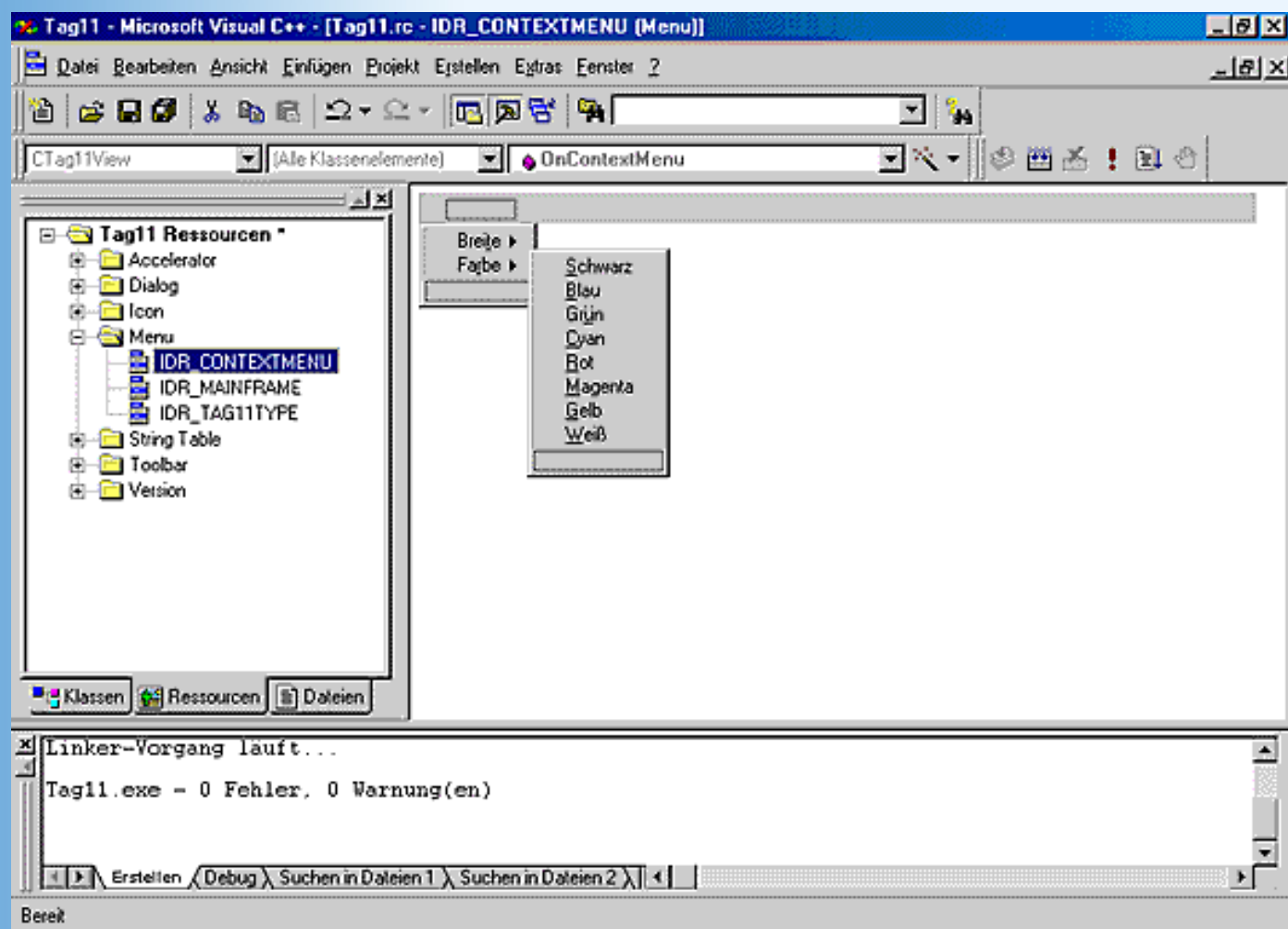


Abbildung 11.4:
Der Entwurf des Kontextmenüs

8. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**.

9. Markieren Sie die Klasse **CTag11View**. Öffnen Sie den Klassen-Assistenten über **Ansicht / Klassen-Assistent**.

10. Fügen Sie eine Funktion für die Nachricht **WM_CONTEXTMENU** in die Klasse **CTag11View** ein.

11. Schreiben Sie in die Funktion den Code aus Listing 11.4.

Listing 11.4: Die Funktion OnContextMenu der Klasse CTag11View

```
1: void CTag11View::OnContextMenu(CWnd* pWnd, CPoint point)
```

```

2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
4:
5: ///////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////
8:
9: CMenu menu;
10:
11: // Kontextmenü laden
12: menu.LoadMenu(IDR_CONTEXTMENU);
13: // Erstes Untermenü (das eigentliche Menü) holen
14: CMenu *pContextMenu = menu.GetSubMenu(0);
15:
16: // Kontextmenü anzeigen
17: pContextMenu->TrackPopupMenu(TPM_LEFTALIGN |
18: TPM_LEFTBUTTON | TPM_RIGHTBUTTON,
19: point.x, point.y, AfxGetMainWnd());
20:
21: ///////////////////////////////////
22: // EIGENER CODE, ENDE
23: ///////////////////////////////////
24: }

```

Dieser Code sollte Ihnen aus Lektion 6 bekannt vorkommen. Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie mit der rechten Maustaste auf das untergeordnete Fenster klicken und eine Zeichenfarbe aus dem daraufhin geöffneten Kontextmenü auswählen, wie es Abbildung 11.5 zeigt.

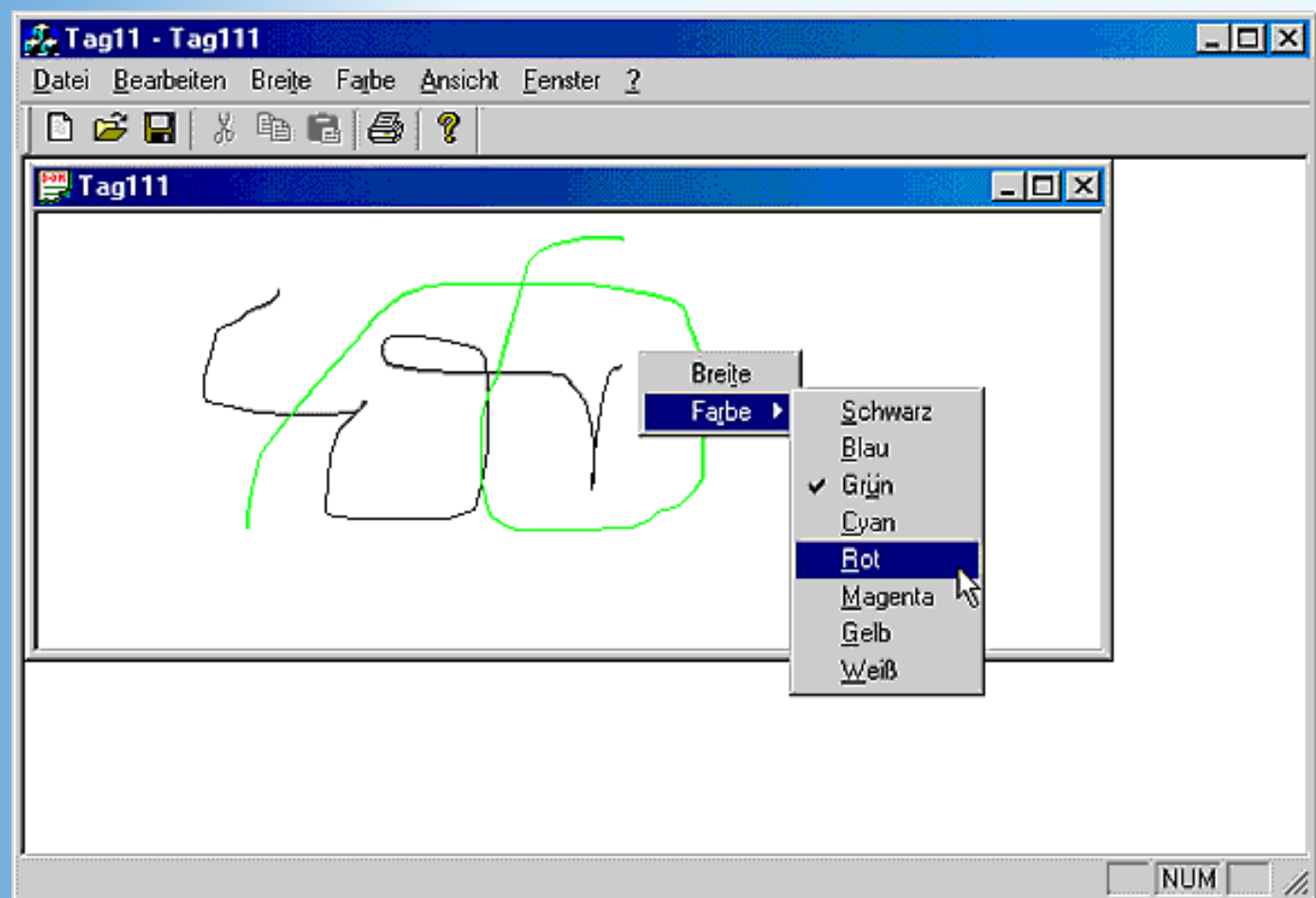


Abbildung 11.5:
Die Zeichenfarben über das Kontextmenü ändern

Zusammenfassung

Das war nicht schlecht, oder? Nach dem anstrengenden gestrigen Tag haben Sie sich sicherlich eine kleine Verschnaufpause mit der heutigen Lektion verdient. Außerdem konnten Sie viele Dinge aus der gestrigen Lektion vertiefen. Es ist aber auch einiges Neues hinzugekommen. Sie haben gelernt, was MDI-Anwendungen sind und wie sie sich von SDI-Anwendungen unterscheiden. Es wurde gezeigt, wie man eine Reihe von Menübefehlen mit einer einzigen Behandlungsroutine bearbeitet. Weiterhin haben Sie gelernt, wie man ein Menü erstellt, das speziell als Kontextmenü vorgesehen ist, und wie man es in eine MDI-Anwendung integriert.

Fragen und Antworten

Frage:

Da es sich bei einer MDI-Anwendung grundsätzlich um den gleichen Code handelt wie bei einer SDI-Anwendung, warum soll ich dann überhaupt eine SDI-Anwendung erstellen? Warum soll ich nicht einfach nur noch MDI-Anwendungen schreiben?

Antwort:

Das hängt von der konkreten Anwendung und deren vorgesehenem Einsatz ab. Wahrscheinlich werden beide Anwendungstypen in Ihrer Programmierstätigkeit zur Routine gehören. Wenn Sie ein Memo schreiben oder an einem Tabellenblatt arbeiten, werden Sie eine MDI-Anwendung bevorzugen. Wollen Sie durch das World Wide Web browsen, wird Ihr Webbrowser wahrscheinlich eine SDI-Anwendung sein. Ein einfacher Texteditor wie der Windows-Editor wäre als MDI-Anwendung für den Benutzer sicherlich unnötig kompliziert. Für die vorgesehenen Aufgaben ist eine SDI-Anwendung besser geeignet. Bei bestimmten Anwendungen ist es von vornherein sinnvoller, sie als SDI- und nicht als MDI-Anwendung zu konzipieren. Stellen Sie sich einfach den möglichen Einsatzbereich der geplanten Anwendung vor, und entscheiden Sie sich dann für das besser geeignete Modell.

Frage:

Einige Befehle in meinem Farbmenü führen zur Auswahl der falschen Farbe. Wie kann ich das Problem ermitteln?

Antwort:

Die Ursache des Problems liegt wahrscheinlich darin, daß die Menü-IDs des Farbmenüs nicht in der richtigen Reihenfolge aufeinanderfolgen. Das können Sie nachprüfen, indem Sie mit der rechten Maustaste im Arbeitsbereich auf der Registerkarte Ressourcen auf Tag11 Ressourcen klicken. Wählen Sie aus dem Kontextmenü den Befehl Ressourcensymbole, um eine Liste der IDs mit den zugeordneten Nummern in alphabetischer Ordnung anzuzeigen. Beginnen Sie mit der ID für Schwarz, und stellen Sie sicher, daß die Nummern mit einem Abstand von 1 aufeinanderfolgen, ohne Nummern zu überspringen. Überprüfen Sie, ob diese IDs in der Reihenfolge stehen, wie sie im Farbmenü (und in der Farbtabelle in der Datei Tag11Doc.cpp) eingetragen sind und nicht in alphabetischer Reihenfolge, wie sie in der Liste erscheinen. Wenn Sie auf Fehler stoßen sollen, müssen Sie Visual C++ schließen und die Datei Resource.h in einem Texteditor öffnen, um die IDs in der richtigen Weise neu zu numerieren. Nachdem Sie diese Korrekturen vorgenommen haben (achten Sie darauf, Duplikate zu löschen), speichern Sie sie, starten Visual C++ erneut und kompilieren die Anwendung. Das Farbmenü sollte nun ordnungsgemäß funktionieren.

Workshop

Kontrollfragen

1. Welche fünf Basisklassen kommen in MDI-Anwendungen zum Einsatz?
2. Warum müssen Sie den Eintrag `ON_COMMAND_RANGE` außerhalb des Abschnitts der vom Klassen-Assistenten verwalteten Nachrichtenzuordnungstabelle unterbringen?
3. Welches Argument übergibt `ON_COMMAND_RANGE` an die Behandlungsfunktion?

4. Welche Nachricht sollten Sie verwenden, um ein Kontextmenü anzuzeigen?

Übung

Fügen Sie die Pulldown- und Kontextmenüs für die Breite hinzu. Verwenden Sie die gleichen Stiftbreiten wie in der gestrigen Lektion.

❖·Kapitel Inhalt Index **SAMS** ❖· Top Kapitel·❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2

Tag 12

Symbolleisten und Statusleisten

Die bisher erstellten SDI- und MDI-Anwendungen verfügen nicht nur über Standardmenüs, sondern weisen auch einfache Symbolleisten auf, die mit den Menüs korrespondieren. Die Symbolleisten sind mit Standardfunktionen (Neu, Öffnen, Speichern, Drucken, Ausschneiden, Kopieren und Einfügen) ausgestattet, die sich in den Symbolleisten der meisten Windows-Anwendungen finden. Die Mehrzahl der Anwendungen beschränkt sich jedoch nicht auf diese vorgegebene Auswahl von Funktionen, sondern paßt die Symbolleisten an, um die konkrete Funktionalität der Anwendung widerzuspiegeln.

Neben den Symbolleisten verfügen die SDI- und MDI-Anwendungen über eine Statusleiste am unteren Rand des Anwendungsfensters, um eine textuelle Beschreibung der Symbolleistenschaltflächen und Menübefehle bereitzustellen. Die Statusleiste verfügt ebenfalls über Standardbereiche, die den Zustand der Feststelltasten für Großbuchstaben, Zahlen und Bildlauf anzeigen.

Heute lernen Sie insbesondere, wie man ...

- eigene Symbolleisten entwirft,
- eine Symbolleiste mit dem Anwendungsgerüst verbindet,
- eine Symbolleiste über einen Menübefehl anzeigt und ausblendet,
- ein Kombinationsfeld in eine Symbolleiste einbindet,
- Beschreibungen der Symbolleistenelemente in der Statusleiste anzeigt,
- eigene Elemente in die Statusleiste einbaut.

Symbolleisten, Statusleisten und Menüs

Eine der treibenden Kräfte hinter der Entwicklung von grafischen Benutzeroberflächen wie Windows ist das Ziel, die Benutzerfreundlichkeit von Computern zu verbessern. In diesem Zusammenhang haben die Entwickler der Oberflächen festgelegt, daß alle Anwendungen mit einem Standardsatz von Menüs auszustatten sind und daß die Menüs in einer standardisierten Art und Weise organisiert sein sollen. Der Gestaltung des Betriebssystems Microsoft Windows liegt die gleiche Philosophie zugrunde, d.h., in den meisten Anwendungen kommt ein standardisierter Satz von Menüs zum Einsatz, die in einer standardisierten Reihenfolge angeordnet sind.

Mit der zunehmenden Verbreitung von Windows ereignete sich etwas Wundersames. Die Anwendungsentwickler stellten fest, daß neue Benutzer trotzdem noch Schwierigkeiten hatten, sich in neue Anwendungen einzuarbeiten, und daß fortgeschrittene Benutzer die Menüs zu umständlich fanden. Daraufhin erfanden die Anwendungsgestalter die Symbolleisten als Lösung für beide Probleme.

Eine Symbolleiste ist ein schmales Band, das an einem Fensterrahmen angebunden oder als unverankertes Dialogfeld im Anwendungsrahmen verschiebbar ist. Dieses Band (oder Dialogfeld) verfügt über eine Anzahl kleiner Schaltflächen, die mit Bildern »beschriftet« sind, die man als Alternative zu Menübefehlen verwenden kann. Die Anwendungsentwickler plazieren die am häufigsten benutzten Funktionen ihrer Anwendungen auf diesen Symbolleisten und versuchen, die Bilder auf den Schaltflächen möglichst so zu gestalten, daß der Benutzer

die Funktionen ohne weiteres erkennen kann.

Nachdem sich die fortgeschrittenen Benutzer an die Symbolleistenschaltflächen gewöhnt hatten, traten die Symbolleisten ihren Siegeszug an. Allerdings hatten neue Benutzer immer noch Probleme, sich mit dem Zweck der Symbolleistenschaltflächen bekanntzumachen. Daraufhin setzten sich die Entwickler wieder ans Reißbrett, um neue Wege auszuknobeln, wie man neuen Benutzern den Umgang mit Symbolleistenschaltflächen erleichtern kann.

Heraus kam eine Lösung, bei der in einer Informationsleiste, die bereits viele Anwendungen am unteren Rand des Anwendungsfensters vorgesehen hatten, detaillierte Beschreibungen sowohl zu den Menüeinträgen als auch den Symbolleistenschaltflächen angezeigt wurden. Eine weitere Lösung war die Anzeige eines kleinen Hilfe-Popup-Fensters mit einer kurzen Beschreibung der Schaltfläche. Dieses Fenster erscheint, wenn man die Maus ein paar Sekunden über der jeweiligen Schaltfläche ruhen läßt. Die erste Lösung wurde als Statusleiste bekannt, die zweite als QuickInfo. Beide Verfahren sind in den meisten heutigen Windows-Anwendungen gängige Praxis.

Wenn Sie eigene Symbolleisten und Statusleisten in Ihren Anwendungen entwerfen und verwenden möchten, sollten Sie daran denken, daß Visual C++ eine Menge Unterstützung für Ihre Anstrengungen bereitstellt und sogar die Implementierung erleichtert. Immerhin haben selbst die Anwendungsentwickler von Microsoft den Vorreiter bei der Entwicklung dieser Elemente gespielt, und viele, wenn auch nicht alle, Windows-Anwendungen von Microsoft werden mit dem Visual C++ aus dem eigenen Hause entwickelt. Diese Tatsache ist sogar in den Tips und Tricks - die beim Start des Visual Studio erscheinen oder sich über das Hilfe-Menü aufrufen lassen - dokumentiert: »Wir haben es vor Ihnen verwendet! Visual C++ wurde mit Hilfe von Visual C++ entwickelt.« Und auch Sie sitzen heute in der ersten Reihe und erfahren, wie Sie eigene Symbolleisten und Statusleisten für Ihre Anwendungen erstellen.

Eine Symbolleiste entwerfen

Zur Einführung in das Thema Symbolleistenentwurf modifizieren wir die am Tag 10 erstellte SDI-Zeichenanwendung. Hier fügen wir eine Symbolleiste ein, mit der sich die Zeichenfarben auswählen lassen.

	Auch wenn die heute verwendete Beispielanwendung eine Erweiterung der am Tag 10 erstellten Anwendung ist, wurden alle Namen für Dateien und Klassen von <code>Tag10</code> in <code>Toolbar</code> geändert. Wenn Sie die Änderungen im Projekt von Tag 10 vornehmen und der nachstehende Text von Änderungen in der Klasse <code>CToolbarDoc</code> spricht, dann nehmen Sie die Änderungen in der Klasse <code>CTag10Doc</code> vor. Die Bearbeitung der Datei <code>Toolbar.rc</code> entspricht analog einer Bearbeitung der Datei <code>Tag10.rc</code>.

Wenn Sie lediglich ein paar zusätzliche Symbolleistenschaltflächen in die vom Anwendungs-Assistenten für eine SDI- oder MDI-Anwendung erzeugte Standardsymbolleiste aufnehmen möchten, können Sie die Symbolleiste im Symbolleisten-Editor (erreichbar über die Registerkarte Ressourcen des Arbeitsbereichs) von Visual C++ bearbeiten. Genau wie im Menü-Editor befindet sich am Ende der Symbolleiste ein leeres Feld, das Sie mit einer weiteren Symbolleistenschaltfläche ausgestalten können, wie es Abbildung 12.1 zeigt. Dazu brauchen Sie lediglich die leere Schaltfläche zu markieren und nach rechts ziehen, wenn ein Zwischenraum (ein Separator) zwischen ihr und der daneben liegenden Schaltfläche entstehen soll, oder die Schaltfläche an eine andere Position ziehen, wenn Sie die Schaltfläche verschieben möchten. Nachdem Sie die Schaltfläche auf der gewünschten Zielposition platziert haben, zeichnen Sie ein Symbol auf die Schaltfläche, das deren Funktion charakterisiert, wenn man die Schaltfläche anklickt. Schließlich doppelklicken Sie auf die Schaltfläche in der Symbolleistenansicht, um das Eigenschaftsdialogfeld der Schaltfläche zu öffnen. Hier geben Sie der Schaltfläche die gleiche ID wie dem Menübefehl, der die betreffende Funktion auslöst. Sobald Sie Ihre Anwendung kompilieren und ausführen, haben Sie eine neue Symbolleistenschaltfläche, die den jeweiligen Menübefehl realisiert. Wenn Sie eine Symbolleistenschaltfläche entfernen möchten, ziehen Sie sie in der Symbolleistenansicht aus der Symbolleiste heraus.

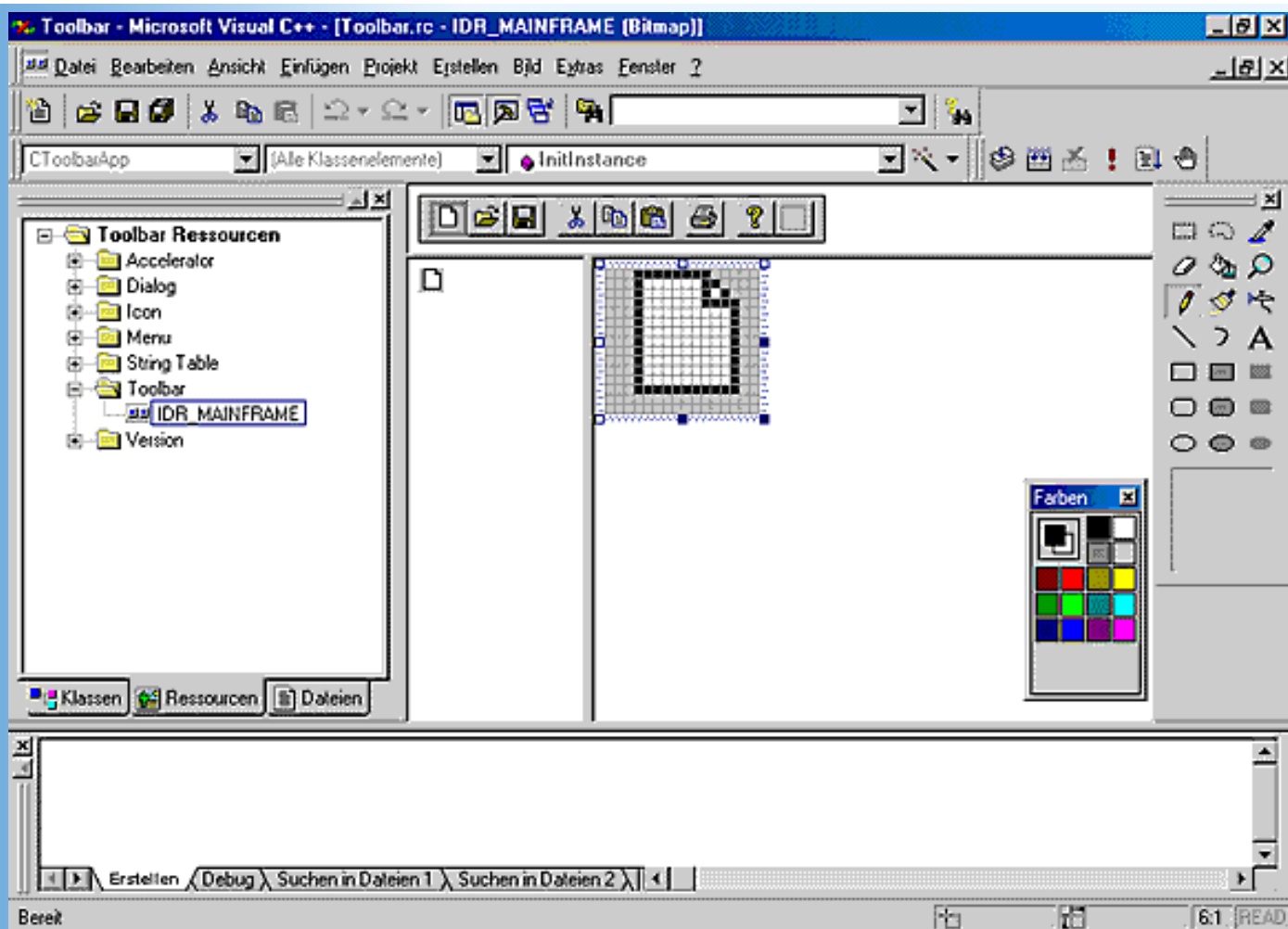


Abbildung 12.1:
Der Symbolleisten-Editor

Eine neue Symbolleiste erstellen

Um eine neue Symbolleiste einzufügen, klicken Sie mit der rechten Maustaste auf den Ordner `Toolbar` und wählen `Toolbar einfügen` aus dem Kontextmenü. Daraufhin wird eine leere Symbolleiste mit einer einzigen leeren Schaltfläche erzeugt. Sobald Sie ein Symbol auf die jeweils leeren Schaltflächen in der Symbolleiste zeichnen, kommt eine weitere leere Schaltfläche am Ende der Symbolleiste hinzu.

Für unsere Zeichenanwendung füllen Sie acht Schaltflächen mit den acht verfügbaren Farben des Zeichenprogramms.

Nachdem Sie die Schaltflächen in der Symbolleiste mit Symbolen ausgestattet haben, doppelklicken Sie in der Symbolleistenansicht auf die erste Schaltfläche. Daraufhin erscheint das Dialogfeld `Schaltfläche für Symbolleiste` Eigenschaften.

Im Feld `ID` geben Sie die ID des Menübefehls ein, den die Schaltfläche realisieren soll (die ID können Sie auch aus der Dropdown-Liste auswählen). In das Feld `Statuszeilentext` tragen Sie die Beschreibung ein, die in der Statusleiste für diese Schaltfläche erscheinen soll. (Wenn Sie bereits eine Aufforderung für den Menübefehl eingegeben haben, ist in diesem Feld schon die Menübeschreibung enthalten.) Am Ende der Beschreibung für die Statusleiste fügen Sie die Zeichen `\n` und einen kurzen Text für die QuickInfo der Symbolleistenschaltfläche an.



In C/C++ steht die Zeichenfolge `\n` für den Befehl »neue Zeile beginnen«. Im Statuszeilentext für Symbolleistschaltflächen und Menübefehle dient diese Zeichenfolge dazu, die Beschreibung der Menübefehle in der Statusleiste und die QuickInfos (die erscheinen, wenn man den Mauszeiger ein paar Sekunden über die Schaltfläche setzt) zu trennen. Die erste Zeile ist für die Beschreibung der Statusleiste vorgesehen, die zweite für den Text in den QuickInfos. Die Beschreibung für QuickInfos kommt nur bei Symbolleisten zum Einsatz, so daß man bei Menübefehlen ohne Symbolleistenäquivalente darauf verzichten kann.

Beispielsweise können Sie `ID_COLOR_BLACK` als ID und Zeichenfarbe `Schwarz\nSchwarz` als Statuszeilentext für die schwarze Schaltfläche auf der Symbolleiste, die Sie für Ihr Zeichenprogramm erstellen, festlegen, wie es Abbildung 12.2 verdeutlicht.

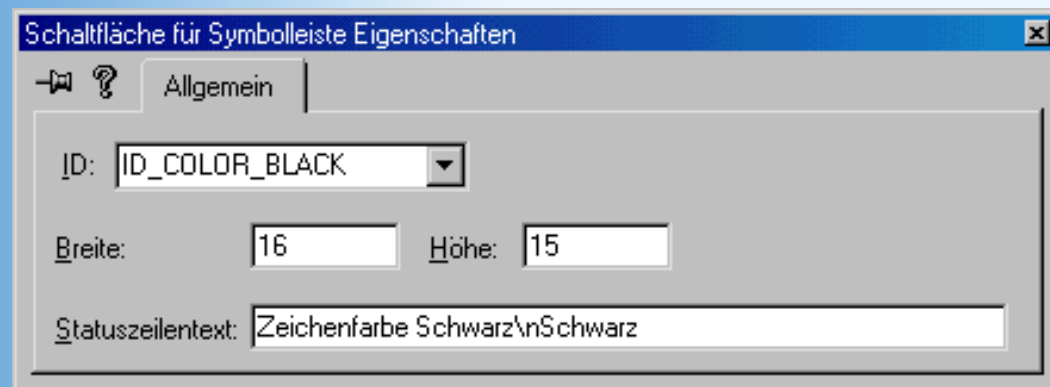


Abbildung 12.2:
Das Dialogfeld Schaltfläche für Symbolleiste Eigenschaften

Nachdem Sie den Entwurf der Symbolleiste fertiggestellt - die Schaltflächen mit Symbolen ausgestattet und die Eigenschaften für jede Schaltfläche festgelegt - haben, ändern Sie die ID der Symbolleiste. Im Arbeitsbereich klicken Sie mit der rechten Maustaste auf die neu hinzugefügte Symbolleiste und öffnen deren Eigenschaftsdialogfeld. Ändern Sie die ID der Symbolleiste in einen aussagekräftigen Namen.

Beispielsweise können Sie für die Symbolleiste Farben, die Sie für das Zeichenprogramm erstellen, die ID der Symbolleiste in `IDR_TBCOLOR` ändern.

Die Symbolleiste mit dem Anwendungsgerüst verbinden

In den bisherigen SDI- und MDI-Anwendungen haben Sie keinerlei Funktionalität hinzugefügt, die eine Arbeit mit dem Rahmenfenster erfordert hätte. Jetzt, da die Symbolleiste mit dem Rahmen verbunden wird, müssen Sie den Code in diesem Modul hinzufügen und modifizieren. Wenn Sie die Klasse `CMainFrame` öffnen und zur Funktion `OnCreate` gehen, sehen Sie, wo die vorhandene Symbolleiste erzeugt und (weiter unten in der Funktion) mit dem Rahmen verbunden wird.

Bevor Sie die Symbolleiste mit dem Anwendungsgerüst verbinden können, müssen Sie zunächst eine Variable in die Klasse `CMainFrame` hinzufügen, um den Wert der neuen Symbolleiste aufzunehmen. Für diese Variable vom Typ `CToolBar` legen Sie den Zugriffsstatus `Protected` fest.

Um die Symbolleiste Farben in Ihre Zeichenanwendung aufzunehmen, klicken Sie mit der rechten Maustaste auf der Registerkarte Klassen des Arbeitsbereichs auf die Klasse `CMainFrame`. Wählen Sie Member-Variable hinzufügen aus dem Kontextmenü, und legen Sie den Variablentyp mit `CToolBar`, den Namen als `m_wndColorBar` und den Zugriff als `Protected` fest.

Jetzt schreiben Sie noch etwas Code in die Funktion `OnCreate` der Klasse `CMainFrame`, um die Symbolleiste hinzuzufügen und sie mit dem Gerüst zu verbinden. Übernehmen Sie die Modifikationen aus Listing 12.1, um die Symbolleiste Farben in Ihre Zeichenanwendung einzubauen.

Listing 12.1: Die modifizierte Funktion `OnCreate` der Klasse `CMainFrame`

```
1: int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```

2: {
3: if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
4: return -1;
5:
6: if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
  CBRSTOP
7: | CBRSGRIPPER | CBRSTOOLTIPS | CBRSTFLYBY | CBRSSIZE_DYNAMIC) ||
8: !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
9: {
10: TRACE0("Symbolleiste konnte nicht erstellt werden\n");
11: return -1; // Fehler bei Erstellung
12: }
13:
14: //////////////////////////////////////
15: // EIGENER CODE, ANFANG
16: //////////////////////////////////////
17:
18: // Symbolleiste Farben hinzufügen
19: int iTBCtlID;
20: int i;
21:
22: // Symbolleiste Farben erzeugen
23: if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
24: WS_VISIBLE | CBRSTOP | CBRSGRIPPER | CBRSTOOLTIPS |
25: CBRSTFLYBY | CBRSSIZE_DYNAMIC) ||
26: !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
27: {
28: TRACE0("Symbolleiste konnte nicht erstellt werden\n");
29: return -1; // Fehler bei Erstellung
30: }
31: // Schaltfläche Schwarz auf der Symbolleiste suchen
32: iTBCtlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
33: if (iTBCtlID >= 0)
34: {
35: // Schleife durch die Schaltflächen, um sie als Optionsfelder wirken zu
  lassen
36: for (i= iTBCtlID; i < (iTBCtlID + 8); i++)
37: m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
38: }
39:
40: //////////////////////////////////////
41: // EIGENER CODE, ENDE
42: //////////////////////////////////////
43:
44: if (!m_wndStatusBar.Create(this) ||
45: !m_wndStatusBar.SetIndicators(indicators,
46: sizeof(indicators)/sizeof(UINT)))
47: {
48: TRACE0("Statusleiste konnte nicht erstellt werden\n");
49: return -1; // Fehler bei Erstellung
50: }
51:
52: // ZU ERLEDIGEN: Löschen Sie diese drei Zeilen, wenn Sie nicht wollen,
  dass die Symbolleiste
53: // andockbar ist.

```

```

54: m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
55:
56: //////////////////////////////////
57: // EIGENER CODE, ANFANG
58: //////////////////////////////////
59:
60: // Andocken für Symbolleiste Farben aktivieren
61: m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
62:
63: //////////////////////////////////
64: // EIGENER CODE, ENDE
65: //////////////////////////////////
66:
67: EnableDocking(CBRS_ALIGN_ANY);
68: DockControlBar(&m_wndToolBar);
69:
70: //////////////////////////////////
71: // EIGENER CODE, ANFANG
72: //////////////////////////////////
73:
74: // Symbolleiste Farben andocken
75: DockControlBar(&m_wndColorBar);
76:
77: //////////////////////////////////
78: // EIGENER CODE, ENDE
79: //////////////////////////////////
80:
81: return 0;
82: }

```

Die Symbolleiste erstellen

Der erste hinzugefügte Codeabschnitt

```

23: if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
24: WS_VISIBLE | CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
25: CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
26: !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))

```

enthält zwei separate Funktionen, die für das Erstellen einer Symbolleiste erforderlich sind. Die erste Funktion, `CreateEx`, erstellt die Symbolleiste selbst, während die zweite Funktion, `LoadToolBar`, die Symbolleiste lädt, die Sie im Symbolleisten-Editor entworfen haben. Die zweite Funktion, `LoadToolBar`, erfordert ein Argument: die ID für die zu erzeugende Symbolleiste.

Der Funktion `CreateEx` können Sie mehrere Argumente übergeben. Das erste - und einzige erforderliche - Argument ist ein Zeiger auf das übergeordnete Fenster. In diesem Fall (der den Normalfall darstellt) ist dieses Argument ein Zeiger auf das Rahmenfenster, mit dem die Symbolleiste verbunden wird.

Das zweite Argument gibt den Stil der Steuerelemente in der zu erzeugenden Symbolleiste an. Es stehen mehrere Stile zur Auswahl, von denen einige mit den beiden letzten Versionen des Internet Explorer eingeführt wurden. Tabelle 12.1 listet die verfügbaren Stile auf.

Tabelle 12.1: Stile für die Steuerelemente von Symbolleisten

Stil	Beschreibung
TBSTYLE_ALTDRAW	Erlaubt dem Benutzer, die Symbolleiste bei niedergehaltener (Alt)-Taste zu verschieben.

TBSTYLE_CUSTOMERASE	Generiert eine NM_CUSTOMDRAW-Nachricht, wenn man die Symbolleiste und den Schaltflächenhintergrund löscht. Damit kann der Programmierer wählen, wann und ob das Löschen des Hintergrunds zu steuern ist.
TBSTYLE_FLAT	Erzeugt eine flache Symbolleiste. Der Schaltflächentext erscheint unter dem Bitmap.
TBSTYLE_LIST	Der Schaltflächentext steht rechts neben dem Bitmap.
TBSTYLE_REGISTERDROP	Beim Drag & Drop von Objekten auf Symbolleistenschaltflächen verwendet.
TBSTYLE_TOOLTIPS	Erzeugt ein QuickInfo-Steuerelement, in dem man eine Beschreibung der Symbolleistenschaltflächen anzeigen kann.
TBSTYLE_TRANSPARENT	Erzeugt eine transparente Symbolleiste.
TBSTYLE_WRAPABLE	Erzeugt eine Symbolleiste, die aus mehreren Schaltflächenreihen bestehen kann.

Das dritte Argument gibt den Stil der Symbolleiste selbst an. Dieses Argument ist normalerweise eine Kombination von Fenster- und Steuerelementstilen. In der Regel verwendet man nur zwei oder drei Fensterstile, während es sich beim Rest der Symbolleistenstile um Steuerelementstile handelt. In Tabelle 12.2 finden Sie die üblicherweise verwendeten Symbolleistenstile.

Tabelle 12.2: Stile für Symbolleisten

Stil	Beschreibung
WS_CHILD	Die Symbolleiste wird als untergeordnetes Fenster geöffnet.
WS_VISIBLE	Die Symbolleiste ist sichtbar, wenn sie erzeugt wird.
CBRS_ALIGN_TOP	Die Symbolleiste lässt sich am oberen Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_BOTTOM	Die Symbolleiste lässt sich am unteren Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_LEFT	Die Symbolleiste lässt sich an der linken Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_RIGHT	Die Symbolleiste lässt sich an der rechten Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_ANY	Die Symbolleiste lässt sich an einer beliebigen Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_BORDER_TOP	Setzt einen Rahmen am oberen Rand der Symbolleiste, wenn der obere Rand der Symbolleiste nicht verankert ist.
CBRS_BORDER_BOTTOM	Setzt einen Rahmen am unteren Rand der Symbolleiste, wenn der obere Rand der Symbolleiste nicht verankert ist.
CBRS_BORDER_LEFT	Setzt einen Rahmen an den linken Rand der Symbolleiste, wenn der obere Rand der Symbolleiste nicht verankert ist.
CBRS_BORDER_RIGHT	Setzt einen Rahmen an den rechten Rand der Symbolleiste, wenn der obere Rand der Symbolleiste nicht verankert ist.
CBRS_FLOAT_MULTI	Gestattet mehrere unverankerte Symbolleisten in einem einzelnen Minirahmenfenster.
CBRS_TOOLTIPS	Bewirkt die Anzeige von QuickInfos für die Symbolleistenschaltflächen.

CBRS_FLYBY	Bewirkt die gleichzeitige Aktualisierung des Textes in der Statusleiste für Symbolleistenschaltflächen und QuickInfos.
CBRS_GRIPPER	Zeichnet eine Griffleiste auf die Symbolleiste.

Das vierte Argument, welches Sie in Ihrem Code nicht bereitstellen, gibt die Größe der Symbolleistenrahmen an. Dieses Argument wird als normale `CRect`-Klasse übergeben, um die gewünschte Länge und Höhe für die Symbolleiste bereitzustellen. Der Standardwert ist `0` für alle Abmessungen des Rechtecks, so daß die Symbolleiste keine Rahmen aufweist.

Das fünfte und letzte Argument, das Sie ebenfalls nicht in Ihrem Code verwenden, stellt die ID des untergeordneten Symbolleistenfensters bereit. Der Standardwert lautet `AFX_IDW_TOOLBAR`, man kann aber alle definierten IDs verwenden, die für die Symbolleiste erforderlich sind oder geeignet erscheinen.

Die Stile der Schaltflächen festlegen

An das Erstellen der Symbolleiste schließt sich ein etwas eigentümlicher Codeabschnitt an:

```
31: // Schaltfläche Schwarz auf der Symbolleiste suchen
32: iTBctlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
33: if (iTBctlID >= 0)
34: {
35: // Schleife durch die Schaltflächen, um sie als Optionsfelder wirken zu
lassen
36: for (i= iTBctlID; i < (iTBctlID + 8); i++)
37: m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
38: }
```

Die erste Zeile in diesem Codefragment verwendet die Symbolleistenfunktion `CommandToIndex`, um die Steuerelementnummer der Schaltfläche `ID_COLOR_BLACK` zu ermitteln. Wenn Sie die Symbolleiste in der Reihenfolge der Farben, wie sie im Menü erscheinen, erstellt haben, sollte es sich dabei um das erste Steuerelement mit einem Index von `0` handeln. Es empfiehlt sich, den Index aller Symbolleistenschaltflächen, die man ändern muß, mit der Funktion `CommandToIndex` zu ermitteln, einfach deshalb, weil man nicht immer davon ausgehen kann, daß die Schaltfläche an der erwarteten Position sitzt. Die Funktion liefert den Index des angegebenen Symbolleistensteuerelements zurück. Diesen Wert verwendet man als Ausgangspunkt, um den Schaltflächenstil der Farbschaltflächen festzulegen.

In der Schleife, in der Sie alle acht Farbschaltflächen der Symbolleiste durchlaufen, steuern Sie mit der Funktion `SetButtonStyle` das Verhalten der Symbolleistenschaltflächen. Das erste Argument an diese Funktion ist der Index der Schaltfläche, die zu ändern ist. Das zweite Argument gibt den Stil für die betreffende Schaltfläche an. In diesem Fall legen wir fest, daß alle Schaltflächen den Stil `TBBS_CHECKGROUP` aufweisen. Damit verhalten sich die Schaltflächen wie Optionsfelder, wobei nur eine Schaltfläche in der Gruppe zu einem bestimmten Zeitpunkt ausgewählt sein kann. Die Liste der verfügbaren Schaltflächenstile gibt Tabelle 12.3 wieder.

Tabelle 12.3: Stile für Symbolleistenschaltflächen

Stil	Beschreibung
TBSTYLE_AUTOSIZE	Die Breite der Schaltfläche wird nach dem Text auf der Schaltfläche berechnet.
TBSTYLE_BUTTON	Erzeugt eine Standardschaltfläche.
TBSTYLE_CHECK	Erzeugt eine Schaltfläche, die sich wie ein Kontrollkästchen verhält und zwischen dem gedrückten und nicht gedrückten Zustand umschaltet.
TBSTYLE_CHECKGROUP	Erzeugt eine Schaltfläche, die sich wie ein Optionsfeld verhält. Die Schaltfläche bleibt im gedrückten Zustand, bis eine andere Schaltfläche in der Gruppe betätigt wird. Es handelt sich hier eigentlich um eine Kombination der Schaltflächenstile <code>TBSTYLE_CHECK</code> und <code>TBSTYLE_GROUP</code> .

TBSTYLE_DROPDOWN	Erzeugt eine Schaltfläche in der Art einer Dropdown-Liste.
TBSTYLE_GROUP	Erstellt eine Schaltfläche, die gedrückt bleibt, bis eine andere Schaltfläche in der Gruppe gedrückt wird.
TBSTYLE_NOPREFIX	Der Schaltflächentext weist kein zugeordnetes Präfix auf.
TBSTYLE_SEP	Erzeugt einen Separator, d.h. eine kleine Lücke zwischen den Schaltflächen zu beiden Seiten.

Die Symbolleiste verankern

Der letzte Codeabschnitt, den Sie in die Funktion `OnCreate` der Klasse `CMainFrame` aufgenommen haben, sieht folgendermaßen aus:

```
60: // Andocken für Symbolleiste Farben aktivieren
61: m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
...
67: EnableDocking(CBRS_ALIGN_ANY); // (Vom Anwendungs-Assistenten generierte
Zeile)
...
73:
74: // Symbolleiste Farben andocken
75: DockControlBar(&m_wndColorBar);
```

In der ersten Zeile steht der Aufruf der Funktion `EnableDocking`. Diese Funktion erlaubt das Verankern der Symbolleiste mit dem Rahmenfenster. Der an diese Symbolleistenfunktion übergebene Wert muß mit dem Wert übereinstimmen, der im nachfolgenden Aufruf der Funktion `EnableDocking` für das Rahmenfenster übergeben wird. Die verfügbaren Werte für diese Funktionen sind in Tabelle 12.4 aufgeführt. Mit diesen Funktionen lassen sich die Rahmen der Symbolleiste und das Rahmenfenster für das Andocken aktivieren. Ruft man diese Funktionen nicht auf, kann man die Symbolleiste nicht mit dem Rahmenfenster verankern. Übergibt man eine bestimmte Seite für das Andocken in diesen Funktionen, und die Seite entspricht nicht diesem Wert, läßt sich die Symbolleiste ebenfalls nicht am Rahmenfenster verankern.

Tabelle 12.4: Seiten, an die man Symbolleisten verankern kann

Stil	Beschreibung
CBRS_ALIGN_TOP	Die Symbolleiste läßt sich am oberen Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_BOTTOM	Die Symbolleiste läßt sich am unteren Rand des Anzeigebereichs im Rahmenfenster verankern.
CBRS_ALIGN_LEFT	Die Symbolleiste läßt sich an der linken Seite des Anzeigebereichs des Rahmenfensters verankern.
CBRS_ALIGN_RIGHT	Die Symbolleiste läßt sich an der rechten Seite des Anzeigebereichs des Rahmenfensters verankern.
CBRS_ALIGN_ANY	Die Symbolleiste läßt sich an einer beliebigen Seite des Anzeigebereichs im Rahmenfenster verankern.
CBRS_FLOAT_MULTI	Gestattet mehrere unverankerte Symbolleisten in einem einzelnen Minirahmenfenster.
0	Die Symbolleiste läßt sich nicht mit dem Rahmen verankern.

Als letzte Funktion wurde die Fensterfunktion `DockControlBar` hinzugefügt, der die Adresse der Symbolleistenvariablen übergeben wird. Diese Funktion verankert die Symbolleiste physisch mit dem

Rahmenfenster. Da der gesamte Code in der Funktion `OnCreate` für das Rahmenfenster erscheint, wird die Symbolleiste verankert, bevor der Benutzer das Fenster oder die Symbolleiste zu Gesicht bekommt.

Nachdem Sie nun den gesamten Code in die Funktion `OnCreate` der Klasse `CMainFrame` aufgenommen haben, können Sie Ihre Anwendung kompilieren und ausführen. Sie finden eine funktionsfähige Symbolleiste Farben vor, über die sich die Zeichenfarbe auswählen läßt (siehe Abbildung 12.3).

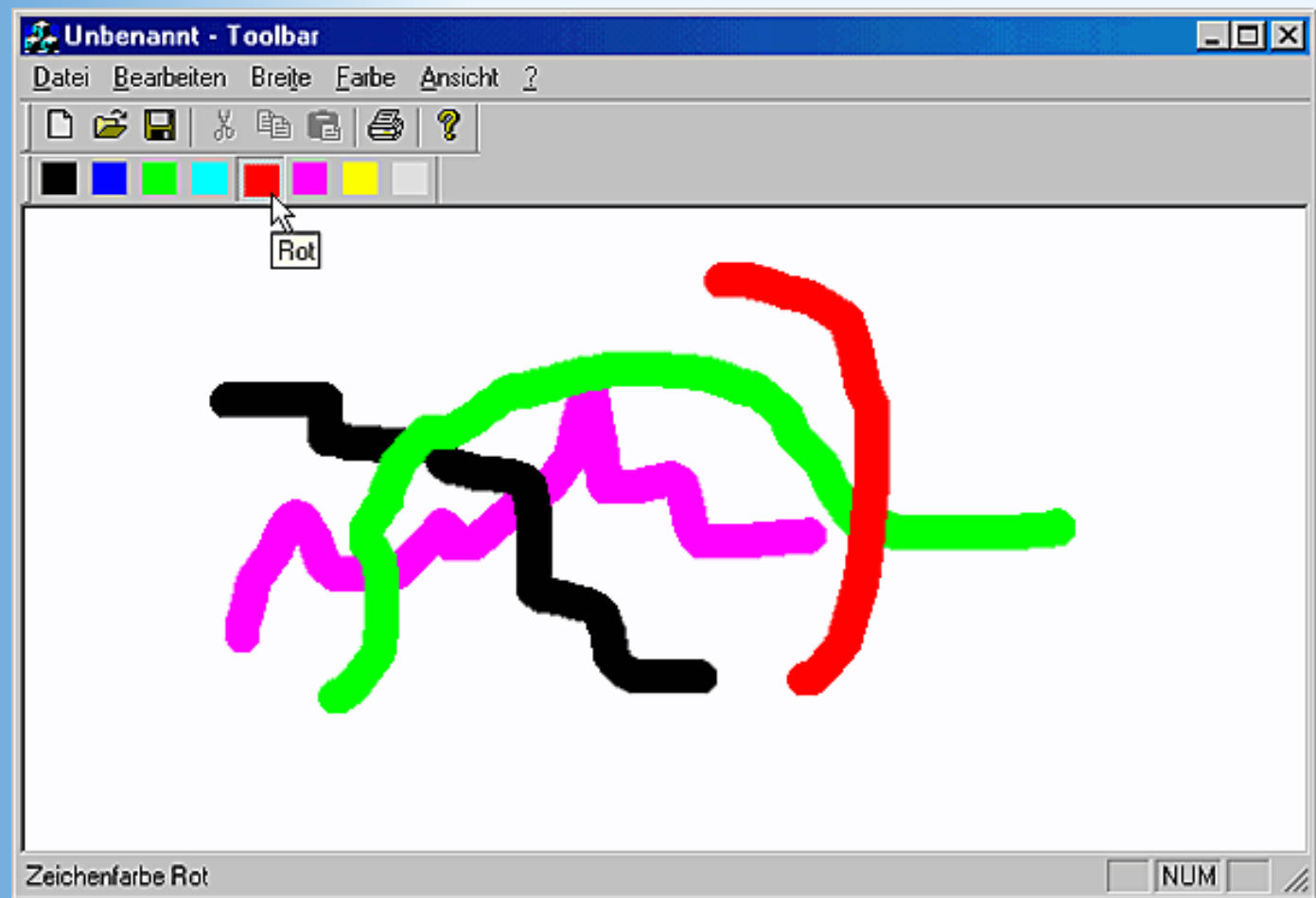


Abbildung 12.3:
Die Symbolleiste Farben im Zeichenprogramm

Die Sichtbarkeit der Symbolleiste steuern

Nunmehr verfügen Sie über eine Symbolleiste Farben, die Sie mit dem Rahmen Ihres Zeichenprogramms verankern können. Es wäre schön, wenn man die Symbolleiste genauso über das Menü `Ansicht` anzeigen und ausblenden könnte, wie es mit der Standardsymbolleiste und der Statusleiste möglich ist. Diese Funktionalität läßt sich recht einfach implementieren, aber es funktioniert nicht unbedingt so, wie Sie es vielleicht erwarten.

Als erstes ist ein Menübefehl hinzuzufügen, um die Sichtbarkeit der Farbpalette umzuschalten. Diese Aufgabe erledigen Sie über den Menü-Editor, mit dem Sie einen neuen Menübefehl in das Menü `Ansicht` aufnehmen. Legen Sie die Eigenschaften des Menüs gemäß Tabelle 12.5 fest.

Tabelle 12.5: Eigenschaften des Menübefehls Farbpalette

Eigenschaft	Einstellung
ID	ID_VIEW_COLORBAR
Titel	&Farbpalette
Statuszeilentext	Blendet die Farbpalette ein oder aus Farbpalette ein/aus

Das Menü aktualisieren

Um zu ermitteln, ob die Symbolleiste sichtbar oder ausgeblendet ist, kann man den aktuellen Stil der Symbolleiste lesen und das Stilflag `WS_VISIBLE` ausmaskieren. Wenn das Flag im aktuellen Symbolleistenstil enthalten ist, ist die Symbolleiste sichtbar. Indem man diese Auswertung in die Funktion `SetCheck` in der Behandlungsroutine `UPDATE_COMMAND_UI` schreibt, läßt sich das Kontrollhäkchen neben dem Menüeintrag für die Farbpalette umschalten.

Diese Funktionalität läßt sich im Zeichenprogramm mit einer Behandlungsroutine für das Ereignis `UPDATE_COMMAND_UI` des Menüs `ID_VIEW_COLORBAR` realisieren. Achten Sie darauf, diese Behandlungsroutine in die Klasse `CMainFrame` aufzunehmen. In die Funktion schreiben Sie den Code von Listing 12.2.

Listing 12.2: Die Funktion `OnUpdateViewColorbar` der Klasse `CMainFrame`

```
1: void CMainFrame::OnUpdateViewColorbar(CCmdUI* pCmdUI)
2: {
3: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
  Benutzeroberfläche hier einfügen
4: //////////////////////////////////
5: // EIGENER CODE, ANFANG
6: //////////////////////////////////
7:
8: // Zustand der Farbpalette prüfen
9: pCmdUI->SetCheck(((m_wndColorBar.GetStyle() & WS_VISIBLE) != 0));
10:
11: //////////////////////////////////
12: // EIGENER CODE, ENDE
13: //////////////////////////////////
14: }
```

Die Sichtbarkeit der Symbolleiste umschalten

Da die Klasse `CToolBar` von der Klasse `CWnd` (über die Klasse `CControlBar`) abgeleitet ist, könnte man meinen, daß man die Funktion `ShowWindow` auf der Symbolleiste selbst aufrufen kann, um die Symbolleiste anzuzeigen und auszublenden. Das ist zwar möglich, aber der Hintergrund für die Symbolleiste wird nicht zusammen mit der Symbolleiste ausgeblendet. Der Benutzer stellt lediglich fest, daß die Symbolleistenschaltflächen erscheinen und verschwinden. (Natürlich ist das die Wirkung, die Sie erzielen möchten, aber Ihre Benutzer werden davon nicht sehr angetan sein.)

Statt dessen greifen Sie auf eine Funktion `ShowControlBar` für das Rahmenfenster zurück, um die Symbolleiste anzuzeigen und auszublenden. Diese Funktion übernimmt drei Argumente. Das erste Argument ist die Adresse für die Symbolleistenvariable. Das zweite Argument ist ein Boolescher Wert, der angibt, ob die Symbolleiste anzuzeigen ist. (`TRUE` zeigt die Symbolleiste an, `FALSE` blendet sie aus.) Schließlich spezifiziert das dritte Argument, ob die Symbolleiste verzögert erscheinen soll. (`TRUE` zeigt die Symbolleiste verzögert, `FALSE` sofort an.)

Nachdem eine Symbolleiste ein- oder ausgeschaltet ist, müssen Sie eine weitere Funktion für das Rahmenfenster, `RecalcLayout`, aufrufen. Diese Funktion bewirkt, daß der Rahmen alle Symbolleisten, Statusleisten und alles innerhalb des Rahmenbereichs neu positioniert. Diese Funktion ist dafür verantwortlich, daß sich die Symbolleiste Farben nach oben oder unten verschiebt, wenn Sie die Standardsymbolleiste ein- und ausschalten.

Diese Funktionalität nehmen Sie in Ihr Zeichenprogramm mit Hilfe einer Behandlungsfunktion für das Ereignis `COMMAND` des Menübefehls `ID_VIEW_COLORBAR` auf. Achten Sie darauf, daß Sie diese Behandlungsfunktion in die Klasse `CMainFrame` aufnehmen. In die Funktion schreiben Sie den Code aus Listing 12.3.

Listing 12.3: Die Funktion `OnViewColorbar` der Klasse `CMainFrame`

```
1: void CMainFrame::OnViewColorbar()
2: {
```

```

3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8: BOOL bVisible;
9:
10: // Zustand der Farbpalette prüfen
11: bVisible = ((m_wndColorBar.GetStyle() & WS_VISIBLE) != 0);
12:
13: // Symbolleiste Farben umschalten
14: ShowControlBar(&m_wndColorBar, !bVisible, FALSE);
15: // Elemente im Rahmenfenster neu anordnen
16: RecalcLayout();
17:
18: //////////////////////////////////
19: // EIGENER CODE, ENDE
20: //////////////////////////////////
21: }

```

Wenn Sie Ihre Anwendung jetzt kompilieren und ausführen, sollten Sie die Symbolleiste Farben über das Menü Ansicht ein- und ausschalten können.

Die Symbolleiste mit einem Kombinationsfeld ausstatten

Mittlerweile ist es üblich, daß die Symbolleisten einer Anwendung neben einfachen Schaltflächen auch noch andere Elemente enthalten. Sehen Sie sich zum Beispiel das Visual Studio an. Hier finden Sie Kombinationsfelder, die Ihnen die Navigation durch den Code erlauben, indem Sie die Klasse, die ID und die zu bearbeitende Funktion einfach über die Symbolleiste auswählen. Wie fügt man nun ein Kombinationsfeld in eine Symbolleiste ein? Der Symbolleisten-Editor stellt es nicht zur Verfügung. Hier finden Sie nur Schaltflächen, auf die Sie Symbole zeichnen können. Mit den Assistenten von Visual C++ lassen sich keine Kombinationsfelder in Symbolleisten aufnehmen. Zu diesem Zweck müssen Sie ein wenig C++-Code schreiben.

Als Einstieg fügen Sie ein Kombinationsfeld in die Symbolleiste Farben ein, die Sie gerade erstellt haben. Mit dem Kombinationsfeld wählt der Benutzer die Breite des Zeichenstifts aus. (Wenn Sie die Unterstützung für unterschiedliche Zeichenbreiten gemäß Übung am Ende von Tag 10 noch nicht realisiert haben, sollten Sie das jetzt nachholen.)

Die Projektressourcen bearbeiten

Um ein Kombinationsfeld in die Symbolleiste aufzunehmen, ist zuerst das zu tun, was Ihnen Visual C++ verweigert. Sie müssen die Ressourcendatei selbst bearbeiten. Über das Visual Studio für Visual C++ ist das nicht möglich. Wenn Sie versuchen, die Ressourcendatei im Visual Studio zu öffnen, gelangen Sie unweigerlich auf die Registerkarte Ressourcen des Arbeitsbereichs, wo die verschiedenen Ressourcen-Editoren untergebracht sind. Nein, Sie müssen diese Datei in einem anderen Editor, wie etwa dem normalen Windows-Editor, bearbeiten.

Schließen Sie Visual C++. Nur so läßt sich garantieren, daß Sie nicht über Ihre Änderungen schreiben. Öffnen Sie den Editor, und gehen Sie in Ihr Projektverzeichnis. Öffnen Sie die Ressourcendatei, die nach dem Projekt benannt ist und eine .rc-Erweiterung aufweist. Blättern Sie in der Datei nach unten bis zu den Symbolleistendefinitionen. (Sie können nach dem Wort »Toolbar« suchen.) Dann gehen Sie ans Ende der Symbolleistendefinition und fügen zwei Separatorlinien am unteren Rand der Symbolleistendefinition ein.

In Ihrer Zeichenanwendung müssen Sie zum Beispiel in das Projektverzeichnis `Toolbar` gehen und die Datei `Toolbar.rc` öffnen. (Falls Sie diese Symbolleisten in die MDI- Zeichenanwendung einfügen, wäre das die

Datei Tag11.rc.) Suchen Sie nach dem Symbolleistenabschnitt, und fügen Sie dann zwei SEPARATOR-Zeilen unmittelbar vor dem Ende des Abschnitts IDR_TBCOLOR ein, wie es Listing 12.4 zeigt. Dann speichern Sie die Datei, verlassen den Editor, starten Visual C++ erneut und laden wieder das Projekt.

Listing 12.4: Die modifizierte Ressourcendatei des Projekts (Toolbar.rc)

```

1:
////////////////////////////////////
2: //
3: // Toolbar
4: //
5:
6: IDR_MAINFRAME TOOLBAR DISCARDABLE 16, 15
7: BEGIN
8:  BUTTON ID_FILE_NEW
9:  BUTTON ID_FILE_OPEN
10:  BUTTON ID_FILE_SAVE
11:  SEPARATOR
12:  BUTTON ID_EDIT_CUT
13:  BUTTON ID_EDIT_COPY
14:  BUTTON ID_EDIT_PASTE
15:  SEPARATOR
16:  BUTTON ID_FILE_PRINT
17:  BUTTON ID_APP_ABOUT
18: END
19:
20: IDR_TBCOLOR TOOLBAR DISCARDABLE 16, 15
21: BEGIN
22:  BUTTON ID_COLOR_BLACK
23:  BUTTON ID_COLOR_BLUE
24:  BUTTON ID_COLOR_GREEN
25:  BUTTON ID_COLOR_CYAN
26:  BUTTON ID_COLOR_RED
27:  BUTTON ID_COLOR_MAGENTA
28:  BUTTON ID_COLOR_YELLOW
29:  BUTTON ID_COLOR_WHITE
30:  SEPARATOR
31:  SEPARATOR
32: END

```

Die beiden SEPARATOR-Zeilen haben Sie in die Symbolleistendefinition aufgenommen, damit der zweite Separator als Platzhalter für das Kombinationsfeld dienen kann, das Sie in die Symbolleiste einbauen wollen. Es gibt zwei Gründe dafür, daß Sie diese Bearbeitung manuell erledigen müssen und nicht den Symbolleisten-Editor von Visual C++ bemühen können. Erstens erlaubt der Symbolleisten-Editor nicht, daß man mehrere Separatoren an das Ende der Symbolleiste anfügt. Zweitens: Wenn Sie nichts anderes nach dem Separator an das Ende der Symbolleiste anfügen, nimmt der Symbolleisten-Editor einen Fehler an und entfernt den Separator. Mit anderen Worten: Mit dem Symbolleisten-Editor von Visual C++ läßt sich kein Platzhalter für das Kombinationsfeld in die Symbolleiste einbauen.

Als nächstes sind die Textzeichenfolgen hinzuzufügen, die Sie in das Kombinationsfeld laden. Um diese Strings aufzunehmen, öffnen Sie die Zeichenfolgentabelle auf der Registerkarte Ressourcen des Arbeitsbereichs. Hier finden Sie alle Strings, die Sie als Statuszeilentext in verschiedenen Eigenschaftsdialogfeldern eingegeben haben. Diese Tabelle enthält eine Anzahl von IDs, die Werte dieser IDs und die Strings, die mit diesen IDs verbunden sind (siehe Abbildung 12.4). Die Strings, die Sie in das Kombinationsfeld eintragen möchten, müssen Sie in die Zeichenfolgentabelle schreiben. Jede Zeile in der Dropdown-Liste muß über eine eindeutige ID und einen Eintrag in der Zeichenfolgentabelle verfügen.

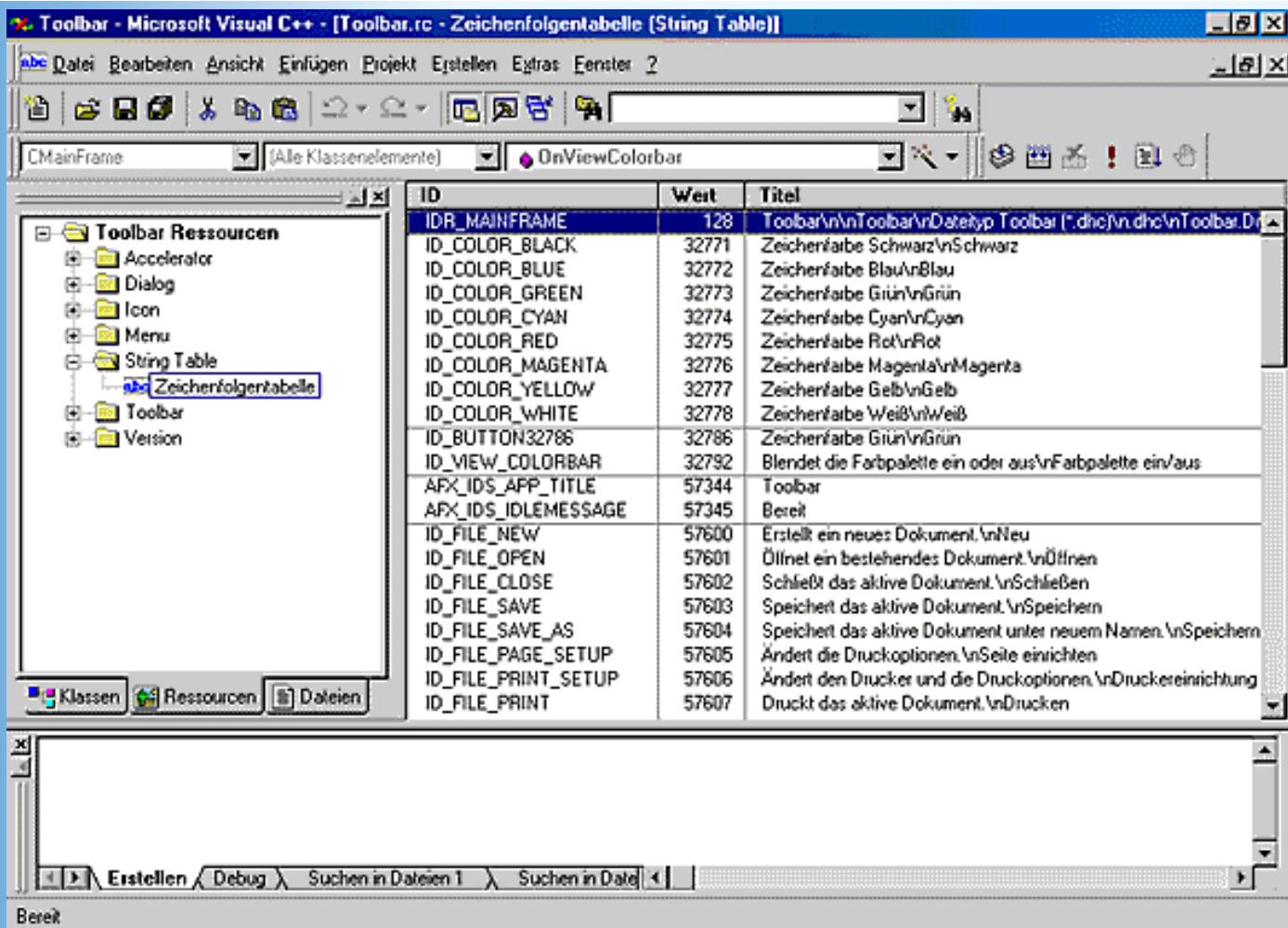


Abbildung 12.4:
Der Zeichenfolgen-Editor

Um zum Beispiel die Strings für das Kombinationsfeld hinzuzufügen, das Sie in die Symbolleiste Farben einbauen möchten, wählen Sie entweder Einfügen / Neue Zeichenfolge oder klicken mit der rechten Maustaste auf die Zeichenfolgentabelle und wählen Neue Zeichenfolge aus dem Kontextmenü.

Im Eigenschaftsdialogfeld des Strings legen Sie die ID für den String fest und geben dann den String so ein, wie er in der Dropdown-Liste erscheinen soll. Schließen Sie das Eigenschaftsdialogfeld, um den String hinzuzufügen. Für die Strings im Kombinationsfeld Breite, das Sie in die Symbolleiste Farben aufnehmen, tragen Sie die Strings gemäß Tabelle 12.6 ein.

**Tabelle 12.6: Strings für das
Kombinationsfeld Breite in der
Symbolleiste**

ID	Titel
IDS_WIDTH_VTHIN	Sehr dünn
IDS_WIDTH_THIN	Dünn
IDS_WIDTH_MEDIUM	Mittel
IDS_WIDTH_THICK	Dick
IDS_WIDTH_VTHICK	Sehr dick

Das Kombinationsfeld für die Symbolleiste erstellen

Bevor Sie das Kombinationsfeld in die Symbolleiste Farben aufnehmen können, müssen Sie eine Variable für das Kombinationsfeld erzeugen. Da sich das Kombinationsfeld nicht über die Editoren hinzufügen läßt, müssen Sie es als Variable in die Klasse `CMainFrame` aufnehmen.

Die Variable für das Kombinationsfeld nehmen Sie über die Registerkarte Klassen des Arbeitsbereichs in die Hauptrahmenklasse für die Symbolleiste Farben auf. Klicken Sie mit der rechten Maustaste auf die Klasse `CMainFrame`, und wählen Sie dann Member- Variable hinzufügen aus dem Kontextmenü. Legen Sie den Variablentyp als `CComboBox` , den Namen als `m_ctlWidth` und den Zugriff als Protected fest.

Nachdem Sie die Variable für das Kombinationsfeld in die Hauptrahmenklasse aufgenommen haben, sind verschiedene Aufgaben zu erledigen, wenn die Symbolleiste erzeugt wurde:

1. Die Breite und die ID des Platzhalters für das Kombinationsfeld in der Symbolleiste auf die Breite und die ID des Kombinationsfelds setzen.
2. Die Position des Platzhalters in der Symbolleiste ermitteln und danach Größe und Lage des Kombinationsfelds festlegen.
3. Das Kombinationsfeld erzeugen. Dabei die Symbolleiste als übergeordnetes Fenster des Kombinationsfelds spezifizieren.
4. Die Strings in die Dropdown-Liste des Kombinationsfelds laden.

Damit diese Abläufe einigermaßen übersichtlich bleiben, empfiehlt es sich, das Erstellen der Symbolleiste Farben in einer eigenen Funktion zu realisieren, die sich aus der Funktion `OnCreate` der Hauptrahmenklasse aufrufen läßt. Um diese Funktion zu erzeugen, klicken Sie im Arbeitsbereich mit der rechten Maustaste auf die Klasse `CMainFrame` und wählen Member-Funktion hinzufügen aus dem Kontextmenü. Legen Sie den Funktionstyp mit `BOOL`, die Funktionsdeklaration als `CreateColorBar` und den Zugriffsstatus als `Public` fest. In die Funktion schreiben Sie den Code von Listing 12.5.

Listing 12.5: Die Funktion `CreateColorBar` der Klasse `CMainFrame`

```
1: BOOL CMainFrame::CreateColorBar()
2: {
3:     int iTBctlID;
4:     int i;
5:
6:     if (!m_wndColorBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
7:     | CBR5_TOP
8:     | CBR5_GRIPPER | CBR5_TOOLTIPS | CBR5_FLYBY | CBR5_SIZE_DYNAMIC) ||
9:     !m_wndColorBar.LoadToolBar(IDR_TBCOLOR))
10:    {
11:        TRACE0("Symbolleiste konnte nicht erstellt werden\n");
12:        return FALSE; // Fehler bei Erstellung
13:    }
14:    iTBctlID = m_wndColorBar.CommandToIndex(ID_COLOR_BLACK);
15:    if (iTBctlID >= 0)
16:    {
17:        for (i= iTBctlID; i < (iTBctlID + 8); i++)
18:            m_wndColorBar.SetButtonStyle(i, TBBS_CHECKGROUP);
19:        // Kombinationsfeld hinzufügen
20:        int nWidth = 100;
21:        int nHeight = 125;
22:
23:        // Platzhalter des Kombinationsfelds konfigurieren
```

```

24: m_wndColorBar.SetButtonInfo(9, IDC_CBWIDTH, TBBS_SEPARATOR, nWidth);
25:
26: // Höhe der Farbpalette ermitteln
27: CRect rect;
28: m_wndColorBar.GetItemRect(9, &rect);
29: rect.bottom = rect.top + nHeight;
30:
31: // Kombinationsfeld erzeugen
32: m_ctlWidth.Create(WS_CHILD | WS_VISIBLE | WS_VSCROLL |
33: CBS_DROPDOWNLIST, rect, &m_wndColorBar, IDC_CBWIDTH);
34:
35: // Kombinationsfeld füllen
36: CString szStyle;
37: if (szStyle.LoadString(IDS_WIDTH_VTHIN))
38: m_ctlWidth.AddString((LPCTSTR)szStyle);
39: if (szStyle.LoadString(IDS_WIDTH_THIN))
40: m_ctlWidth.AddString((LPCTSTR)szStyle);
41: if (szStyle.LoadString(IDS_WIDTH_MEDIUM))
42: m_ctlWidth.AddString((LPCTSTR)szStyle);
43: if (szStyle.LoadString(IDS_WIDTH_THICK))
44: m_ctlWidth.AddString((LPCTSTR)szStyle);
45: if (szStyle.LoadString(IDS_WIDTH_VTHICK))
46: m_ctlWidth.AddString((LPCTSTR)szStyle);
47:
48: return TRUE;
49: }

```

Zeile 24 von Listing 12.5 gibt an, daß das Kombinationsfeld mit der Objekt-ID `IDC_CBWIDTH` zu erstellen ist. Diese Objekt-ID kennzeichnet das Kombinationsfeld, wenn es eine Ereignisnachricht an die Anwendung sendet oder wenn man festlegen muß, welcher Listeneintrag im Bearbeitungsfeld anzuzeigen ist. Allerdings existiert diese Objekt-ID nicht in Ihrer Anwendung. Bevor Sie die Anwendung kompilieren können, müssen Sie diese ID zu den Projektressourcen-IDs hinzufügen, wie Sie es am Tag 4 bei der Behandlung von Timern kennengelernt haben. Gehen Sie dazu im Arbeitsbereich auf die Registerkarte Ressourcen. Markieren Sie den Beginn des Ressourcenbaumes, und klicken Sie mit der rechten Maustaste, um das Kontextmenü einzublenden. Wählen Sie Ressourcensymbole aus dem Kontextmenü, und fügen Sie die Objekt-ID `IDC_CBWIDTH` hinzu. Vergewissern Sie sich, daß Sie für die neue Objekt-ID einen eindeutigen numerischen Wert vergeben, damit keine Konflikte mit anderen Objekten Ihrer Anwendung auftreten.

Den Platzhalter konfigurieren

Nachdem die Symbolleiste erstellt und die Symbolleisteneigenschaften festgelegt sind, konfigurieren Sie als nächstes den Separator, der als Platzhalter für das zu erzeugende Kombinationsfeld dient. Dazu verwenden Sie die Symbolleistenfunktion `SetButtonInfo` wie folgt:

```
m_wndColorBar.SetButtonInfo(9, IDC_CBWIDTH, TBBS_SEPARATOR, nWidth);
```

Diese Funktion übernimmt vier Argumente. Das erste gibt den aktuellen Index des Steuerelements in der Symbolleiste an - in diesem Fall das zehnte Steuerelement in der Symbolleiste (acht Farbschaltflächen und zwei Separatoren). Im zweiten Argument übergibt man die neue ID des Symbolleistensteuerelements. Dabei handelt es sich um die ID, die in die Nachrichtenwarteschlange gestellt wird, wenn ein Steuerelementereignis auftritt. Das dritte Argument ist der Typ des Symbolleistensteuerelements. Das vierte und letzte Argument ist etwas irreführend. Wenn Sie sich die Funktionsdokumentation ansehen, gibt das vierte Argument den neuen Index des Steuerelements in der Symbolleiste an. Das ist die Position, auf die das Steuerelement verschoben wird. Wenn das Steuerelement jedoch ein Separator ist, gibt dieses Argument die Breite des Steuerelements an und verschiebt es nicht an eine andere Stelle. Da es sich im vorliegenden Fall um einen Separator handelt, legt man mit diesem Argument die Breite des zu erzeugenden Kombinationsfelds fest.

Die Position des Kombinationsfelds in der Symbolleiste ermitteln

Der Separator als Platzhalter für das Kombinationsfeld in der Symbolleiste ist nun konfiguriert. Als nächstes ermitteln Sie die Position des Platzhalters auf der Symbolleiste, damit sich die Lage des Kombinationsfelds festlegen läßt:

```
m_wndColorBar.GetItemRect(9, &rect);
rect.bottom = rect.top + nHeight;
```

In der ersten Zeile steht der Aufruf der Funktion `GetItemRect`, um die Position und Größe des Platzhalters für das Kombinationsfeld zu ermitteln. Die nächste Zeile addiert die Höhe der Dropdown-Liste zur Höhe, die das Kombinationsfeld schließlich einnimmt.

Das Kombinationsfeld erstellen

Die Größe des Platzhalters ist nun genau festgelegt, und Sie verfügen über Lage und Größe für das Kombinationsfeld. Nun ist es an der Zeit, das Kombinationsfeld zu erzeugen. Dazu dient die Funktion `Create` des Kombinationsfelds:

```
m_ctlWidth.Create(WS_CHILD | WS_VISIBLE | WS_VSCROLL |
CBS_DROPDOWNLIST, rect, &m_wndColorBar, IDC_CBWIDTH);
```

Das erste Argument der Funktion `Create` für das Kombinationsfeld ist der Stil des Kombinationsfelds. Normalerweise kombiniert man verschiedene Stilattribute, um einen Stilwert zu bilden. Tabelle 12.7 listet die Attribute auf, die Sie in diesem Wert verwenden können.

Tabelle 12.7: Stile für Kombinationsfelder

Stil	Beschreibung
WS_CHILD	Kennzeichnet das Kombinationsfeld als untergeordnetes Fenster (erforderlich).
WS_VISIBLE	Macht das Kombinationsfeld sichtbar.
WS_DISABLED	Deaktiviert das Kombinationsfeld.
WS_VSCROLL	Stattet die Dropdown-Liste mit vertikalen Bildlaufleisten aus.
WS_HSCROLL	Stattet die Dropdown-Liste mit horizontalen Bildlaufleisten aus.
WS_GROUP	Gruppiert Steuerelemente.
WS_TABSTOP	Schließt das Kombinationsfeld in die Tabulator-Reihenfolge ein.
CBS_AUTOHSCROLL	Rollt den Text im Bearbeitungsfeld automatisch nach rechts, wenn der Benutzer ein Zeichen am Ende der Zeile eingibt. Damit kann der Benutzer Text eingeben, der breiter ist, als das Eingabefeld des Kombinationsfelds.
CBS_DROPDOWN	Ähnlich zu <code>CBS_SIMPLE</code> . Allerdings wird die Liste nur dann angezeigt, wenn der Benutzer das Symbol neben dem Eingabefeld wählt.
CBS_DROPDOWNLIST	Ähnlich zu <code>CBS_DROPDOWN</code> . Allerdings wird das Eingabefeld durch ein statisches Textfeld ersetzt, das den momentan in der Liste markierten Eintrag anzeigt.
CBS_HASSTRINGS	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Elemente des Listenfelds bestehen aus Strings.
CBS_OEMCONVERT	Der in das Eingabefeld eingegebene Text wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und wieder zurück nach ANSI konvertiert.
CBS_OWNERDRAWFIXED	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Einträge in der Liste haben alle die gleiche Höhe.

CBS_OWNERDRAWVARIABLE	Der Besitzer des Listenfelds ist für den Inhalt des Listenfelds verantwortlich. Die Höhe der Listeneinträge ist variabel.
CBS_SIMPLE	Das Listenfeld wird immer angezeigt.
CBS_SORT	Sortiert automatisch die Strings im Listenfeld.
CBS_DISABLENOSCROLL	Die Liste zeigt eine deaktivierte Bildlaufleiste, wenn nur soviel Einträge in der Liste vorhanden sind, daß sich ein Bildlauf erübrigt.
CBS_NOINTEGRALHEIGHT	Legt fest, daß das Kombinationsfeld genau die angegebene Größe einnimmt.

Das zweite Argument beschreibt das Rechteck, das das Kombinationsfeld einnehmen soll. Dieses Argument gibt die Position innerhalb des übergeordneten Fensters - in diesem Fall der Symbolleiste - an, wo das Kombinationsfeld untergebracht ist. Es verschiebt sich mit dem übergeordneten Fenster (der Symbolleiste) und bleibt immer auf der gleichen Relativposition.

Das dritte Argument ist ein Zeiger auf das übergeordnete Fenster. Es handelt sich hierbei um die Adresse der Variablen für die Symbolleiste Farben.

Im vierten Argument wird die Objekt-ID für das Kombinationsfeld übergeben.

Das Kombinationsfeld füllen

Als letztes beim Erstellen des Kombinationsfelds auf der Symbolleiste Farben ist noch die Dropdown-Liste mit den verfügbaren Elementen, aus denen der Benutzer auswählen kann, zu füllen. Das Ganze wird mit einer Kombination von zwei Funktionen realisiert:

```
if (szStyle.LoadString(IDS_WIDTH_VTHIN))
m_ctlwidth.AddString((LPCTSTR)szStyle);
```

Die erste Funktion ist die Funktion `LoadString` der Klasse `CString`. Diese Funktion übernimmt eine String-ID und lädt den zugehörigen String aus der Zeichenfolgentabelle. Die zweite Funktion ist die Funktion `AddString` des Kombinationsfelds, die den als Argument übergebenen String in die Dropdown-Liste hinzufügt. Indem man diese Kombination von Funktionen für alle Elemente, die in der Dropdown-Liste erscheinen sollen, aufruft, kann man das Kombinationsfeld aus der Zeichenfolgentabelle der Anwendung füllen.

Die Funktion `OnCreate` aktualisieren

Nachdem wir den gesamten Code zum Erstellen der Symbolleiste Farben in eine separate Funktion verschoben haben, können wir die Funktion `OnCreate` auf den neuesten Stand bringen, so daß sie die Funktion `CreateColorBar` aufruft, wo die Symbolleiste Farben erzeugt wird. Der entsprechende Code ist in Listing 12.6 wiedergegeben.

Listing 12.6: Die modifizierte Funktion `OnCreate` der Klasse `CMainFrame`

```
1: int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
2: {
3: if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
4: return -1;
5:
6: if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
  ÂCBRS_TOP
7: | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
8: !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
9: {
10: TRACE0("Symbolleiste konnte nicht erstellt werden\n");
11: return -1; // Fehler bei Erstellung
12: }
```

```

13:
14: //////////////////////////////////////
15: // EIGENER CODE, ANFANG
16: //////////////////////////////////////
17:
18: // Symbolleiste Farben hinzufügen
19: if (!CreateColorBar())
20: {
21: TRACE0("Symbolleiste Farben konnte nicht erstellt werden\n");
22: return -1; // // Fehler bei Erstellung
23: }
24:
25: //////////////////////////////////////
26: // EIGENER CODE, ENDE
27: //////////////////////////////////////
28:
29: if (!m_wndStatusBar.Create(this) ||
30: !m_wndStatusBar.SetIndicators(indicators,
31: sizeof(indicators)/sizeof(UINT)))
32: {
33: TRACE0("Statusleiste konnte nicht erstellt werden\n");
34: return -1; // Fehler bei Erstellung
35: }
36:
37: // ZU ERLEDIGEN: Löschen Sie diese drei Zeilen, wenn Sie nicht wollen,
38: // dass die Symbolleiste
39: // andockbar ist.
40: m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
41:
42: //////////////////////////////////////
43: // EIGENER CODE, ANFANG
44: //////////////////////////////////////
45: // Andocken für Symbolleiste Farben aktivieren
46: m_wndColorBar.EnableDocking(CBRS_ALIGN_ANY);
47:
48: //////////////////////////////////////
49: // EIGENER CODE, ENDE
50: //////////////////////////////////////
51:
52: EnableDocking(CBRS_ALIGN_ANY);
53: DockControlBar(&m_wndToolBar);
54:
55: //////////////////////////////////////
56: // EIGENER CODE, ANFANG
57: //////////////////////////////////////
58:
59: // Symbolleiste Farben andocken
60: DockControlBar(&m_wndColorBar);
61:
62: //////////////////////////////////////
63: // EIGENER CODE, ENDE
64: //////////////////////////////////////
65:
66: return 0;

```

67: }

Wenn Sie nun die Anwendung kompilieren und ausführen, sollte sich am Ende der Symbolleiste Farben ein Kombinationsfeld befinden, wie es Abbildung 12.5 zeigt. Allerdings bewirkt das Kombinationsfeld noch keine Aktionen.



Abbildung 12.5:
Die Symbolleiste Farben mit dem Kombinationsfeld Breite

Ereignisse des Kombinationsfelds der Symbolleiste behandeln

Eine Behandlungsroutine für das Kombinationsfeld läßt sich leicht erstellen, auch wenn man das manuell erledigen muß (da der Klassen-Assistent gar nicht weiß, daß das Kombinationsfeld existiert). Es ist ein Eintrag `ON_CBN_SELCHANGE` in die Nachrichtenzuordnungstabelle aufzunehmen und dann die eigentliche Behandlungsfunktion in die Klasse `CMainFrame` einzufügen.

Als erstes nehmen wir die Behandlungsfunktion auf. Markieren Sie dazu die Klasse `CMainFrame` im Arbeitsbereich, und wählen Sie Member-Funktion hinzufügen aus dem Kontextmenü. Geben Sie den Funktionstyp mit `afx_msg void`, die Funktionsdeklaration mit `OnSelChangeWidth` und den Zugriff mit `Protected` an. Den Code für die Funktion übernehmen Sie aus dem ersten Teil von Listing 12.7. Um die Breite im Dokument zu setzen, wird in `OnSelChangeWidth` die Funktion `SetWidth` aufgerufen.

Diese Funktion müssen Sie noch in die Klasse `CToolBarDoc` einfügen. Markieren Sie im Arbeitsbereich auf der Registerkarte Klassen die Klasse `CToolBarDoc`, und wählen Sie Member-Funktion hinzufügen aus dem Kontextmenü. Legen Sie als Funktionstyp `void`, als Deklaration `SetWidth(UINT nWidth)` und als Zugriff `Public` fest. In die Funktion schreiben Sie den zweiten Teil von Listing 12.7.

Listing 12.7: Die Funktionen `OnSelChangeWidth` und `SetWidth`

```
1: void CMainFrame::OnSelChangeWidth()
2: {
3: // Neue Auswahl des Kombinationsfelds ermitteln
4: int nIndex = m_ctlWidth.GetCurSel();
5: if (nIndex == CB_ERR)
6: return;
7:
8: // Aktives Dokument ermitteln
9: CToolBarDoc* pDoc = (CToolbarDoc*)GetActiveDocument();
10: // Handelt es sich um ein gültiges Dokument?
11: if (pDoc)
12: // Neue Zeichenbreite setzen
13: pDoc->SetWidth(nIndex);
14:
15: }
16:
17: void CToolBarDoc::SetWidth(UINT nWidth)
18: {
19: // Breite setzen
20: m_nWidth = nWidth;
21: }
```

Die Funktion `OnSelChangeWidth` ermittelt zuerst die aktuelle Auswahl des Kombinationsfelds. Da wir die Einträge in der richtigen Reihenfolge eingetragen und beim Erstellen des Kombinationsfelds das Flag `CBS_SORT` (zum Sortieren der Einträge) nicht angegeben haben, entspricht der Index den Breiten im Dokument. Daher kann man einen Zeiger auf die aktuelle Instanz des Dokuments mit der Funktion `GetActiveDocument` ermitteln

und dann die neue Breite an das Dokument mit der Funktion `SetWidth` übergeben.

Um die Behandlungsfunktion aufzurufen, wenn der Benutzer die Auswahl im Kombinationsfeld ändert, müssen Sie den entsprechenden Eintrag in die Nachrichtenzuordnungstabelle von `CMainFrame` einfügen. Gehen Sie im Quellcode von `CMainFrame` nach oben, bis Sie den Abschnitt der Nachrichtenzuordnungstabelle finden, und fügen Sie hier Zeile 12 von Listing 12.8 ein.

Listing 12.8: Die modifizierte Nachrichtenzuordnungstabelle von `CMainFrame`

```

1:
////////////////////////////////////
2: // CMainFrame
3:
4: IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
5:
6: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
7: //{{AFX_MSG_MAP(CMainFrame)
8: ON_WM_CREATE()
9: ON_COMMAND(ID_VIEW_COLORBAR, OnViewColorbar)
10: ON_UPDATE_COMMAND_UI(ID_VIEW_COLORBAR, OnUpdateViewColorbar)
11: //}}AFX_MSG_MAP
12: ON_CBN_SELCHANGE(IDC_CBWIDTH, OnSelChangeWidth)
13: END_MESSAGE_MAP()

```

Der Eintrag

```
ON_CBN_SELCHANGE(IDC_CBWIDTH, OnSelChangeWidth)
```

in der Nachrichtenzuordnungstabelle legt fest, daß die Funktion `OnSelChangeWidth` aufzurufen ist, wenn Ereignisse mit der Objekt-ID des Kombinationsfelds in der Symbolleiste Farben aufgrund von Änderungen der Auswahl im Kombinationsfeld auftreten. Wenn Sie nun die Anwendung kompilieren und ausführen, sollten Sie die Stiftbreite zum Zeichnen über das Kombinationsfeld der Symbolleiste Farben ändern können.

Das Kombinationsfeld der Symbolleiste aktualisieren

Für das Kombinationsfeld ist noch ein Problem zu lösen. Falls der Benutzer einen neuen Wert aus dem Menü und nicht über das Kombinationsfeld auswählt, soll das Kombinationsfeld ebenfalls den neuen Wert widerspiegeln. Die aktuelle Auswahl im Kombinationsfeld läßt sich am effizientesten setzen, wenn das beim Auslösen irgendeines Menübefehls geschieht. Dazu ist eine Funktion in der Hauptrahmenklasse erforderlich, die sich von der Dokumentklasse aufrufen läßt. Die Funktion im Hauptrahmen muß lediglich die aktuelle Auswahl im Kombinationsfeld setzen.

Um diese Funktion im Hauptrahmen zu implementieren, fügen Sie eine neue Member-Funktion in die Klasse `CMainFrame` ein und legen den Funktionstyp mit `void`, die Deklaration mit `UpdateWidthCB(int nIndex)` und den Zugriff mit `Public` fest. In die Funktion schreiben Sie den Code aus Listing 12.9.

Listing 12.9: Die Funktion `UpdateWidthCB` der Klasse `CMainFrame`

```

1: void CMainFrame::UpdateWidthCB(int nIndex)
2: {
3: // Die neue Auswahl im Kombinationsfeld setzen
4: m_wndColorBar.m_ctlWidth.SetCurSel(nIndex);
5: }

```

Die Funktion `UpdateWidthCB` ruft die Kombinationsfeldfunktion `SetCurSel` auf, die die aktuelle Auswahl in der Dropdown-Liste des Kombinationsfelds auf den Eintrag mit dem angegebenen Index setzt. Das Eingabefeld des Kombinationsfelds wird mit dem neu ausgewählten Listeneintrag aktualisiert. Wenn man dem Kombinationsfeld einen Index anbietet, der nicht in der Dropdown-Liste existiert, liefert die Funktion einen Fehler zurück.

Auf der Seite des Dokuments muß man die Funktion im Hauptrahmen immer dann aufrufen, wenn die

entsprechenden Behandlungsfunktionen für das Menü aufgerufen werden. Da das in verschiedenen Funktionen vorkommen kann, ist es sinnvoll, die erforderliche Funktionalität in einer einzigen Funktion unterzubringen. Diese Funktion benötigt einen Zeiger auf die mit dem Dokument verbundene Ansicht und ermittelt dann - über die Ansicht - einen Zeiger auf den Rahmen. Diesen Zeiger kann man dann im Aufruf der Funktion `UpdateWidthCB`, die wir eben der Hauptrahmenklasse hinzugefügt haben, verwenden.

Um die Funktion in die Anwendung aufzunehmen, markieren Sie die Klasse `CToolBarDoc` im Arbeitsbereich und wählen Member-Funktion hinzufügen aus dem Kontextmenü. Legen Sie als Funktionstyp `void`, als Funktionsdeklaration `UpdateColorbar(int nIndex)` und als Zugriffsstatus Privat fest. In die Funktion übernehmen Sie den Code aus Listing 12.10.

Listing 12.10: Die Funktion `UpdateColorbar` der Klasse `CToolBarDoc`

```
1: void CToolBarDoc::UpdateColorbar(int nIndex)
2: {
3: // Position der ersten Ansicht ermitteln
4: POSITION pos = GetFirstViewPosition();
5: // Ist die Position gültig?
6: if (pos != NULL)
7: {
8: // Zeiger auf Ansicht in dieser Position holen
9: CView* pView = GetNextView(pos);
10: // Ist Zeiger auf Ansicht gültig?
11: if (pView)
12: {
13: // Zeiger auf Rahmen über Ansicht holen
14: CMainFrame* pFrame = (CMainFrame*)pView->GetTopLevelFrame();
15: // Zeiger auf Rahmen erhalten?
16: if (pFrame)
17: // Kombinationsfeld in Symbolleiste Farben
18: // über den Rahmen aktualisieren
19: pFrame->UpdateWidthCB(nIndex);
20: }
21: }
22: }
```

Diese Funktion folgt dem Pfad, um den Anwendungsrahmen aus der Dokumentklasse zu erhalten. Da in einem Dokument mehrere Ansichten gleichzeitig geöffnet sein können, ermitteln Sie mit der Funktion `GetFirstViewPosition` die Position der ersten Ansicht, die mit dem Dokument verbunden ist.

Die nächste Funktion, `GetNextView`, gibt einen Zeiger auf die Ansicht zurück, die durch die Position spezifiziert wird. Diese Funktion aktualisiert ebenfalls die Positionsvariable mit dem Zeiger auf die nächste Ansicht in der Liste der Ansichten, die mit dem aktuellen Dokument verbunden sind.

Nachdem Sie über einen Zeiger auf die Ansicht verfügen, können Sie die Fensterfunktion `GetTopLevelFrame` aufrufen, die einen Zeiger auf das Rahmenfenster der Anwendung zurückgibt. Diese Funktion müssen Sie über die Ansicht aufrufen, da das Dokument nicht von der Klasse `CWnd` abgeleitet ist, während das bei der Ansicht der Fall ist.

Nunmehr besitzen Sie einen Zeiger auf das Rahmenfenster und können diesen Zeiger verwenden, um die weiter vorn erzeugte Funktion aufzurufen und das Kombinationsfeld in der Symbolleiste zu aktualisieren. Wenn Sie jetzt diese neue Funktion aus der Behandlungsroutine des Menübefehls Breite aufrufen, wie es in Listing 12.11 geschieht, wird das Kombinationsfeld in der Symbolleiste Farben automatisch aktualisiert und spiegelt die momentan ausgewählte Stiftbreite zum Zeichnen wider, unabhängig davon, ob die Auswahl über das Menü oder das Kombinationsfeld erfolgt ist.

Listing 12.11: Eine aktualisierte Behandlungsroutine für den Menübefehl Breite

```
1: void CToolBarDoc::OnWidthVthin()
```

```

2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4: // Neue Breite setzen
5: m_nWidth = 0;
6: // Kombinationsfeld in der Symbolleiste Farben aktualisieren
7: UpdateColorbar(0);
8: }

```

Ein neues Element für die Statusleiste

Weiter oben in der heutigen Lektion haben Sie gelernt, wie man Meldungen in der Statusleiste und QuickInfos für Symbolleistenschaltflächen und Menübefehle spezifiziert. Wie verhält es sich nun, wenn Sie dem Benutzer in der Statusleiste weitergehende Informationen anbieten möchten? Wie läßt sich analog zu Visual Studio für Visual C++ erreichen, daß man Informationen über die momentan ausgeführten Aktionen, die gerade im Dokument bearbeitete Stelle oder den aktuellen Zustand der Anwendung anzeigen kann? Diese Angaben gehen über die Anzeigen der Feststelltasten hinaus, die Visual C++ automatisch in der Statusleiste meldet.

Es ist tatsächlich einfach, zusätzliche Ausschnitte in die Statusleiste aufzunehmen oder auch bereits vorhandene Ausschnitte zu entfernen. Um sich mit diesen Vorgängen bekannt zu machen, nehmen Sie einen neuen Ausschnitt in die Statusleiste Ihrer Zeichenanwendung auf, der die momentan gültige Farbe meldet.

Ein neuer Ausschnitt für die Statusleiste

Bevor Sie einen neuen Ausschnitt in die Statusleiste einbinden, müssen Sie einen neuen Eintrag in der Zeichenfolgentabelle der Anwendung für den Statusleisten-Ausschnitt erstellen. Der Eintrag in der Zeichenfolgentabelle realisiert für den Statusleisten-Ausschnitt zwei Aufgaben. Erstens stellt er die Objekt-ID für den Statusleisten-Ausschnitt bereit. Diese ID verwenden Sie, um den Ausschnitt in der Statusleiste zu aktualisieren, wenn ein neuer Text in diesem Feld auszugeben ist. Die zweite Funktion des Eintrags in der Zeichenfolgentabelle bezieht sich auf die Größe des Ausschnitts. Um die korrekte Größe des Ausschnitts festzulegen, müssen Sie einen Titel für den Eintrag in der Zeichenfolgentabelle bereitstellen, der mindestens so lang wie der längste String ist, den Sie im Statusleisten-Ausschnitt anzeigen möchten.

In die Zeichenfolgentabelle fügen Sie einen neuen Eintrag mit den gleichen Schritten hinzu, die Sie weiter vorn beim Text für das Kombinationsfeld in der Symbolleiste Farben ausgeführt haben. Legen Sie die String-ID mit `ID_INDICATOR_COLOR` und den Titel mit `MAGENTA` (der längste String, der im Ausschnitt der Statusleiste anzuzeigen ist) fest.

Ein kleiner Teil im ersten Abschnitt des Quellcodes für den Hauptrahmen definiert das Layout der Statusleiste. Diese kleine Tabelle enthält die Objekt-IDs der Statusleisten-Ausschnitte als Tabellenelemente, und zwar in der Reihenfolge, in der sie von links nach rechts in der Statusleiste erscheinen.

Um den Ausschnitt für die Farbe in die Statusleiste aufzunehmen, fügen Sie die ID des Farbausschnittes in die Tabellendefinition für die Statusleistenanzeige unmittelbar im Anschluß an die Nachrichtenzuordnungstabelle in der Quellcodedatei für den Hauptrahmen ein. Schreiben Sie die ID für den Farbausschnitt in der Tabellendefinition an die Stelle, die der Ausschnitt auf der Statusleiste einnehmen soll, wie es aus Zeile 18 von Listing 12.12 hervorgeht.

Listing 12.12: Eine modifizierte Tabellendefinition für Statusleistenanzeigen

```

1:
2: //////////////////////////////////////
3: // CMainFrame
4: IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
5:
6: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
7: //{{AFX_MSG_MAP(CMainFrame)

```



```

8: ON_WM_CREATE()
9: ON_COMMAND(ID_VIEW_COLORBAR, OnViewColorbar)
10: ON_UPDATE_COMMAND_UI(ID_VIEW_COLORBAR, OnUpdateViewColorbar)
11: //}}AFX_MSG_MAP
12: ON_CBN_SELCHANGE(IDC_CBWIDTH, OnSelChangeWidth)
13: END_MESSAGE_MAP()
14:
15: static UINT indicators[] =
16: {
17: ID_SEPARATOR, // Statusleistanzeige
18: ID_INDICATOR_COLOR,
19: ID_INDICATOR_CAPS,
20: ID_INDICATOR_NUM,
21: ID_INDICATOR_SCRL,
22: };
23:
24:
////////////////////////////////////
25: // CMainFrame Konstruktion/Zerstörung

```

Falls Sie eine der Anzeigen für die Feststelltasten aus der Statusleiste entfernen möchten, löschen Sie einfach den entsprechenden Eintrag in der Tabellendefinition. Weiter unten in der Funktion `OnCreate`, die die Statusleiste (unmittelbar nach den Symbolleisten) erstellt, sehen Sie auch, wo diese Tabelle zum Einsatz kommt, um die Statusleiste mit dem folgenden Code zu erzeugen:

```

if (!m_wndStatusBar.Create(this) ||
!m_wndStatusBar.SetIndicators(indicators,
sizeof(indicators)/sizeof(UINT)))

```

Nachdem die Statusleiste erzeugt wurde, folgt ein Aufruf der Funktion `SetIndicators` auf der Statusleiste, um die Ausschnitte gemäß der Definition in der Tabelle der Statusleistenanzeigen in die Statusleiste aufzunehmen. Die mit den IDs in der Tabelle der Statusleistenanzeigen verbundenen Strings dienen dazu, die Ausschnitte zu initialisieren und deren Größe festzulegen. Wenn Sie jetzt die Anwendung kompilieren und ausführen, sehen Sie den neuen Farbausschnitt in der Statusleiste. Der Ausschnitt zeigt den Titel aus der Zeichenfolgentabelle an.

Text für einen Statusleisten-Ausschnitt festlegen

Nachdem Sie den Ausschnitt in die Statusleiste aufgenommen haben, können Sie über das Ereignis `UPDATE_COMMAND_UI` alle Aktualisierungen des Ausschnitts erledigen lassen. Dazu müssen Sie lediglich eine Behandlungsroutine für das Ereignis auf der Objekt-ID des Ausschnitts hinzufügen und dieses Ereignis verwenden, um den Text im Ausschnitt festzulegen. Da die Statusleiste immer sichtbar ist, wird das Ereignis `UPDATE_COMMAND_UI` für die Ausschnitte in der Statusleiste jedesmal ausgelöst, wenn die Anwendung im Leerlauf arbeitet. Das bedeutet, daß das Ereignis auftritt, nachdem die Anwendung die Abarbeitung aller Tastenbetätigungen und Mausbewegungen abgeschlossen hat. In etwa einer Woche, am Tag 18, lernen Sie bei der Behandlung des Multitasking, wie häufig und wann Tasks, die im Leerlauf einer Anwendung arbeiten, ausgelöst werden.

In der Behandlungsroutine müssen Sie einen String erzeugen, der den Namen der aktuellen Farbe enthält (oder was auch immer für ein Text im Ausschnitt der Statusleiste erscheinen soll). Als nächstes ist sicherzustellen, daß der Ausschnitt aktiviert ist. Schließlich müssen Sie den Text des Ausschnitts auf den erzeugten String setzen.

Um das Ganze in Ihrer Anwendung zu implementieren, erzeugen Sie eine Behandlungsroutine für die Nachricht `UPDATE_COMMAND_UI`. Auch in diesem Fall ist der Klassen-Assistent nicht über den Ausschnitt in der Statusleiste informiert, so daß Sie die Behandlungsroutine manuell erzeugen und in die Nachrichtenzuordnungstabelle eintragen müssen. Um die Behandlungsroutine zu erstellen, fügen Sie der Dokumentklasse (`CToolBarDoc`) eine neue Member-Funktion mit dem Typ `afx_msg void`, der Deklaration `OnUpdateIndicatorColor (CCommandUI * pCmdUI)` und dem Zugriffsstatus `Protected` hinzu. In die

Funktion nehmen Sie den Code von Listing 12.13 auf.

Listing 12.13: Die Funktion OnUpdateIndicatorColor

```

1: void CToolBarDoc::OnUpdateIndicatorColor(CCmdUI *pCmdUI)
2: {
3:   CString strColor;
4:
5:   // Wie heit die aktuelle Farbe?
6:   switch (m_nColor)
7:   {
8:   case 0: // Schwarz
9:     strColor = "SCHWARZ";
10:    break;
11:   case 1: // Blau
12:     strColor = "BLAU";
13:    break;
14:   case 2: // Grn
15:     strColor = "GRN";
16:    break;
17:   case 3: // Cyan
18:     strColor = "CYAN";
19:    break;
20:   case 4: // Rot
21:     strColor = "ROT";
22:    break;
23:   case 5: // Magenta
24:     strColor = "MAGENTA";
25:    break;
26:   case 6: // Gelb
27:     strColor = "GELB";
28:    break;
29:   case 7: // Wei
30:     strColor = "WEISS";
31:    break;
32:   }
33:   // Statusleisten-Ausschnitt aktivieren
34:   pCmdUI->Enable(TRUE);
35:   // Text im Ausschnitt der Statusleiste
36:   // auf die aktuelle Farbe setzen
37:   pCmdUI->SetText(strColor);
38: }

```

Diese Funktion befolgt genau drei Schritte: sie erzeugt einen String mit dem Namen der aktuellen Farbe, stellt sicher, da der Ausschnitt in der Statusleiste aktiviert ist, und setzt den Text im Feld auf den erzeugten String.

Um nun sicherzustellen, da der Aufruf der neuen Behandlungsroutine wie erwartet erfolgt, mssen Sie einen ON_UPDATE_COMMAND_UI-Eintrag in die Nachrichtenzuordnungstabelle am Beginn des Quellcodes der Dokumentdatei aufnehmen, wie es Listing 12.14 zeigt.

Listing 12.14: Die modifizierte Nachrichtenzuordnungstabelle von CToolBarDoc

```

1:
2: //////////////////////////////////////
3: // CToolBarDoc
4: IMPLEMENT_DYNCREATE(CToolBarDoc, CDocument)
5:

```

```

6: BEGIN_MESSAGE_MAP(CToolbarDoc, CDocument)
7: ON_UPDATE_COMMAND_UI(ID_INDICATOR_COLOR, OnUpdateIndicatorColor)
8: //{AFX_MSG_MAP(CToolbarDoc)
9: ON_UPDATE_COMMAND_UI(ID_WIDTH_VTHIN, OnUpdateWidthVthin)
10: .
11: .
12: ON_COMMAND(ID_WIDTH_VTHIN, OnWidthVthin)
13: //}}AFX_MSG_MAP
14: END_MESSAGE_MAP()

```

Nachdem Sie die Behandlungsroutine und den Eintrag in die Nachrichtenzuordnungstabelle hinzugefügt haben, können Sie nun Ihre Anwendung kompilieren und ausführen. In der Statusleiste sollte der Farbenausschnitt automatisch aktualisiert werden, um die momentane Zeichenfarbe anzugeben, wie es Abbildung 12.6 zeigt.

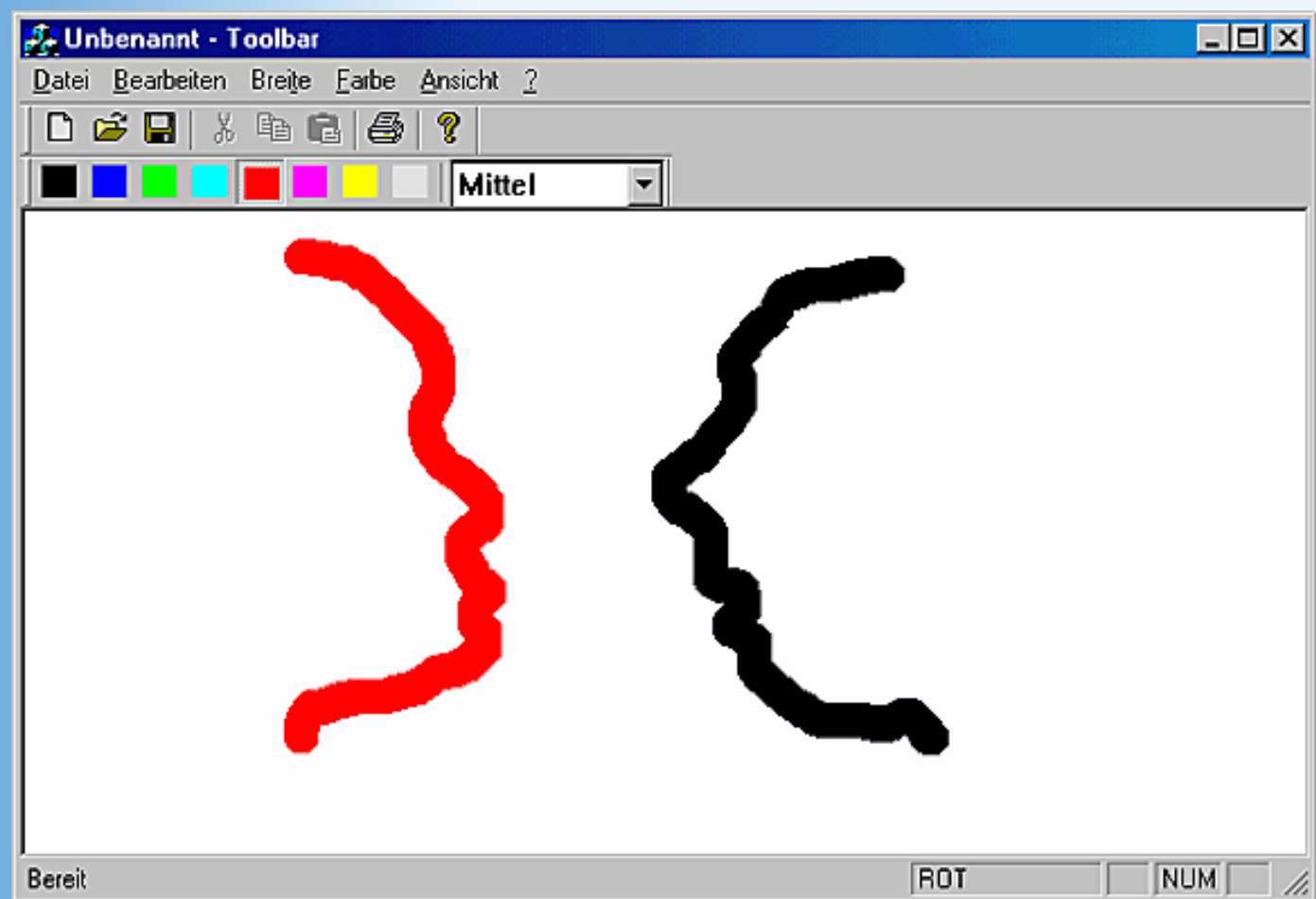


Abbildung 12.6:
Die Zeichenanwendung zeigt die aktuelle Farbe in der Statusleiste an.

Zusammenfassung

Der heutige Lehrstoff war recht umfangreich. (Zeichnet sich hier ein Trend ab?) Sie haben gelernt, wie man eigene Symbolleisten entwirft und erstellt. In diesem Zusammenhang haben Sie auch erfahren, wie man Aufforderungen in der Statusleiste für die Symbolleistenschaltflächen und Menüs festlegt sowie QuickInfos anzeigt, wenn man die Maus ein paar Sekunden über einer Symbolleistenschaltfläche ruhen läßt. Sie wissen nun, wie man diese Symbolleisten erstellt und sie mit dem Anwendungsgerüst verbindet. Weiterhin haben Sie gelernt, wie man über einen Menübefehl steuert, ob die Symbolleiste sichtbar ist.

Als nächstes sind wir darauf eingegangen, wie man ein Kombinationsfeld in einer Symbolleiste unterbringt, damit man dem Benutzer der Anwendung die gleiche Bequemlichkeit bieten kann, die er von anderen bekannten Softwarepaketen her gewohnt ist. Gleichzeitig haben Sie erfahren, wie man ein Kombinationsfeld per Code erzeugt, ohne daß man mit dem Dialog-Editor arbeiten muß, und wie man die Dropdown-Liste des

Kombinationsfelds mit Texteinträgen füllt. Dann wurde gezeigt, wie man das Kombinationsfeld in die Anwendung einbindet, indem man Behandlungsroutinen für die Ereignisse des Kombinationsfelds erstellt, und wie man das Kombinationsfeld aktualisiert, um die am Anwendungsmenü vorgenommenen Änderungen widerzuspiegeln.

Schließlich hat der heutige Tag gezeigt, wie man eigene Ausschnitte in die Statusleiste aufnimmt und wie man den Ausschnitt aktualisiert, um den momentanen Status der Anwendung wiederzugeben.

Fragen und Antworten

Frage:

In manchen Anwendungen hat man bei Symbolleisten die Option, Text anzuzeigen, beispielsweise im Internet Explorer. Wie läßt sich Text für meine Symbolleistenschaltflächen hinzufügen?

Antwort:

Leider bietet der Symbolleisten-Editor keine Möglichkeit, um Symbolleistenschaltflächen einen Text zuzuordnen. Das bedeutet, daß man den Text per Anwendungscode in die Schaltflächen aufnehmen muß, genauso wie man für alle Schaltflächen der Symbolleiste Farbe das Verhalten als Optionsfelder realisiert. Mit der Funktion `SetButtonText` legen Sie den Text auf jeder Symbolleistenschaltfläche individuell fest. Diese Funktion übernimmt zwei Argumente: das erste ist der Index der Schaltfläche, das zweite enthält den Text für die Schaltfläche. Wenn Sie wirklich Text auf den Symbolleistenschaltflächen unterbringen möchten, müssen Sie außerdem die Symbolleiste in der Größe ändern, um Platz für den anzuzeigenden Text zu schaffen.

Frage:

Ich habe im Symbolleisten-Editor einige Änderungen an der Symbolleiste Farben vorgenommen und erhalte nun jedesmal einen Assertion-Fehler, wenn ich die Anwendung auszuführen versuche. Was ist passiert?

Antwort:

Das Problem besteht darin, daß der Symbolleisten-Editor die Separatoren gefunden hat, die Sie in die Ressourcendatei als Platzhalter für das Kombinationsfeld aufgenommen haben. Der Symbolleisten-Editor nimmt an, daß es sich hier um Fehler handelt, und entfernt die Platzhalter automatisch. Der Fehler, den Sie erhalten, tritt deshalb auf, weil Sie versuchen, mit einem Steuerelement in der Symbolleiste Farben zu arbeiten, das nicht existiert. Um das Problem zu umgehen, öffnen Sie die Ressourcendatei erneut im Editor und fügen die beiden Separatoren wieder am Ende der Definition der Farben-Symbolleiste an. Laden Sie dann das Projekt in Visual C++, und kompilieren Sie die Anwendung neu.

Frage:

Das Kombinationsfeld in meinen Symbolleisten sieht zu groß aus. Wie kann ich erreichen, daß es sich harmonischer in die Symbolleiste einfügt?

Antwort:

Um das Kombinationsfeld in die Symbolleiste einzupassen, wie es zum Beispiel im Visual Studio von Visual C++ zu sehen ist, sind eine Reihe von Dingen zu erledigen. Zuerst verringern Sie den oberen Teil des Kombinationsfelds um 3. Damit entsteht ein schmaler Rand zwischen dem oberen Rand des Kombinationsfelds und dem Rand der Symbolleiste. Als nächstes wählen Sie für das Kombinationsfeld eine kleinere Schrift, die sich besser in die Symbolleiste einfügt. Experimentieren Sie etwas mit Schriften und Schriftabständen, bis Sie das Optimum für das Kombinationsfeld in der Symbolleiste gefunden haben.

Frage:

Wie kann ich den Text im ersten Abschnitt der Statusleiste festlegen, ohne Menü- und Symbolleistenaufforderungen zu verwenden?

Antwort:

Mit `SetWindowText` können Sie den Text im ersten Ausschnitt der Statusleiste festlegen. Per Voreinstellung ist der erste Ausschnitt in der Statusleiste ein Separator, der sich automatisch erweitert und dann die Breite der Statusleiste abzüglich der anderen nach rechts ausgerichteten Ausschnitte in der Leiste einnimmt. Ruft man die Funktion `SetWindowText` auf der Statusleistenvariablen auf, wird nur der Text im ersten Ausschnitt festgelegt.

Wenn Sie den Text in einem der anderen Ausschnitte ändern möchten und das zu anderen Zeiten als in der Behandlungsroutine der Nachricht `ON_UPDATE_COMMAND_UI`, können Sie mit der Funktion `SetPaneText` arbeiten. Es gibt zwei Möglichkeiten, um den Text im Hauptteil der Statusleiste zu setzen. Die erste sieht folgendermaßen aus:

```
CString myString = "Das ist mein Statusleistertext"
m_wndStatusBar.SetWindowText(myString);
```

Die zweite Methode lautet:

```
CString* myString = " Das ist mein Statusleistertext "
m_wndStatusBar.SetPaneText(0, myString);
```

Workshop

Kontrollfragen

1. Wie verbinden Sie eine Symbolleistenschaltfläche mit einem Menübefehl, der dieselbe Funktion auslöst?
2. Wie kann man sicherstellen, daß eine Symbolleiste mit dem Rahmenfenster verankert ist?
3. Wie läßt sich die Statusanzeige für die numerische Feststelltaste aus der Statusleiste entfernen?
4. Warum muß man die Ressourcendatei bearbeiten, um ein Kombinationsfeld in eine Symbolleiste aufzunehmen?

Übungen

1. Fügen Sie einen weiteren Ausschnitt in die Statusleiste ein, um die momentan ausgewählte Breite anzuzeigen.
2. Nehmen Sie in die Hauptsymbolleiste eine Schaltfläche auf, mit der man die Symbolleiste **Farben** ein-/ausschalten kann, wie es Abbildung 12.7 zeigt.

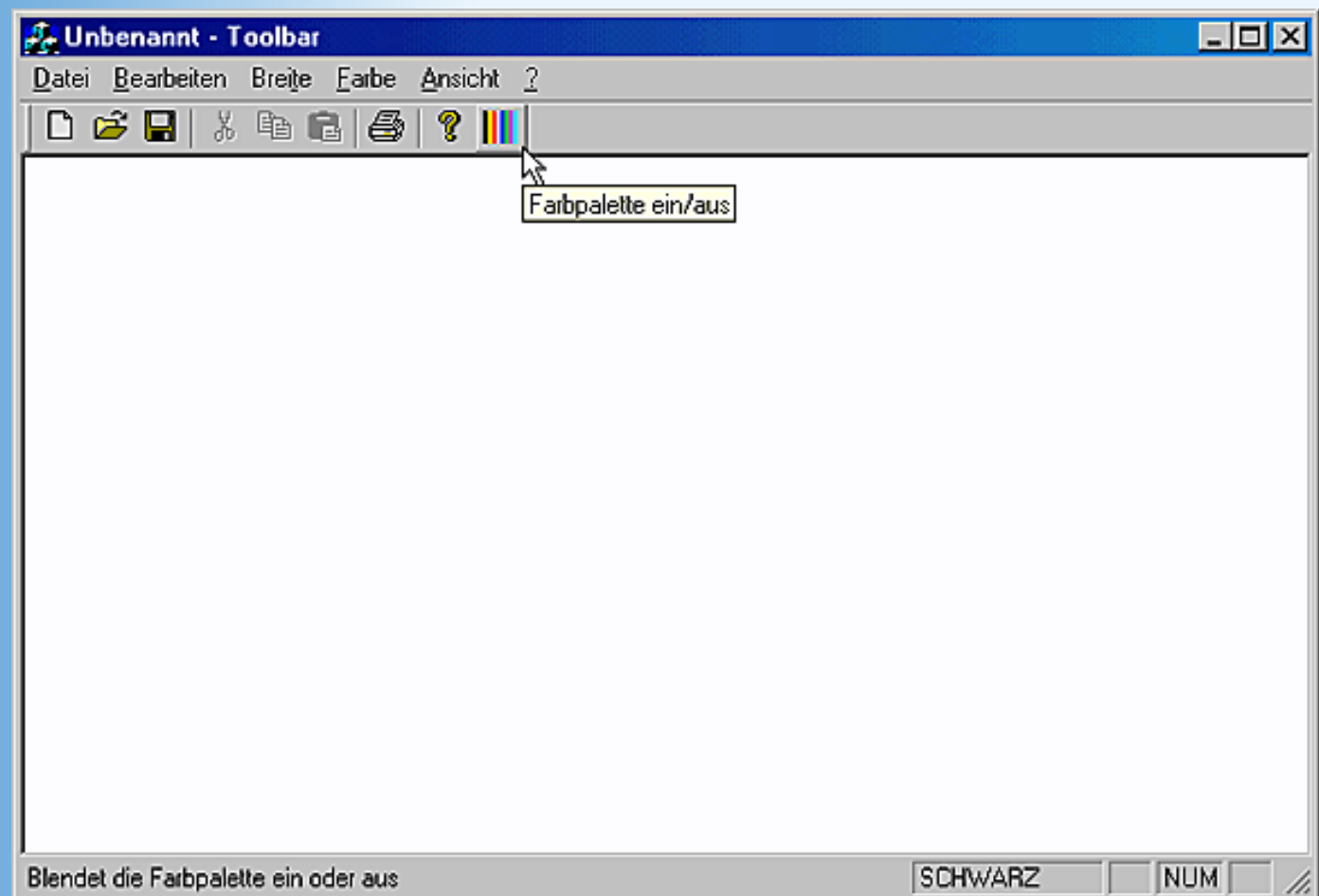


Abbildung 12.7:

Der Umschalter für die Symbolleiste Farben

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2

Tag 13

Dateizugriff

Die meisten Anwendungen bieten dem Benutzer die Möglichkeit, die eigene Arbeit zu speichern. Dabei kann es sich um ein Dokument einer Textverarbeitung, ein Tabellenblatt, eine Zeichnung oder eine Menge von Datensätzen handeln. In der heutigen Lektion untersuchen wir, welche Mittel Visual C++ bereitstellt, um diese Funktionalität in einfacher Weise zu implementieren.

Heute geht es darum, wie man ...

- C++-Streams in Visual C++ einsetzt, um Informationen zur Anwendung zu speichern,
- die Anwendungsdaten in Binärdateien speichert,
- die Objekte einer Anwendung »serialisierbar« macht,
- Variablen unterschiedlicher Datentypen in ein und derselben Datei ablegt.

Serialisierung

Die sogenannte Serialisierung zerfällt in zwei Teile. Wenn Anwendungsdaten auf dem Systemlaufwerk in Form einer Datei gespeichert werden, spricht man von Serialisierung. Wird der Anwendungszustand aus der Datei wiederhergestellt, bezeichnet man diesen Vorgang als Deserialisierung. Die Kombination der beiden Teile ergibt die Serialisierung von Anwendungsobjekten in Visual C++.

Die Klassen CArchive und CFile

Die Serialisierung in Visual C++-Anwendungen läuft über die Klasse CArchive. Diese Klasse agiert als Eingabe-/Ausgabe- (E/A) Stream für ein CFile-Objekt, wie es Abbildung 13.1 zeigt. Die C++-Streams gewährleisten einen effizienten Datenfluß in und aus einer Datei, die als Speicher der Anwendungsdaten dient. Die Klasse CArchive kann nicht ohne ein Objekt der Klasse CFile existieren, an die sie gebunden ist.

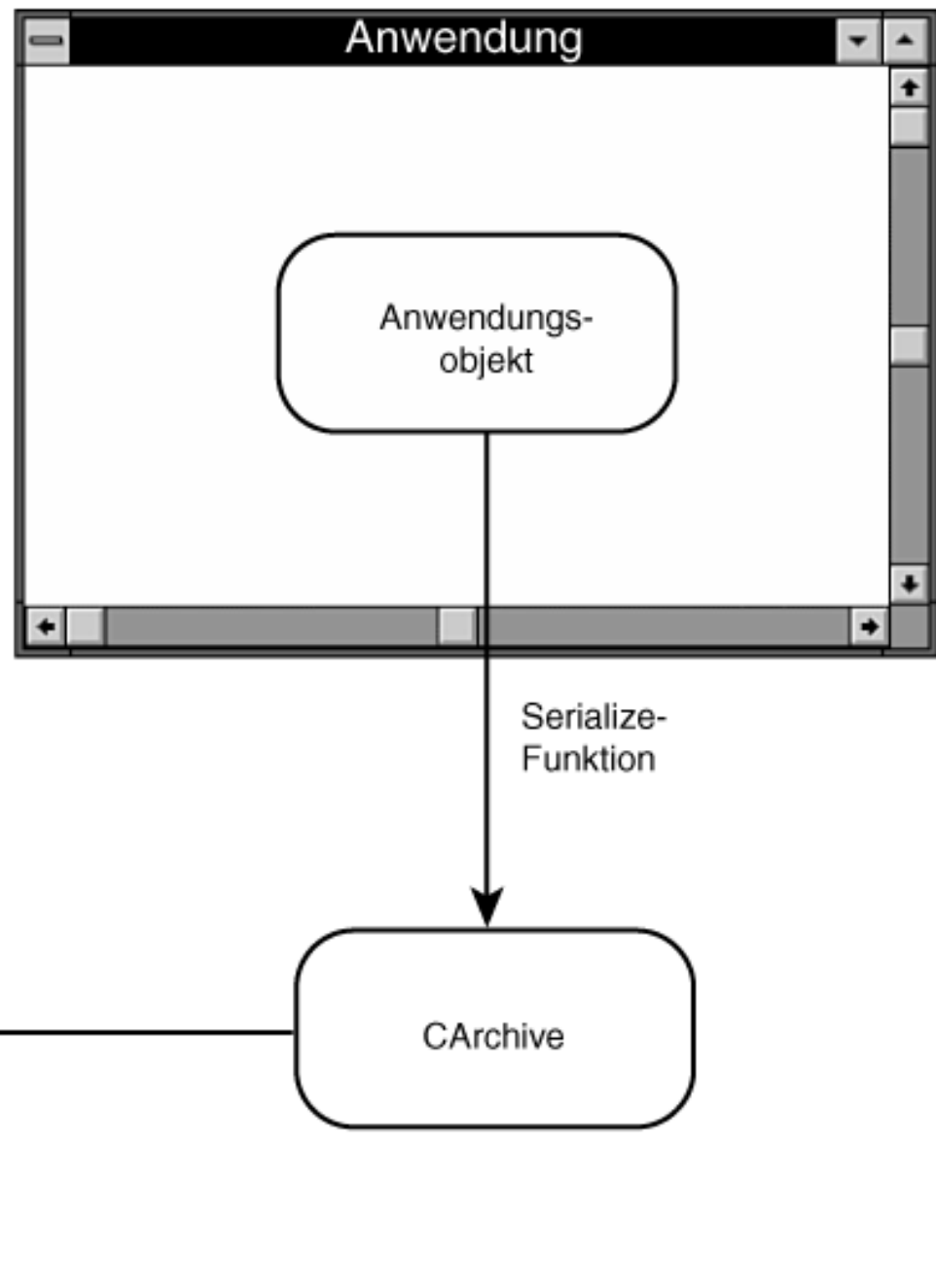


Abbildung 13.1:
Die Klasse CArchive speichert Anwendungsdaten in einem CFile-Objekt.

Die Klasse CArchive kann Daten in einer Vielzahl verschiedener Dateitypen speichern, die alle von der Klasse CFile abgeleitet sind. In der Standardeinstellung bindet der Anwendungs-Assistent die gesamte Funktionalität ein, um normale CFile-Objekte für den Einsatz mit CArchive zu erzeugen und zu öffnen. Wenn Sie mit einem der anderen Dateitypen arbeiten wollen, müssen Sie zusätzlichen Code in Ihre Anwendung einbinden, um die Verwendung abweichender Dateitypen zu ermöglichen.

Die Funktion Serialize

Die Klasse CArchive kommt in der Funktion Serialize auf den Dokument- und Datenobjekten in Visual C++-Anwendungen zum Einsatz. Wenn eine Anwendung eine Datei liest oder schreibt, ruft sie die Funktion Serialize des Dokumentobjekts auf und übergibt das CArchive-Objekt, das das Schreiben in oder Lesen aus der Datei abwickelt. In der Funktion Serialize ist zunächst zu bestimmen, ob das Archiv zu schreiben oder zu lesen ist. Dazu ruft man die Funktionen IsStoring oder IsLoading der Klasse CArchive auf. Aus dem Rückgabewert dieser Funktionen lässt sich ablesen, ob die Anwendung in/aus E/A-Streams der

Klasse `CArchive` schreiben bzw. lesen muß. Eine typische `Serialize`-Funktion in der Ansichtsklasse ist in Listing 13.1 wiedergegeben.

Listing 13.1: Eine typische `Serialize`-Funktion

```

1: void CAppDoc::Serialize(CArchive& ar)
2: {
3: // Ist in das Archiv zu schreiben?
4: if (ar.IsStoring())
5: {
6: // Ja, Variable schreiben
7: ar << m_MyVar;
8: }
9: else
10: {
11: // Nein, Variable lesen
12: ar >> m_MyVar;
13: }
14: }

```

Eine `Serialize`-Funktion können Sie in alle von Ihnen erstellten Klassen aufnehmen und damit deren `Serialize`-Funktion von der Funktion `Serialize` des Dokuments aufrufen. Wenn Sie Ihre benutzerdefinierten Objekte in ein Objektarray stellen, wie zum Beispiel das in der Zeichenanwendung der letzten drei Tage verwendete `CObArray`, läßt sich die `Serialize`-Funktion des Arrays von der `Serialize`-Funktion des Dokuments aufrufen. Das Objektarray ruft seinerseits die `Serialize`-Funktion des jeweiligen Objekts auf, das im Array gespeichert ist.

Objekte serialisierbar machen

Als Sie am Tag 10 die Klasse `CLine` erstellt haben, mußten Sie zwei Makros einbinden, bevor Sie die Zeichnungen speichern und wiederherstellen konnten. Die beiden Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` binden in Ihre Klassen die erforderliche Funktionalität ein, damit die Funktion `Serialize` ordnungsgemäß arbeiten kann.

Das Makro `DECLARE_SERIAL` einbinden

In die Klassendeklaration ist das Makro `DECLARE_SERIAL` gemäß Listing 13.2 einzubinden. Das Makro `DECLARE_SERIAL` übernimmt als einziges Argument den Klassennamen und fügt automatisch in Ihre Klasse bestimmte Standardfunktionen und Operatordeklarationen ein, die für den korrekten Ablauf der Serialisierung notwendig sind.

Listing 13.2: Das Makro `DECLARE_SERIAL` in die Klassendeklaration einbinden

```

1: class CMyClass : public CObject
2: {
3: DECLARE_SERIAL (CMyClass)
4: public:
5: virtual void Serialize(CArchive &ar);
6: CMyClass();
7: virtual ~CMyClass();
8: };

```

Das Makro `IMPLEMENT_SERIAL` einbinden

Das Makro `IMPLEMENT_SERIAL` ist in die Implementierung Ihrer Klasse einzubinden. Es muß außerhalb aller anderer Klassenfunktionen erscheinen, da es den Code für diejenigen Klassenfunktionen hinzufügt, die

mit dem Makro `DECLARE_SERIAL` deklariert werden.

Das Makro `IMPLEMENT_SERIAL` übernimmt drei Argumente. Das erste gibt wie beim Makro `DECLARE_SERIAL` den Klassennamen an. Das zweite Argument übergibt den Namen der Basisklasse, von der Ihre Klasse abgeleitet ist. Das dritte Argument bezeichnet eine Versionsnummer, aus der man bestimmen kann, ob eine in die Anwendung einzulesende Datei in der richtigen Version vorliegt. Die Versionsnummer muß positiv sein und sollte jedesmal inkrementiert werden, wenn Sie Änderungen an der Serialisierungsmethode der Klasse vornehmen und sich dabei die mit einer Datei auszutauschenden Datenformate ändern. Listing 13.3 zeigt einen typischen Einsatz des Makros `IMPLEMENT_SERIAL`.

Listing 13.3: Das Makro `IMPLEMENT_SERIAL` in die Implementierung der Klasse einbinden

```

1: // MyClass.cpp: Implementierung der Klasse CMyClass.
2: //
3: //////////////////////////////////////
4:
5: #include "stdafx.h"
6: #include "MyClass.h"
7:
8: #ifdef _DEBUG
9: #undef THIS_FILE
10: static char THIS_FILE[]=__FILE__;
11: #define new DEBUG_NEW
12: #endif
13:
14: IMPLEMENT_SERIAL (CMyClass, CObject, 1)
15: //////////////////////////////////////
16: // Konstruktion/Destruktion
17: //////////////////////////////////////
18:
19: CMyClass::CMyClass()
20: {
21: }
22:
23: CMyClass::~~CMyClass()
24: {
25: }
```

Die Funktion `Serialize` definieren

Neben den beiden Makros müssen Sie eine `Serialize`-Funktion in Ihre Klasse einbinden. Diese Funktion sollte vom Typ `void` mit einem einzelnen Argument (`CArchive &ar`), dem Zugriffsstatus `Public` und eingeschaltetem Kontrollkästchen `Virtual` deklariert sein, woraus die Funktionsdeklaration in Listing 13.2 hervorgeht. Wenn Sie die Funktion `Serialize` für Ihre Klasse implementieren, gehen Sie normalerweise genauso vor, wie bei der in Listing 13.1 gezeigten Dokumentklasse. Auch hier ist zu prüfen, ob die Datei zu lesen oder zu schreiben ist.

Eine serialisierbare Klasse implementieren

Wenn Sie eine neue Anwendung entwerfen, müssen Sie als eine der ersten Aufgaben entscheiden, wie die Daten in der Dokumentklasse, die Ihre Anwendung erzeugt und darauf arbeitet, zu speichern sind. Erstellen Sie zum Beispiel eine datenbezogene Anwendung, die in der Art einer Kontakt-Datenbank Datenmengen vom Benutzer entgegennimmt, wie wollen Sie diese Daten im Arbeitsspeicher der Anwendung halten? Oder wenn Sie eine Textverarbeitung erstellen - wie wollen Sie das geschriebene Dokument im Speicher unterbringen?

Oder ein Tabellenblatt? Oder ein Zeichenprogramm? Oder - na ja, Sie ahnen schon, worum es geht.

Nachdem Sie die Datenstrukturen entworfen haben, mit denen Ihre Anwendung arbeitet, können Sie bestimmen, wie sich Ihre Anwendung und die Klassen am besten serialisieren lassen. Wenn Sie alle Daten direkt in der Dokumentklasse aufbewahren, brauchen Sie sich nur darum zu kümmern, wie die Daten über das `CArchive`-Objekt in der Funktion `Serialize` des Dokuments geschrieben und gelesen werden. Wenn Sie Ihre eigene Klasse erstellen, um die Anwendungsdaten darin zu halten, müssen Sie die Funktionalität der Serialisierung in Ihre Datenklassen einbinden, damit sich die Objekte selbst speichern und wiederherstellen können.

In der heutigen Beispielanwendung erstellen Sie eine einfache, lineare Datenbankanwendung, die verdeutlicht, wie man eine Mischung von Datentypen in einem einzigen Datenstrom bei der Serialisierung der Anwendung kombinieren kann. Die Anwendung zeigt ein paar Datenfelder an, von denen einige als Strings mit variabler Länge und andere als Integer oder Boolesch festgelegt sind. Diese Variablen werden über einen einzigen Datenstrom in das `CArchive`-Objekt geschrieben bzw. daraus wiederhergestellt.

Eine serialisierte Anwendung erstellen

Für eine SDI- oder MDI-Anwendung kann man eigene Klassen erzeugen, die sich ebenfalls serialisieren lassen. Kurz gesagt kann jede Anwendung, die in irgendeiner Form mit Daten arbeitet, ob es sich nun um eine Datenbank oder ein Dokument handelt, serialisiert werden. Sie erstellen nun eine einfache, lineare Datenbankanwendung, die Sie dann serialisieren.



Eine lineare - oder »flache« - Datenbank ist der Urahn aller Datenbanken. Es handelt sich dabei um eine einfache, dateibasierte Datenbank, bei der die Datensätze sequentiell gespeichert sind und jeweils an den vorherigen Datensatz anschließen. Man vermißt hier die märchenhafte Funktionalität von relationalen Datenbanken, die mittlerweile zum Standard in der Datenbanktechnologie avanciert sind. Die Datenbank, die Sie in der heutigen Lektion erstellen, kommt einer alten - ohne Indexe arbeitenden - dBASE- oder Paradox-Datenbank näher als einer Datenbank wie Access oder SQL Server.

Das Anwendungsgerüst erstellen

Zunächst erstellen Sie mit dem Anwendungs-Assistenten eine neue Anwendung. Geben Sie der Anwendung einen treffenden Namen - der Kürze wegen etwa `Serial` - und klicken Sie auf OK, um den Anwendungs-Assistenten zu starten.

Im Anwendungs-Assistenten wählen Sie die Option Einzelnes Dokument (SDI) und lassen das Kontrollkästchen Unterstützung der Dokument-/Ansicht-Architektur eingeschaltet. Im dritten Dialogfeld des Assistenten können Sie die Unterstützung für ActiveX-Steuerelemente beibehalten, obwohl es für das heutige Beispiel eigentlich nicht erforderlich ist.

Im vierten Dialogfeld des Assistenten geben Sie die Dateierweiterung für die Dateien an, die Ihre Anwendung erzeugt und liest. Ein Beispiel einer Dateierweiterung, die Sie verwenden können, ist `ser` für serialisieren oder `fdb` für flat-file database (lineare Datenbank).

Im sechsten Schritt des Anwendungs-Assistenten müssen Sie die Basisklasse für die Ansichtsklasse Ihrer Anwendung festlegen. Eine Erläuterung der verschiedenen Basisklassen, von der Sie die Ansichtsklasse ableiten können, finden Sie in der Lektion 10. Da es sich beim heutigen Beispiel um eine Datenbankanwendung handelt, ist es am einfachsten, `CFormView` als Basisklasse zu verwenden, von der Sie Ihre Ansichtsklasse ableiten. Das bietet nebenbei die Möglichkeit, die Ansicht Ihrer Anwendung mit dem Dialog-Editor zu gestalten.

Nachdem Sie mit dem Anwendungs-Assistenten das Gerüst Ihrer Anwendung erstellt haben, bietet Ihnen der

Dialog-Editor eine große Fensterfläche wie bei einer dialogfeldbasierenden Anwendung, nur daß die Schaltflächen OK und Abbrechen nicht vorhanden sind (siehe Abbildung 13.2).

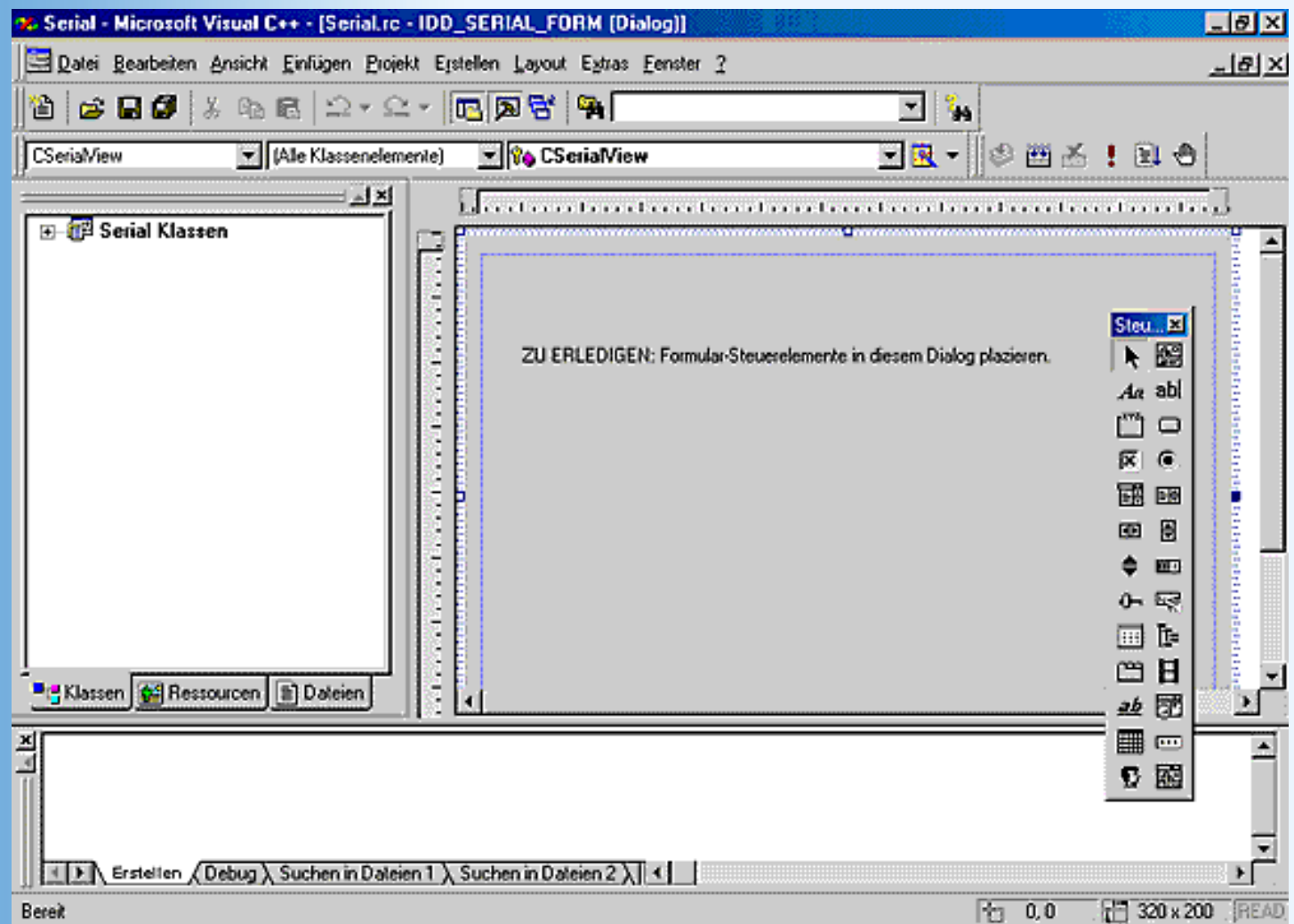


Abbildung 13.2:
Der Dialog-Editor für eine SDI-Anwendung

Das Anwendungsfenster entwerfen

Nachdem Sie eine SDI- oder MDI-Anwendung erstellt haben, in der die Ansichtsklasse auf der Klasse `CFormView` basiert, müssen Sie nun die Ansicht der Anwendung entwerfen. Das geht fast genauso wie beim Entwurf eines Dialogfelds, allerdings brauchen Sie sich nicht um irgendwelche Schaltflächen zum Schließen des Fensters zu kümmern, wobei der Benutzer die Arbeit entweder speichert oder abbricht. Bei einer SDI- oder MDI-Anwendung ist die gesamte Funktionalität zum Speichern und Verlassen des Fensters traditionsgemäß in den Anwendungsmenüs oder der Symbolleiste untergebracht. Daher brauchen Sie nur Steuerelemente für die vom Anwendungsfenster zu realisierenden Funktionen vorzusehen.



Wenn Sie dialogfeldbasierende Anwendungen erstellen, nimmt der Anwendungs-Assistent keinerlei Code für die Serialisierung in Ihr Anwendungsgerüst auf. Wenn Sie eine derartige Anwendung serialisieren müssen, sind Sie für den entsprechenden Code selbst verantwortlich.

Für die heute zu erstellende Beispielanwendung platzieren Sie die Steuerelemente in der Fensterfläche gemäß Abbildung 13.3. Die Eigenschaften der jeweiligen Steuerelemente sind in Tabelle 13.1 aufgeführt.

Tabelle 13.1: Einstellungen der Steuerelementeigenschaften

Objekt	Eigenschaft	Einstellung

Text	ID	IDC_STATIC
	Titel	&Name:
Eingabefeld	ID	IDC_ENAME
Text	ID	IDC_STATIC
	Titel	&Alter:
Eingabefeld	ID	IDC_EAGE
Text	ID	IDC_STATIC
	Titel	Familienstand:
Optionsfeld	ID	IDC_RSINGLE
	Titel	&ledig
	Gruppe	eingeschaltet
Optionsfeld	ID	IDC_RMARRIED
	Titel	&verheiratet
Optionsfeld	ID	IDC_RDIVORCED
	Titel	&geschieden
Optionsfeld	ID	IDC_RWIDOW
	Titel	ver&witwet
Kontrollkästchen	ID	IDC_CBEMPLOYED
	Titel	erwerbs&tätig
Schaltfläche	ID	IDC_BFIRST
	Titel	&Erster
Schaltfläche	ID	IDC_BPREV
	Titel	Vor&heriger
Schaltfläche	ID	IDC_BNEXT
	Titel	N&ächster
Schaltfläche	ID	IDC_BLAST
	Titel	Let&zter
Text	ID	IDC_SPOSITION
	Titel	Datensatz 0 von 0

Abbildung 13.3:
Das Layout der Beispielanwendung

Wenn Sie dialogfeldbasierende Anwendungen oder Fenster entwickeln, weisen Sie den Steuerelementen im Fenster Variablen in der Dialogfeldklasse zu. Wie sieht es nun bei einer SDI- oder MDI-Anwendung aus? In welcher Klasse erzeugen Sie die Variablen? Da die Funktion `UpdateData` ein Element der Klasse `CWnd` ist, und die Ansichtsklasse von der Klasse `CWnd` abgeleitet ist, auch wenn das nicht für das Dokument zutrifft, dann bietet sich die Ansichtsklasse als logischer Platz an, um die Variablen aufzunehmen, die Sie den Steuerelementen im Fenster zuweisen.

Um Variablen mit den Steuerelementen in Ihrer Beispielanwendung zu verbinden, öffnen Sie den Klassen-Assistenten und fügen der Ansichtsklasse (in diesem Fall `CSerialView`) Variablen für die Steuerelemente hinzu. Nehmen Sie für die Beispielanwendung die in Tabelle 13.2 aufgeführten Variablen für die angegebenen Steuerelemente auf.

Tabelle 13.2: Variablen für die Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_CBEMPLOYED	m_bEmployed	Wert	BOOL
IDC_EAGE	m_iAge	Wert	int
IDC_ENAME	m_sName	Wert	CString
IDC_RSINGLE	m_iMaritalStatus	Wert	int
IDC_SPOSITION	m_sPosition	Wert	CString

Wenn Sie sich den Quellcode für die Ansichtsklasse ansehen, stellen Sie fest, daß es keine Funktion `OnDraw` gibt. Wenn Sie für Ihre SDI- oder MDI-Anwendung die Basisklasse `CFormView` verwenden, brauchen Sie sich um die Funktion `OnDraw` nicht zu kümmern. Statt dessen behandeln Sie die Ansichtsklasse genauso wie die Dialogfeldklasse in einem Dialogfenster oder einer dialogfeldbasierenden Anwendung. Der Hauptunterschied besteht darin, daß die Daten, mit denen Sie die Steuerelemente im Fenster füllen, nicht in der Ansichtsklasse, sondern in der Dokumentklasse gespeichert sind. Im Ergebnis müssen Sie die Interaktion

zwischen diesen beiden Klassen realisieren, um die Daten für die Steuerelemente in beiden Richtungen zu übertragen.

Eine serialisierbare Klasse erstellen

Wenn Sie eine formularbasierte Anwendung erstellen, kann man davon ausgehen, daß Ihre Anwendung mehrere Datensätze im Formular aufnimmt und der Benutzer durch die Datensätze navigieren kann, um Änderungen vorzunehmen. Der Benutzer kann zusätzliche Datensätze einfügen oder sogar Datensätze aus dem Recordset entfernen. Das eigentlich Interessante in dieser Phase der Anwendungsentwicklung ist es, wie man diese Menge von Datensätzen darstellt und dabei die gesamte erforderliche Funktionalität unterstützt.

Eine Lösung besteht darin, eine Klasse zu erzeugen, die jeden Datensatz verkapselt, und dann diese Datensätze in ein Array aufzunehmen. Dieses Vorgehen entspricht der Zeichenanwendung, die Sie in den vergangenen Tagen erstellt und erweitert haben. Diese Klasse ist von der Klasse `CObject` abzuleiten und muß Variablen für alle Steuerelementvariablen enthalten, die Sie der Ansichtsklasse hinzugefügt haben. Außerdem gehören noch die Methoden dazu, um diese Variablen lesen und schreiben zu können. Sie müssen aber nicht nur die Methoden hinzufügen, um die Variablen zu setzen und zu lesen, Sie müssen auch die Klasse serialisierbar machen, indem Sie sowohl die Funktion `Serialize` als auch die beiden Makros, die die Serialisierung der Klasse realisieren, in die Klasse aufnehmen.

Die Basisklasse erstellen

Wie Sie in Lektion 10 gesehen haben, können Sie das Projekt auf der Registerkarte Klassen des Arbeitsbereichs markieren, mit der rechten Maustaste klicken und Neue Klasse aus dem Kontextmenü wählen, wenn Sie eine neue Klasse erstellen möchten. Daraufhin öffnet sich das Dialogfeld Neue Klasse.

Im Dialogfeld Neue Klasse legen Sie den Typ der Klasse fest, d.h., ob es sich um eine MFC-Klasse, eine allgemeine Klasse oder eine Formulkasse handelt. Um eine Klasse zu erzeugen, die die Daten eines Datensatzes enthalten kann, müssen Sie höchstwahrscheinlich eine allgemeine Klasse erstellen. Mehr dazu, wie man den passenden Klassentyp bestimmt, finden Sie am Tag 16. Weiterhin müssen Sie Ihrer Klasse einen Namen verleihen und die Basisklasse angeben, von der die neue Klasse abzuleiten ist.

Da das erstellte Formular der Beispielanwendung Informationen zu einer Person enthält, bietet es sich an, die neue Klasse `CPerson` zu nennen. Damit Sie die Klasse im Objektarray speichern können, müssen Sie die Klasse von `CObject` als Basisklasse ableiten. Wie schon am Tag 10 wird das Dialogfeld Neue Klasse feststellen, daß sich die Header-Datei mit der Basisklasse nicht finden läßt und Sie diese manuell hinzufügen müssen. Nun, die Header-Datei ist bereits eingebunden, so daß Sie diese Meldung ignorieren können. (Tag 16 geht näher darauf ein, wann Sie dieser Meldung Beachtung schenken müssen.)

Nachdem Sie die neue Klasse erstellt haben, sind noch die Variablen hinzuzufügen, um die auf dem Bildschirm für den Benutzer anzuzeigenden Datenelemente aufzunehmen. Gemäß einem guten objektorientierten Entwurf deklariert man diesen Variablen als privat, damit sie sich nicht durch andere Klassen manipulieren lassen. Die Variablentypen sollten den Typen der Variablen der Ansichtsklasse, die mit dem Steuerelementen des Fensters verbunden sind, entsprechen.

Für die Beispielanwendung fügen Sie die Variablen gemäß Tabelle 13.3 hinzu.

Tabelle 13.3:
Klassenvariablen für die
Klasse CPerson

Name	Typ
m_bEmployed	BOOL
m_iAge	int

m_sName	CString
m_iMaritalStatus	int

Methoden zum Lesen und Schreiben von Variablen hinzufügen

Nachdem die Klasse erstellt ist, müssen Sie Mittel bereitstellen, um die Variablen in der Klasse zu lesen und zu schreiben. Am einfachsten läßt sich das mit Inline-Funktionen in der Klassendefinition realisieren. Mit der einen Gruppe von Inline-Funktionen setzen Sie die Variablen, während Sie mit einer anderen Gruppe von Inline-Funktionen die aktuellen Werte der Variablen abrufen.



Eine Inline-Funktion ist eine kurze C++-Funktion, bei der der Funktionskörper beim Kompilieren der Anwendung anstelle des Funktionsaufrufs kopiert wird. Wenn die kompilierte Anwendung dann läuft, wird der Funktionscode ausgeführt, ohne daß ein Kontextsprung zur Funktion und nach Beendigung der Funktion wieder zurück erfolgen muß. Damit verringert sich der Overhead in der laufenden Anwendung, die Ausführungsgeschwindigkeit ist etwas besser, aber die ausführbare Anwendungsdatei nimmt etwas an Umfang zu. Je mehr Aufrufe der Inline-Funktion stattfinden, desto größer ist schließlich die Anwendung. Weitere Informationen zu Inline-Funktionen finden Sie im Anhang.

Wenn Sie die Variablenfunktionen `Get` und `Set` für Ihre Klasse `CPerson` der Beispielanwendung implementieren wollen, bearbeiten Sie die Header-Datei `Person.h` und fügen hier die Zeilen aus Listing 13.4 ein.

Listing 13.4: Die Deklarationen der Inline-Funktionen `Get` und `Set`

```
1: class CPerson : public CObject
2: {
3: public:
4: // Funktionen zum Setzen der Variablen
5: void SetEmployed(BOOL bEmployed) { m_bEmployed = bEmployed;}
6: void SetMaritalStat(int iStat) { m_iMaritalStatus = iStat;}
7: void SetAge(int iAge) { m_iAge = iAge;}
8: void SetName(CString sName) { m_sName = sName;}
9: // Funktionen zum Holen der aktuellen Variableneinstellungen
10: BOOL GetEmployed() { return m_bEmployed;}
11: int GetMaritalStatus() { return m_iMaritalStatus;}
12: int GetAge() {return m_iAge;}
13: CString GetName() {return m_sName;}
14: CPerson();
15: virtual ~CPerson();
16:
17: private:
18: BOOL m_bEmployed;
19: int m_iMaritalStatus;
20: int m_iAge;
21: CString m_sName;
22: };
```

Mit diesen Methoden können Sie nun die Werte von Variablen in Ihrer benutzerdefinierten Klasse setzen und abrufen. Normalerweise initialisiert man die Variablen auch, wenn die Klasse erstmalig erzeugt wird. Das läßt sich im Klassenkonstruktor erreichen, indem man allen Variablen einen Standardwert zuweist. Für die heutige Beispielanwendung zeigt Listing 13.5 den Konstruktor der Klasse `CPerson`.

Listing 13.5: Der Konstruktor der Klasse CPerson

```

1: CPerson::CPerson()
2: {
3: // Klassenvariablen initialisieren
4: m_iMaritalStatus = 0;
5: m_iAge = 0;
6: m_bEmployed = FALSE;
7: m_sName = "";
8: }

```

Die Klasse serialisieren

Nachdem Sie Ihre benutzerdefinierte Klasse mit allen Variablen definiert und initialisiert haben, müssen Sie die Klasse serialisierbar machen. Das läuft in drei Schritten ab. Im ersten Schritt nehmen Sie die Funktion `Serialize` in die Klasse auf. Diese Funktion schreibt die Variablenwerte mit Hilfe von C++-Streams in das `CArchive`-Objekt und liest sie auf die gleiche Weise von dort wieder aus. In den beiden anderen Schritten fügen Sie die Makros `DECLARE_SERIAL` und `IMPLEMENT_SERIAL` hinzu. Mit diesen Elementen ist Ihre benutzerdefinierte Klasse serialisierbar und bereit zum Einsatz.

Um die Funktion `Serialize` in die benutzerdefinierte Klasse aufzunehmen, fügen Sie über die Registerkarte Klassen des Arbeitsbereichs der Klasse `CPerson` eine Member-Funktion hinzu. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `Serialize(CArchive &ar)` und den Zugriffsstatus als `Public` fest. Schalten Sie außerdem das Kontrollkästchen `Virtual` ein. Die Funktion `Serialize` wird nun hinzugefügt und im Editor zur Bearbeitung angezeigt.

Als erstes ist in der Funktion `Serialize` die gleichnamige Funktion der Basisklasse aufzurufen. Wenn Sie die Funktion der Basisklasse zuerst aufrufen, werden alle gespeicherten Basisinformationen zuerst wiederhergestellt. Damit steht die erforderliche Unterstützung für Ihre Klasse bereit, bevor die Variablen in Ihrer Klasse wiederhergestellt werden. Nach dem Aufruf der Funktion der Basisklasse müssen Sie ermitteln, ob die Klassenvariablen zu lesen oder zu schreiben sind. Dazu können Sie die Methode `IsStoring` der Klasse `CArchive` aufrufen. Diese Funktion liefert `TRUE` zurück, wenn das Archiv zu schreiben ist, und `FALSE`, wenn es zu lesen ist. Wenn die Funktion `IsStoring` den Wert `TRUE` zurückgibt, können Sie C++-E/A-Streams einsetzen, um alle Klassenvariablen in das Archiv zu schreiben. Liefert die Funktion `FALSE`, lesen Sie mit C++-Streams aus dem Archiv. Insbesondere ist darauf zu achten, daß die Variablen sowohl beim Lesen als auch beim Schreiben in derselben Reihenfolge erscheinen. Mehr über C++-Streams finden Sie in Anhang A.

Listing 13.6 zeigt eine typische `Serialize`-Funktion für Ihre benutzerdefinierte Klasse. Beachten Sie, daß die Variablen von `CPerson` sowohl beim Schreiben in als auch beim Lesen aus dem Archiv in der gleichen Reihenfolge vorliegen.

Listing 13.6: Die Funktion Serialize der Klasse CPerson

```

1: void CPerson::Serialize(CArchive &ar)
2: {
3: // Funktion der Basisklasse aufrufen
4: CObject::Serialize(ar);
5:
6: // Wird geschrieben?
7: if (ar.IsStoring())
8: // Alle Variablen in der richtigen Reihenfolge schreiben
9: ar << m_sName << m_iAge << m_iMaritalStatus << m_bEmployed;
10: else
11: // Alle Variablen in der richtigen Reihenfolge lesen
12: ar >> m_sName >> m_iAge >> m_iMaritalStatus >> m_bEmployed;

```

13: }

Nachdem die Funktion `Serialize` an Ort und Stelle ist, müssen Sie noch die Makros in die benutzerdefinierte Klasse aufnehmen. Das erste Makro - `DECLARE_SERIAL` - gehört in die Header-Datei der Klasse und übernimmt den Klassennamen als einziges Argument.

Um zum Beispiel das Makro `DECLARE_SERIAL` in die benutzerdefinierte Klasse `CPerson` der Beispielanwendung hinzuzufügen, schreiben Sie das Makro direkt nach dem Anfang der Klassendeklaration, wo es den Standardzugriff für die Klasse erhält. Legen Sie den Klassennamen `CPerson` als einziges Argument an das Makro fest, wie es Listing 13.7 zeigt.

Listing 13.7: Die serialisierte Klassendeklaration von `CPerson`

```
1: class CPerson : public CObject
2: {
3: DECLARE_SERIAL (CPerson)
4: public:
5: // Funktionen zum Setzen der Variablen
6: void SetEmployed(BOOL bEmployed) { m_bEmployed = bEmployed;}
7: void SetMaritalStat(int iStat) { m_iMaritalStatus = iStat;}
8: void SetAge(int iAge) { m_iAge = iAge;}
9: void SetName(CString sName) { m_sName = sName;}
10: // Funktionen zum Holen der aktuellen Variableneinstellungen
11: BOOL GetEmployed() { return m_bEmployed;}
12: int GetMaritalStatus() { return m_iMaritalStatus;}
13: int GetAge() {return m_iAge;}
14: CString GetName() {return m_sName;}
15: CPerson();
16: virtual ~CPerson();
17:
18: private:
19: BOOL m_bEmployed;
20: int m_iMaritalStatus;
21: int m_iAge;
22: CString m_sName;
23: };
```



Als Zugriffsstatus für Funktionen und Variablen in C++-Klassen ist **Public** (öffentlich) voreingestellt. Alle Funktionen und Variablen, die vor der ersten Zugriffsdeklaration deklariert werden, sind per Vorgabe öffentlich. Es ist natürlich am einfachsten, alle öffentlichen Funktionen und Variablen von Klassen in diesem Bereich der Klassendeklaration unterzubringen, aber die explizite Deklaration der Zugriffsrechte für alle Funktionen und Variablen gehört zum besseren Programmierstil, weil damit eindeutig geklärt ist, wie die Funktionen oder Variablen der Klasse zugänglich (sichtbar) sind.



Die meisten C++-Funktionen erfordern ein Semikolon am Ende der Codezeile. Bei den beiden Makros zur Serialisierung ist das nicht notwendig, weil der C-Präprozessor die Makros insgesamt durch den Code ersetzt, bevor die Anwendung an den Compiler übergeben wird. Es schadet allerdings nicht, die Semikolons zu schreiben. Der Präprozessor ignoriert sie einfach.

Um die Serialisierung der benutzerdefinierten Klasse fertigzustellen, müssen Sie noch das Makro `IMPLEMENT_SERIAL` in die Klassendefinition aufnehmen. Der beste Platz für dieses Makro ist vor der Konstruktordefinition in der `.cpp`-Datei, die den Quellcode der Klasse enthält. Das Makro übernimmt drei

Argumente: den Namen der benutzerdefinierten Klasse, den Namen der Basisklasse und die Versionsnummer. Wenn Sie irgendwelche Änderungen an der Funktion `Serialize` vornehmen, sollten Sie die an das Makro `IMPLEMENT_SERIAL` übergebene Versionsnummer inkrementieren. Die Versionsnummer gibt an, wenn eine Datei mit einer vorherigen Version der Funktion `Serialize` geschrieben wurde und sich daher unter Umständen nicht durch die aktuelle Version der Anwendung lesen läßt.



Wenn Sie eine Datei mit einer älteren Version der Funktion `Serialize` in Ihrer Klasse geschrieben haben, löst die Anwendung in der Praxis eine Exception aus, die man mit den üblichen Verfahren der Ausnahmebehandlung in C++ abfangen kann. Damit können Sie in Ihrer Anwendung Code vorsehen, um ältere Dateiversionen zu erkennen und geeignet zu konvertieren. Weitere Informationen zur Ausnahmebehandlung in C++ finden Sie im Anhang A.

Um das Makro `IMPLEMENT_SERIAL` in die Beispielanwendung aufzunehmen, fügen Sie es in die Datei `Person.cpp` unmittelbar vor dem Klassenkonstruktor von `CPerson` ein. Übergeben Sie `CPerson` (den Klassennamen) als erstes Argument, `CObject` (die Basisklasse) als zweites Argument und `1` als Versionsnummer, wie es Listing 13.8 zeigt.

Listing 13.8: Das Makro `IMPLEMENT_SERIAL` im Code von `CPerson`

```
1: // Person.cpp: Implementierung der Klasse CPerson.
2: //
3: //////////////////////////////////////
4:
5: #include "stdafx.h"
6: #include "Serial.h"
7: #include "Person.h"
8:
9: #ifdef _DEBUG
10: #undef THIS_FILE
11: static char THIS_FILE[]=__FILE__;
12: #define new DEBUG_NEW
13: #endif
14:
15: IMPLEMENT_SERIAL (CPerson, CObject, 1)
16: //////////////////////////////////////
17: // Konstruktion/Destruktion
18: //////////////////////////////////////
19:
20: CPerson::CPerson()
21: {
22: // Klassenvariablen initialisieren
23: m_iMaritalStatus = 0;
24: m_iAge = 0;
25: m_bEmployed = FALSE;
26: m_sName = "";
27: }
```

Unterstützung in der Dokumentklasse

Wenn Sie eine formularbasierte Anwendung erstellen, in der das Formular im Fenster der vorrangige Platz für die Benutzerinteraktion mit der Anwendung ist, gibt es eine stillschweigende Übereinkunft, daß Ihre Anwendung dem Benutzer die Arbeit mit einer Anzahl von Datensätzen gestattet. Sie müssen also eine Unterstützung vorsehen, um die Datensätze zwischenspeichern und die Navigation durch den Recordset zu

ermöglichen. Am einfachsten läßt sich das realisieren, indem man ein Objektarray als Variable in die Dokumentklasse aufnimmt, wie Sie es am Tag 10 gesehen haben. Damit kann man bei Bedarf zusätzliche Datensatzobjekte hinzufügen. Die Navigation implementiert man als Funktionen, die das erste, letzte, nächste oder vorherige Datensatzobjekt abrufen. Schließlich müssen Sie noch die Funktionalität realisieren, um bestimmen zu können, welchen Datensatz im Recordset der Benutzer gerade bearbeitet.

Um diese Funktionalität zu unterstützen, braucht die Dokumentklasse zunächst einmal zwei Variablen für das Objektarray und die aktuelle Datensatznummer im Array. Diese beiden Variablen bieten die erforderliche Unterstützung, um den Recordset zu speichern und die Navigation zu ermöglichen.

In die Beispielanwendung fügen Sie die beiden Variablen für die Unterstützung des Recordsets von CPerson-Objekten gemäß Tabelle 13.4 hinzu. Legen Sie für beide Variablen den Zugriff als Privat fest.

Tabelle 13.4: Variablen der Dokumentklasse

Name	Typ
m_iCurPosition	int
m_oaPeople	CObArray

Weiterhin ist in der Dokumentklasse noch zu gewährleisten, daß das Dokument über das zu speichernde Datensatzobjekt informiert ist. Dazu binden Sie in die Quelldatei der Dokumentklasse die Header-Datei der benutzerdefinierten Klasse ein, und zwar vor der Header-Datei für die Dokumentklasse. Da die Dokumentklasse Aktionen in der Ansichtsklasse auslösen muß, empfiehlt es sich, auch die Header-Datei für die Ansichtsklasse in die Dokumentklasse aufzunehmen.

Um diese Header-Dateien in die Beispielanwendung einzubinden, öffnen Sie die Quellcodedatei für die Dokumentklasse und fügen die beiden `#include`-Anweisungen gemäß Listing 13.9 ein.

Listing 13.9: Die benutzerdefinierten und Ansichtsklassen in die Implementierung der Dokumentklasse einbinden

```

1: // SerialDoc.cpp : Implementierung der Klasse CSerialDoc
2: //
3:
4: #include "stdafx.h"
5: #include "Serial.h"
6:
7: #include "Person.h"
8: #include "SerialDoc.h"
9: #include "SerialView.h"
10:
11: #ifdef _DEBUG
12: #define new DEBUG_NEW
13: #undef THIS_FILE
14: static char THIS_FILE[] = __FILE__;
15: #endif
16:
17:
18: //////////////////////////////////////
```

Neue Datensätze hinzufügen

Bevor Sie durch den Recordset navigieren können, müssen Sie neue Datensätze in das Objektarray aufnehmen können. Wenn Sie eine private Funktion für das Hinzufügen neuer Datensätze vorsehen, können Sie neue Datensätze dynamisch in den Recordset je nach Bedarf einfügen. Da neue Datensätze dem Benutzer leere Datenfelder präsentieren sollten, brauchen Sie keine Datensatzvariablen zu setzen, wenn Sie einen neuen Datensatz in das Objektarray aufnehmen. Damit können Sie auf den Standardkonstruktor zurückgreifen.

Nach der gleichen Logik, mit der Sie neue Liniendatensätze am Tag 10 hinzugefügt haben, nehmen Sie in der heutigen Beispielanwendung einen neuen Personendatensatz in das Objektarray der Dokumentklasse auf. Nachdem Sie einen neuen Datensatz hinzugefügt haben, können Sie einen Zeiger darauf zurückgeben, so daß die Ansichtsklasse direkt die Variablen im Datensatzobjekt aktualisieren kann.

Sobald der neue Datensatz hinzugefügt ist, setzen Sie den aktuellen Datensatzzeiger auf den neuen Datensatz im Array. Auf diese Weise läßt sich die aktuelle Datensatznummer leicht anhand des Positionszählers bestimmen.

Falls irgendwelche Probleme beim Erstellen des neuen Persondatensatzobjekts auftreten, weisen Sie den Benutzer darauf hin, daß der Anwendung nicht mehr ausreichend Speicher zur Verfügung steht, und löschen das reservierte Objekte genau wie am Tag 10.

Um diese Funktionalität in der Beispielanwendung zu realisieren, fügen Sie der Dokumentklasse eine neue Member-Funktion hinzu. Legen Sie den Typ als Zeiger auf Ihre benutzerdefinierte Klasse fest. Wenn die benutzerdefinierte Klasse CPerson heißt, lautet der Funktionstyp CPerson*. Die Funktion benötigt keine Argumente. Geben Sie der Funktion einen Namen, der ihren Zweck widerspiegelt, beispielsweise AddNewRecord . Legen Sie den Zugriff auf die Funktion mit Privat fest, da sie nur von anderen Funktionen innerhalb der Dokumentklasse aufgerufen wird. In die Funktion übernehmen Sie den Code aus Listing 13.10.

Listing 13.10: Die Funktion AddNewRecord der Klasse CSerialDoc

```

1: CPerson * CSerialDoc::AddNewRecord()
2: {
3: // Ein neues CPerson-Objekt erzeugen
4: CPerson *pPerson = new CPerson();
5: try
6: {
7: // Neue Person in Objektarray hinzufügen
8: m_oaPeople.Add(pPerson);
9: // Dokument als bearbeitet markieren
10: SetModifiedFlag();
11: // Neue Position festhalten
12: m_iCurPosition = (m_oaPeople.GetSize() - 1);
13: }
14: // Speicherausnahme aufgetreten?
15: catch (CMemoryException* perr)
16: {
17: // Benutzer über schlechte Neuigkeiten
18: // informieren
19: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
20: // Wurde Person-Objekt erzeugt?
21: if (pPerson)
22: {
23: // Objekt löschen
24: delete pPerson;
25: pPerson = NULL;
26: }
27: // Ausnahmeobjekt löschen

```



```

28: perr->Delete();
29: }
30: return pPerson;
31: }

```

Die aktuelle Position ermitteln

Um dem Benutzer die Navigation durch den Recordset zu erleichtern, ist es immer hilfreich, wenn man einen Anhaltspunkt liefert, wo sich der Benutzer im Recordset befindet. Um diese Informationen bereitzustellen, müssen Sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze aus dem Dokument ermitteln und sie dem Benutzer anzeigen.

Die betreffenden Funktionen sind recht einfach. Für die Gesamtzahl der Datensätze im Objektarray brauchen Sie lediglich die Größe des Arrays zu ermitteln und diesen Wert an den Aufrufer zurückgeben.

Fügen Sie für die Beispielanwendung eine neue Member-Funktion in die Dokumentklasse ein. Legen Sie den Funktionstyp als `int`, den Funktionsnamen mit `GetTotalRecords` und den Zugriff als `Public` fest. In die Funktion übernehmen Sie den Code aus Listing 13.11.

Listing 13.11: Die Funktion `GetTotalRecords` der Klasse `CSerialDoc`

```

1: int CSerialDoc::GetTotalRecords()
2: {
3: // Anzahl der Datensätze im Array zurückgeben
4: return m_oaPeople.GetSize();
5: }

```

Die aktuelle Datensatznummer läßt sich genauso einfach ermitteln. Wenn Sie einen Positionszähler in der Dokumentklasse verwalten, enthält diese Variable die Nummer des Datensatzes, den der Benutzer gerade bearbeitet. Letztendlich brauchen Sie nur den Wert dieser Variablen an den Aufrufer zurückgeben. Da das Objektarray mit Position 0 beginnt, addieren Sie zunächst eine 1 zur aktuellen Position, bevor Sie den Wert zurückgeben und dem Benutzer anzeigen.

Fügen Sie eine weitere Member-Funktion in die Dokumentklasse der Beispielanwendung ein. Legen Sie den Typ mit `int`, den Funktionsnamen mit `GetCurRecordNbr` und den Zugriff als `Public` fest. Übernehmen Sie den Code aus Listing 13.12 in die Funktion.

Listing 13.12: Die Funktion `GetCurRecordNbr` der Klasse `CSerialDoc`

```

1: int CSerialDoc::GetCurRecordNbr()
2: {
3: // Aktuelle Position zurückgeben
4: return (m_iCurPosition + 1);
5: }

```

Durch den Recordset navigieren

Damit die Anwendung auch einen echten Nutzen bringt, müssen Sie dem Benutzer eine Möglichkeit bieten, durch den Recordset zu navigieren. Für die Minimalausstattung erstellen Sie einen Satz von Funktionen in der Dokumentklasse, um die Zeiger auf bestimmte Datensätze im Recordset zu ermitteln. Die erste Funktion holt einen Zeiger auf den aktuellen Datensatz. Zwei weitere Funktionen liefern die Zeiger auf den ersten bzw. letzten Datensatz im Recordset. Schließlich sind noch Funktionen erforderlich, um den vorherigen und den nächsten Datensatz im Recordset anzusprechen. Wenn der Benutzer bereits den letzten Datensatz im Recordset bearbeitet und versucht, zum nächsten Datensatz weiterzuschalten, können Sie automatisch einen neuen Datensatz in den Recordset aufnehmen und dem Benutzer als neuen, leeren Datensatz anzeigen.

Als erstes erstellen wir die Funktion, die den aktuellen Datensatz zurückgibt. Diese Funktion muß den Wert

im Datensatzzeiger untersuchen, um sicherzustellen, daß der aktuelle Datensatz eine gültige Arrayposition ist. Nachdem Sie geprüft haben, daß die aktuelle Position gültig ist, kann die Funktion einen Zeiger auf den aktuellen Datensatz im Array zurückgeben.

Fügen Sie eine neue Member-Funktion in die Dokumentklasse der Beispielanwendung ein. Legen Sie den Funktionstyp mit `CPerson*` (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration als `GetCurRecord` und den Zugriff als `Public` fest. Übernehmen Sie den Code aus Listing 13.13 in die Funktion.

Listing 13.13: Die Funktion `GetCurRecord` der Klasse `CSerialDoc`.

```
1: CPerson* CSerialDoc::GetCurRecord()
2: {
3: // Ist Datensatznummer gültig?
4: if (m_iCurPosition >= 0)
5: // Ja, aktuellen Datensatz zurückgeben
6: return (CPerson*)m_oaPeople[m_iCurPosition];
7: else
8: // Nein, NULL zurückgeben
9: return NULL;
10: }
```

Als nächste Aufgabe realisieren Sie die Funktion, die den ersten Datensatz im Array zurückgibt. In dieser Funktion müssen Sie prüfen, ob das Array überhaupt Datensätze enthält. Sind Datensätze im Array vorhanden, setzen Sie den aktuellen Datensatzzeiger auf 0 und liefern einen Zeiger auf den ersten Datensatz im Array zurück.

Fügen Sie wiederum eine neue Member-Funktion in die Dokumentklasse der Beispielanwendung ein. Legen Sie den Funktionstyp als `CPerson*` (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration als `GetFirstRecord` und den Zugriff als `Public` fest. In die Funktion übernehmen Sie den Code aus Listing 13.14.

Listing 13.14: Die Funktion `GetFirstRecord` der Klasse `CSerialDoc`

```
1: CPerson* CSerialDoc::GetFirstRecord()
2: {
3: // Enthält das Array Datensätze?
4: if (m_oaPeople.GetSize() > 0)
5: {
6: // Ja, zur Position 0 gehen
7: m_iCurPosition = 0;
8: // Datensatz bei Position 0 zurückgeben
9: return (CPerson*)m_oaPeople[0];
10: }
11: else
12: // Keine Datensätze, NULL zurückgeben
13: return NULL;
14: }
```

Um zum nächsten Datensatz im Recordset zu navigieren, müssen Sie den aktuellen Datensatzzeiger inkrementieren und dann prüfen, ob das Ende des Arrays noch nicht überschritten ist. Wenn der Zeiger innerhalb der Arraygrenzen liegt, geben Sie einen Zeiger auf den aktuellen Datensatz im Array zurück. Ist das Ende des Arrays überschritten, fügen Sie einen neuen Datensatz an das Array an.

Zu diesem Zweck fügen Sie eine neue Member-Funktion in die Dokumentklasse der Beispielanwendung hinzu. Legen Sie den Funktionstyp als `CPerson*` (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration als `GetNextRecord`, und den Zugriff als `Public` fest. In die Funktion übernehmen Sie

den Code aus Listing 13.15.

Listing 13.15: Die Funktion `GetNextRecord` der Klasse `CSerialDoc`

```

1: CPerson * CSerialDoc::GetNextRecord()
2: {
3: // Arraygrenze überschritten nach Inkrementieren
4: // des Positionszählers?
5: if (++m_iCurPosition < m_oaPeople.GetSize())
6: // Nein, Datensatz an der neuen aktuellen Position zurückgeben
7: return (CPerson*)m_oaPeople[m_iCurPosition];
8: else
9: // Ja, neuen Datensatz hinzufügen
10: return AddNewRecord();
11: }

```

In der Funktion, mit der Sie zum vorherigen Datensatz im Array navigieren, sind verschiedene Prüfungen auszuführen. Erstens ist zu testen, ob das Array über Datensätze verfügt. Ist das der Fall, dekrementieren Sie den aktuellen Datensatzzeiger. Wird dieser Zeiger kleiner als null, setzen Sie den aktuellen Datensatzzeiger gleich 0 und zeigen damit auf den ersten Datensatz im Array. Nach diesen Schritten können Sie einen Zeiger auf den aktuellen Datensatz im Array zurückgeben.

Fügen Sie auch hier wieder eine neue Member-Funktion in die Dokumentklasse der Beispielanwendung ein. Legen Sie den Funktionstyp als `CPerson*` (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration mit `GetPrevRecord` und den Zugriff als `Public` fest. In die Funktion übernehmen Sie den Code aus Listing 13.16.

Listing 13.16: Die Funktion `GetPrevRecord` der Klasse `CSerialDoc`

```

1: CPerson * CSerialDoc::GetPrevRecord()
2: {
3: // Enthält das Array Datensätze?
4: if (m_oaPeople.GetSize() > 0)
5: {
6: // Position nach Dekrementieren der aktuellen
7: // Position kleiner als 0?
8: if (--m_iCurPosition < 0)
9: // Ja, Datensatzzeiger auf 0 setzen
10: m_iCurPosition = 0;
11: // Datensatz an der neuen aktuellen Position zurückgeben
12: return (CPerson*)m_oaPeople[m_iCurPosition];
13: }
14: else
15: // Keine Datensätze, NULL zurückgeben
16: return NULL;
17: }

```

Für die Funktion, die zum letzten Datensatz im Array navigiert, müssen Sie ebenfalls prüfen, ob Datensätze im Array vorhanden sind. Ist das der Fall, können Sie die aktuelle Größe des Arrays ermitteln und den aktuellen Datensatzzeiger auf die um 1 verminderte Zahl der Datensätze im Array setzen. Dabei handelt es sich tatsächlich um den letzten Datensatz im Array, da die Zählung der Datensätze im Array bei 0 beginnt. Nachdem Sie den aktuellen Datensatzzeiger gesetzt haben, können Sie einen Zeiger auf den letzten Datensatz im Array zurückgeben.

Nehmen Sie eine neue Member-Funktion in die Dokumentklasse der Beispielanwendung auf. Legen Sie den Funktionstyp als `CPerson*` (ein Zeiger auf die benutzerdefinierte Klasse), die Funktionsdeklaration mit

GetLastRecord und den Zugriff als Public fest. In die Funktion schreiben Sie den Code aus Listing 13.17.

Listing 13.17: Die Funktion GetLastRecord der Klasse CSerialDoc

```

1: CPerson * CSerialDoc::GetLastRecord()
2: {
3: // Enthält das Array Datensätze?
4: if (m_oaPeople.GetSize() > 0)
5: {
6: // Zur letzten Position im Array gehen
7: m_iCurPosition = (m_oaPeople.GetSize() - 1);
8: // Datensatz auf dieser Position zurückgeben
9: return (CPerson*)m_oaPeople[m_iCurPosition];
10: }
11: else
12: // Keine Datensätze, NULL zurückgeben
13: return NULL;
14: }

```

Den Recordset serialisieren

Wenn Sie die Funktion `Serialize` in die Dokumentklasse aufnehmen, brauchen Sie nichts weiter zu tun, als das `CArchive`-Objekt an die Funktion `Serialize` des Objektarrays zu übergeben, genauso wie Sie es von Tag 10 her kennen.

Beim Lesen der Daten aus dem Archiv fragt das Objektarray das `CArchive`-Objekt ab, um zu bestimmen, welcher Objekttyp und wie viele Objekte zu erzeugen sind. Das Objektarray erzeugt dann jedes Objekt im Array, ruft dessen `Serialize`-Funktion auf und übergibt dabei das `CArchive`-Objekt. Damit können die Objekte im Objektarray ihre eigenen Variablenwerte aus dem `CArchive`-Objekt in der gleichen Reihenfolge lesen, wie sie in die Datei geschrieben wurden.

Beim Schreiben von Daten in das Dateiarhiv ruft das Objektarray alle `Serialize`-Funktionen der Reihe nach auf und übergibt das `CArchive`-Objekt (genau wie beim Lesen aus dem Archiv). Damit kann das Objekt im Array die eigenen Variablen in der erforderlichen Form und Reihenfolge in das Archiv schreiben.

Für die Beispielanwendung bearbeiten Sie die Funktion `Serialize` der Dokumentklasse, um das Objekt `CArchive` an die Funktion `Serialize` des Objektarrays zu übergeben, wie es Listing 13.18 zeigt.

Listing 13.18: Die Funktion Serialize der Klasse CSerialDoc

```

1: void CSerialDoc::Serialize(CArchive& ar)
2: {
3: // Serialisierung an Objektarray übergeben
4: m_oaPeople.Serialize(ar);
5: }

```

Aufräumarbeiten

Es ist noch der Code zu realisieren, um das Dokument »aufzuräumen«, wenn es geschlossen oder ein neues Dokument geöffnet wird. Dazu sind alle Objekte im Objektarray zu durchlaufen und zu löschen. Anschließend kann man das Objektarray über die Funktion `RemoveAll` zurücksetzen.

Um diese Funktionalität in der Beispielanwendung zu realisieren, fügen Sie mit dem Klassen-Assistenten eine Behandlungsroutine in die Dokumentklasse für die Nachricht `DeleteContents` hinzu. In diese Funktion übernehmen Sie den Code aus Listing 13.19.

Listing 13.19: Die Funktion DeleteContents der Klasse CSerialDoc


```

1: void CSerialDoc::DeleteContents()
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen
4:
5: ///////////////////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: ///////////////////////////////////////////////////
8:
9: // Anzahl der Datensätze im Objektarray ermitteln
10: int liCount = m_oaPeople.GetSize();
11: int liPos;
12:
13: // Enthält das Array Objekte?
14: if (liCount)
15: {
16: // Schleife durch das Array, dabei jedes Objekt löschen
17: for (liPos = 0; liPos < liCount; liPos++)
18: delete m_oaPeople[liPos];
19: // Array zurücksetzen
20: m_oaPeople.RemoveAll();
21: }
22:
23: ///////////////////////////////////////////////////
24: // EIGENER CODE, ENDE
25: ///////////////////////////////////////////////////
26:
27: CDocument::DeleteContents();
28: }

```

Ein neues Dokument öffnen

Wenn der Benutzer ein neues Dokument anlegt, soll ein leeres Formular erscheinen, in das er neue Informationen eingeben kann. Um diesen leeren Datensatz für die Aufnahme neuer Informationen vorzubereiten, nimmt man einen Datensatz in das Objektarray auf, der im übrigen leer ist. Damit steht im Objektarray ein einziger Datensatz. Im Anschluß muß man die Ansicht modifizieren, um den neuen Datensatz auch tatsächlich anzuzeigen. Andernfalls zeigt die Ansicht den letzten bearbeiteten Datensatz aus dem vorherigen Recordset an (und der Benutzer wundert sich, warum die Anwendung keinen neuen Datensatz angelegt hat).

Um diese Funktionalität zu implementieren, ist die Funktion `OnNewDocument` in der Dokumentklasse zu bearbeiten. Diese Funktion ist bereits in der Dokumentklasse vorhanden, so daß man sie nicht extra über den Klassen-Assistenten hinzufügen muß. In der Funktion fügt man zunächst einen neuen Datensatz in das Objektarray ein. Anschließend holt man einen Zeiger auf das Ansichtsojekt. Die Position des Ansichtsojekts läßt sich mit der Funktion `GetFirstViewPosition` ermitteln. Mit der für das Ansichtsojekt zurückgegebenen Position ruft man die Funktion `GetNextView` auf, um einen Zeiger auf das Ansichtsojekt zu erhalten. Mit dem gültigen Zeiger kann man eine Funktion - die Sie in der Ansichtsklasse erstellen - aufrufen, um die Ansicht anzuweisen, die aktuellen Datensatzinformationen auf dem Formular anzuzeigen.



Wenn Sie diesen Code schreiben, sollten Sie beachten, daß der Typ des Zeigers auf die Ansicht in einen Zeiger der Klasse des Ansichtsbereichs umzuwandeln ist. Die Funktion `GetNextView` liefert einen Zeiger vom Typ `CView`, so daß keine Aufrufe der zusätzlichen Funktionen in der Ansichtsklasse möglich sind, bis Sie den Zeiger in Ihre Ansichtsklasse umwandeln. Die Typumwandlung des Zeigers teilt dem Compiler mit, daß der Zeiger eigentlich ein Zeiger auf die Objekte Ihrer Ansichtsklasse ist und demzufolge alle Funktionen enthält, die Sie hinzugefügt haben. Wenn Sie den Zeiger nicht umwandeln, nimmt der Compiler an, daß das Ansichtsbereich keine der von Ihnen hinzugefügten Funktionen enthält, und läßt das Kompilieren der Anwendung nicht zu.

Suchen Sie die Funktion `OnNewDocument` im Quellcode der Dokumentklasse auf, und übernehmen Sie in diese Funktion den Code aus Listing 13.20. Bevor Sie Ihre Anwendung kompilieren können, müssen Sie die Funktion `NewDataSet` in die Ansichtsklasse aufnehmen.

Listing 13.20: Die Funktion `OnNewDocument` der Klasse `CSerialDoc`

```

1: BOOL CSerialDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: ///////////////////////////////////
10: // EIGENER CODE, ANFANG
11: ///////////////////////////////////
12:
13: // Läßt sich kein neuer Datensatz hinzufügen, FALSE zurückgeben
14: if (!AddNewRecord())
15: return FALSE;
16:
17: // Einen Zeiger auf die Ansicht holen
18: POSITION pos = GetFirstViewPosition();
19: CSerialView* pView = (CSerialView*)GetNextView(pos);
20: // Der Ansicht mitteilen, daß eine neue Datenmenge eingetroffen ist
21: if (pView)
22: pView->NewDataSet();
23:
24: ///////////////////////////////////
25: // EIGENER CODE, ENDE
26: ///////////////////////////////////
27:
28: return TRUE;
29: }

```

Wenn Sie einen vorhandenen Recordset öffnen, brauchen Sie keinen neuen Datensatz hinzuzufügen. Dennoch müssen Sie das Ansichtsbereich anweisen, den anzuzeigenden Datensatz zu aktualisieren. Praktisch können Sie den gleichen Code in die Funktion `OnOpenDocument` schreiben, wie in die Funktion `OnNewDocument`, nur daß Sie den ersten Teil weglassen, der einen neuen Datensatz in das Objektarray einfügt.

Nehmen Sie mit dem Klassen-Assistenten eine neue Behandlungsroutine für das Ereignis `OnOpenDocument` in die Dokumentklasse auf. Schreiben Sie in diese Funktion den Code gemäß Listing

13.21.

Listing 13.21: Die Funktion OnOpenDocument der Klasse CSerialDoc

```

1: BOOL CSerialDoc::OnOpenDocument(LPCTSTR lpszPathName)
2: {
3: if (!CDocument::OnOpenDocument(lpszPathName))
4: return FALSE;
5:
6: // TODO: Speziellen Erstellungscode hier einfügen
7:
8: //////////////////////////////////////
9: // EIGENER CODE, ANFANG
10: //////////////////////////////////////
11:
12: // Einen Zeiger auf die Ansicht holen
13: POSITION pos = GetFirstViewPosition();
14: CSerialView* pView = (CSerialView*)GetNextView(pos);
15: // Der Ansicht mitteilen, daß sie eine neue Datenmenge erhalten hat
16: if (pView)
17: pView->NewDataSet();
18:
19: //////////////////////////////////////
20: // EIGENER CODE, ENDE
21: //////////////////////////////////////
22:
23: return TRUE;
24: }

```

Navigation und Bearbeitung in der Ansichtsklasse unterstützen

In die Dokumentklasse haben Sie die Unterstützung für den Recordset aufgenommen. Diese Funktionalität ist nun noch in der Ansichtsklasse zu realisieren, um durch die Datensätze zu navigieren, diese anzuzeigen und zu aktualisieren. Beim ersten Entwurf Ihrer Ansichtsklasse haben Sie eine Reihe von Steuerelementen im Fenster untergebracht, die für die Anzeige und Bearbeitung der verschiedenen Datenelemente der Datensätze vorgesehen sind. Weiterhin haben Sie Steuerelemente für die Navigation durch den Recordset aufgenommen. Diesen Steuerelementen müssen Sie nun »Leben einhauchen«, damit sie die Navigation durch die Datensätze und die Aktualisierung eines Datensatzes mit den vom Benutzer vorgenommenen Änderungen durchführen.

Durch den Umfang der direkten Interaktion, die das Formular mit dem Datensatzobjekt hat (Lesen der Variablenwerte aus dem Datensatz und Schreiben der neuen Werte in den Datensatz), ist es sinnvoll, einen Datensatzzeiger auf die Ansichtsklasse als private Variable hinzuzufügen. Nehmen Sie für die Beispielanwendung eine neue Member-Variable in die Ansichtsklasse auf, legen Sie den Typ mit `CPerson*`, den Namen mit `m_pCurPerson` und den Zugriff mit `Privat` fest. Als nächstes bearbeiten Sie den Quellcode der Ansicht und binden die Header-Datei für die Personenklasse gemäß Listing 13.22 ein.

Listing 13.22: Die Header-Datei für das benutzerdefinierte Objekt in den Quellcode der Ansichtsklasse einbinden

```

1: // SerialView.cpp : Implementierung der Klasse CSerialView
2: //
3:
4: #include "stdafx.h"
5: #include "Serial.h"
6:

```



```

7: #include "Person.h"
8: #include "SerialDoc.h"
9: #include "SerialView.h"
10:
11: #ifdef _DEBUG
12: .
13: .
14: .

```

Den aktuellen Datensatz anzeigen

Fügen Sie der Ansichtsklasse zuerst die Funktionalität hinzu, um den aktuellen Datensatz anzuzeigen. Da verschiedene Stellen in der Ansichtsklasse auf diese Funktionalität zurückgreifen, erstellen Sie am besten eine eigene Funktion dafür. In dieser Funktion holen Sie die aktuellen Werte aller Variablen im Datensatzobjekt und schreiben sie in die Variablen der Ansichtsklasse, die mit den Steuerelementen im Fenster verbunden sind. Weiterhin ermitteln Sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze im Recordset und zeigen diese Werte an, damit der Benutzer die relative Position innerhalb des Recordsets kennt.

Fügen Sie der Beispielanwendung eine neue Member-Funktion hinzu. Legen Sie den Funktionstyp mit `void`, die Funktionsdeklaration mit `PopulateView` und den Zugriff als Privat fest. In der Funktion ermitteln Sie einen Zeiger auf das Dokumentobjekt. Nachdem Sie über einen gültigen Zeiger für das Dokument verfügen, geben Sie die aktuelle Datensatznummer und die Gesamtzahl der Datensätze im Recordset - die Sie mit den bereits in die Dokumentklasse hinzugefügten Funktionen `GetCurRecordNbr` und `GetTotalRecords` ermitteln - formatiert aus. Mit einem gültigen Zeiger auf ein Datensatzobjekt setzen Sie als nächstes alle Variablen auf die Werte ihrer korrespondierenden Felder im Datensatzobjekt. Anschließend aktualisieren Sie das Fenster mit den Variablenwerten, wie es Listing 13.23 zeigt.

Listing 13.23: Die Funktion `PopulateView` der Klasse `CSerialView`

```

1: void CSerialView::PopulateView()
2: {
3: // Einen Zeiger auf das aktuelle Dokument holen
4: CSerialDoc* pDoc = GetDocument();
5: if (pDoc)
6: {
7: // Aktuelle Datensatzposition in der Menge anzeigen
8: m_sPosition.Format("Datensatz %d von %d", pDoc->GetCurRecordNbr(),
9: pDoc->GetTotalRecords());
10: }
11: // Ist Datensatzobjekt gültig?
12: if (m_pCurPerson)
13: {
14: // Ja, alle Werte des Datensatzes holen
15: m_bEmployed = m_pCurPerson->GetEmployed();
16: m_iAge = m_pCurPerson->GetAge();
17: m_sName = m_pCurPerson->GetName();
18: m_iMaritalStatus = m_pCurPerson->GetMaritalStatus();
19: }
20: // Anzeige aktualisieren
21: UpdateData(FALSE);
22: }

```

Durch den Recordset navigieren

Da Sie beim Entwurf des Formulars Schaltflächen zur Navigation hinzugefügt haben, lässt sich die Navigation

in einfacher Weise realisieren, indem Sie Behandlungsroutinen für alle Navigationsschaltflächen hinzufügen und die korrespondierende Navigationsfunktion im Dokument aufrufen. Nachdem das Dokument zum entsprechenden Datensatz im Recordset gegangen ist, müssen Sie die Funktion aufrufen, die Sie für die Anzeige des aktuellen Datensatzes erstellt haben. Wenn die Navigationsfunktionen des Dokuments Zeiger auf das neue aktuelle Datensatzobjekt zurückgeben, sollten Sie diese Zeiger zwischenspeichern, bevor Sie die Funktion zur Anzeige des aktuellen Datensatzes aufrufen.

Fügen Sie in Ihre Beispielanwendung mit dem Klassen-Assistenten eine Behandlungsfunktion für das Klickereignis der Schaltfläche Erster hinzu. In der Funktion holen Sie einen Zeiger auf das Dokumentobjekt. Nachdem Sie über einen gültigen Zeiger auf das Dokument verfügen, rufen Sie die Funktion `GetFirstRecord` des Dokumentobjekts auf und speichern den zurückgegebenen Objektzeiger in der Zeigervariablen `CPerson` der Ansicht ab. Wenn Sie einen gültigen Zeiger erhalten, rufen Sie die Funktion `PopulateView` auf, um die Daten des Datensatzes anzuzeigen. Der zugehörige Code ist in Listing 13.24 wiedergegeben.

Listing 13.24: Die Funktion `OnBfirst` der Klasse `CSerialView`

```
1: void CSerialView::OnBfirst()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: // Zeiger auf das aktuelle Dokument holen
6: CSerialDoc * pDoc = GetDocument();
7: if (pDoc)
8: {
9: // Den ersten Datensatz aus dem Dokument holen
10: m_pCurPerson = pDoc->GetFirstRecord();
11: if (m_pCurPerson)
12: {
13: // Aktuellen Datensatz anzeigen
14: PopulateView();
15: }
16: }
17: }
```

Für die Schaltfläche Letzter führen Sie die gleichen Schritte aus wie für die Schaltfläche Erster, rufen aber die Funktion `GetLastRecord` des Dokumentobjekts auf, wie es Listing 13.25 zeigt.

Listing 13.25: Die Funktion `OnBlast` der Klasse `CSerialView`

```
1: void CSerialView::OnBlast()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: // Zeiger auf das aktuelle Dokument holen
6: CSerialDoc * pDoc = GetDocument();
7: if (pDoc)
8: {
9: // Letzten Datensatz vom Dokument holen
10: m_pCurPerson = pDoc->GetLastRecord();
11: if (m_pCurPerson)
12: {
13: // Aktuellen Datensatz anzeigen
```

```

14: PopulateView();
15: }
16: }
17: }

```

Die gleichen Schritte wiederholen Sie für die Schaltflächen Vorheriger und Nächster, wobei Sie die Funktionen `GetPrevRecord` bzw. `GetNextRecord` des Dokumentobjekts aufrufen. Diese abschließenden Schritte realisieren in Ihrer Anwendung die gesamte Funktionalität, die man für die Navigation durch den Recordset benötigt. Da außerdem der Aufruf der Funktion `GetNextRecord` des Dokuments für den letzten Datensatz im Recordset automatisch einen neuen Datensatz in den Recordset hinzufügt, haben Sie auch die Möglichkeit, bei Bedarf neue Datensätze in den Recordset aufzunehmen.

Änderungen speichern

Wenn der Benutzer Änderungen an den Daten in den Steuerelementen auf dem Bildschirm vornimmt, müssen diese Änderungen irgendwie in den aktuellen Datensatz im Dokument gelangen. Wenn Sie im Ansichtsobjekt einen Zeiger auf das aktuelle Datensatzobjekt verwalten, können Sie die verschiedenen Funktionen zum Setzen von Werten des Datensatzobjekts aufrufen und dabei den neuen Wert übergeben, um den Wert im Datensatzobjekt zu setzen.

Um dies in der Beispielanwendung zu implementieren, fügen Sie mit dem Klassen-Assistenten eine Behandlungsroutine für das Ereignis `BN_CLICKED` für das Kontrollkästchen erwerbstätig hinzu. In der erzeugten Funktion rufen Sie zuerst `UpdateData` auf, um die Werte aus dem Formular in die Variablen der Ansicht zu kopieren. Prüfen Sie, ob Sie einen gültigen Zeiger auf das aktuelle Datensatzobjekt erhalten, und rufen Sie dann die entsprechende `Set`-Funktion auf dem Datensatzobjekt auf (in diesem Fall die Funktion `SetEmployed` wie in Listing 13.26).

Listing 13.26: Die Funktion `OnCbemployed` der Klasse `CSerialView`

```

1: void CSerialView::OnCbemployed()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: // Daten im Formular mit den Variablen synchronisieren
6: UpdateData(TRUE);
7: // Wenn gültiges Person-Objekt vorhanden, Daten an das Objekt übergeben
8: if (m_pCurPerson)
9: m_pCurPerson->SetEmployed(m_bEmployed);
10: }

```

Wiederholen Sie die gleichen Schritte für die anderen Steuerelemente, wobei Sie die jeweiligen Funktionen des Datensatzobjekts aufrufen. Fügen Sie eine Behandlungsroutine für das Ereignis `EN_CHANGE` der Eingabefelder Name und Alter hinzu, und rufen Sie die Funktionen `SetName` und `SetAge` auf. Die Optionsfelder für den Familienstand statuen Sie ebenfalls mit einer Behandlungsroutine aus, und zwar für das `BN_CLICKED`- Ereignis. Rufen Sie für alle vier Optionsfelder dieselbe Behandlungsfunktion auf. In dieser Funktion rufen Sie die Funktion `SetMaritalStat` im Datensatzobjekt auf.

Einen neuen Recordset anzeigen

Als letzte Funktionalität fügen Sie eine Funktion hinzu, um die Ansicht zurückzusetzen, wenn der Benutzer einen neuen Recordset anlegt oder öffnet, damit der alte Recordset nicht mehr angezeigt wird. Dazu rufen Sie die Behandlungsroutine für die Schaltfläche Erster auf und erzwingen, daß die Ansicht den ersten Datensatz im neuen Recordset anzeigt.

Um diese Funktionalität in der Beispielanwendung umzusetzen, fügen Sie eine neue Member-Funktion in die

Ansichtsklasse ein und legen deren Typ mit `void` fest. Als Funktionsnamen geben Sie den Namen an, den Sie vom Dokumentobjekt aus aufrufen (`NewDataSet`), und für den Zugriff wählen Sie die Option `Public` (damit sich die Funktion aus der Dokumentklasse aufrufen läßt). In der Funktion rufen Sie die Behandlungsroutine für die Schaltfläche Erster auf, wie es Listing 13.27 zeigt.

Listing 13.27: Die Funktion `NewDataSet` der Klasse `CSerialView`

```
1: void CSerialView::NewDataSet()
2: {
3: // Ersten Datensatz in der Menge anzeigen
4: OnBfirst();
5: }
```

Das Projekt umhüllen

Bevor Sie Ihre Anwendung kompilieren und ausführen können, müssen Sie die Header-Datei für Ihre benutzerdefinierte Klasse in den Quellcode der Hauptanwendung einbinden. Diese Datei trägt den gleichen Namen wie Ihr Projekt, hat aber die Erweiterung `.cpp`. Ihre benutzerdefinierte Header-Datei sollte vor den Header-Dateien sowohl der Dokument- als auch der Ansichtsklasse eingebunden werden. Für die Beispielanwendung fügen Sie in die Datei `Serialize.cpp` die Zeile 8 aus Listing 13.28 hinzu.

Listing 13.28: Die Header-Datei der Datensatzklasse in die Quelldatei der Hauptanwendung einbinden

```
1: // Serial.cpp : Legt das Klassenverhalten für die Anwendung fest.
2: //
3:
4: #include "stdafx.h"
5: #include "Serial.h"
6:
7: #include "MainFrm.h"
8: #include "Person.h"
9: #include "SerialDoc.h"
10: #include "SerialView.h"
11:
12: #ifdef _DEBUG
13: .
14: .
15: .
```

Jetzt können Sie mit Ihrer Anwendung Gruppen von Datensätzen hinzufügen, bearbeiten, speichern und wiederherstellen. Wenn Sie die Anwendung kompilieren und ausführen, können Sie Datensätze für sich selbst, für alle Ihre Familienangehörigen, Ihre Freunde oder jeden, den Sie in diese Anwendung aufnehmen möchten, anlegen. Wenn Sie den erstellten Recordset speichern und erneut öffnen, sobald Sie das nächste Mal mit der Anwendung arbeiten, sollten die Datensätze in genau dem gleichen Zustand wiederhergestellt sein, wie Sie sie eingegeben haben (siehe Abbildung 13.4).

Person1.ser - Serial

Datei Bearbeiten Ansicht ?

Name: Davis Chapman

Alter: 37

Familienstand: ☐ ledig ☐ geschieden
☒ verheiratet ☐ verwitwet

☒ erwerbstätig

Erster Vorheriger Nächster Letzter

Datensatz 1 von 5

Bereit NUM

Abbildung 13.4:
Die laufende Serialisierungsanwendung

Zusammenfassung

Die heutige Lektion hat im Detail gezeigt, was Serialisierung ist und wie sie funktioniert. Sie haben gelernt, wie man eine Klasse serialisierbar macht, und warum und wie man die beiden Makros einsetzt, die zur Serialisierung einer Klasse erforderlich sind. Weiterhin wurde dargestellt, wie man eine formularbasierte SDI-Anwendung entwirft und erstellt, wobei eine Gruppe von Datensätzen in einer linearen Datenbank für den Einsatz in der Anwendung verwaltet wurde. Sie haben gelernt, wie man die Serialisierung einsetzt, um die lineare Datenbank zu erstellen und zu verwalten, und wie man die Funktionalität in der Dokument- und der Ansichtsklasse realisiert, um die Navigation und Bearbeitung für diese Recordsets bereitzustellen.

Fragen und Antworten

Frage:

Wenn ich irgendwelche Änderungen an einem der Datensätze in meinem Recordset vornehme, nachdem ich den Recordset gespeichert und dann die Anwendung geschlossen habe oder einen anderen Recordset öffne, fordert die Anwendung keine Bestätigung, ob ich die Änderungen speichern möchte. Wie kann ich erreichen, daß die Anwendung eine entsprechende Bestätigung abrufen, wenn Daten geändert wurden?

Antwort:

Den Schlüssel zu diesem Problem liefert ein Funktionsaufruf in der Funktion AddNewRecord im Dokumentobjekt. Nachdem Sie einen neuen Datensatz in das Objektarray aufgenommen haben, rufen Sie die

Funktion SetModifiedFlag auf. Diese Funktion markiert das Dokument als geändert. Wenn Sie den Recordset speichern, wird die Markierung automatisch wieder zurückgesetzt (außer wenn die Anwendung den Recordset aus irgendwelchen Gründen nicht speichern kann). Beim Übernehmen der Änderungen müssen Sie das Dokument in den geänderten Zustand setzen, damit die Anwendung weiß, daß das Dokument ungesicherte Änderungen enthält.

Schreiben Sie zu diesem Zweck den entsprechenden Code in die Behandlungsroutinen für Ihre Datensteuerelemente. Nachdem Sie den neuen Wert im aktuellen Datensatz gespeichert haben, holen Sie einen Zeiger auf das Dokumentobjekt und rufen die Funktion SetModifiedFlag des Dokuments auf, wie es Listing 13.29 zeigt. Wenn Sie die gleichen Änderungen an allen Behandlungsroutinen für Datensteuerelemente vornehmen, fragt Ihre Anwendung nach, ob Sie die Änderungen, die Sie seit der letzten Sicherung des Recordsets vorgenommen haben, speichern möchten.

Listing 13.29: Die modifizierte Funktion OnCbemployed der Klasse CSerialView

```
1: void CSerialView::OnCbemployed()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: // Daten im Formular mit den Variablen synchronisieren
6: UpdateData(TRUE);
7: // Wenn gültiges Person-Objekt vorhanden, Daten an das Objekt übergeben
8: if (m_pCurPerson)
9: m_pCurPerson->SetEmployed(m_bEmployed);
10: // Zeiger auf das Dokument holen
11: CSerialDoc * pDoc = GetDocument();
12: if (pDoc)
13: // Veränderungsflag im Dokument setzen
14: pDoc->SetModifiedFlag();
15: }
```

Frage:

Warum muß ich die Versionsnummer im Makro IMPLEMENT_SERIAL ändern, wenn ich die Funktion Serialize in der benutzerdefinierten Datensatzklasse überarbeite?

Antwort:

Ob Sie die Versionsnummer inkrementieren müssen, hängt von der Art der vorgenommenen Änderungen ab. Wenn Sie beispielsweise ein berechnetes Feld in die Datensatzklasse aufnehmen und den Code hinzufügen, um diese neue Variable aus den Werten zu berechnen, die Sie in die Variablen aus dem CArchive-Objekt eingelesen haben, brauchen Sie die Versionsnummer eigentlich nicht inkrementieren, weil sich die Variablen und die Reihenfolge der Variablen, die Sie in das Archiv schreiben oder daraus lesen, nicht geändert haben. Wenn Sie allerdings ein neues Feld in die Datensatzklasse aufnehmen und das neue Feld in den E/A-Stream übernehmen, der in das CArchive-Objekt geschrieben oder daraus gelesen wird, dann hat sich das Format der mit dem Archiv ausgetauschten Daten geändert, und Sie müssen die Versionsnummer inkrementieren. Wenn Sie die Versionsnummer unverändert lassen und Dateien lesen, die mit vorherigen Versionen Ihrer Anwendung erzeugt wurden, dann erhalten Sie eine Meldung wie »Ungültiges Dateiformat«, und die Datei wird nicht gelesen. Nachdem Sie die Versionsnummer erhöht haben und eine Datei lesen, die mit einer älteren Versionsnummer geschrieben wurde, erhalten Sie zwar die gleiche Meldung, haben aber die Möglichkeit, die Ausnahme mit eigenem Code zu behandeln und das Archiv in eine Umwandlungsroutine umzuleiten, die die Datei in das neue Dateiformat konvertiert.

Workshop

Kontrollfragen

1. Welche zwei Makros müssen Sie in eine Klasse einfügen, um sie serialisierbar zu machen?
2. Wie kann man bestimmen, ob das `CArchive`-Objekt aus der Archivdatei zu lesen oder in die Archivdatei zu schreiben ist?
3. Welche Argumente sind an das Makro `IMPLEMENT_SERIAL` zu übergeben?
4. Von welcher Klasse müssen Sie die Ansichtsklasse ableiten, damit Sie ein Formular für das Hauptfenster einer SDI- oder MDI-Anwendung mit dem Dialog-Editor erstellen können?
5. In welchen Dateityp schreibt `CArchive` per Vorgabe?

Übung

Nehmen Sie Optionsfelder in das Formular auf, um das Geschlecht der Person festzulegen, wie es Abbildung 13.5 zeigt. Binden Sie diese Änderungen in die Klasse `CPerson` ein, um das Feld dauerhaft zu speichern.

Person1.ser - Serial

Datei Bearbeiten Ansicht ?

Name: Davis Chapman

Alter: 37

Familienstand: ☐ Jedig ☐ geschieden
☒ verheiratet ☐ verwitwet

Geschlecht: ☒ männlich ☐ weiblich

☒ erwerbstätig

Erster Vorheriger Nächster Letzter

Datensatz 1 von 5

Bereit NUM

Abbildung 13.5:
Die laufende Serialisierungsanwendung mit dem Geschlecht einer Person

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2

Tag 14

Daten aus ODBC- Datenbanken abrufen

Eine große Zahl von Anwendungen greift auf Datenbanken zu. Angefangen beim persönlichen Organizer bis hin zu großen Personalsystemen in Unternehmen speichern und verwalten Datenbanken alle Datensätze, die eine Anwendung verwendet und manipuliert. Visual C++ bietet vier verschiedene Technologien für die Verwendung und den Zugriff auf Datenbanken in Ihren Anwendungen: Data Access Objects (DAO), ODBC, OLE DB und ActiveX Data Objects (ADO). In der heutigen und morgigen Lektion gehen wir auf zwei dieser Technologien näher ein und befassen uns mit den Unterschieden sowie dem Einsatz in eigenen Anwendungen.

Heute lernen Sie, wie ...

- die ODBC-Schnittstelle einen einheitlichen Zugriff auf eine Datenbank ermöglicht,
- Visual C++ mit der Klasse `CRecordset` den Zugriff auf eine ODBC-Datenquelle bereitstellt,
- man eine einfache Datenbankanwendung mit Hilfe der Assistenten in Visual C++ erstellt,
- man in Visual C++ Datensätze in eine ODBC-Datenbank hinzufügt und daraus löscht.

Datenbankzugriff und ODBC

Die meisten Anwendungen im geschäftlichen Bereich arbeiten mit Daten. Diese Anwendungen verwalten und manipulieren Datensätze, die in Datenbanken gespeichert sind. Wenn Sie eine geschäftliche Anwendung erstellen, kommen Sie um den Zugriff auf eine Datenbank nicht herum. Die Frage ist, welche Datenbank?

Auf dem Markt ist eine ganze Anzahl von Datenbanken verfügbar. Wenn Sie eine Einzelbenutzeranwendung erstellen müssen, die eigenständig auf einem einzelnen Computer läuft, können Sie die verschiedenen PC-basierten Datenbanken wie Access oder FoxPro von Microsoft oder Paradox von Borland ins Auge fassen. Bei Anwendungen, die auf große, gemeinsam genutzte Datenbanken zugreifen, kommt höchstwahrscheinlich nur eine SQL-basierte Datenbank wie SQL Server oder Oracle in Frage. Alle Datenbanken bieten grundsätzlich die gleiche Funktionalität, um Datensätze zu verwalten. Bei jeder Datenbank lassen sich je nach Bedarf einzelne oder mehrere Datensätze abrufen. Man kann auch beliebige Datensätze hinzufügen, aktualisieren oder löschen. Alle Datenbanken können die Anforderungen Ihrer Anwendungen erfüllen, so daß Sie in der Lage sein sollten, eine bestimmte Datenbank für die eine Anwendung einzusetzen und für die nächste Anwendung zu einer anderen Datenbank zu wechseln, je nachdem, welche konkreten Anforderungen in Ihrer Anwendung zu realisieren sind, welche Datenbank sich am besten für die Anwendung eignet,

oder welche Datenbank Ihr Auftraggeber bevorzugt.



In der Praxis gibt es zahlreiche Unterschiede der heutzutage verfügbaren Datenbanken zu beachten. Jede Datenbank hat spezielle Stärken und Schwächen, was die eine Datenbank für eine bestimmte Situation geeigneter erscheinen läßt als eine andere. Allerdings würde es im Rahmen dieses Buches zu weit führen, auf die konkreten Unterschiede zwischen den einzelnen Produkten näher einzugehen. In bezug auf die heute und morgen zu behandelnden Datenbankfunktionen kann man davon ausgehen, daß alle Datenbanken funktionell gleichwertig und gegeneinander austauschbar sind.

Wenn man von einer Datenbank zu einer anderen wechselt, begegnet man dem Problem, daß jede Datenbank mit einer anderen Schnittstelle für den Zugriff arbeitet. Das bedeutet, daß man sich mit komplett neuen Programmierverfahren und Funktionen für jede einzusetzende Datenbank beschäftigen müßte. Diese Probleme lassen sich mit der eigens für diesen Zweck entwickelten ODBC-Schnittstelle in den Griff bekommen.

Die Schnittstelle ODBC

Microsoft hat die Inkompatibilität der Datenbankschnittstellen als Problem erkannt. Jede Datenbank hatte ihre eigene Anwendungsentwicklungssprache, die zwar gut in die Datenbank integriert war, aber nicht mit irgendeiner anderen Datenbank zusammenarbeitete. Daraus ergaben sich für jeden Entwickler Probleme, der eine Datenbank für die eine Anwendung und dann eine andere Datenbank für die nächste Anwendung benötigte. Der Entwickler mußte sich mit der speziellen Datenbanksprache der jeweiligen Datenbank beschäftigen und konnte nicht auf eine bekannte Sprache zurückgreifen. Damit der Programmierer mit einer beliebigen Datenbank in der bevorzugten Programmiersprache arbeiten konnte, war eine standardisierte Schnittstelle erforderlich, die mit jeder Datenbank funktioniert.

Die Schnittstelle Open Database Connectivity (ODBC) ist als standardisierte, SQL-basierte Schnittstelle integriert und in das Betriebssystem Windows integriert. Hinter dieser Schnittstelle verbergen sich für die jeweilige Datenbank Plugins, die die ODBC- Funktionsaufrufe entgegennehmen und in Aufrufe für die spezielle Schnittstelle der betreffenden Datenbank umwandeln. Die ODBC-Schnittstelle verwendet auch einen zentralen Satz von Konfigurationen zur Datenbankverbindung, wobei auch die Art und Weise standardisiert ist, wie man diese Konfigurationen festlegt und verwaltet. Damit braucht der Programmierer nur noch eine einzige Datenbankschnittstelle für alle Datenbanken zu erlernen und einzusetzen. Darüber hinaus ist es den Anbietern von Programmiersprachen möglich, ODBC-Unterstützung in ihre Sprachen und Entwicklungswerkzeuge zu integrieren, um den Datenbankzugriff nahezu transparent zu gestalten.

Die Klasse CRecordset

In der Visual C++-Entwicklungsumgebung ist der größte Teil der ODBC-Funktionalität in zwei Klassen verkapselt: **CRecordset** und **CDatabase**. Die Klasse **CDatabase** enthält die Verbindungsinformationen zur Datenbank und kann in der gesamten Anwendung gemeinsam genutzt werden. Die Klasse **CRecordset** verkapselt einen Satz von Datensätze *n* aus der Datenbank. Über diese Klasse kann man eine auszuführende SQL-Abfrage spezifizieren. Die Klasse führt die Abfrage aus und verwaltet die Ergebnismenge, die die Datenbank zurückgibt. Die Datensätze im Recordset kann man modifizieren und aktualisieren, die Änderungen werden zurück an die Datenbank übergeben. Im Recordset lassen sich Datensätze hinzufügen und löschen. Auch diese Änderungen

kann man zurück in die Datenbank schreiben.

Mit der Datenbank verbinden

Bevor die Klasse `CRecordset` irgendwelche Funktionen ausführen kann, muß sie mit einer Datenbank verbunden werden. Das realisiert man mit Hilfe der Klasse `CDatabase`. Es ist keine Instanz der Klasse `CDatabase` explizit zu erzeugen oder einzurichten - die erste Instanz der Klasse `CRecordset` übernimmt das automatisch. Wenn man eine Anwendung mit dem Anwendungs-Assistenten erstellt und die Einbindung der ODBC- Datenbankunterstützung wählt, bindet der Anwendungs-Assistent die Verbindungsinformationen zur Datenbank in die erste von ihm erzeugte und von `CRecordset` abgeleitete Klasse ein. Wenn man diese `CRecordset`-Klasse erzeugt, ohne ein `CDatabase`- Objekt zu übergeben, verwendet sie die vorgegebenen Verbindungsinformationen, die durch den Anwendungs-Assistenten hinzugefügt wurden, um ihre eigene Datenbankverbindung herzustellen.

Den Recordset öffnen und schließen

Nachdem das `CRecordset`-Objekt erstellt und mit der Datenbank verbunden ist, müssen Sie den Recordset öffnen, um die Gruppe der Datensätze aus der Datenbank abzurufen. Dazu verwenden Sie die Member-Funktion `Open` des `CRecordset`-Objekts. Diese Funktion kann man ohne irgendwelche Argumente aufrufen, wenn man für alles die Standardwerte übernehmen möchte, einschließlich der auszuführenden SQL- Anweisung.

Das erste Argument an die Funktion `Open` ist der Typ des Recordsets. Der Standardwert dafür lautet `AFX_DB_USE_DEFAULT_TYPE`. Er öffnet den Recordset als Snapshot von Datensätzen. Tabelle 14.1 listet die vier Typen von Recordsets auf. Nur zwei dieser Recordset-Typen sind im Anwendungs-Assistenten verfügbar, wenn Sie die Datenquelle spezifizieren.

Tabelle 14.1: Typen von Recordsets

Typ	Beschreibung
<code>CRecordset::dynaset</code>	Eine Gruppe von Datensätzen, die sich durch Aufruf der Funktion <code>Fetch</code> aktualisieren lassen, so daß Änderungen, die durch andere Benutzer am Recordset vorgenommen wurden, sichtbar sind.
<code>CRecordset::snapshot</code>	Eine Gruppe von Datensätzen, die sich nur aktualisieren läßt, wenn man den Recordset schließt und erneut öffnet.
<code>CRecordset::dynamic</code>	Nahezu ähnlich zum Typ <code>CRecordset::dynaset</code> , aber in vielen ODBC-Treibern nicht verfügbar.
<code>CRecordset::forwardOnly</code>	Eine schreibgeschützte Gruppe von Datensätzen, die sich nur vom ersten bis zum letzten Datensatz durchblättern lassen.

Das zweite Argument an die Funktion `Open` ist die SQL-Anweisung, die auszuführen ist, um den Recordset zu füllen. Übergibt man für dieses Argument eine `NULL`, wird die vom Anwendungs-Assistenten erzeugte und vorgegebene SQL-Anweisung ausgeführt.

Das dritte Argument ist eine Gruppe von Flags, mit denen Sie angeben können, wie die Gruppe der Datensätze in den Recordset abzurufen ist. Die meisten dieser Flags erfordern ein tiefergehendes Verständnis der ODBC-Schnittstelle, damit Sie Sinn und Zweck dieser Flags in Ihrer Anwendung

einschätzen können. Aus diesem Grund zeigt Tabelle 14.2 lediglich ein paar dieser Flags.

Tabelle 14.2: Flags für das Öffnen eines Recordsets

Flag	Beschreibung
CRecordset::none	Der Standardwert für dieses Argument. Legt fest, daß keine Optionen die Art und Weise beeinflussen, wie der Recordset geöffnet und verwendet wird.
CRecordset::appendOnly	Verhindert, daß der Benutzer die existierenden Datensätze im Recordset bearbeiten oder löschen kann. Der Benutzer kann ausschließlich neue Datensätze in den Recordset einfügen. Diese Option kann nicht in Verbindung mit dem Flag CRecordset::readOnly verwendet werden.
CRecordset::readOnly	Legt fest, daß sich der Recordset nur lesen läßt und der Benutzer keinerlei Änderungen daran vornehmen kann. Diese Option kann nicht in Verbindung mit dem Flag CRecordset::appendOnly verwendet werden.

Nachdem der Benutzer die Arbeit mit dem Recordset abgeschlossen hat, können Sie die Funktion `Close` aufrufen, um den Recordset zu schließen und alle vom Recordset belegten Ressourcen freizugeben. Die Funktion `Close` übernimmt keine Argumente.

Durch den Recordset navigieren

Wenn Sie eine Gruppe von Datensätzen aus der Datenbank abgerufen haben, müssen Sie in der Lage sein, durch diese Menge von Datensätzen zu navigieren (es sei denn, Sie haben es nur mit einem einzigen Datensatz zu tun). Die Klasse `CRecordset` stellt mehrere Funktionen bereit, mit denen man im Recordset navigieren und den Benutzer zu einem beliebigen Datensatz führen kann. Tabelle 14.3 listet die verfügbaren Funktionen zur Navigation im Recordset auf.

Tabelle 14.3: Funktionen zur Navigation im Recordset

Funktion	Beschreibung
MoveFirst	Geht zum ersten Datensatz im Recordset.
MoveLast	Geht zum letzten Datensatz im Recordset.
MoveNext	Geht zum nächsten Datensatz im Recordset.
MovePrev	Geht zum vorherigen Datensatz im Recordset.
Move	Schaltet um die angegebene Zahl von Datensätzen ausgehend vom aktuellen Datensatz oder vom ersten Datensatz im Recordset weiter.
SetAbsolutePosition	Geht zum angegebenen Datensatz im Recordset.
IsBOF	Liefert TRUE zurück, wenn es sich beim aktuellen Datensatz um den ersten Datensatz im Recordset handelt.
IsEOF	Liefert TRUE, wenn es sich beim aktuellen Datensatz um den letzten Datensatz im Recordset handelt.

GetRecordCount	Gibt die Anzahl der Datensätze im Recordset zurück.

Nur zwei dieser Navigations- und Informationsfunktionen - **Move** und **SetAbsolutePosition** - weisen Argumente auf. Die Funktion **SetAbsolutePosition** übernimmt ein einzelnes numerisches Argument, das die Zeile des Zieldatensatzes spezifiziert. Bei Übergabe von 0 gelangen Sie zur Position **BOF** (Begin Of File - Dateianfang), bei 1 zum ersten Datensatz im Recordset. Negative Werte bewirken, daß vom letzten Datensatz im Recordset rückwärts gezählt wird. (Zum Beispiel kommen Sie mit -1 zum letzten, mit -2 zum vorletzten Datensatz.)

Die Funktion **Move** übernimmt zwei Argumente. Das erste gibt die Anzahl der zu überspringenden Zeilen an. Es lassen sich positive und negative Werte angeben. Mit negativen Werten navigieren Sie rückwärts durch den Recordset. Das zweite Argument gibt die Art und Weise der Navigation durch die Menge der Zeilen an. Die möglichen Werte für das zweite Argument sind in Tabelle 14.4 aufgeführt.

Tabelle 14.4: Navigationstypen in der Funktion Move

Typ	Beschreibung
SQL_FETCH_RELATIVE	Bewegung um die angegebene Anzahl von Zeilen ab der aktuellen Zeile.
SQL_FETCH_NEXT	Verschiebung zur nächsten Zeile, wobei die angegebene Anzahl von Zeilen ignoriert wird. Das gleiche wie der Aufruf der Funktion MoveNext .
SQL_FETCH_PRIOR	Verschiebung zur vorherigen Zeile, wobei die angegebene Anzahl von Zeilen ignoriert wird. Das gleiche wie der Aufruf der Funktion MovePrev .
SQL_FETCH_FIRST	Geht zur ersten Zeile, wobei die angegebene Anzahl von Zeilen ignoriert wird. Das gleiche wie der Aufruf der Funktion MoveFirst .
SQL_FETCH_LAST	Geht zur letzten Zeile, wobei die angegebene Anzahl der Zeilen ignoriert wird. Das gleiche wie der Aufruf der Funktion MoveLast .
SQL_FETCH_ABSOLUTE	Verschiebung um die angegebene Anzahl von Zeilen vom Beginn des Recordsets. Das gleiche wie der Aufruf der Funktion SetAbsolutePosition .

Datensätze hinzufügen, löschen und aktualisieren

In der Regel genügt es nicht, wenn man lediglich durch die Menge der Datensätze navigieren kann. Man muß auch in der Lage sein, neue Datensätze in den Recordset aufzunehmen, vorhandene Datensätze zu bearbeiten und zu aktualisieren sowie Datensätze zu löschen. Diese Aktionen sind alle möglich über die Funktionen, die die Klasse **CRecordset** bereithält. Die Funktionen, mit denen Sie diese Funktionalität realisieren können, sind in Tabelle 14.5 aufgeführt.

Tabelle 14.5: Bearbeitungsfunktionen für Recordsets

Funktion	Beschreibung

AddNew	Fügt einen neuen Datensatz in den Recordset ein.
Delete	Löscht den aktuellen Datensatz aus dem Recordset.
Edit	Erlaubt die Bearbeitung des aktuellen Datensatzes.
Update	Speichert die aktuellen Änderungen in der Datenbank.
Requery	Führt die aktuelle SQL-Abfrage erneut aus, um den Recordset zu aktualisieren.

Diese Funktionen weisen keine Argumente auf. Allerdings erfordern einige, daß man bestimmte Schritte einhält, damit die Funktionen ordnungsgemäß arbeiten können.

Um einen neuen Datensatz in die Datenbank aufzunehmen, rufen Sie die Funktion **AddNew** auf. Als nächstes müssen Sie Vorgabewerte in allen Feldern festlegen, die Werte erfordern, beispielsweise Schlüsselfelder. Dann ist die Funktion **Update** aufzurufen, um den neuen Datensatz in die Datenbank aufzunehmen. Wenn Sie versuchen, zu einem anderen Datensatz zu navigieren, bevor Sie die Funktion **Update** aufgerufen haben, geht der neue Datensatz verloren. Nachdem Sie den neuen Datensatz gespeichert haben, müssen Sie die Funktion **Requery** aufrufen, um den Recordset zu aktualisieren, damit Sie zum neuen Datensatz navigieren und dem Benutzer die Bearbeitung des neuen Datensatzes ermöglichen können. Diese Sequenz der Funktionsaufrufe sieht in der Regel folgendermaßen aus:

```
// Einen neuen Datensatz in den Recordset einfügen
m_pSet.AddNew();
// Das Schlüsselfeld auf den neuen Datensatz setzen
m_pSet.m_AddressID = m_lNewID;
// Den neuen Datensatz in der Datenbank speichern
m_pSet.Update();
// Den Recordset aktualisieren
m_pSet.Requery();
// Zum neuen Datensatz gehen
m_pSet.MoveLast();
```

Den aktuellen Datensatz können Sie einfach durch Aufruf der Funktion **Delete** löschen. Ist der aktuelle Datensatz gelöscht, müssen Sie zu einem anderen Datensatz navigieren, damit der Benutzer nicht mehr den gerade gelöschten Datensatz sieht. Sobald der aktuelle Datensatz gelöscht wurde, gibt es keinen aktuellen Datensatz mehr, bis Sie zu einem anderen Datensatz navigieren. Die Funktion **Update** müssen Sie nicht explizit aufrufen, weil das die Navigationsfunktionen automatisch erledigen. Damit läßt sich der folgende Code schreiben, um den aktuellen Datensatz zu löschen:

```
// Den aktuellen Datensatz löschen
m_pSet.Delete();
// Zum vorherigen Datensatz gehen
m_pSet.MovePrev();
```

Um schließlich dem Benutzer zu ermöglichen, den aktuellen Datensatz zu bearbeiten, rufen Sie die Funktion **Edit** auf. Damit können Sie die Felder im Datensatz mit den neuen, vom Benutzer eingegebenen oder durch Ihre Anwendung berechneten Werten aktualisieren. Nachdem alle Änderungen am aktuellen Datensatz ausgeführt sind, müssen Sie die Funktion **Update** aufrufen, um die Änderungen zu speichern:

```
// Dem Benutzer die Bearbeitung des aktuellen Datensatzes
```

```
ermöglichen
```

```
m_pSet.Edit();
```

```
// Datenaustausch durchführen, dabei die Felder im Recordset  
aktualisieren
```

```
.
```

```
.
```

```
// Änderungen des Benutzers im aktuellen Datensatz speichern
```

```
m_pSet.Update();
```

Vielleicht fragen Sie sich, wie man zu den Feldern in den Datensätzen gelangt, um sie zu aktualisieren. Wenn der Anwendungs-Assistent die von **CRecordset** abgeleitete Klasse für Ihre Anwendung erstellt, fügt er alle Felder in den Datensätzen, die im Recordset erscheinen werden, als Member-Variablen der Recordset-Klasse hinzu. Im Ergebnis können Sie auf die Member-Variablen zugreifen, um die Datenelemente in den Datenbankdatensätzen, die Elemente des Recordsets sind, anzusprechen und zu manipulieren.

Eine Datenbankanwendung mit ODBC

Die heute zu erstellende Beispielanwendung konzipieren Sie als SDI-Anwendung mit ODBC-Datenbankunterstützung. Die Anwendung ruft Datensätze aus einer ODBC- Datenbank ab und erlaubt dem Benutzer, beliebige Datensätze zu bearbeiten und zu aktualisieren. Außerdem realisieren Sie die Funktionalität, um dem Benutzer zu ermöglichen, neue Datensätze in die Datenbank aufzunehmen und Datensätze aus der Datenbank zu löschen.

Die Datenbank vorbereiten

Bevor Sie eine Anwendung erstellen können, die eine Datenbank verwendet, brauchen Sie eine entsprechende Datenbank für Ihre Anwendung. Fast zu jeder Datenbank, die Sie für Ihre Anwendungen kaufen können, gehören Werkzeuge, mit denen sich eine neue Datenbank anlegen läßt. Zuerst müssen Sie Ihre Datenbank mit diesen Werkzeugen erstellen und dann mit dem ODBC-Administrator eine ODBC-Datenquelle für Ihre neue Datenbank konfigurieren.

Die neue Datenbank wurde für die Beispielanwendung in diesem Kapitel mit dem Datenbank-Assistent en von Access 97 erstellt. Als Vorlage für die Datenbank diente das Adreßbuch. Beim Start des Datenbank-Assistenten wählen Sie die vorgegebenen Felder, die in die Datenbank aufzunehmen sind, und die Option Beispieldaten einfügen, wie es Abbildung 14.1 zeigt. Übernehmen Sie die übrigen Einstellungen, die der Datenbank-Assistent vorgibt.

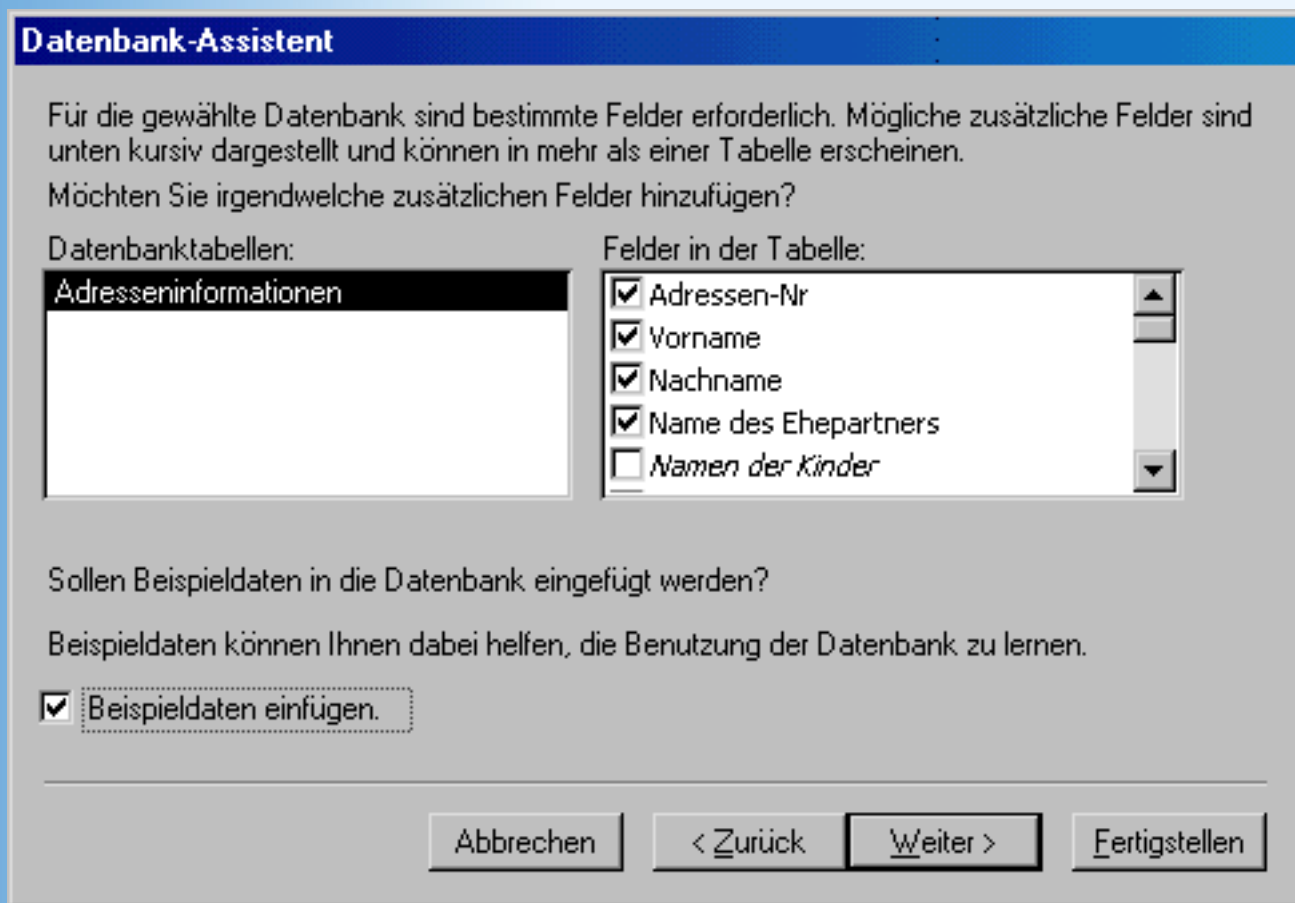


Abbildung 14.1:
Beispieldaten in die Datenbank aufnehmen

Nachdem Sie die Datenbank erstellt haben, müssen Sie eine ODBC-Datenquelle konfigurieren, die auf die gerade erstellte Datenbank verweist. Rufen Sie dazu den ODBC- Administrator auf, der sich in der Systemsteuerung von Windows 95/98 befindet.

Im Dialogfeld ODBC-Datenquellen-Administrator fügen Sie eine neue Datenquelle hinzu. Klicken Sie dazu auf die Schaltfläche Hinzufügen (siehe Abbildung 14.2). Es öffnet sich ein weiteres Dialogfeld, in dem Sie den Datenbanktreiber für die neue Datenquelle auswählen (siehe Abbildung 14.3).

Da Sie die heutige Beispielanwendung mit einer Access-Datenbank erstellen, markieren Sie den Microsoft Access-Treiber und klicken auf die Schaltfläche **Fertigstellen**.

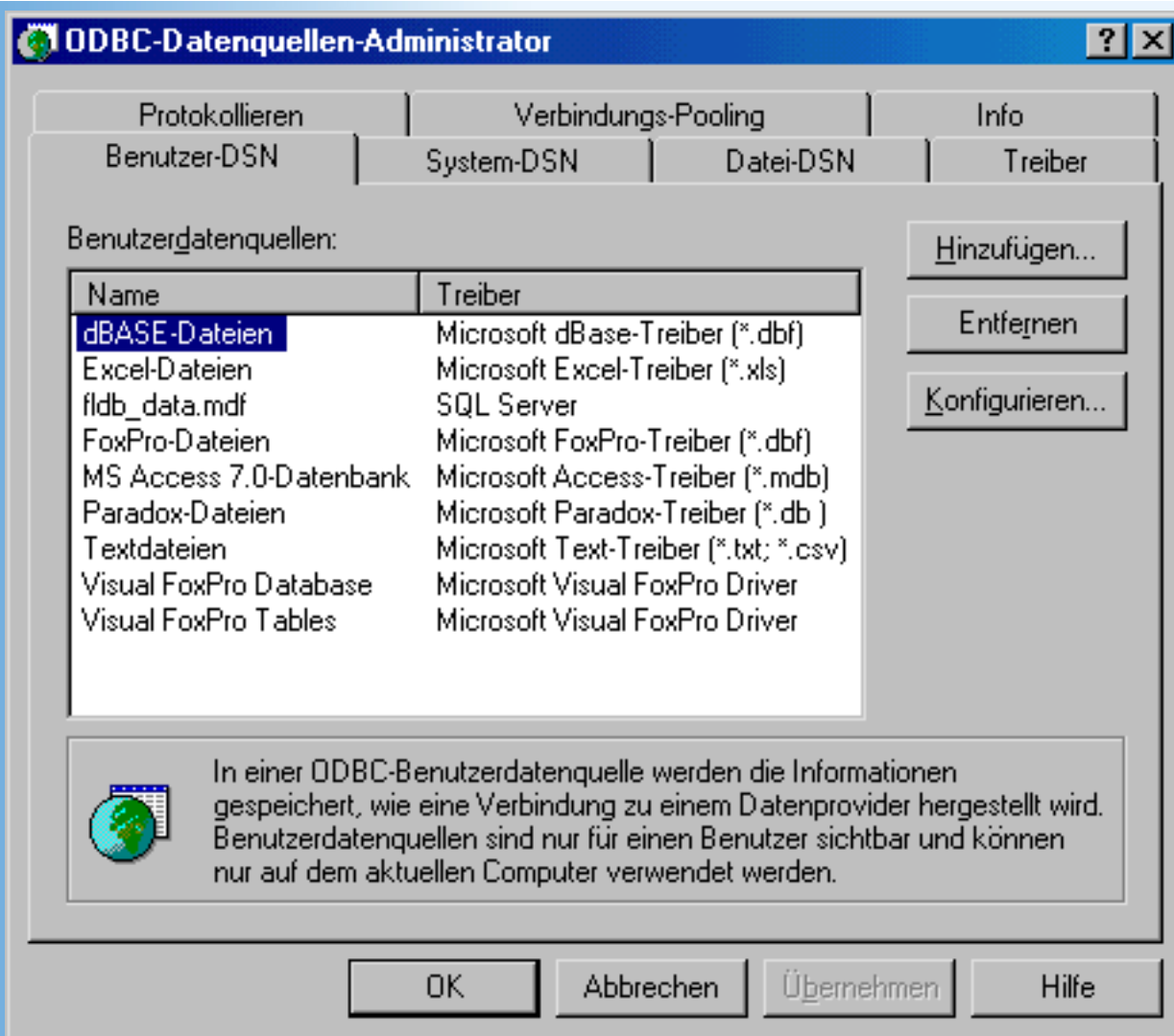


Abbildung 14.2:
Der ODBC-Datenquellen-Administrator

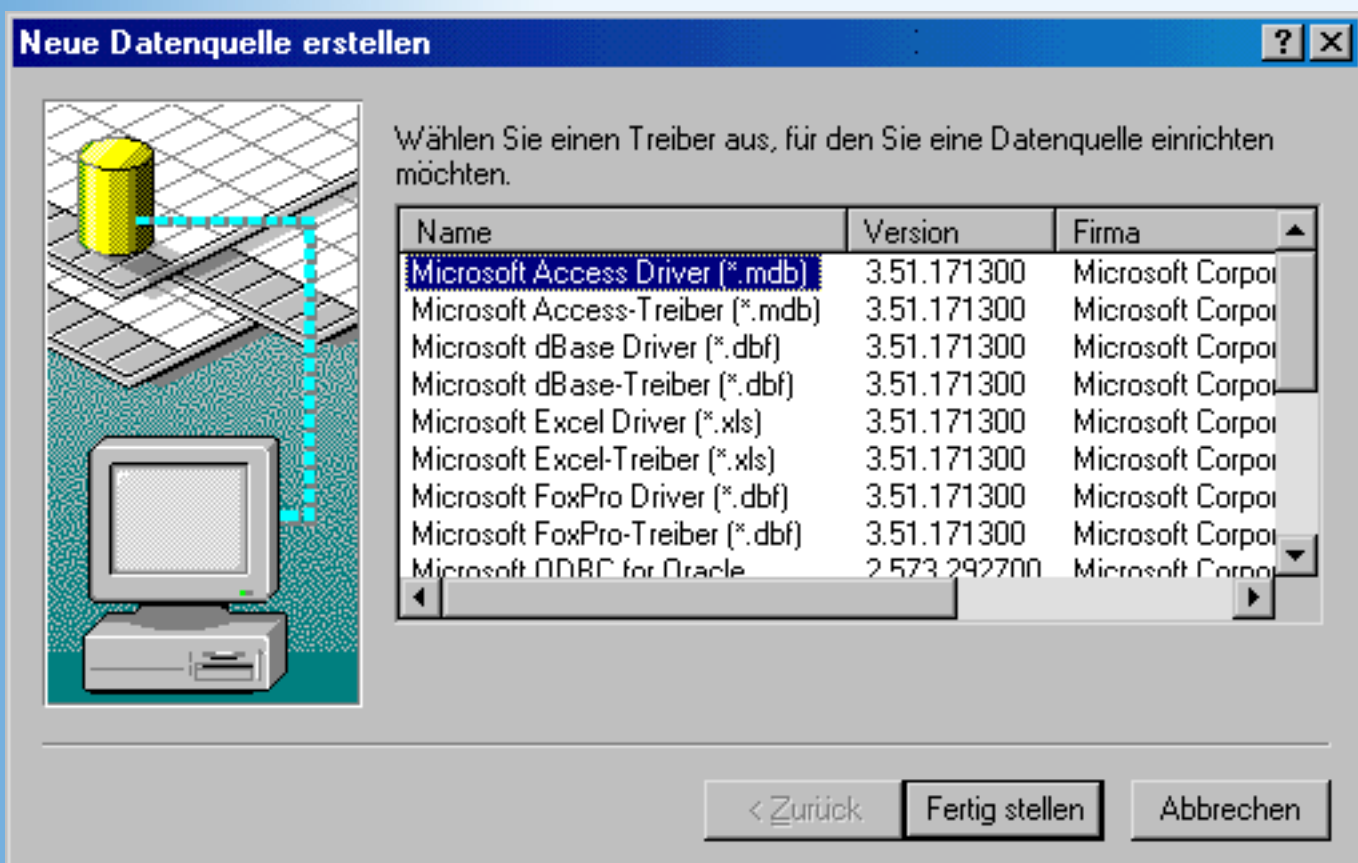


Abbildung 14.3: Das Dialogfeld Neue Datenquelle erstellen

Im Dialogfeld ODBC Microsoft Access 97-Setup (siehe Abbildung 14.4) geben Sie einen kurzen, einfachen Namen für die Datenquelle an. Ihre Anwendung verwendet diesen Namen, um die für die Datenbankverbindung zu verwendende Konfiguration der ODBC-Datenquelle zu spezifizieren. Daher sollte der Name den Zweck der Datenbank umreißen oder auf die Anwendung hinweisen, die mit der Datenbank arbeitet. Für die heutige Beispielanwendung wählen Sie etwa den Namen VCPP21DB (für Visual C++ in 21 Tagen, Datenbank) und geben dann eine Beschreibung für die Datenbank in das nächste Feld ein.

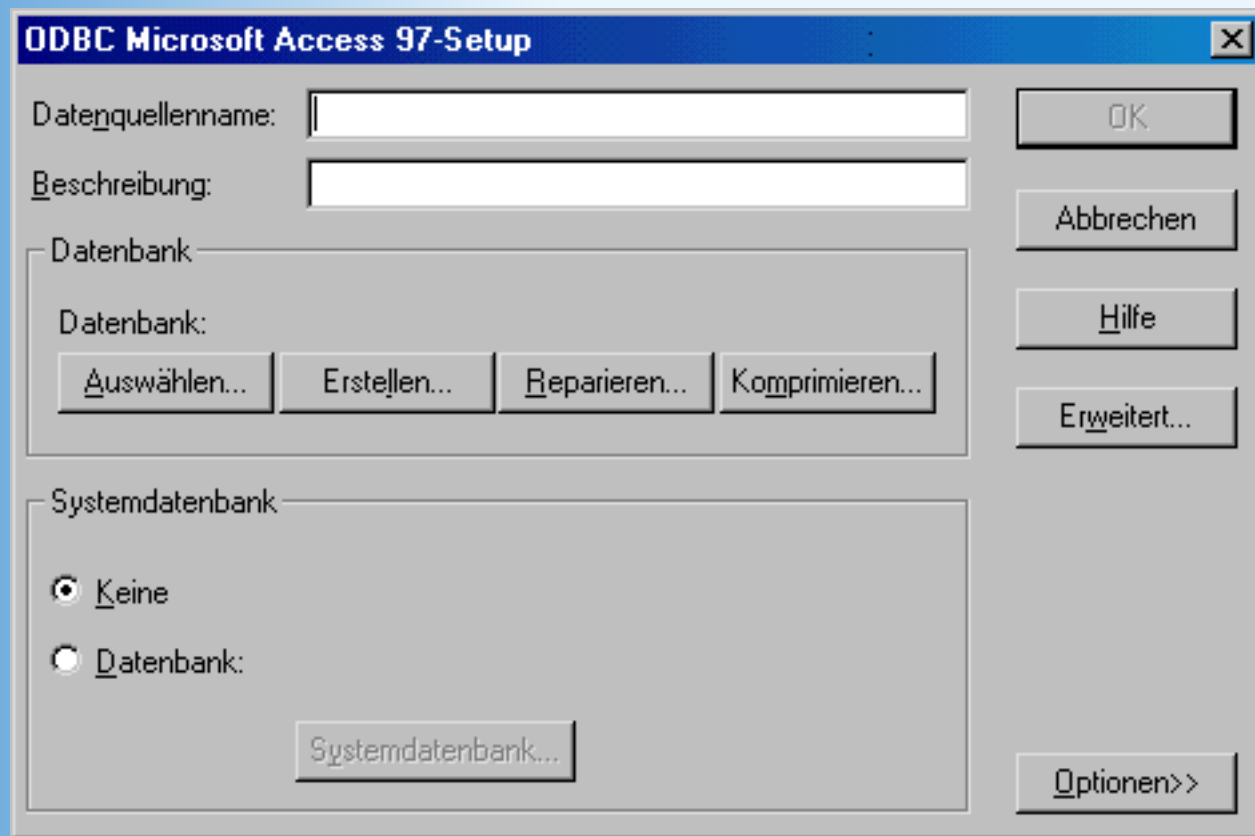


Abbildung 14.4: Das Dialogfeld ODBC Microsoft Access 97-Setup

Nachdem Sie einen Namen und eine Beschreibung für die Datenquelle eingegeben haben, müssen Sie festlegen, wo sich die Datenbank befindet. Klicken Sie auf die Schaltfläche **Auswählen**, und spezifizieren Sie dann die Access-Datenbank, die Sie bereits erstellt haben. Nachdem Sie die ODBC-Datenquelle für Ihre Datenbank konfiguriert haben, klicken Sie auf die Schaltfläche OK, um die neue Datenquelle in den ODBC-Datenquellen-Administrator aufzunehmen. Klicken Sie erneut auf OK, um die Konfiguration zu beenden und den ODBC-Datenquellen-Administrator zu schließen, da Sie nun Ihre Aufmerksamkeit auf das Erstellen der Anwendung richten können.

Das Anwendungsgerüst erstellen

Die heutige Beispielanwendung legen Sie als SDI-Anwendung mit Datenbankunterstützung an. Beginnen Sie ein neues Projekt, wählen Sie den Anwendungs-Assistenten, und geben Sie einen passenden Namen wie etwa DbOdbc ein.

Im ersten Dialogfeld des Anwendungs-Assistenten wählen Sie die Option **Einzelnes Dokument (SDI)**. Im zweiten Dialogfeld wählen Sie die Option **Datenbankansicht mit Dateiunterstützung** und

klicken auf die Schaltfläche **Datenquelle**, um die Datenquelle für Ihre Anwendung festzulegen. Im Dialogfeld **Datenbankoptionen** klicken Sie auf die Option ODBC und wählen die für Ihre Access-Datenbank erstellte ODBC- Konfiguration aus dem Listefeld aus, wie es Abbildung 14.5 zeigt. Als Recordset-Typ können Sie entweder **Snapshot** oder **Dynaset** wählen.

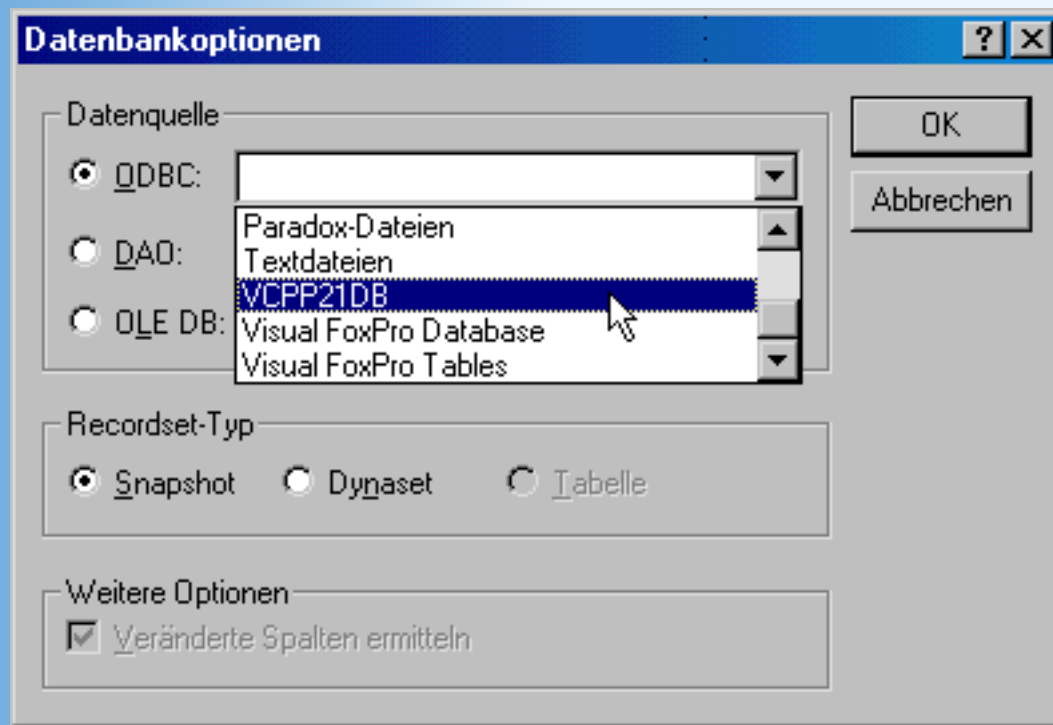


Abbildung 14.5:
Das Dialogfeld Datenbankoptionen

Wenn Sie auf OK klicken, erscheint ein weiteres Dialogfeld, in dem die verfügbaren Tabellen der ausgewählten Datenbank aufgelistet sind. Markieren Sie die Tabelle **Adressen**, wie es Abbildung 14.6 zeigt, und klicken Sie auf OK, um das Dialogfeld zu schließen und zum Anwendungs-Assistenten zurückzukehren.

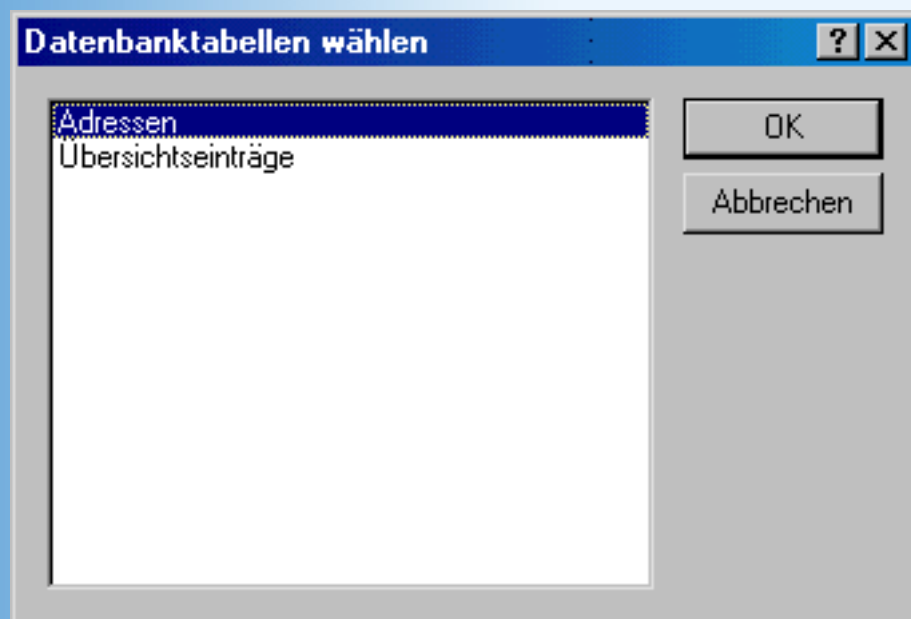


Abbildung 14.6:
Das Dialogfeld Datenbanktabellen wählen

In den übrigen Dialogfeldern des Anwendungs-Assistenten übernehmen Sie die Standardeinstellungen. Das letzte Dialogfeld des Assistenten zeigt, daß der Assistent eine zusätzliche

Klasse angelegt hat. Wenn Sie diese Klasse markieren, sehen Sie, daß sie von der Klasse `CRecordset` abgeleitet ist. Außerdem ist festzustellen, daß die Ansichtsklasse von der Klasse `CRecordView` abgeleitet ist, die ein Nachfolger der Klasse `CFormView` ist und zusätzlich die Datenbankunterstützung enthält.

Das Hauptformular entwerfen

Nachdem Sie das Anwendungsgerüst erstellt haben, entwerfen Sie das Layout des Hauptformulars, in dem Sie die Datensätze aus der Datenbank anzeigen und bearbeiten können. Dieses Formular läßt sich mit den Standardsteuerelementen von Visual C++ erstellen, ohne spezielle ActiveX-Steuerelemente zu benötigen. Für das Layout des Hauptformulars Ihrer Beispielanwendung können Sie sich an Abbildung 14.7 orientieren. Die Steuerelemente konfigurieren Sie mit den in Tabelle 14.6 angegebenen Eigenschaften.



Wenn Sie die Beispielanwendung etwas zeitsparender erstellen möchten, können Sie die meisten Steuerelemente und Datenbankfelder aus der Anwendung weglassen. Die benötigten Schlüsselfelder sind Adressen-Nr, Vorname, Nachname, Geburtsdatum und Karte senden. Auf die anderen Felder können Sie ohne weiteres verzichten.

Tabelle 14.6: Einstellungen der Eigenschaften für die Steuerelemente

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Adressen-Nr:
Eingabefeld	ID	IDC_EID
Text	ID	IDC_STATIC
	Titel	Vorname:
Eingabefeld	ID	IDC_EFNAME
Text	ID	IDC_STATIC
	Titel	Nachname:
Eingabefeld	ID	IDC_ELNAME
Text	ID	IDC_STATIC
	Titel	Name des Ehepartners:
Eingabefeld	ID	IDC_ESNAME
Text	ID	IDC_STATIC
	Titel	Adresse:
Eingabefeld	ID	IDC_EADDR
	Mehrzeilig	eingeschaltet

Text	ID	IDC_STATIC
	Titel	Ort:
Eingabefeld	ID	IDC_ECITY
Text	ID	IDC_STATIC
	Titel	Bundesland:
Eingabefeld	ID	IDC_ESTATE
Text	ID	IDC_STATIC
	Titel	Postleitzahl:
Eingabefeld	ID	IDC_EZIP
Text	ID	IDC_STATIC
	Titel	Land:
Eingabefeld	ID	IDC_ECOUNTRY
Text	ID	IDC_STATIC
	Titel	Email-Adresse:
Eingabefeld	ID	IDC_EEMAIL
Text	ID	IDC_STATIC
	Titel	Telefon/privat:
Eingabefeld	ID	IDC_EHPHONE
Text	ID	IDC_STATIC
	Titel	Telefon/beruflich:
Eingabefeld	ID	IDC_EWPHONE
Text	ID	IDC_STATIC
	Titel	Durchwahl Büro:
Eingabefeld	ID	IDC_EWEXT
Text	ID	IDC_STATIC
	Titel	Faxnummer:
Eingabefeld	ID	IDC_EFAX
Text	ID	IDC_STATIC
	Titel	Geburtsdatum:
Eingabefeld	ID	IDC_EDOB
Kontrollkästchen	ID	IDC_CBCARD
	Titel	Karte senden

Text	ID	IDC_STATIC
	Titel	Anmerkungen:
Eingabefeld	ID	IDC_ENOTES
	Mehrzeilig	eingeschaltet

Abbildung 14.7:
Der Entwurf des Hauptformulars

Nachdem Sie alle Steuerelemente auf dem Hauptformular Ihrer Anwendung hinzugefügt und konfiguriert haben, können Sie die Steuerelemente mit den Feldern der Datenbank verbinden. Wenn Sie im Klassen-Assistenten auf die Registerkarte **Member- Variablen** gehen, ein Steuerelement markieren und auf die Schaltfläche **Variable hinzufügen** klicken, erscheint das Dialogfeld **Member-Variable hinzufügen**. Dieses Dialogfeld enthält ein Dropdown-Kombinationsfeld, in das Sie den Variablennamen eingeben können. Wenn Sie auf den Pfeil der Dropdown-Liste klicken, öffnet sich die Liste, die bereits mit den Feldern des Recordsets gefüllt ist, wie es Abbildung 14.8 zeigt. Damit können Sie die Datenbankfelder direkt mit den Steuerelementen auf dem Formular verbinden. Fügen Sie also die in Tabelle 14.7 angegebenen Variablen hinzu.

Tabelle 14.7: Variablen für die Steuerelemente des Hauptformulars

Objekt	Name
IDC_CBCARD	m_pSet->m_SendenKarte
IDC_EADDR	m_pSet->m_Adresse
IDC_ECITY	m_pSet->m_Ort

IDC_COUNTRY	m_pSet->m_Land
IDC_EEMAIL	m_pSet->m_EmailAdresse
IDC_EFAX	m_pSet->m_Faxnummer
IDC_EFNAME	m_pSet->m_Vorname
IDC_EHPHONE	m_pSet->m_TelefonPrivat
IDC_EID	m_pSet->m_Adressen_Nr
IDC_ELNAME	m_pSet->m_Nachname
IDC_ENOTES	m_pSet->m_Anmerkungen
IDC_ESNAME	m_pSet->m_EhepartnerName
IDC_ESTATE	m_pSet->m_Bundesland
IDC_EWEXT	m_pSet->m_Durchwahl_B_ro
IDC_EWPHONE	m_pSet->m_TelefonBeruflich
IDC_EZIP	m_pSet->m_Postleitzahl

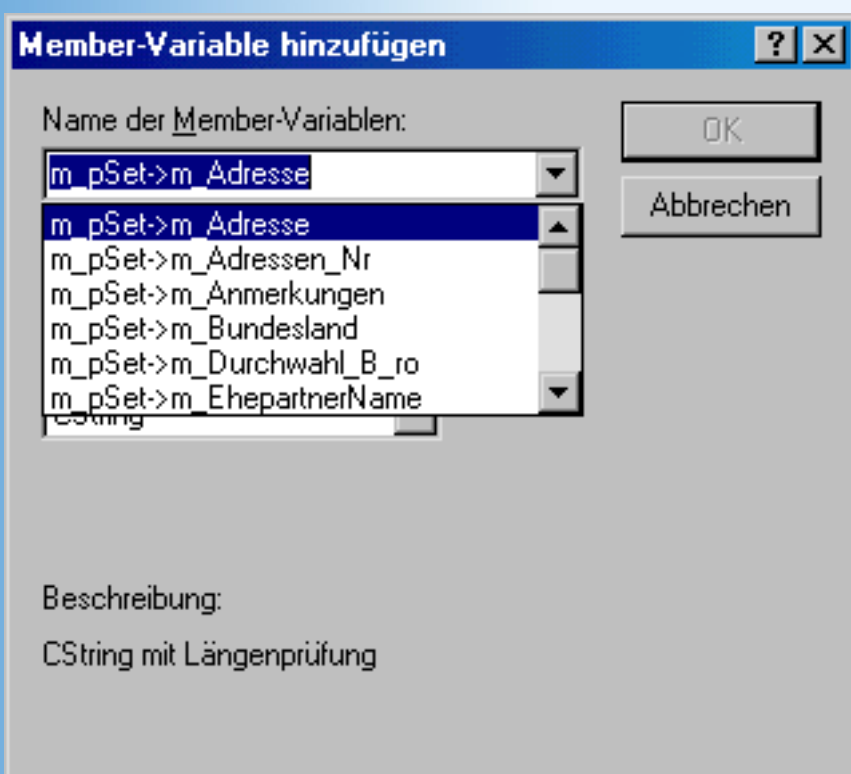


Abbildung 14.8:
Das Dialogfeld Member-Variable hinzufügen mit Datensatzfeldern

Sicherlich haben Sie bemerkt, daß es in der Liste der Datenbankfelder kein Feld für das Geburtsdatum gibt, das Sie mit dem Steuerelement Geburtsdatum verbinden könnten. Wenn Sie sich die Recordset-Klasse in der Klassenansicht ansehen und den Baum erweitern, stellen Sie fest, daß das Geburtsdatum zwar als Datenbankfeld aufgeführt, aber nicht in der Liste der vorhandenen Spalten für die Steuerelemente verfügbar ist. Doppelklicken Sie in der Recordset-Klasse auf das Feld für das Geburtsdatum, um dessen Definition anzuzeigen. Hier ist zu sehen, daß die Variable

`m_Geburtsdatum` mit dem Typ `CTime` deklariert ist. Aus diesem Grund ist die Variable nicht in der Liste der Datenbankfelder aufgeführt, die sich mit den Steuerelementen verbinden lassen. Es gibt weder ein Makro noch eine Funktion, mit denen man Daten zwischen einem Steuerelement und einer Variablen vom Typ `CTime` austauschen könnte. Das ist auch deshalb ein Problem, weil der Variablentyp `CTime` keine Datumswerte vor dem 31. Dezember 1969 behandeln kann. Um dieses Datenbankfeld nutzen zu können, muß man seine Definition vom Typ `CTime` in eine Variable vom Typ `COleDateTime` ändern, wie es in Zeile 17 von Listing 14.1 geschieht. Nachdem Sie den Variablentyp dieses Datenbankfelds geändert haben, können Sie auch die Variable mit dem Steuerelement `IDC_EDOB` verbinden.

Listing 14.1: Die Variablendeklarationen für die Datenbankfelder

```
1: // Feld-/Parameterdaten
2: //{AFX_FIELD(CDbOdbcSet, CRecordset)
3: long m_Adressen_Nr;
4: CString m_Vorname;
5: CString m_Nachname;
6: CString m_EhepartnerName;
7: CString m_Adresse;
8: CString m_Ort;
9: CString m_Bundesland;
10: CString m_Postleitzahl;
11: CString m_Land;
12: CString m_EmailAdresse;
13: CString m_TelefonPrivat;
14: CString m_TelefonBeruflich;
15: CString m_Durchwahl_B_ro;
16: CString m_Faxnummer;
17: COleDateTime m_Geburtsdatum;
18: BOOL m_SendenKarte;
19: CString m_Anmerkungen;
20: //}}AFX_FIELD
```



Normalerweise tasten Sie die Codeabschnitte, die von den verschiedenen Assistenten erzeugt und verwaltet werden, nicht an. Die hier geschilderte Änderung gehört zu den wenigen Ausnahmen von dieser Regel. Man könnte dieses Hindernis als Bug von Visual C++ annehmen, obwohl es aus technischer Sicht kein Bug ist. Datenbankfelder für Datums-/Zeitwerte lassen sich in mehrere Variablentypen umwandeln, wenn man eine Klassenvariable erzeugt, die dieses Feld repräsentiert. `CTime` ist einer dieser Variablentypen, `COleDateTime` ein anderer. Da es sich bei beiden um gültige Auswahlen handelt und die Funktionen zum Füllen dieser Variablen mit beiden Typen arbeiten können, bleibt diese Änderung wahrscheinlich ohne gräßliche Konsequenzen.

Nachdem Sie den Variablentyp für die Variable `m_Geburtsdatum` in der Recordset- Klasse (`CDbOdbcSet`) geändert und dieses Datenbankfeld dem Steuerelement Geburtstag auf dem Formular zugeordnet haben, nehmen Sie vielleicht an, daß Sie die Anwendung nun kompilieren und ausführen können. Leider erhalten Sie eine Fehlermeldung des Compilers, daß `DDX_FieldText` den Variablentyp `COleDateTime` nicht konvertieren kann. Sie müssen also den Code zur Umwandlung in eigener Regie hinzufügen. Kehren Sie zum Klassen-Assistenten zurück, und löschen Sie die

Variable, die Sie für das Steuerelement IDC_EDOB hinzugefügt haben. Nehmen Sie eine neue Variable für dieses Steuerelement auf. Legen Sie für die Variable den Typ `COleDateTime` fest, und geben Sie der Variablen einen Namen wie `m_oledtDOB`. Laden Sie die Funktion `DoDataExchange` der Sichtklasse (`CDbOdbcView`) in den Editor, und fügen Sie die Zeilen 4 bis 6 sowie die Zeilen 25 bis 28 gemäß Listing 14.2 in die Funktion ein.

Listing 14.2: Die Funktion `DoDataExchange` der Klasse `CDbOdbcView`

```

1: void CDbOdbcView::DoDataExchange(CDataExchange* pDX)
2: {
3: CRecordView::DoDataExchange(pDX);
4: // Geburtsdatum aus Recordset in Ansichtsvariable kopieren
5: if (pDX->m_bSaveAndValidate == FALSE)
6: m_oledtDOB = m_pSet->m_Geburtsdatum;
7: //{{AFX_DATA_MAP(CDbOdbcView)
8: DDX_FieldText(pDX, IDC_EADDR, m_pSet->m_Adresse, m_pSet);
9: DDX_FieldCheck(pDX, IDC_CBCARD, m_pSet->m_SendenKarte, m_pSet);
10: DDX_FieldText(pDX, IDC_ECITY, m_pSet->m_Ort, m_pSet);
11: DDX_FieldText(pDX, IDC_ECOUNTRY, m_pSet->m_Land, m_pSet);
12: DDX_FieldText(pDX, IDC_EEMAIL, m_pSet->m_EmailAdresse, m_pSet);
13: DDX_FieldText(pDX, IDC_EFAX, m_pSet->m_Faxnummer, m_pSet);
14: DDX_FieldText(pDX, IDC_EFNAME, m_pSet->m_Vorname, m_pSet);
15: DDX_FieldText(pDX, IDC_EHPHONE, m_pSet->m_TelefonPrivat,
m_pSet);
16: DDX_FieldText(pDX, IDC_EID, m_pSet->m_Adressen_Nr, m_pSet);
17: DDX_FieldText(pDX, IDC_ELNAME, m_pSet->m_Nachname, m_pSet);
18: DDX_FieldText(pDX, IDC_ENOTES, m_pSet->m_Anmerkungen, m_pSet);
19: DDX_FieldText(pDX, IDC_ESNAME, m_pSet->m_EhepartnerName,
m_pSet);
20: DDX_FieldText(pDX, IDC_ESTATE, m_pSet->m_Bundesland, m_pSet);
21: DDX_FieldText(pDX, IDC_EWEXT, m_pSet->m_Durchwahl_B_ro, m_pSet);
22: DDX_FieldText(pDX, IDC_EWPHONE, m_pSet->m_TelefonBeruflich,
m_pSet);
23: DDX_FieldText(pDX, IDC_EZIP, m_pSet->m_Postleitzahl, m_pSet);
24: DDX_Text(pDX, IDC_EDOB, m_oledtDOB);
25: //}}AFX_DATA_MAP
26: // Geburtsdatum aus der Ansichtsvariablen zurück in den
Recordset kopieren
27: if (pDX->m_bSaveAndValidate == TRUE)
28: m_pSet->m_Geburtsdatum = m_oledtDOB;
29: }
```

Zusätzlich zu den obigen Änderungen ist noch die Initialisierung der Variablen `m_Geburtsdatum` in der `Recordset`-Klasse zu entfernen. Diesen Code hat ebenfalls der Anwendungs-Assistenten hinzugefügt. Auch hier müssen Sie die Regel durchbrechen, den Code zu modifizieren, den Sie eigentlich niemals berühren sollten. Um diese Änderung durchzuführen, können Sie einfach die Initialisierung dieser Variablen im Konstruktor der `Recordset`-Klasse auskommentieren, wie es Zeile 19 von Listing 14.3 zeigt.

Listing 14.3: Der Konstruktor von `CDbOdbcSet`

```

1: CDbOdbcSet::CDbOdbcSet(CDatabase* pdb)
2: : CRecordset(pdb)
3: {
4: //{{AFX_FIELD_INIT(CDbOdbcSet)
5: m_Adressen_Nr = 0;
6: m_Vorname = _T("");
7: m_Nachname = _T("");
8: m_EhepartnerName = _T("");
9: m_Adresse = _T("");
10: m_Ort = _T("");
11: m_Bundesland = _T("");
12: m_Postleitzahl = _T("");
13: m_Land = _T("");
14: m_EmailAdresse = _T("");
15: m_TelefonPrivat = _T("");
16: m_TelefonBeruflich = _T("");
17: m_Durchwahl_B_ro = _T("");
18: m_Faxnummer = _T("");
19: //m_Geburtsdatum = 0;
20: m_SendenKarte = FALSE;
21: m_Anmerkungen = _T("");
22: m_nFields = 17;
23: //}}AFX_FIELD_INIT
24: m_nDefaultType = snapshot;
25: }

```

Jetzt können Sie die Anwendung erneut kompilieren und ausführen. Sie haben nun eine voll funktionsfähige Datenbankanwendung vor sich, die eine Menge von Datensätzen aus der Datenbank abrufen und es ermöglicht, durch diese Datensätze zu blättern und Änderungen an den Daten vorzunehmen (siehe Abbildung 14.9).

Unbenannt - DbOdbc

Datei Bearbeiten Datensatz Ansicht ?

Adressen-Nr: 1

Vorname: Nicole

Nachname: Dauster

Name des Ehepartners: Paul

Adresse: Hofstr. 1
App. 2a

Ort: Hamburg

Bundesland:

Postleitzahl: 21079

Land: Deutschland

Email-Adresse: nicoled@irgendwo.co

Telefon/privat: (040)5559857

Telefon/beruflich: (040)5559922

Durchwahl Büro:

Faxnummer: (040)5557722

Geburtsdatum: 04.08.72

☐ Karte senden

Anmerkungen:

Bereit NUM

Abbildung 14.9:
Die laufende Anwendung

Neue Datensätze hinzufügen

Sie haben bereits eine voll funktionsfähige Datenbankanwendung erstellt, ohne eine einzige Zeile Code zu schreiben. Allerdings fehlen noch ein paar Funktionen. In den meisten Datenbankanwendungen kann der Benutzer neue Datensätze in die Datenbank aufnehmen. Um einen neuen Datensatz in die Datenbank hinzuzufügen, müssen Sie herausfinden, wie die nächste ID-Nummer lauten soll. Gehen Sie zum letzten Datensatz im Recordset, um die ID zu ermitteln, und inkrementieren Sie diese um 1. Als nächstes rufen Sie die Funktion `AddNew` auf, um einen neuen Datensatz hinzuzufügen, setzen das ID-Feld auf die neu berechnete ID und rufen dann die Funktion `Update` auf, um den neuen Datensatz zu speichern. Schließlich rufen Sie die Funktion `Requery` auf, um die Ergebnismenge zu aktualisieren, und gehen dann zum letzten Datensatz im Recordset, um dem Benutzer die Eingabe von Daten in den neuen Datensatz anzubieten.



Da das ID-Feld in der Datenbank als `AutoWert`-Feld definiert ist, legen Sie normalerweise die ID für das Feld nicht selbst fest. Da allerdings der Recordset einen neuen Datensatz mit dem ID-Feld erzeugt, müssen Sie dem Datensatz eine gültige ID zuweisen, da Sie den Datensatz ansonsten nicht in die Datenbank aufnehmen können. Die in der Beispielanwendung verwendete Methode funktioniert nicht in einer Mehrbenutzerdatenbank, da jeder Benutzer dieselbe ID für neue Datensätze erzeugen würde. In dieser Situation stellt eine zentrale Methode für das Generieren neuer IDs, wie etwa ein Zählerfeld in der Datenbank, die bessere Lösung dar. Weiterhin haben Sie die Möglichkeit, eine SQL-Anweisung zu erstellen, die einen neuen Datensatz in die Datenbank einfügt. In dieser Anweisung lassen Sie das ID-Feld aus, so daß die `AutoWert`-Funktion korrekt arbeiten kann.

Um die beschriebene Funktionalität in der Beispielanwendung zu realisieren, fügen Sie zunächst in die Recordset-Klasse eine Funktion ein, mit der Sie die nächste zu verwendende ID-Nummer bestimmen. Nehmen Sie dazu eine Member-Funktion in die Recordset-Klasse (`CDbOdbcSet`) auf. Legen Sie den Funktionstyp als `long`, die Funktionsdeklaration mit `GetMaxID` und den Zugriffsstatus als `Public` fest. Schreiben Sie den Code gemäß Listing 14.4 in die Funktion.

Listing 14.4: Die Funktion `GetMaxID` der Klasse `CDbOdbcSet`

```
1: long CDbOdbcSet::GetMaxID( )
2: {
3: // Zum letzten Datensatz gehen
4: MoveLast();
5: // ID dieses Datensatzes zurückgeben
6: return m_Adressen_Nr;
7: }
```

Als nächstes stattdessen Sie das Menü Datensatz mit einem neuen Befehl aus, damit der Benutzer einen neuen Datensatz in die Datenbank aufnehmen kann. Konfigurieren Sie den neuen Menübefehl mit den Eigenschaften gemäß Tabelle 14.8.

Tabelle 14.8: Eigenschaftseinstellungen für das Menü

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_RECORD_NEW
	Titel	Neuer &Datensatz
	Statuszeilentext	Fügt einen neuen Datensatz hinzu\nNeuer Datensatz

Mit dem Klassen-Assistenten erstellen Sie eine Behandlungsroutine für die Nachricht `COMMAND` dieses Menüs in der Ansichtsklasse `CDbOdbcView`. In die Funktion übernehmen Sie den Code aus Listing 14.5.

Listing 14.5: Die Funktion `OnRecordNew` der Klasse `CDbOdbcView`

```
1: void CDbOdbcView::OnRecordNew( )
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
```

```

4: // Zeiger auf Recordset holen
5: CRecordset* pSet = OnGetRecordset();
6: // Sicherstellen, daß alle Änderungen am aktuellen
7: // Datensatz gespeichert wurden.
8: if (pSet->CanUpdate() && !pSet->IsDeleted())
9: {
10: pSet->Edit();
11: if (!UpdateData())
12: return;
13:
14: pSet->Update();
15: }
16: // ID für den neuen Datensatz ermitteln
17: long m_lNewID = m_pSet->GetMaxID() + 1;
18: // Den neuen Datensatz hinzufügen
19: m_pSet->AddNew();
20: // Die ID in den neuen Datensatz schreiben
21: m_pSet->m_Adressen_Nr = m_lNewID;
22: // Den neuen Datensatz speichern
23: m_pSet->Update();
24: // Den Recordset aktualisieren
25: m_pSet->Requery();
26: // Zum neuen Datensatz gehen
27: m_pSet->MoveLast();
28: // Formular aktualisieren
29: UpdateData(FALSE);
30: }

```

Korrespondierend zum Menübefehl **Neuer Datensatz** fügen Sie eine neue Schaltfläche in die Symbolleiste ein. Dann können Sie Ihre Anwendung kompilieren und ausführen. Nunmehr sollte es möglich sein, neue Datensätze in die Datenbank aufzunehmen, wobei Sie in die neuen Datensätze Daten eingeben können.

Datensätze löschen

Als letztes ist noch die Funktionalität zum Löschen des aktuellen Datensatzes aus der Datenbank zu implementieren. Um diese Aktion auszulösen, fügen Sie einen neuen Menübefehl hinzu. Wenn der Benutzer diesen Befehl wählt, fragen Sie zur Sicherheit nach, ob er den aktuellen Datensatz wirklich löschen möchte, und rufen dann die Funktion **Delete** auf, um den Datensatz zu entfernen. Anschließend rufen Sie die Funktion **MovePrev** auf, um zum vorherigen Datensatz im Recordset zu navigieren.

Diese Funktionalität realisieren Sie über einen neuen Menübefehl, den der Benutzer auswählen kann, um den aktuellen Datensatz aus der Datenbank zu löschen. Fügen Sie den neuen Befehl in das Menü **Datensatz** ein, und konfigurieren Sie den Befehl mit den Eigenschaften gemäß Tabelle 14.9.

Tabelle 14.9: Eigenschaftseinstellungen für das Menü

Objekt	Eigenschaft	Einstellung

Menübefehl	ID	IDM_RECORD_DELETE
	Titel	Datensatz l&öschen
	Statuszeilentext	Löscht den aktuellen Datensatz \nDatensatz löschen

Mit dem Klassen-Assistenten erstellen Sie eine Behandlungsroutine für die Nachricht **COMMAND** dieses Menübefehls in der Ansichtsklasse **CDbOdbcView**. In die Funktion schreiben Sie den Code aus Listing 14.6.

Listing 14.6: Die Funktion OnRecordDelete der Klasse CDbOdbcView

```

1: void CDbOdbcView::OnRecordDelete()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4: // Fragen, ob Benutzer wirklich den Datensatz löschen will
5: if (MessageBox("Möchten Sie diesen Datensatz wirklich löschen?",
6: "Diesen Datensatz löschen?", MB_YESNO | MB_ICONQUESTION) ==
IDYES)
7: {
8: // Den Datensatz löschen
9: m_pSet->Delete();
10: // Zum vorherigen Datensatz gehen
11: m_pSet->MovePrev();
12: // Formular aktualisieren
13: UpdateData(FALSE);
14: }
15: }
```

In die Symbolleiste nehmen Sie eine neue Schaltfläche auf und verbinden sie mit der Menü-ID **IDM_RECORD_DELETE**, damit der Benutzer den aktuellen Datensatz löschen kann, ohne erst über das Menü zu gehen. Wenn Sie jetzt Ihre Anwendung kompilieren und ausführen, haben Sie eine voll funktionsfähige Datenbankenanwendung vor sich, in der Sie Datensätze hinzufügen, bearbeiten und löschen können (siehe Abbildung 14.10).

The screenshot shows a Windows application window titled "Unbenannt - DbOdbc". The window has a menu bar with "Datei", "Bearbeiten", "Datensatz", and "Ansicht". Below the menu is a toolbar with icons for file operations and navigation. The main area contains a form with two columns of input fields. The left column includes fields for "Adressen-Nr.", "Vorname", "Nachname", "Name des Ehepartners", "Adresse", "Ort", "Bundesland", "Postleitzahl", and "Land". The right column includes fields for "Email-Adresse", "Telefon/privat", "Telefon/beruflich", "Durchwahl Büro", "Faxnummer", "Geburtsdatum", a checkbox for "Karte senden", and a large text area for "Anmerkungen". The status bar at the bottom shows "Bereit" and a "NUM" button.

Adressen-Nr:	6	Email-Adresse:	davischa@onramp.ne
Vorname:	Davis	Telefon/privat:	123-456-7890
Nachname:	Chapman	Telefon/beruflich:	987-654-3210
Name des Ehepartners:	Dore	Durchwahl Büro:	123
Adresse:	123 Irgendwo	Faxnummer:	987-654-6211
Ort:	Nirgendwo	Geburtsdatum:	10.01.61
Bundesland:	TX	☒ Karte senden	
Postleitzahl:	75222	Anmerkungen:	
Land:			

Abbildung 14.10:
Die fertiggestellte Anwendung

Zusammenfassung

Die heutige Lektion hat gezeigt, wie man mit Hilfe der ODBC-Schnittstelle Datenbankanwendungen erstellt, die sich in Verbindung mit einer beliebigen Datenbank einsetzen lassen. Sie haben gesehen, wie die Klasse `CRecordset` einen Großteil der Funktionalität bereitstellt, so daß Sie Datenbankaktionen in Ihre Anwendungen einbauen können. Weiterhin wurde erläutert, wie der Anwendungs-Assistent bereits einen wesentlichen Teil der Datenbankfunktionalität realisiert, ohne daß Sie eine einzige Codezeile schreiben müssen.

Morgen geht es um die neueste Technologie für den Datenbankzugriff von Microsoft, ActiveX Data Objects. Sie werden erfahren, wie man diese Technologie in Verbindung mit der ODBC-Schnittstelle einsetzt, um den Zugriff auf Datenbanken noch leichter zu realisieren.

Fragen und Antworten

Frage:

Warum soll ich mit der ODBC-Schnittstelle arbeiten und nicht mit den Datenzugriffsobjekten (DAO)?

Antwort:

Die Datenzugriffsobjekte stützen sich auf die Microsoft Jet Database Engine, um den gesamten Datenbankzugriff durchzuführen. Durch den zusätzlichen Verwaltungsaufwand (Overhead) bläht sich Ihre Anwendung um mindestens 1 Mbyte auf. Falls Sie mit einer SQL-basierten Datenbank arbeiten, kann die Datenbank bereits alle Aufgaben realisieren, die die Jet Engine ausführt. Darüber hinaus verwendet die Jet Database Engine die ODBC-Schnittstelle, um auf SQL-basierte Datenbanken zuzugreifen. Solange Sie also mit PC-basierten Datenbanken wie Access, FoxPro oder Paradox arbeiten, erzielen Sie eine bessere Leistungsbilanz, wenn Sie direkt in eigener Regie über die ODBC-Schnittstelle arbeiten.

Frage:
Wie kann ich unterschiedliche Recordsets in eine MDI-Anwendung aufnehmen?

Antwort:
 Mit dem Klassen-Assistenten können Sie zusätzliche, von CRecordset abgeleitete Klassen in ein MDI-Anwendungsprojekt aufnehmen. Dazu legen Sie fest, daß die neue Klasse eine MFC-Klasse ist und als Basisklasse die Klasse CRecordset verwendet. Der Klassen-Assistent fordert die Angabe der Datenquelle, genau wie Sie beim Anwendungs-Assistenten das Anwendungsgerüst für die heutige Beispielanwendung erstellt haben. Nachdem Sie die Recordset-Klasse erzeugt haben, können Sie eine neue Klasse auf die gleiche Weise anlegen, wobei Sie CRecordView als Basisklasse spezifizieren. Wenn Sie auf OK klicken, müssen Sie im Klassen-Assistenten festlegen, welche der Recordset-Klassen mit der neuen Datensatzansichtsklasse zu verwenden ist.

Workshop

Kontrollfragen

1. Wofür steht ODBC?
2. Mit welchen Funktionen können Sie durch den Recordset in einem CRecordset- Objekt navigieren?
3. Welche Ansichtsklasse sollten Sie für Ihre ODBC-Anwendung wählen?
4. Welche Funktionssequenz ist aufzurufen, um einen neuen Datensatz in einen Recordset einzufügen?
5. Welche Funktion müssen Sie aufrufen, bevor sich Felder im CRecordset-Objekt mit allen Änderungen aktualisieren lassen?

Übung

Fügen Sie einen Menübefehl und ein Dialogfeld hinzu, um dem Benutzer die Möglichkeit zu geben, eine Datensatznummer als Ziel einzugeben. Verschieben Sie dann den Datensatzzeiger auf diesen Datensatz.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 2 im Rückblick

Mit Abschluß der zweiten Woche besitzen Sie genügend Kenntnisse, um einigermaßen flüssig mit Visual C++ arbeiten zu können. Einen ersten Eindruck von der MFC-Klassenhierarchie haben Sie bereits gewonnen und wissen nun, wie man die vorhandene Funktionalität in eigenen Anwendungen bereitstellt. Außerdem kennen Sie jetzt die Grundzüge der unterstützenden Infrastruktur, die die Assistenten von Visual C++ einer Anwendung »als Startpaket« mit auf den Weg geben.

An dieser Stelle empfiehlt es sich, das bisher Gelernte selbst auszuprobieren. Erstellen Sie eine MDI-Anwendung, und setzen Sie dazu einen Dokumenttyp ein, den Sie selbst entwickeln. Sehen Sie sich an, wie man das Dokument speichern und wiederherstellen sowie verwalten kann. Eigene Übungen sind der Schlüssel dazu, Ihre bisherigen Kenntnisse zu vertiefen. Dabei erkennen Sie auch Schwachstellen und können Themen wiederholen, die Ihnen noch nicht geläufig sind, aber Sie wissen auch, wo Sie bereits sattelfest sind.

Die Dokument/Ansicht-Architektur ist Ihnen kein Buch mit sieben Siegeln mehr. Mit dieser Architektur verwalten Sie die Trennung der eigentlichen Daten von der Darstellung der Daten, die für den Benutzer angezeigt werden. Auf der Basis dieses Modells haben Sie sowohl SDI- (Single Document Interface) als auch MDI- (Multiple Document Interface) Anwendungen entwickelt und eingesetzt, um Dateien auf den Datenträger zu schreiben und wieder in die Anwendung einzulesen. Diese Architektur gehört zu den Grundpfeilern beim Erstellen von MFC-Anwendungen mit Visual C++. Sie wissen, wo man die Initialisierungen für einen neuen Satz von Daten unterbringt, und wo man Aufräumarbeiten erledigen muß, wenn man einen Satz von Daten schließt.

Weiterhin kennen Sie die Grundzüge von SDI- und MDI-Anwendungen und wissen, wie sie sich untereinander und von dialogbasierten Anwendungen unterscheiden. Sie haben bereits ein Gefühl dafür entwickelt, wann eine zu erstellende Anwendung in einem dieser Stile zu erstellen ist und wann sich ein anderer Stil anbietet. Sie können problemlos Ihre eigenen SDI- und MDI-Anwendungen je nach Ihren Anforderungen erstellen. Falls noch irgendwelche Fragen zu diesen Themen auftauchen, sehen Sie sich noch einmal in den Lektionen 10 und 11 an, wie die Dokument/Ansicht-Architektur sowohl in SDI- als auch in MDI-Anwendungen funktioniert.

Zu Ihren Kenntnissen gehört mittlerweile auch, daß Sie in SDI- und MDI-Anwendungen komplexe Datenstrukturen in Dateien auf der Festplatte des Systems speichern und in die Anwendung einlesen können. Sie erstellen Objekte mit gemischten Typen, die Sie im Dokumentobjekt Ihrer Anwendung erzeugen und verwalten. Dazu gehört auch der Umgang mit der Funktion `Serialize` in Verbindung mit dem `CArchive`-Objekt, um die Objekte in eine Datei zu schreiben und die Objekte zu einem späteren Zeitpunkt wiederherstellen zu können. Bei Problemen in diesem Themenbereich wiederholen Sie Tag 13.

Beim Themenkomplex Lesen und Schreiben von Dateien haben Sie gelernt, wie man Symbolleisten für SDI- und MDI-Anwendungen entwirft und erstellt. Nunmehr können Sie eigene Symbolleisten entwerfen und erstellen sowie in Ihren Anwendungen einsetzen. Ihnen ist bekannt, daß die ID der

Symbolleisten-Schaltfläche mit der ID des Menüs, für das die Schaltfläche als Ersatz gedacht ist, übereinstimmen muß. Außerdem wissen Sie, wie man eigene benutzerdefinierte Elemente in der Statusleiste von SDI- und MDI-Anwendungen erzeugt und verwendet. Die Nachricht `UPDATE_COMMAND_UI` setzen Sie ein, um den Status von Elementen im Menü, auf Symbolleisten und in der Statusleiste auszuwerten und zu ändern sowie deren Aussehen und Zustand in eigener Regie zu verwalten, wobei Sie diese Elemente nicht selbst einrichten müssen. Wenn Sie bezüglich dieser Dinge noch Fragen haben, gehen Sie noch einmal zu Tag 12 zurück.

Der letzte Tag dieser Woche hat eine einfache Datenbankanwendung vorgestellt, mit der Sie Daten aus einer Datenbank über die ODBC-Schnittstelle abrufen. Ihnen sind jetzt die Grundlagen bekannt, wie man Datenbankanwendungen mit diesem Verfahren erstellt, wie man die Daten verwaltet, wie man neue Datensätze hinzufügt und wie man Datensätze löscht. Sie wissen, wie die `Recordset`-Klasse die gesamte Interaktion mit der Datenbank abwickelt und wie Sie die Daten über dieses Objekt direkt beeinflussen. Falls Ihnen in diesem Themenbereich noch etwas unklar ist, wiederholen Sie den Stoff von Tag 14.

Es wurde gezeigt, wie einfach man ActiveX-Steuerelemente in ein Projekt aufnehmen kann und wie Visual C++ um das Steuerelement herum C++-Klassen aufbaut, die Ihnen die Interaktion mit dem Steuerelement wie mit jedem anderen C++-Objekt ermöglichen. Sie können nun beliebige ActiveX-Steuerelemente (ausgerüstet mit der Dokumentation für das Steuerelement) in Ihre Anwendung einbauen und nahtlos mit ihnen zusammenarbeiten. Dabei deklarieren Sie Variablen für das Steuerelement, stellen die Eigenschaften ein, rufen dessen Methoden auf und reagieren auf seine Ereignisse - genau wie bei einem Standardelement der Entwicklungsumgebung von Visual C++. Wenn bei Ihnen diesbezüglich noch Fragen auftauchen, gehen Sie noch einmal zurück zu Tag 9.

Die Woche haben Sie mit einer Zeichenanwendung eingeleitet und dabei gelernt, wie man Linien, Kreise und Quadrate mit einer Vielzahl von Stiften und Pinseln zeichnet. Sie haben sogar einen benutzerdefinierten Pinsel aus einem Bitmap erstellt. Es wurde erläutert, wie man ein Bitmap aus einer Datei lädt und für den Benutzer anzeigt. Vor allem aber ging es darum, was ein Gerätekontext ist und wie man ihn einsetzt, um alle diese Features im Fenster der Anwendung darzustellen. Mit den Methoden eines Gerätekontextes können Sie nun diese und andere Figuren zeichnen, um beliebige Grafiken im Fenster zu erstellen, die der Benutzer ansehen und damit in Wechselwirkung treten kann. Wenn hierzu noch Fragen offen sind, wiederholen Sie Tag 8.

Mittlerweile haben Sie sich ein umfangreiches Repertoire an Programmierkenntnissen mit Visual C++ erarbeitet. Kleinere Programmieraufgaben können Sie nun ohne weiteres lösen und sich möglicherweise auch bereits mit etwas größeren Programmen auseinandersetzen. Sie sind auf dem besten Weg, ein perfekter Visual C++-Programmierer zu werden!

Woche 3 im Überblick

In der dritten und letzten Woche lernen Sie einige der komplizierteren Aspekte beim Erstellen von Anwendungen mit Visual C++ kennen. In Ihrer weiteren Programmierpraxis haben Sie mit manchen Themen öfter zu tun als mit anderen. Wenn Visual C++ aber zu Ihrem täglichen Brot gehört, kommen Sie doch auf lange Sicht mit fast allen dieser Bereiche in Berührung.

Die Woche setzt da ein, wo wir in der letzten Woche aufgehört haben - mit dem Erstellen von Datenbankanwendungen. Am Tag 15 geht es um die neueste Datenbanktechnologie von Microsoft, die ActiveX Data Objects (ADO) und wie man sie in Visual C++ einbindet, um dem Benutzer den Datenbankzugriff zu ermöglichen. Sie erfahren, welche Gemeinsamkeiten und Unterschiede beim Einsatz von ADO gegenüber dem Erstellen von Datenbankanwendungen mit ODBC vorhanden sind.

Am Tag 16 erstellen Sie eigene, benutzerdefinierte Klassen und erfahren, wie man den Entwurf dieser Klassen angeht. Weiterhin verlagern Sie funktionelle Bausteine in Bibliothekmodule, die Sie an andere Visual C++-Programmierer für deren Anwendungen weitergeben können.

Am siebzehnten Tag lernen Sie ein weiteres Mittel kennen, um anderen Programmierern Ihren Code zugänglich zu machen: DLLs. Dabei erstellen Sie zwei verschiedene Arten von DLLs: der eine Typ läßt sich nur zusammen mit anderen Visual C++-Anwendungen einsetzen, den anderen kann man mit jeder anderen Entwicklungssprache von Windows nutzen.

Tag 18 zeigt, wie sich in einer Anwendung zwei oder mehrere separate Aufgaben gleichzeitig ausführen lassen. Sie lösen Hintergrundverarbeitungen aus, sobald eine Anwendung im Leerlauf arbeitet, und erstellen unabhängige Threads (Programmfäden), die selbst dann arbeiten, wenn die Anwendung beschäftigt ist.

In der neunzehnten Lektion lernen Sie, wie Sie eigene ActiveX-Steuerelemente erstellen, die sich in anderen Anwendungen oder sogar in Webseiten einsetzen lassen. Sie definieren die Eigenschaften und Methoden für das Steuerelement und lösen über Ihr Steuerelement Ereignisse in der Containeranwendung aus.

Tag 20 erläutert, wie Internet-Anwendungen über die Winsock-Schnittstelle untereinander kommunizieren. Sie setzen die gleiche Schnittstelle ein, um in Ihren Anwendungen den Datenaustausch über ein Netzwerk oder auf ein und derselben Maschine zu realisieren.

Schließlich demonstriert Tag 21, wie einfach es ist, den Webbrowser Internet Explorer von Microsoft in eigene Visual C++-Anwendungen zu integrieren. Dabei geht es um die Steuerung des Webbrowsers, Sie legen die anzuzeigenden Webseiten fest und bieten dem Benutzer Informationen darüber, was der Browser gerade ausführt.

Wenn Sie diese letzte Woche abgeschlossen haben, besitzen Sie Kenntnisse über die meisten Bereiche der Visual C++-Programmierung. Obwohl es noch weitere Gebiete und Technologien gibt, die ein tieferes Eindringen erfordern, haben Sie zumindest eine Vorstellung, worum es im wesentlichen geht.

Sie sind darauf vorbereitet, in allen Bereichen der Visual C++-Programmierung zu arbeiten. Und mittlerweile haben Sie vielleicht einige Gebiete herausgegriffen, über die Sie mehr erfahren möchten als dieses Buch Ihnen vermitteln kann.

Eine Woche ist noch zu meistern, gehen Sie es an.

❖Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 3

Tag 15

Datenbanken per ADO bearbeiten

Nachdem Sie einen Eindruck von der Arbeitsweise einer Anwendung mit einer ODBC-Datenbank - einer der ältesten Technologien des Datenbankzugriffs von Microsoft - gewonnen haben, wenden wir uns nun der neuesten Datenbanktechnologie von Microsoft zu, den ActiveX Data Objects (ADO). ADO ist für alle Programmier- und Skriptingwerkzeuge von Microsoft vorgesehen und bietet dem Visual C++-Programmierer neue Möglichkeiten bei der Datenbankprogrammierung, wobei gleichzeitig die Funktionalität vertraut bleibt. Heute lernen Sie, wie ...

- ADO arbeitet und wie es auf die Technologie OLE DB zurückgreift, um einen einfachen Datenbankzugriff zu bieten,
- man einfache ADO-Anwendungen in wenigen Minuten mit Hilfe von ActiveX-Steurelementen erstellt,
- man eine komplette Datenbankanwendung mit regulären Formularsteuerelementen aufbaut,
- sich mit speziellen ADO-Makros benutzerdefinierte Datensatzklassen für Datenbankanwendungen erstellen lassen.



In diesem Kapitel kommen einige Merkmale zum Einsatz, die nicht in allen Versionen von Visual C++ enthalten sind. Obwohl ADO ein wichtiges neues Gebiet der Programmierung mit den Technologien des Datenbankzugriffs von Microsoft darstellt, behandelt dieses Kapitel bestimmte Dinge, die Sie mit Ihrer Version von Visual C++ eventuell nicht nachvollziehen können.

Was ist ADO?

Microsoft hat vor einigen Jahren OLE DB als neue Technologie des Datenbankzugriffs auf den Markt gebracht. Diese Technologie zielte nicht nur darauf ab, Daten einfach mit Datenbanken auszutauschen, sondern sollte auch den Zugriff auf Daten ermöglichen, die an einer beliebigen Stelle gespeichert sind. Über die Technologie OLE DB konnte man auf Mail-Nachrichten, Tabellenblätter, Dateien usw. zugreifen - kurz gesagt, auf alles, was in irgendeiner Form mit Daten zu tun hatte. Diese Technologie war eine der ersten, die zur Forschung und Entwicklung des objektorientierten Dateisystems gehörte, das bei Microsoft in den vergangenen Jahren unter der Bezeichnung »Cairo« firmierte.

Wie man sich gut vorstellen kann, ist es bei der umfangreichen Funktionalität, die OLE DB für den

Zugriff auf Daten in den unterschiedlichsten Quellen braucht, ziemlich kompliziert, mit dieser Technologie zu arbeiten. An dieser Stelle kommen die ActiveX Data Objects ins Spiel. Das Konzept von ADO stellt praktisch eine zusätzliche Schicht über OLE DB dar, die speziell für den Datenbankzugriff vorgesehen ist.

Eines der Ziele bei der Entwicklung von ADO bestand darin, ein Steuerelement zu schaffen, das den Datenzugriff und die Steuerung in Webseiten gestattet, wobei die Datensätze auf dem Client zwischengespeichert werden. Zum Teil begründet sich dieses Ziel damit, daß der Benutzer eines Webbrowsers auf eine ganze Gruppe von Datensätzen zugreifen kann, ohne die einzelnen Datensätze nacheinander herunterladen zu müssen, um durch die Datensätze navigieren und Änderungen daran vornehmen zu können. Aus diesen Gründen gehört das ADO-Steuerelement zum Lieferumfang des Webbrowsers Internet Explorer von Microsoft (ab der Version 4.0 aufwärts).

ADO-Objekte

Um den Einsatz von ADO in Skriptsprachen wie VBScript genauso einfach zu gestalten wie in Programmierungsumgebungen wie Visual Basic, hat Microsoft versucht, die Anzahl der Objekte auf einem Minimum zu halten. Im Ergebnis haben Sie es mit einer kleinen Anzahl von Basisobjekten zu tun:

- **Connection**
- **Error**
- **Command**
- **Parameter**
- **Recordset**
- **Field**

Daneben existieren noch Sammelobjekte für die Aufnahme von Kollektionen der Objekte **Error**, **Parameter** und **Field**.

Das Objekt Connection

Das Objekt **Connection** dient dazu, eine Verbindung zu einer Datenbank einzurichten und zu verwalten. Man konfiguriert das Objekt mit der Verbindungsinformation, einschließlich dem Standort der Datenbank, der Benutzer-ID und dem Kennwort, bevor man die Verbindung öffnet. Wenn alle Angaben ordnungsgemäß konfiguriert sind, sollte das Verbindungsobjekt seine **Open**-Methode aufrufen, um die Verbindung zu öffnen. Sobald das **Connection**-Objekt seinen Gültigkeitsbereich verliert, wird die Verbindung automatisch geschlossen. Wenn Sie weitergehenden Einfluß auf das Schließen und Öffnen der Datenbankverbindung nehmen möchten, können Sie auch die **Close**-Methode des Objekts **Connection** aufrufen, um die Verbindung zu schließen.

Über das **Connection**-Objekt wird auch die gesamte Funktionalität der höheren Ebenen gesteuert. Dazu gehört die Kontrolle der Transaktionsverarbeitung mit den Methoden **BeginTrans**, **CommitTrans** und **RollbackTrans** des **Connection**-Objekts.

Das Objekt Error

Bei jedem Datenbankfehler werden die Fehlerinformationen aus der Datenbank in ein ADO **Error**-Objekt geschrieben. Bei den Fehlerinformationen im **Error**-Objekt handelt es sich um die Fehlerinformationen aus der Datenbank und nicht um Fehlerinformationen von ADO. Wenn Sie einen Fehler erhalten und anhand der Fehlerinformationen die Ursachen ermitteln wollen, müssen Sie die

Fehlercodes der Datenbank und nicht die von ADO prüfen.

Das Objekt Command

Über das Objekt **Command** werden die Befehle in der Datenbank ausgeführt. Mit diesem Objekt lassen sich SQL-Anweisungen ausführen oder gespeicherte Prozeduren (SQL- Funktionen, die in der Datenbank gespeichert sind) aufrufen. Bei jedem Befehl, der Datenzeilen zurückgibt, müssen Sie das **Command**-Objekt mit einem **Recordset**-Objekt verbinden, in dem sich die zurückgegebenen Daten ablegen lassen.

Beim Aufruf einer gespeicherten Prozedur sind oftmals - wie bei Funktionen in anderen Programmiersprachen - Parameter zu übergeben. Dazu weist man dem **Command**- Objekt eine Reihe von **Parameter**-Objekten zu. Jedes **Parameter**-Objekt verfügt über den Namen des Parameters, für den es den Wert aufnimmt, sowie den Wert, der an die Datenbank für den betreffenden Parameter zu übergeben ist.

Das Objekt Parameter

Mit dem **Parameter**-Objekt lassen sich Variablen für den Aufruf gespeicherter Prozeduren oder parametrisierter Abfragen übergeben. Diese werden an ein **Command**-Objekt gebunden und beim Aufruf des Befehls verwendet, der im **Command**-Objekt programmiert wurde.

Das Objekt Recordset

Das **Recordset**-Objekt enthält eine Gruppe von Datensätzen aus der Datenbank. Die Menge der Datensätze ist das Ergebnis eines Befehls, der an die Datenbank gesendet wurde und die Rückgabe einer Gruppe von Datensätzen bewirkt. Durch den **Recordset** können Sie genauso navigieren, wie es bei **Recordset**-Objekten in anderen Technologien des Datenbankzugriffs üblich ist. Auf die einzelnen Felder eines Datensatzes im **Recordset** kann man über die **Field**-Objekte zugreifen, die mit dem **Recordset** verbunden sind. Man kann die Datensätze im **Recordset** aktualisieren und dann mit dem **Recordset** die Datenbank auf den neuesten Stand bringen. Es lassen sich auch neue Datensätze in den **Recordset** einfügen oder daraus löschen. Diese Änderungen kann man dann an die Datenbank übertragen.

Das Objekt Field

Das **Field**-Objekt repräsentiert eine einzelne Spalte im **Recordset**. Jedes **Field**-Objekt enthält den Spaltennamen, den Datenwert und wie der Datenwert darzustellen ist. Da ADO für den Einsatz in den Skriptsprachen von Microsoft vorgesehen ist und in diesen Skriptsprachen lediglich der Datentyp **Variant** zur Verfügung steht, enthalten **Field**-Objekte immer einen Datenwert vom Typ **Variant**. Der Datenwert wird automatisch in den jeweiligen Datentyp konvertiert, wenn die Datenbank aktualisiert wird. Als Programmierer, der mit den ADO-Objekten arbeitet, müssen Sie den Wert vom Datentyp **Variant** in den benötigten Datentyp umwandeln und beim Aktualisieren des Wertes diese Umwandlung in der anderen Richtung vornehmen.

Das ActiveX-Steuerelement ADO

In Ihren Visual C++-Anwendungen können Sie das ADO-Steuerelement nach zwei verschiedenen Methoden einsetzen. Der einfache Weg zum Einbinden von ADO in Ihre Anwendung verläuft über ActiveX-Steuerelemente. Das ADO-Datensteuerelement können Sie in Ihr Visual C++-Projekt genau

wie jedes andere ActiveX-Steuerelement aufnehmen (siehe Abbildung 15.1).

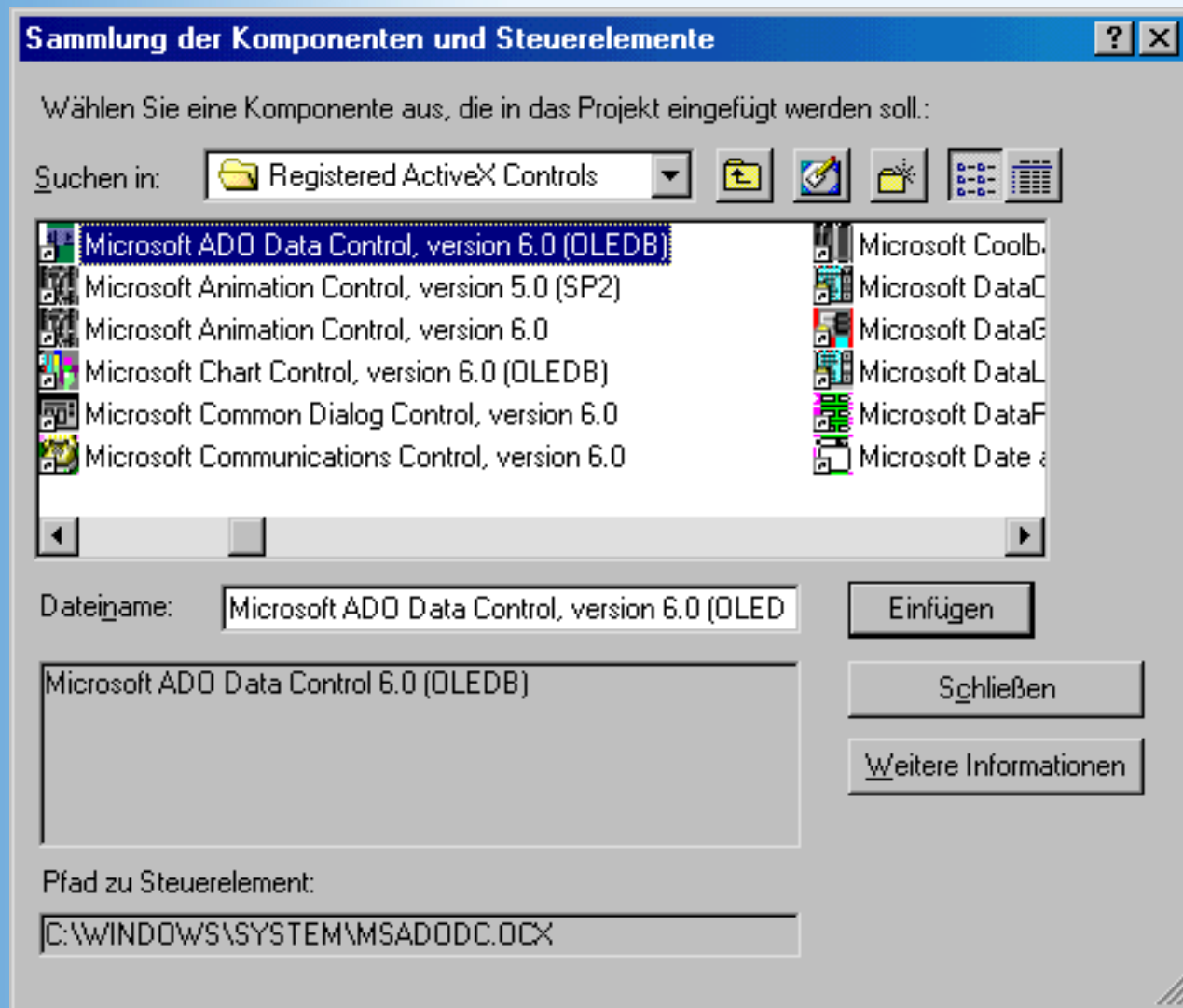


Abbildung 15.1:
Das ActiveX-Steuerelement ADO in ein Projekt aufnehmen

Nachdem Sie das ADO-Steuerelement in Ihr Projekt eingebunden und in einem Fenster platziert haben, müssen Sie die Datenverbindung in den Steuerelementeigenschaften festlegen, wie es Abbildung 15.2 zeigt. Außerdem ist die Quelle für die Datensätze anzugeben, die das Steuerelement abrufen (siehe Abbildung 15.3).

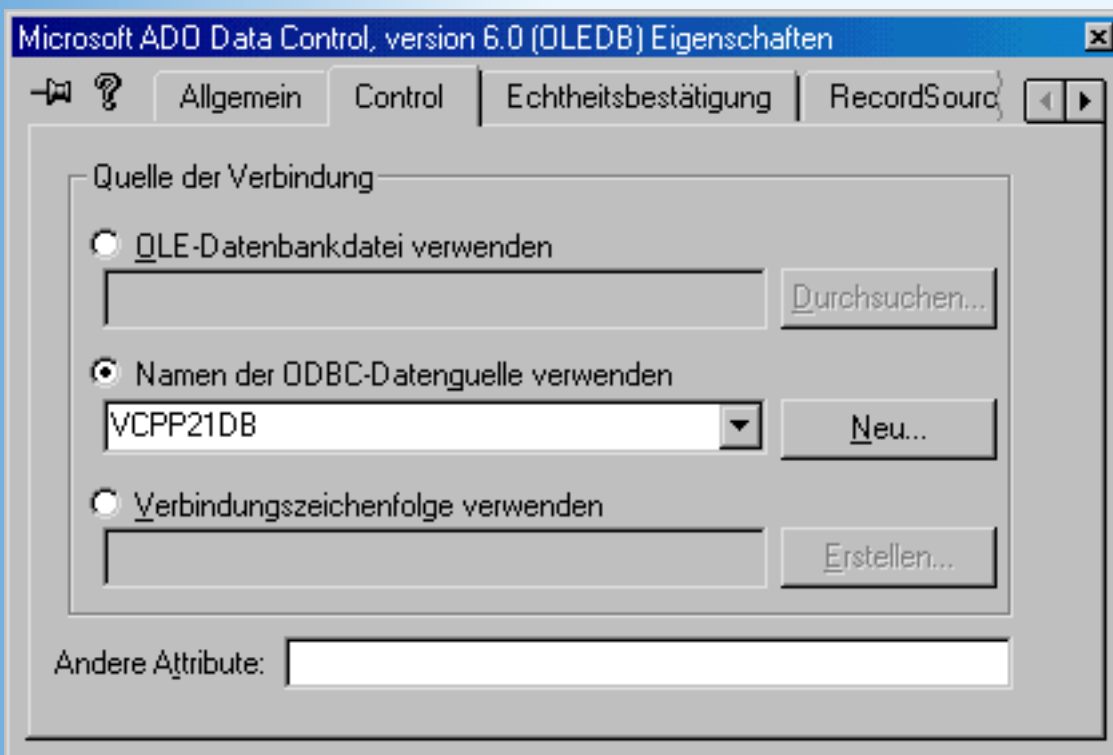


Abbildung 15.2:
Die Datenbankverbindung festlegen

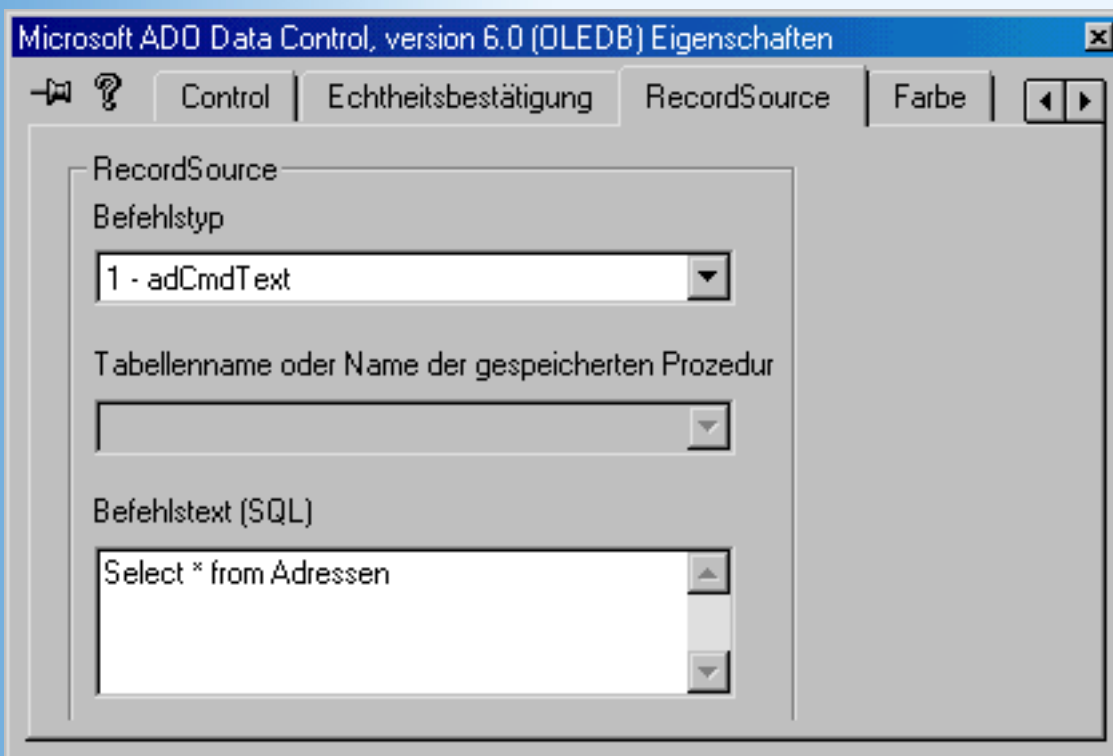


Abbildung 15.3:
Die Datenquelle (Record Source) festlegen

Um das ADO-Steuerelement effizient nutzen zu können, wird man auch datengebundene Steuerelemente, die ADO-fähig sind, einsetzen, beispielsweise das Steuerelement DataGrid von Microsoft. Wenn man diese Steuerelemente in das Fenster mit dem ADO-Steuerelement aufnimmt, legt man das ADO-Steuerelement als Datenquelle für das Steuerelement fest, wie es Abbildung 15.4 zeigt. Wenn das Steuerelement lediglich den Zugriff für ein einzelnes Feld in einem Recordset bereitstellen soll, muß man auch festlegen, welches Feld für das Steuerelement zu verwenden ist.

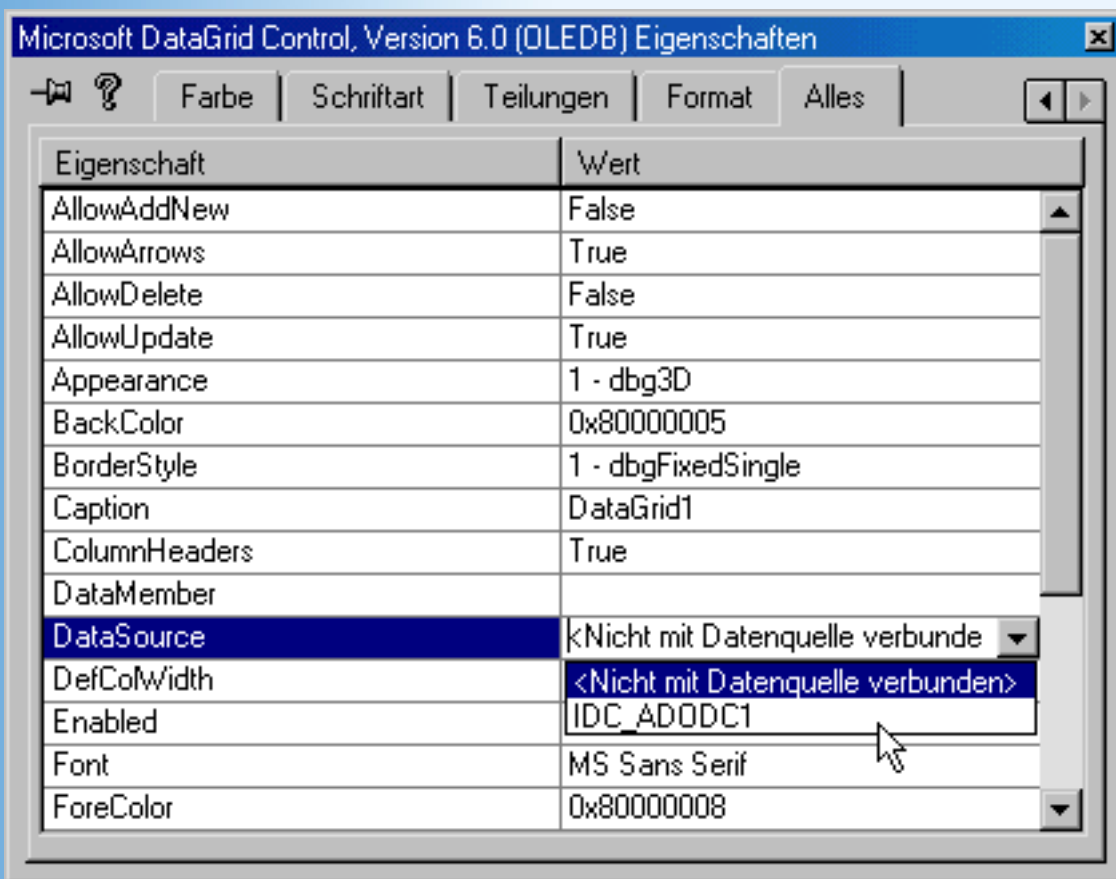


Abbildung 15.4:
Die Datenquelle (Data Source) festlegen

Nachdem Sie alle derartigen Steuerelemente in das Fenster aufgenommen und konfiguriert haben, bietet Ihre Anwendung vollen Datenbankzugriff über ADO, ohne daß Sie dafür eine einzige Codezeile schreiben mußten (siehe Abbildung 15.5).

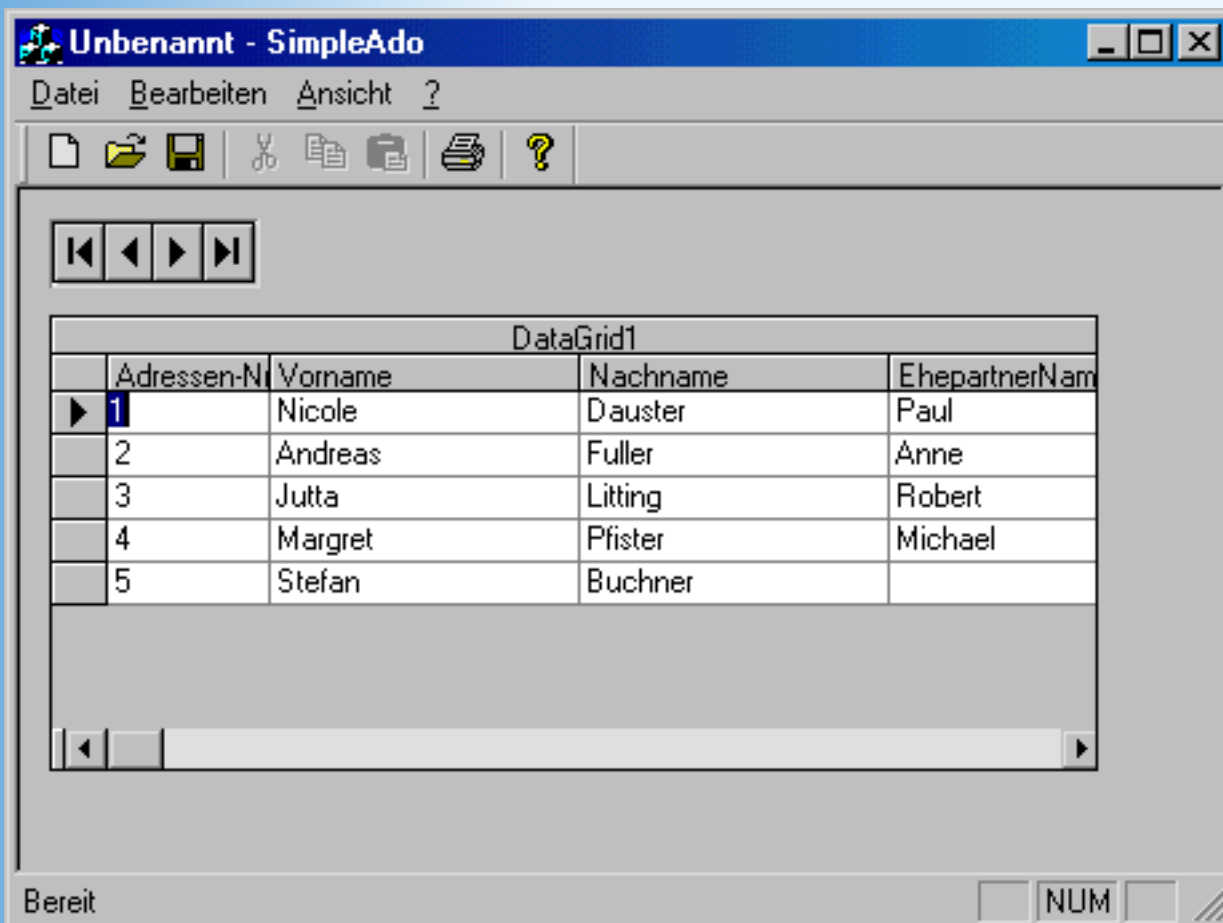


Abbildung 15.5: Eine laufende Anwendung mit dem ADO-Steuerelement

Datenbankanwendungen lassen sich also sehr einfach erstellen: Man plaziert Steuerelemente in einem Fenster und konfiguriert deren Eigenschaften, um die Datenpfade zu beschreiben. Wo liegt nun der Nachteil, wenn man ADO-Anwendungen auf diese Weise erstellt? Erstens bringt diese Lösung eine Menge unnötigen Verwaltungsaufwand mit sich. Für jede SQL-Abfrage oder Tabelle, die man in einen separaten Recordset stellen möchte, ist ein eigenes ADO-Steuerelement erforderlich. Jedes dieser ADO-Steuerelemente richtet eine eigene Verbindung zur Datenbank ein, was bei Datenbanken mit einer begrenzten Anzahl von verfügbaren Verbindungen zu Problemen führen kann (ganz zu schweigen vom zusätzlichen Verwaltungsaufwand in der Anwendung). Schließlich sind nicht alle datengebundenen Steuerelemente auch ADO-fähig. Da die ADO-Technologie ziemlich neu ist, gibt es momentan nur wenige Steuerelemente, die sich dafür eignen. Manche Steuerelemente erlauben zwar, daß Sie Daten abrufen und für den Benutzer anzeigen, aber der Benutzer kann die Daten nicht ändern oder bearbeiten. Andere Steuerelemente bieten nicht einmal diese Unterstützung.

Die ADO-DLL importieren

Wenn Sie sich in der MFC-Klassenhierarchie umsehen, werden Sie Klassen für den Einsatz mit ADO vermissen. Wenn Sie die Lösung mit Steuerelementen nicht einsetzen möchten, welche Möglichkeiten gibt es sonst? Müssen Sie Klassen in eigener Regie erstellen? Nein. Microsoft hat andere Werkzeuge bereitgestellt, um Klassen für die Objekte in ADO zu erstellen und zu verwenden. Zu diesem Zweck gibt es eine neue C++-Präcompilerdirektive namens `#import`.

Die Präcompilerdirektive `#import` ist erst mit der Version 5.0 zu Visual C++ hinzugekommen. Mit dieser Direktive kann man eine ActiveX-DLL importieren, die mit der Schnittstellenbeschreibung `IDispatch` erstellt wurde. Diese Direktive weist den Visual C++-Compiler an, die Objektinformationen aus der DLL herauszuziehen, wobei eine Reihe von Header-Dateien erstellt und automatisch in Ihr Projekt aufgenommen werden. Diese Header-Dateien weisen die Dateierweiterungen `.t lh` und `.t li` auf und befinden sich im Ausgabeverzeichnis Ihres Projekts (dem Verzeichnis Debug oder Release, demselben Verzeichnis, in dem Sie die ausführbare Anwendung finden, nachdem Sie das Projekt kompiliert haben). Diese beiden Dateien enthalten Klassendefinitionen für alle Objekte in der DLL, die Sie in Ihrem Code verwenden können. Die Direktive `#import` weist den Compiler auch an, die DLL als Teil des Projekts einzubinden, wodurch die Notwendigkeit entfällt, die `.lib`-Datei für die DLL in das Projekt aufzunehmen.

Um die ADO-DLL zu importieren, schreiben Sie den folgenden Code an den Beginn der Header-Datei, in der Sie alle Datenbankobjekte definieren:

```
#define INITGUID
#import "C:\Programme\Gemeinsame Dateien\System\ADO\Msado15.dll"
rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"
```

In diesen vier Zeilen definiert die erste Direktive eine Konstante, die für ADO erforderlich ist. Die zweite Zeile importiert die ADO-DLL und erstellt die erwähnten zwei Header-Dateien. Nach dem Namen der zu importierenden Datei enthält diese Direktive zwei Attribute zur `#import`-Direktive. Das erste, `rename_namespace`, benennt den Namensbereich um, in den die DLL importiert wurde. Daran schließt sich die Zeile nach der `#import`-Direktive an, wo der umbenannte Namensbereich

als der verwendete spezifiziert wird. Das zweite Attribut, `rename`, benennt ein Element in den Header-Dateien um, die mit der `#import`-Direktive erzeugt wurden. Die Elemente in diesen Header-Dateien sind umzubenennen, um zu verhindern, daß Konflikte mit einem anderen Element, das an anderer Stelle benannt wurde, auftreten. Wenn Sie sich die Header- Datei ansehen, ist das angegebene Element nicht in der Datei umbenannt, wenn aber der Compiler die Datei liest, benennt er das Element um. Die letzte Zeile bindet die ADO-Header-Datei mit den Definitionen verschiedener Makros ein, die Sie beim Schreiben Ihrer ADO-Anwendungen benötigen.

Mit einer Datenbank verbinden

Bevor man ADO-Objekte verwenden kann, muß man die COM-Umgebung für die Anwendung initialisieren. Dazu ruft man die API-Funktion `CoInitialize` auf und übergibt `NULL` als einzigen Parameter:

```
::CoInitialize(NULL);
```

Damit können Sie nun ActiveX-Objekte aufrufen. Wenn Sie diese Codezeile in Ihrer Anwendung auslassen oder erst nach der Interaktion mit den Objekten schreiben, erhalten Sie einen COM-Fehler, sobald Sie die Anwendung ausführen.

Wenn Sie alle Aktivitäten in bezug auf ADO beendet haben, müssen Sie auch die COM-Umgebung herunterfahren. Rufen Sie dazu die Funktion `CoUninitialize` wie folgt auf:

```
CoUninitialize;
```

Diese Funktion räumt die COM-Umgebung auf und bereitet Ihre Anwendung auf das Herunterfahren vor.

Sobald die COM-Umgebung initialisiert ist, können Sie eine Verbindung zur Datenbank herstellen. Statt eine `Connection`-Objektvariable zu erzeugen, deklariert man besser einen `Connection`-Objektzeiger, `_ConnectionPtr`, und verwendet ihn für alle Zugriffe auf das `Connection`-Objekt. Diesen `Connection`-Objektzeiger können Sie dann mit einer Instanz des `Connection`-Objekts initialisieren. Dazu rufen Sie die Funktion `CreateInstance` auf und übergeben die `UUID` des `Connection`-Objekts als einzigen Parameter:

```
_ConnectionPtr pConn;  
pConn.CreateInstance(__uuidof(Connection));
```



Wenn Sie mit diesen Objekten und Funktionen arbeiten, ist auf die richtige Anzahl von Unterstrichen vor den verschiedenen Objekt- und Funktionsnamen zu achten. Das Objekt `_ConnectionPtr` hat nur einen einzigen Unterstrich, während in `__uuidof` zwei Unterstriche zu schreiben sind.

Nachdem Sie das Objekt erzeugt haben, können Sie die Funktion `Open` aufrufen, um die Verbindung zur Datenbank einzurichten. Die Funktion `Open` übernimmt vier Parameter. Der erste gibt die Verbindungszeichenfolge an. Dieser String definiert die OLE DB-Datenquelle für die Datenbank. Das kann ein ODBC OLE DB-Treiber sein, wobei OLE DB über einer ODBC-Datenquelle sitzt, wie Sie sie in Ihrer Beispielanwendung einsetzen. Wenn Sie mit SQL Server oder Oracle-Datenbanken arbeiten, kann es eine direkte Verbindung zur OLE DB-Schnittstelle sein, die von der Datenbank selbst bereitgestellt wird. Der zweite Parameter ist die Benutzer-ID für die Verbindung zur Datenbank. Im dritten Parameter ist das zugehörige Kennwort anzugeben. Der vierte Parameter bezeichnet den Cursortyp, der mit der Datenbank zum Einsatz kommt. Diese Typen sind in der

Header-Datei `Msado15.tlh` definiert, die durch die `#import`-Direktive angelegt wird. Das folgende Beispiel zeigt einen typischen Einsatzfall der Funktion `Open`, um eine Verbindung zu einer ODBC-Datenquelle herzustellen, die weder Benutzer-ID noch Kennwort benötigt:

```
pConn->Open(L"Provider=MSDASQL.1;Data Source=VCP21DB", L"", L"",
adOpenUnspecified);
```

Befehle ausführen und Daten abrufen

Nachdem Sie die Verbindung geöffnet haben, können Sie mit einem `Command`-Objekt SQL-Befehle an die Datenbank übergeben. Das ist die normale Methode, um SQL- Befehle mit ADO auszuführen. Um ein `Command`-Objekt zu erzeugen, gehen Sie genauso vor, wie bei einem `Connection`-Objekt. Sie deklarieren einen `Command`-Objektzeiger, `_CommandPtr`, und erzeugen dann eine Instanz dieses Objekts mit der UUID des `Command` -Objekts:

```
_CommandPtr pCmd;
pCmd.CreateInstance(__uuidof(Command));
```

Wenn Sie Ihr `Command`-Objekt erstellt und bereits eine Verbindung zur Datenbank eingerichtet haben, setzen Sie die Eigenschaft `ActiveConnection` (für die aktive Verbindung) des `Command`-Objekts auf den geöffneten `Connection`-Objektzeiger:

```
pCmd->ActiveConnection = pConn;
```

Als nächstes können Sie den auszuführenden SQL-Befehl spezifizieren, indem Sie die `CommandText`-Eigenschaft des `Command`-Objekts festlegen:

```
pCmd->CommandText = "Select * from Adressen";
```

Jetzt haben Sie zwei Möglichkeiten, um diesen Befehl auszuführen und die Datensätze abzurufen. Erstens können Sie die Methode `Execute` des `Command`-Objekts aufrufen. Die Methode gibt ein neues `Recordset`-Objekt zurück, das Sie auf einen `Recordset`- Objektzeiger setzen:

```
_RecordsetPtr pRs;
pRs = pCmd->Execute();
```

Die andere Lösung, den Befehl auszuführen und die Datensätze abzurufen, besteht darin, das `Command`-Objekt als Quelle für die Datensätze im `Recordset` festzulegen. Dazu ist es erforderlich, das `Recordset`-Objekt wie folgt zu erstellen:

```
_RecordsetPtr pRs;
pRs.CreateInstance(__uuidof(Recordset));
pRs->PutRefSource(pCmd);
```

Jetzt sind zwei `NULL`-Werte vom Typ `Variant` zu erzeugen, um sie als die beiden ersten Parameter an die Methode `Open` des `Recordset`-Objekts zu übergeben. Der dritte Parameter ist der zu verwendende Cursortyp gefolgt von der zu verwendenden Sperrmethode. Schließlich enthält der fünfte Parameter an die Methode `Open` des `Recordset`- Objekts ein Optionsflag, das angibt, wie die Datenbank den übergebenen Befehl auswerten soll. Dazu schreiben Sie folgenden Code:

```
// NULL vom Typ Variant erzeugen
_variant_t vNull;
vNull.vt = VT_ERROR;
vNull.scode = DISP_E_PARAMNOTFOUND;
```

```
// Recordset öffnen
```



```
pRs->Open(vNull, vNull, adOpenDynamic, adLockOptimistic,
adCmdUnknown);
```

Es gibt eine weitere Lösung, um alle obigen Aufgaben in nur ein paar Codezeilen zu formulieren. Man läßt die Objekte `Command` und `Connection` einfach links liegen und schreibt die gesamten Verbindungsinformationen in die Funktion `Open` des `Recordset`- Objekts. Den SQL-Befehl kann man als ersten Parameter spezifizieren und die Verbindungsinformationen als zweiten, statt der beiden `NULL`-Werte, die Sie vorher übergeben haben. Bei dieser Methode reduziert sich der gesamte obige Code auf die folgenden Zeilen:

```
_RecordsetPtr pRs;
pRs.CreateInstance(__uuidof(Recordset));
pRs->Open(_T("Provider=MSDASQL.1;Data Source=VCP21DB"),
_T("select * from Adressen"), adOpenDynamic,
adLockOptimistic, adCmdUnknown);
```



Auch wenn es bei einer einfachen Anwendung - wie der heute zu erstellenden - ohne weiteres möglich ist, alle Befehls- und Verbindungsinformationen in der Funktion `Open` des `Recordset`-Objekts unterzubringen, empfiehlt es sich bei Anwendungen, die mehr als ein paar Datenbankabfragen ausführen, das `Connection`-Objekt zu verwenden. Damit kann man eine einzelne Datenbankverbindung einrichten und diese Verbindung zur gesamten Interaktion mit der Datenbank nutzen.

Durch den Recordset navigieren

Nachdem Sie eine Gruppe von Datensätzen aus der Datenbank abgerufen und in einem `Recordset`-Objekt abgelegt haben, müssen Sie durch die Datensätze navigieren. Diese Funktionalität ist erwartungsgemäß über die Funktionen `MoveFirst` (zum ersten Datensatz), `MoveLast` (zum letzten Datensatz), `MovePrevious` (zum vorherigen Datensatz) und `MoveNext` (zum nächsten Datensatz) verfügbar. Die Funktionen übernehmen keine Parameter, da sie genau die beschriebenen Aufgaben ausführen.

Neben diesen Funktionen verfügt das `Recordset`-Objekt noch über die beiden Eigenschaften `BOF` und `EOF` (die Sie normalerweise umbenennen sollten, um Konflikte mit der Standarddefinition von `EOF` zu vermeiden). Aus diesen Eigenschaften geht hervor, ob der aktuelle Datensatz in der Gruppe außerhalb der Datensatzgruppe liegt.

Auf Feldwerte zugreifen

Sobald Sie auf die Datenwerte in den einzelnen Feldern zugreifen müssen, beginnt das Arbeiten mit ADO in Visual C++ interessant zu werden. Das Konzept von ADO zielt auf einen einfachen Einsatz in den Skriptsprachen VBScript und JScript von Microsoft ab. Allerdings kennen diese Sprachen nur den Datentyp `Variant`, so daß alle Datenelemente, die Sie aus den Feldern im ADO-`Recordset` abrufen, `Variant`-Werte sind. Man muß sie in die Datentypen konvertieren, die man eigentlich benötigt. Dazu gibt es zwei Wege. Beim einfacheren ruft man die Werte in einen `Variant` ab und konvertiert sie dann, wie es der folgende Code zeigt:

```
_variant_t vFirstName;
CString strFirstName;
```

```
vFirstName = pRs->GetCollect(_variant_t("Vorname"));
vFirstName.ChangeType(VT_BSTR);
strFirstName = vFirstName.bstrVal;
```

Der etwas mühsamere Weg ist tatsächlich der bessere und erleichtert auf lange Sicht die Arbeit. Microsoft hat eine Reihe von Makros geschaffen, die die Konvertierung automatisch vornehmen und einen Satz von Variablen der Datensätze in der Ergebnismenge verwalten. Um damit zu arbeiten, definiert man eine neue Klasse als Schnittstelle zum Recordset. Diese Klasse leitet man von der Klasse `CADORecordBinding` ab, die in der Header-Datei `icrsint.h` definiert ist und die man unmittelbar nach der `#import` -Direktive einbindet. Diese Klasse weist weder Konstruktor noch Destruktor auf, enthält aber verschiedene Makros und eine Anzahl von Variablen. Jedes Feld in der Datensatzmenge hat zwei Variablen, eine `unsigned long`, die den Status der Variablen verwaltet, und die Feldvariable an sich. Diese Variablen müssen reguläre C-Variablen sein und dürfen nicht als C++-Klassen wie etwa `CString` angelegt sein. Der folgende Code zeigt ein einfaches Beispiel für diese Klassendeklaration:

```
class CCustomRs :
public CADORecordBinding
{
BEGIN_ADO_BINDING(CCustomRs)
ADO_FIXED_LENGTH_ENTRY(1, adInteger, m_lAddressID, lAddressIDStatus,
FALSE)
ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szFirstName,
sizeof(m_szFirstName), lFirstNameStatus, TRUE)
ADO_FIXED_LENGTH_ENTRY(3, adDate, m_dtBirthdate, lBirthdateStatus,
TRUE)
ADO_FIXED_LENGTH_ENTRY(4, adBoolean, m_bSendCard, lSendCardStatus,
TRUE)
END_ADO_BINDING()

public:
LONG m_lAddressID;
ULONG lAddressIDStatus;
CHAR m_szFirstName[51];
ULONG lFirstNameStatus;
DATE m_dtBirthdate;
ULONG lBirthdateStatus;
VARIANT_BOOL m_bSendCard;
ULONG lSendCardStatus;
};
```

Den Aufbau dieser Klasse definieren Sie entsprechend der Anordnung der Datensätze, die die Datenbankabfrage zurückgibt. Dann können Sie eine Variable dieser Klasse für Ihre Anwendung wie folgt deklarieren:

```
CCustomRs m_rsRecSet;
```

Als nächstes ist ein Zeiger auf eine `IADORecordBinding`-Schnittstelle zu erzeugen:

```
IADORecordBinding *picRs = NULL;
```

Es handelt sich um einen Zeiger auf eine COM-Schnittstelle, die Teil des ADO-Recordset

-Objekts ist. Wenn Sie die Gruppe der Datensätze abrufen, müssen Sie den Zeiger auf die `IADORecordBinding`-Schnittstelle abrufen und die benutzerdefinierte Recordset-Klasse an das Recordset-Objekt wie im folgenden Codebeispiel binden:

```
if (FAILED(pRs->QueryInterface(__uuidof(IADORecordBinding), (LPVOID *)&picRs))) _com_issue_error(E_NOINTERFACE);
picRs->BindToRecordset(&m_rsRecSet);
```

Wenn Sie nun durch die Datensätze in der Ergebnismenge navigieren, brauchen Sie nur auf die Member-Variablen Ihrer benutzerdefinierten Datensatzklasse zuzugreifen, um den aktuellen Wert für jedes Feld abzurufen.

Die Makros `BEGIN_ADO_BINDING` und `END_ADO_BINDING`

Den Schlüssel zur zweiten Methode des Zugriffs auf Datenwerte im Recordset bilden die Makros, mit denen Sie die Datensatzklasse definieren. Die Gruppe der Makros beginnt mit `BEGIN_ADO_BINDING`, das den Klassennamen als einzigen Parameter übernimmt. Das Makro richtet die Strukturdefinition ein, die mit den nachstehend beschriebenen restlichen Makros erzeugt wird.

Das Makro `END_ADO_BINDING` schließt die Makrogruppe ab. Dieses Makro übernimmt keine Parameter und beendet die Definition der Datensatzbindungsstruktur, die in der Klasse erzeugt wurde. In den übrigen Makros, die zwischen diesen beiden stehen, läuft die eigentliche Arbeit ab.

Die Makros `ADO_FIXED_LENGTH_ENTRY`

Das Makro `ADO_FIXED_LENGTH_ENTRY` kommt bei Datenbankfeldern mit fester Größe zum Einsatz. Es arbeitet mit Feldern vom Typ Datum oder Boolesch sowie mit Textfeldern, denen eine feste Größe zugewiesen ist und es keine Möglichkeit einer Änderung in der Datenbank gibt. Das Makro existiert in zwei Versionen, wobei man für die zweite Version eine 2 an den Makronamen anhängt (`ADO_FIXED_LENGTH_ENTRY2`).

Beide Versionen erfordern die gleichen ersten drei und den letzten Parameter. In der ersten Version ist ein zusätzlicher Parameter erforderlich, der in der zweiten Version optional ist. Der erste Parameter gibt die Ordinalzahl des Feldes im Recordset an. Es handelt sich dabei um die Feldreihenfolge, wie sie die SQL-Abfrage zurückgibt, die zum Füllen des Recordset ausgeführt wird. Der zweite Parameter bezeichnet den Datentyp des Feldes. Die verfügbaren Dateitypen sind in der Header-Datei definiert, die durch die `#import`-Direktive erzeugt wird. Im dritten Parameter steht die Variable, in die der Datenwert kopiert wird. Bei der ersten Version des Makros ist der vierte Parameter die Variable für den Feldstatus (der `unsigned long`, den Sie mit der Variablen für den eigentlichen Wert definiert haben). Die letzte Variable ist ein Boolescher Wert, der angibt, ob das Feld modifiziert werden kann.

Die Makros `ADO_NUMERIC_ENTRY`

Die Makros `ADO_NUMERIC_ENTRY` verwendet man nur in Verbindung mit numerischen Feldern. Wie die Makros `ADO_FIXED_LENGTH_ENTRY` liegen sie ebenfalls in zwei Versionen vor, die in der gleichen Weise benannt sind. Die ersten fünf und der letzte Parameter sind in beiden Versionen gleich. Wie bei den Makros `ADO_FIXED_LENGTH_ENTRY` hat die erste Version einen zusätzlichen Parameter, der in der zweiten Version nicht verwendet wird.

Die drei ersten Parameter für die Makros `ADO_NUMERIC_ENTRY` entsprechen den Parametern für die Makros `ADO_FIXED_LENGTH_ENTRY`. Gleiches gilt für den letzten Parameter und bei der

zweiten Version für den vorletzten. In den vierten und fünften Parametern unterscheiden sich diese Makros. Der vierte Parameter legt die Genauigkeit des Wertes in diesem Feld des Recordsets fest. Der fünfte Parameter spezifiziert die Skalierung des Wertes. Beide Parameter sind für die Konvertierung des Wertes aus und in den Datentyp `Variant` entscheidend.

Die Makros `ADO_VARIABLE_LENGTH_ENTRY`

Die Makros `ADO_VARIABLE_LENGTH_ENTRY` bilden den Abschluß in der Reihe der Makros. Diese Makros verwendet man für Datenbankfelder, die sich in der Länge ändern können. Bei einer SQL-basierten Datenbank kommen diese Makros bei allen Spalten vom Typ `varchar` (Strings variabler Länge) zum Einsatz. Das Makro existiert in drei Versionen. Alle Versionen stimmen in den ersten vier und dem letzten Parameter überein. In dazwischen vorkommenden Parametern unterscheiden sich die Makros.

Der erste Parameter gibt die Ordinalzahl der Spalte im Recordset an, wie er von der SQL-Abfrage zurückgegeben wird. Der zweite Parameter gibt den Datentyp an. Im dritten Parameter steht die Variable, in die der Datenwert zu übernehmen ist. Der vierte Parameter spezifiziert für alle Versionen des Makros die Größe der Variablen, in die der Wert einzutragen ist. Damit wird verhindert, daß die Daten über das Ende der Variablen, die für die Aufnahme des Wertes definiert wurde, hinaus geschrieben werden. Wie bei den vorherigen Makros gibt der letzte Parameter an, ob das Feld aktualisierbar ist.

In der ersten Version des Makros stehen zwei Parameter zwischen dem vierten und dem letzten Parameter. Die zweite Version weist an dieser Stelle nur einen Parameter auf, während in der dritten Version des Makros nur der zweite der beiden zusätzlichen Parameter vorkommt. Der erste der beiden Parameter ist die Statusvariable für das betreffende Feld. Im zweiten dieser beiden Parameter steht die Länge des Feldes in der Datenbank. Das obige Beispiel verwendet die zweite Version des Makros.

Datensätze aktualisieren

Wenn man Werte in einem Datensatz des Recordsets aktualisieren muß, hängt dessen Behandlung von den beiden Methoden ab, nach denen man die Datenelemente aus dem Recordset abrufen. Wenn Sie jedes Feld abgerufen und in eigener Regie aus einem `Variant` konvertiert haben, müssen Sie jedes einzelne Feld, das sich geändert hat, aktualisieren. Zu diesem Zweck verwendet man die Methode `Update` des `Recordset`- Objekts. Diese Methode übernimmt zwei Variablen: das zu aktualisierende Feld und den neuen Wert für das Feld. Der folgende Code zeigt ein Beispiel für eine derartige Aktualisierung:

```
_variant_t vName, vValue;
vName.SetString("Vorname");
vValue.SetString("John");
pRs->Update(vName, vValue);
```

Haben Sie eine Datensatzklasse erzeugt und an den Recordset gebunden, vereinfacht sich die Aktualisierung des Datensatzes etwas. Nachdem Sie die neuen Werte in die Variablen der Datensatzklasse kopiert haben, können Sie die datensatzgebundene Version der Funktion `Update` aufrufen, wie es das folgende Beispiel zeigt:

```
picRs->Update(&m_rsRecSet);
```

Hier erhält das zu aktualisierende `Recordset`-Objekt die Werte aus der Datensatzklasse, die Sie an den Recordset gebunden haben.

Hinzufügen und Löschen

Datensätze lassen sich in einem ADO-Recordset in ähnlicher Weise hinzufügen und löschen, wie man es von anderen Technologien des Datenbankzugriffs kennt. Allerdings gibt es kleinere Besonderheiten zu beachten, wenn man neue Datensätze hinzufügt.

Um den aktuellen Datensatz zu löschen, ruft man die `Delete`-Methode des `Recordset`- Objekts auf. Diese Methode erfordert einen einzelnen Parameter, der festlegt, wie das Löschen erfolgen soll. Höchstwahrscheinlich übergeben Sie den Wert `adAffectCurrent` , so daß nur der aktuelle Datensatz im Recordset gelöscht wird, wie es der folgende Code zeigt:

```
pRs->Delete(adAffectCurrent);
pRs->MovePrevious();
```

Analog zu anderen Technologien des Datenbankzugriffs gibt es keinen aktuellen Datensatz mehr, nachdem Sie den aktuellen Datensatz gelöscht haben. Man muß zu einem anderen Datensatz navigieren, bevor man dem Benutzer weitere Aktionen gestatten kann.

Um einen neuen Datensatz hinzuzufügen, ruft man die Methode `AddNew` des `Recordset` -Objekts auf. Der neu hinzugefügte Datensatz wird zum aktuellen Datensatz im Recordset. Wenn Sie die Variablen in der von Ihnen erzeugten Datensatzklasse untersuchen, stellen Sie fest, daß sie alle leer sind. Allerdings können Sie nicht einfach beginnen, Datenwerte in diese Felder einzugeben. Um dem Benutzer die sofortige Eingabe der verschiedenen Datenelemente in den neuen Datensatz zu ermöglichen, leeren Sie die Werte in der Datensatzklasse und übergeben diese Variable als einzigen Parameter an die Klasse `AddNew`. Diese müssen Sie über den datensatzgebundenen Schnittstellenzeiger wie im folgenden Beispiel aufrufen:

```
CString strBlank = " ";
ColeDateTime dtBlank;

m_rsRecSet.m_lAddressID = 0;
strcpy(m_rsRecSet.m_szFirstName, (LPCTSTR)strBlank);
m_rsRecSet.m_dtBirthdate = (DATE)dtBlank;
m_rsRecSet.m_bSendCard = VARIANT_FALSE;
picRs->AddNew(&m_rsRecSet);
```

Auf diese Weise kann man dem Benutzer einen leeren Datensatz zur Verfügung stellen, der zur Bearbeitung bereit ist. Nachdem der Benutzer die verschiedenen Werte in den Datensatz eingetragen hat, kopieren Sie alle diese Werte zurück in die Datensatzvariable. Rufen Sie dann die Methode `Update` auf, um den Datensatz zu speichern.

Die Objekte Recordset und Connection schließen

Wenn Sie die Arbeit an einem Recordset beendet haben, schließen Sie den Recordset durch Aufruf der Methode `Close`, wie es der folgende Code zeigt:

```
pRs->Close();
```

Haben Sie die Datenbankinteraktion für die gesamte Anwendung beendet, schließen Sie auch die Verbindung zur Datenbank. Rufen Sie dazu die Methode `Close` des `Connection`-Objekts auf:

```
pConn->Close();
```

Eine Datenbankanwendung mit ADO erstellen

Die Beispielanwendung, die Sie heute erstellen, ist eine weitere einfache Datenbankanwendung, die grundsätzlich das gleiche wie die Anwendung der gestrigen Lektion realisiert. Allerdings rufen Sie jetzt eine Datensatzgruppe per ADO aus einer Access- Datenbank ab und stellen die Funktionalität bereit, um durch den Recordset zu navigieren. Der Benutzer kann Änderungen an den Daten im Recordset vornehmen und diese Änderungen in die Datenbank übernehmen. Außerdem lassen sich Datensätze nach Belieben neu hinzufügen und löschen. Alles das erreichen Sie mit ADO als Mittel für den Zugriff auf die Datenbank, wobei der Zugriff über den gestern konfigurierten ODBC-Treiber verläuft.

Das Anwendungsgerüst erstellen

Die Beispielanwendung ist als SDI-Anwendung konzipiert. Wie bei verschiedenen anderen Anwendungen, die Sie im Verlauf dieses Buches erstellen, lassen sich alle Elemente der heutigen Anwendung ebenso auf MDI-Anwendungen oder dialogbasierte Anwendungen übertragen. Als erstes erstellen Sie für die Anwendung ein Gerüst mit dem MFC-Anwendungs-Assistenten, wobei Sie die meisten der vorgegebenen Einstellungen für eine SDI-Anwendung übernehmen.

Legen Sie zunächst mit dem Anwendungs-Assistenten ein neues Projekt an, das Sie zum Beispiel mit DbAdo benennen. Im ersten Dialogfeld des Assistenten wählen Sie die Option Einzelnes Dokument (SDI). Übernehmen Sie die Standardeinstellungen im zweiten bis fünften Schritt des Assistenten. Achten Sie darauf, daß keine Datenbankunterstützung in die Anwendung eingebunden wird. Im letzten Schritt des Assistenten legen Sie fest, daß die Sichtklasse (CDbAdoView) von der Basisklasse CFormView abzuleiten ist.

Nachdem Sie das Anwendungsgerüst erstellt haben, entwerfen Sie das Hauptdialogfeld. Fügen Sie gemäß Abbildung 15.6 die Steuerelemente für alle Felder der Tabelle Adressen aus der gestern verwendeten Datenbank hinzu (falls Sie mit einer anderen Datenbank gearbeitet haben, nehmen Sie die Steuerelemente für die Felder der tatsächlich verwendeten Tabelle auf). Legen Sie die Eigenschaften der Steuerelemente entsprechend Tabelle 15.1 fest.



Wenn Ihre Zeit knapp bemessen ist, können Sie das Beispiel auch vereinfachen und verschiedene Steuerelemente und Datenbankfelder auslassen. Als Schlüsselfelder brauchen Sie Adressen-Nr, Vorname, Nachname, Geburtsdatum und Karte senden. Die anderen Felder können Sie ohne weiteres in der Anwendung weglassen. Diese Felder müssen Sie in die Klasse CCustomRS einbinden, die Sie weiter hinten in diesem Kapitel erstellen.

The form layout is as follows:

Adressen-Nr:	Bearbeiten	Email-Adresse:	Bearbeiten
Vorname:	Bearbeiten	Telefon/privat:	Bearbeiten
Nachname:	Bearbeiten	Telefon/beruflich:	Bearbeiten
Name des Ehepartners:	Bearbeiten	Durchwahl Büro:	Bearbeiten
Adresse:	Bearbeiten	Faxnummer:	Bearbeiten
Ort:	Bearbeiten	Geburtsdatum:	Bearbeiten
Bundesland:	Bearbeiten	☐ Karte senden	
Postleitzahl:	Bearbeiten	Anmerkungen:	Bearbeiten
Land:	Bearbeiten		

Abbildung 15.6:
Das Layout des Hauptformulars

Tabelle 15.1: Eigenschaften der Steuerelemente für das Hauptformular der Datenbankanwendung

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Adressen-Nr:
Bearbeitungsfeld	ID	IDC_EDIT_ADDRESSID
Text	ID	IDC_STATIC
	Titel	Vorname:
Bearbeitungsfeld	ID	IDC_EDIT_FIRSTNAME
Text	ID	IDC_STATIC
	Titel	Nachname:
Bearbeitungsfeld	ID	IDC_EDIT_LASTNAME
Text	ID	IDC_STATIC
	Titel	Name des Ehepartners:
Bearbeitungsfeld	ID	IDC_EDIT_SPOUSENAME
Text	ID	IDC_STATIC
	Titel	Adresse:

Bearbeitungsfeld	ID	IDC_EDIT_ADDRESS
Text	ID	IDC_STATIC
	Titel	Ort:
Bearbeitungsfeld	ID	IDC_EDIT_CITY
Text	ID	IDC_STATIC
	Titel	Bundesland:
Bearbeitungsfeld	ID	IDC_EDIT_STATEORPROVINCE
Text	ID	IDC_STATIC
	Titel	Postleitzahl:
Bearbeitungsfeld	ID	IDC_EDIT_POSTALCODE
Text	ID	IDC_STATIC
	Titel	Land:
Bearbeitungsfeld	ID	IDC_EDIT_COUNTRY
Text	ID	IDC_STATIC
	Titel	Email-Adresse:
Bearbeitungsfeld	ID	IDC_EDIT_EMAILADDRESS
Text	ID	IDC_STATIC
	Titel	Telefon/privat:
Bearbeitungsfeld	ID	IDC_EDIT_HOMEPHONE
Text	ID	IDC_STATIC
	Titel	Telefon/beruflich:
Bearbeitungsfeld	ID	IDC_EDIT_WORKPHONE
Text	ID	IDC_STATIC
	Titel	Durchwahl Büro:
Bearbeitungsfeld	ID	IDC_EDIT_WORKEXTENSION
Text	ID	IDC_STATIC
	Titel	Faxnummer:
Bearbeitungsfeld	ID	IDC_EDIT_FAXNUMBER
Text	ID	IDC_STATIC
	Titel	Geburtsdatum:
Bearbeitungsfeld	ID	IDC_EDIT_BIRTHDATE

Text	ID	IDC_STATIC
	Titel	Karte senden
Kontrollkästchen	ID	IDC_CHECK_SENDCARD
Text	ID	IDC_STATIC
	Titel	Anmerkungen:
Bearbeitungsfeld	ID	IDC_EDIT_NOTES

Nachdem Sie diese Steuerelemente auf dem Formular untergebracht haben, weisen Sie den Steuerelementen mit dem Klassen-Assistenten Variablen zu, wie es aus Tabelle 15.2 hervorgeht. Die Variablen sollten mit den Datentypen der Datenbankspalten, die das jeweilige Steuerelement anzeigt, übereinstimmen.

Tabelle 15.2: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_CHECK_SENDCARD	m_bSendCard	Wert	BOOL
IDC_EDIT_ADDRESS	m_strAddress	Wert	CString
IDC_EDIT_ADRESSID	n_lAddressID	Wert	long
IDC_EDIT_BIRTHDATE	m_oleDtBirthdate	Wert	COleDateTime
IDC_EDIT_CITY	m_strCity	Wert	CString
IDC_EDIT_COUNTRY	m_strCountry	Wert	CString
IDC_EDIT_EMAILADDRESS	m_strEmailAddress	Wert	CString
IDC_EDIT_FAXNUMBER	m_strFaxNumber	Wert	CString
IDC_EDIT_FIRSTNAME	m_strFirstName	Wert	CString
IDC_EDIT_HOMEPHONE	m_strHomePhone	Wert	CString
IDC_EDIT_LASTNAME	m_strLastName	Wert	CString
IDC_EDIT_NOTES	m_strNotes	Wert	CString
IDC_EDIT_POSTALCODE	m_strPostalCode	Wert	CString
IDC_EDIT_SPOUSENAME	m_strSpouseName	Wert	CString
IDC_EDIT_STATEORPROVINCE	m_strStateOrProvince	Wert	CString
IDC_EDIT_WORKEXTENSION	m_strWorkExtension	Wert	CString
IDC_EDIT_WORKPHONE	m_strWorkPhone	Wert	CString

Eine benutzerdefinierte Datensatzklasse erstellen

Bevor Sie weiter an Ihrer Anwendung arbeiten, müssen Sie erst Ihre benutzerdefinierte Datensatzklasse erstellen, die Sie an den Recordset binden. In dieser Klasse sind öffentliche Variablen für alle Spalten in der ausgewählten Datenbanktabelle sowie Statusvariablen für diese Spalten erforderlich. Weiterhin erstellen Sie die Gruppe von Makros, um die Spaltenwerte zwischen dem Recordset und den Klassenvariablen auszutauschen. Um die Klasse zu erstellen, legen Sie eine neue Klasse nach der gleichen Methode wie in den vergangenen Tagen an. Als Typ wählen Sie Allgemeine Klasse. Vergeben Sie einen passenden Klassennamen wie etwa CCustomRs, und legen Sie CADORecordBinding als Basisklasse mit Zugriffsstatus `public` fest.

In der neu erstellten Klasse löschen Sie den Konstruktor und den Destruktor sowohl aus den Header- als auch den Quellcode Dateien für die neue Klasse. Bearbeiten Sie die Header-Datei für die neue Klasse, um die ADO-DLL zu importieren sowie die Makros und Variablen aufzunehmen, wie es Listing 15.1 zeigt.

Listing 15.1: Die benutzerdefinierte Datensatzklasse

```
1: #define INITGUID
2: #import "C:\Programme\Gemeinsame Dateien\System\ADO\msado15.dll"
   Ârename_namespace("ADOCG") rename("EOF", "EndOfFile")
3: using namespace ADOCG;
4: #include "icrsint.h"
5:
6: class CCustomRs :
7: public CADORecordBinding
8: {
9: BEGIN_ADO_BINDING(CCustomRs)
10: ADO_FIXED_LENGTH_ENTRY(1, adInteger, m_lAddressID,
   lAddressIDStatus, ÂFALSE)
11: ADO_VARIABLE_LENGTH_ENTRY2(2, adVarChar, m_szFirstName,
   Âsizeof(m_szFirstName), lFirstNameStatus, TRUE)
12: ADO_VARIABLE_LENGTH_ENTRY2(3, adVarChar, m_szLastName,
   Âsizeof(m_szLastName), lLastNameStatus, TRUE)
13: ADO_VARIABLE_LENGTH_ENTRY2(4, adVarChar, m_szSpouseName,
   Âsizeof(m_szSpouseName), lSpouseNameStatus, TRUE)
14: ADO_VARIABLE_LENGTH_ENTRY2(5, adVarChar, m_szAddress,
   sizeof(m_szAddress), ÂlAddressStatus, TRUE)
15: ADO_VARIABLE_LENGTH_ENTRY2(6, adVarChar, m_szCity,
   sizeof(m_szCity), ÂlCityStatus, TRUE)
16: ADO_VARIABLE_LENGTH_ENTRY2(7, adVarChar, m_szStateOrProvince,
   Âsizeof(m_szStateOrProvince), lStateOrProvinceStatus, TRUE)
17: ADO_VARIABLE_LENGTH_ENTRY2(8, adVarChar, m_szPostalCode,
   Âsizeof(m_szPostalCode), lPostalCodeStatus, TRUE)
18: ADO_VARIABLE_LENGTH_ENTRY2(9, adVarChar, m_szCountry,
   sizeof(m_szCountry), ÂlCountryStatus, TRUE)
19: ADO_VARIABLE_LENGTH_ENTRY2(10, adVarChar, m_szEmailAddress,
   Âsizeof(m_szEmailAddress), lEmailAddressStatus, TRUE)
20: ADO_VARIABLE_LENGTH_ENTRY2(11, adVarChar, m_szHomePhone,
   Âsizeof(m_szHomePhone), lHomePhoneStatus, TRUE)
```

```
21: ADO_VARIABLE_LENGTH_ENTRY2(12, adVarChar, m_szWorkPhone,
Âsizeof(m_szWorkPhone), lWorkPhoneStatus, TRUE)
22: ADO_VARIABLE_LENGTH_ENTRY2(13, adVarChar, m_szWorkExtension,
Âsizeof(m_szWorkExtension), lWorkExtensionStatus, TRUE)
23: ADO_VARIABLE_LENGTH_ENTRY2(14, adVarChar, m_szFaxNumber,
Âsizeof(m_szFaxNumber), lFaxNumberStatus, TRUE)
24: ADO_FIXED_LENGTH_ENTRY(15, adDate, m_dtBirthdate,
lBirthdateStatus, TRUE)
25: ADO_FIXED_LENGTH_ENTRY(16, adBoolean, m_bSendCard,
lSendCardStatus, TRUE)
26: ADO_VARIABLE_LENGTH_ENTRY2(17, adLongVarChar, m_szNotes,
Âsizeof(m_szNotes), lNotesStatus, TRUE)
27: END_ADO_BINDING()
28:
29: public:
30: LONG m_lAddressID;
31: ULONG lAddressIDStatus;
32: CHAR m_szFirstName[51];
33: ULONG lFirstNameStatus;
34: CHAR m_szLastName[51];
35: ULONG lLastNameStatus;
36: CHAR m_szSpouseName[51];
37: ULONG lSpouseNameStatus;
38: CHAR m_szAddress[256];
39: ULONG lAddressStatus;
40: CHAR m_szCity[51];
41: ULONG lCityStatus;
42: CHAR m_szStateOrProvince[21];
43: ULONG lStateOrProvinceStatus;
44: CHAR m_szPostalCode[21];
45: ULONG lPostalCodeStatus;
46: CHAR m_szCountry[51];
47: ULONG lCountryStatus;
48: CHAR m_szEmailAddress[51];
49: ULONG lEmailAddressStatus;
50: CHAR m_szHomePhone[31];
51: ULONG lHomePhoneStatus;
52: CHAR m_szWorkPhone[31];
53: ULONG lWorkPhoneStatus;
54: CHAR m_szWorkExtension[21];
55: ULONG lWorkExtensionStatus;
56: CHAR m_szFaxNumber[31];
57: ULONG lFaxNumberStatus;
58: DATE m_dtBirthdate;
59: ULONG lBirthdateStatus;
60: VARIANT_BOOL m_bSendCard;
61: ULONG lSendCardStatus;
62: CHAR m_szNotes[65536];
63: ULONG lNotesStatus;
```

```
64: };
```

Nachdem Sie diese Klasse erstellt haben, ist eine Variable in die Dokumentklasse aufzunehmen. Fügen Sie eine neue Member-Variable in die Dokumentklasse ein, wobei Sie den Variablentyp als `CCustomRs`, den Namen als `m_rsRecSet` und den Zugriff als Privat festlegen. Weiterhin müssen Sie die Header-Datei der benutzerdefinierten Datensatzklasse in die Quelldatei des Dokuments aufnehmen, wie es aus Listing 15.2 hervorgeht.

Listing 15.2: Die #include-Dateien in der Quelldatei des Dokuments

```
1: // DbAdoDoc.cpp : Implementierung der Klasse CDbAdoDoc
2: //
3:
4: #include "stdafx.h"
5: #include "dbado.h"
6:
7: #include "CustomRs.h"
8: #include "DbAadoDoc.h"
9: #include "DbAdoView.h"
```

Bevor es weitergeht, müssen Sie erst darauf achten, daß Sie der Ansicht eine Möglichkeit bieten, einen Zeiger auf die Datensatzklasse von der Dokumentklasse zu erhalten. Diese Funktion sollte einen Zeiger auf die Variable der Datensatzklasse zurückgeben. Um die Funktion in Ihre Anwendung aufzunehmen, fügen Sie der Dokumentklasse eine neue Member-Funktion hinzu, wobei Sie den Funktionstyp mit `CCustomRs*`, die Funktionsdeklaration mit `GetRecSet` und den Zugriff als Public festlegen. In die Funktion schreiben Sie den Code aus Listing 15.3.

Listing 15.3: Die Funktion GetRecSet der Klasse CDbAdoDoc

```
1: CCustomRs* CDbAdoDoc::GetRecSet()
2: {
3: // Zeiger auf das Datensatzobjekt zurückgeben
4: return &m_rsRecSet;
5: }
```

Der letzte Teil der Funktionalität, die Sie realisieren müssen, bevor es wirklich in medias res mit der ADO-Programmierung geht, ist die Funktion, die Fehler von ADO und der Datenbank meldet. Diese Funktion zeigt dem Benutzer eine Meldung mit dem Fehlercode und einer Beschreibung an. Fügen Sie dazu eine neue Member-Funktion in die Dokumentklasse ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `GenerateError(HRESULT hr, PWSTR pwszDescription)` und den Zugriff als Public fest. Geben Sie in die Funktion den Code gemäß Listing 15.4 ein.

Listing 15.4: Die Funktion GenerateError der Klasse CDbAdoDoc

```
1: void CDbAdoDoc::GenerateError(HRESULT hr, PWSTR pwszDescription)
2: {
3: CString strError;
4:
5: // Fehlermeldung formatieren und anzeigen
6: strError.Format("Laufzeitfehler '%d (%x)'", hr, hr);
7: strError += "\n\n";
8: strError += pwszDescription;
```



```

9:
10: AfxMessageBox(strError);
11: }

```

Verbinden und Daten abrufen

In der Funktion `OnNewDocument` in der Dokumentklasse lassen sich alle Aufgaben in bezug auf die Verbindung zur Datenbank und das Abrufen des Recordsets realisieren. Bevor Sie diese Funktionalität hinzufügen können, müssen Sie weitere Variablen in die Dokumentklasse aufnehmen. Es sind ein `Recordset`-Objektzeiger, ein `IADORecordBinding`-Schnittstellenzeiger, verschiedene Zeichenfolgenvariablen zur Aufnahme der Verbindungszeichenfolge für die Datenbank und des auszuführenden SQL-Befehls, der den Recordset füllt, erforderlich. Diese Variablen fügen Sie der Dokumentklasse gemäß Tabelle 15.3 hinzu.

Tabelle 15.3: Member-Variablen der Dokumentklasse

Name	Typ	Zugriff
<code>m_pRs</code>	<code>_RecordsetPtr</code>	Privat
<code>m_piAdoRecordBinding</code>	<code>IADORecordBinding*</code>	Privat
<code>m_strConnection</code>	<code>CString</code>	Privat
<code>m_strCmdText</code>	<code>CString</code>	Privat

In der Funktion `OnNewDocument` führen Sie verschiedene Schritte aus, um die Verbindung herzustellen und den Recordset abzurufen. Zuerst legen Sie die Strings für die Datenbankverbindung und den auszuführenden SQL-Befehl fest. Als nächstes initialisieren Sie die COM-Umgebung und setzen die beiden Zeiger auf den Anfangswert `NULL`. Mit der Funktion `CreateInstance` erzeugen Sie das `Recordset`-Objekt. Öffnen Sie das `Recordset`-Objekt, wobei Sie gleichzeitig die Verbindung zur Datenbank herstellen und den SQL-Befehl ausführen. Binden Sie die Datensatzklasse mit Hilfe des Schnittstellenzeigers `IADORecordBinding` an den Recordset. Schließlich weisen Sie die Ansichtsklasse an, die gebundenen Daten zu aktualisieren und dabei mit einer in Kürze zu erstellenden Funktion den anfänglichen Datensatz für den Benutzer anzuzeigen. Um die genannte Funktionalität zu realisieren, bearbeiten Sie die Funktion `OnNewDocument` in der Dokumentklasse und nehmen den Code ab Zeile 8 von Listing 15.5 auf.

Listing 15.5: Die Funktion `OnNewDocument` der Klasse `CDbAdoDoc`

```

1: BOOL CDbAdoDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8: // Verbindungszeichenfolge und SQL-Befehl festlegen
9: m_strConnection = _T("Provider=MSDASQL.1;Data Source=VCP21DB");
10: m_strCmdText = _T("select * from Adressen");
11:
12: // Zeiger auf Recordset und Bindung initialisieren

```

```

13: m_pRs = NULL;
14: m_piAdoRecordBinding = NULL;
15: // COM-Umgebung initialisieren
16: ::CoInitialize(NULL);
17: try
18: {
19: // Datensatzobjekt erzeugen
20: m_pRs.CreateInstance(__uuidof(Recordset));
21:
22: // Datensatzobjekt öffnen
23: m_pRs->Open((LPCTSTR)m_strCmdText, (LPCTSTR)m_strConnection,
24: adOpenDynamic, adLockOptimistic, adCmdUnknown);
25:
26: // Zeiger auf Bindungsschnittstelle des Datensatzes holen
27: if (FAILED(m_pRs->QueryInterface(__uuidof(IADORecordBinding),
28: (LPVOID *)&m_piAdoRecordBinding)))
29: _com_issue_error(E_NOINTERFACE);
30: // Datensatzklasse an Recordset binden
31: m_piAdoRecordBinding->BindToRecordset(&m_rsRecSet);
32:
33: // Zeiger auf Ansicht holen
34: POSITION pos = GetFirstViewPosition();
35: CDbAdoView* pView = (CDbAdoView*)GetNextView(pos);
36: if (pView)
37: // Datensatzgruppe mit Formular synchronisieren
38: pView->RefreshBoundData();
39: }
40: // Fehler vorhanden?
41: catch (_com_error &e)
42: {
43: // Fehler anzeigen
44: GenerateError(e.Error(), e.Description());
45: }
46:
47: return TRUE;
48: }

```

Bevor Sie weitergehen, sollten Sie zunächst mit entsprechendem Code sicherstellen, daß beim Schließen der Anwendung alles ordnungsgemäß aufgeräumt wird. Sie müssen den Recordset schließen und den Zeiger auf die Bindungsschnittstelle des Datensatzes freigeben. Weiterhin ist die COM-Umgebung herunterzufahren. Zu diesem Zweck fügen Sie eine Behandlungsfunktion für die Nachricht `DeleteContents` in die Dokumentklasse ein. Nehmen Sie in die Funktion den Code aus Listing 15.6 auf.

Listing 15.6: Die Funktion `DeleteContents` der Klasse `CDbAdoDoc`

```

1: void CDbAdoDoc::DeleteContents()
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
   aufrufen

```



```

4: // Recordset schließen
5: if (m_pRs)
6: m_pRs->Close();
7: // Ist Zeiger auf Datensatzbindung gültig?
8: if (m_piAdoRecordBinding)
9: // Freigeben
10: m_piAdoRecordBinding->Release();
11: // Recordset-Zeiger auf NULL setzen
12: m_pRs = NULL;
13:
14: // COM-Umgebung herunterfahren
15: CoUninitialize();
16:
17: CDocument::DeleteContents();
18: }

```

Das Formular füllen

Um die Spaltenwerte des Datensatzes für den Benutzer anzuzeigen, fügen Sie eine Funktion hinzu, die die Werte aus der Datensatzklasse in die Ansichtsvariablen kopiert. Diese Funktion muß zunächst einen Zeiger auf die Datensatzklasse aus der Dokumentklasse ermitteln. Als nächstes prüft die Funktion den Status der einzelnen Felder in der Datensatzklasse, um sich zu vergewissern, daß der Kopiervorgang starten kann. Anschließend wird der Wert kopiert. Nachdem Sie alle Werte kopiert haben, rufen Sie die Funktion `UpdateData` auf, um die Werte in den Steuerelementen auf dem Formular anzuzeigen. Fügen Sie dazu eine neue Member-Funktion in die Ansichtsklasse ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `RefreshBoundData` und den Zugriff als `Public` fest. In die Funktion übernehmen Sie den Code aus Listing 15.7.

Listing 15.7: Die Funktion `RefreshBoundData` der Klasse `CDbAdoView`

```

1: void CDbAdoView::RefreshBoundData()
2: {
3: CCustomRs* pRs;
4:
5: // Zeiger auf Dokumentobjekt holen
6: pRs = GetDocument()->GetRecSet();
7:
8: // Ist das Feld OK?
9: if (adFldOK == pRs->lAddressIDStatus)
10: // Wert kopieren
11: m_lAddressID = pRs->m_lAddressID;
12: else
13: // Andernfalls Wert auf 0 setzen
14: m_lAddressID = 0;
15: // Ist das Feld OK?
16: if (adFldOK == pRs->lFirstNameStatus)
17: // Wert kopieren
18: m_strFirstName = pRs->m_szFirstName;
19: else
20: // Andernfalls Wert auf 0 setzen

```

```
21: m_strFirstName = _T("");
22: if (adFldOK == pRs->lLastNameStatus)
23: m_strLastName = pRs->m_szLastName;
24: else
25: m_strLastName = _T("");
26: if (adFldOK == pRs->lSpouseNameStatus)
27: m_strSpouseName = pRs->m_szSpouseName;
28: else
29: m_strSpouseName = _T("");
30: if (adFldOK == pRs->lAddressStatus)
31: m_strAddress = pRs->m_szAddress;
32: else
33: m_strAddress = _T("");
34: if (adFldOK == pRs->lCityStatus)
35: m_strCity = pRs->m_szCity;
36: else
37: m_strCity = _T("");
38: if (adFldOK == pRs->lStateOrProvinceStatus)
39: m_strStateOrProvince = pRs->m_szStateOrProvince;
40: else
41: m_strStateOrProvince = _T("");
42: if (adFldOK == pRs->lPostalCodeStatus)
43: m_strPostalCode = pRs->m_szPostalCode;
44: else
45: m_strPostalCode = _T("");
46: if (adFldOK == pRs->lCountryStatus)
47: m_strCountry = pRs->m_szCountry;
48: else
49: m_strCountry = _T("");
50: if (adFldOK == pRs->lEmailAddressStatus)
51: m_strEmailAddress = pRs->m_szEmailAddress;
52: else
53: m_strEmailAddress = _T("");
54: if (adFldOK == pRs->lHomePhoneStatus)
55: m_strHomePhone = pRs->m_szHomePhone;
56: else
57: m_strHomePhone = _T("");
58: if (adFldOK == pRs->lWorkPhoneStatus)
59: m_strWorkPhone = pRs->m_szWorkPhone;
60: else
61: m_strWorkPhone = _T("");
62: if (adFldOK == pRs->lWorkExtensionStatus)
63: m_strWorkExtension = pRs->m_szWorkExtension;
64: else
65: m_strWorkExtension = _T("");
66: if (adFldOK == pRs->lFaxNumberStatus)
67: m_strFaxNumber = pRs->m_szFaxNumber;
68: else
69: m_strFaxNumber = _T("");
```

```

70: if (adFldOK == pRs->lBirthdateStatus)
71: m_oledtBirthdate = pRs->m_dtBirthdate;
72: else
73: m_oledtBirthdate = 0L;
74: if (adFldOK == pRs->lSendCardStatus)
75: m_bSendCard = VARIANT_FALSE == pRs->m_bSendCard ? FALSE : TRUE;
76: else
77: m_bSendCard = FALSE;
78: if (adFldOK == pRs->lNotesStatus)
79: m_strNotes = pRs->m_szNotes;
80: else
81: m_strNotes = _T("");
82:
83: // Daten mit Steuerelementen synchronisieren
84: UpdateData(FALSE);
85: }

```



Da Sie direkt mit der benutzerdefinierten Datensatzklasse, die Sie in dieser Funktion erzeugt haben, arbeiten, müssen Sie die Header-Datei für Ihre benutzerdefinierte Datensatzklasse in die Quelldatei der Ansichtsklasse einbinden, genau wie Sie es bei der Quelldatei der Dokumentklasse getan haben.

Aktualisierungen speichern

Wenn Sie Änderungen zurück in den Recordset kopieren müssen, verläuft das Kopieren der Daten aus den Steuerelementen auf dem Formular in die Variablen der Datensatzklasse in umgekehrter Richtung. Dabei können Sie alle Werte kopieren, ob diese sich geändert haben oder nicht, oder Sie vergleichen die beiden Werte auf Änderungen, um zu entscheiden, welche zurückkopiert werden müssen. Die betreffende Funktion rufen Sie auf, bevor der Benutzer zu anderen Datensätzen im Recordset navigieren kann, damit alle vom Benutzer vorgenommenen Änderungen in der Datenbank gespeichert werden. Zu diesem Zweck fügen Sie eine neue Member-Funktion in die Ansichtsklasse ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `UpdateBoundData` und den Zugriff als Privat fest. Nehmen Sie den Code aus Listing 15.8 in die Funktion auf.

Listing 15.8: Die Funktion `UpdateBoundData` der Klasse `CDbAdoView`

```

1: void CDbAdoView::UpdateBoundData()
2: {
3: CCustomRs* pRs;
4:
5: // Zeiger auf Dokument holen
6: pRs = GetDocument()->GetRecSet();
7:
8: // Steuerelemente mit Variablen synchronisieren
9: UpdateData(TRUE);
10:
11: // Feld geändert? Wenn ja, Wert zurück kopieren
12: if (m_lAddressID != pRs->m_lAddressID)
13: pRs->m_lAddressID = m_lAddressID;

```

```

14: if (m_strFirstName != pRs->m_szFirstName)
15: strcpy(pRs->m_szFirstName, (LPCTSTR)m_strFirstName);
16: if (m_strLastName != pRs->m_szLastName)
17: strcpy(pRs->m_szLastName, (LPCTSTR)m_strLastName);
18: if (m_strSpouseName != pRs->m_szSpouseName)
19: strcpy(pRs->m_szSpouseName, (LPCTSTR)m_strSpouseName);
20: if (m_strAddress != pRs->m_szAddress)
21: strcpy(pRs->m_szAddress, (LPCTSTR)m_strAddress);
22: if (m_strCity != pRs->m_szCity)
23: strcpy(pRs->m_szCity, (LPCTSTR)m_strCity);
24: if (m_strStateOrProvince != pRs->m_szStateOrProvince)
25: strcpy(pRs->m_szStateOrProvince, (LPCTSTR)m_strStateOrProvince);
26: if (m_strPostalCode != pRs->m_szPostalCode)
27: strcpy(pRs->m_szPostalCode, (LPCTSTR)m_strPostalCode);
28: if (m_strCountry != pRs->m_szCountry)
29: strcpy(pRs->m_szCountry, (LPCTSTR)m_strCountry);
30: if (m_strEmailAddress != pRs->m_szEmailAddress)
31: strcpy(pRs->m_szEmailAddress, (LPCTSTR)m_strEmailAddress);
32: if (m_strHomePhone != pRs->m_szHomePhone)
33: strcpy(pRs->m_szHomePhone, (LPCTSTR)m_strHomePhone);
34: if (m_strWorkPhone != pRs->m_szWorkPhone)
35: strcpy(pRs->m_szWorkPhone, (LPCTSTR)m_strWorkPhone);
36: if (m_strWorkExtension != pRs->m_szWorkExtension)
37: strcpy(pRs->m_szWorkExtension, (LPCTSTR)m_strWorkExtension);
38: if (m_strFaxNumber != pRs->m_szFaxNumber)
39: strcpy(pRs->m_szFaxNumber, (LPCTSTR)m_strFaxNumber);
40: if (((DATE)m_oleDtBirthdate) != pRs->m_dtBirthdate)
41: pRs->m_dtBirthdate = (DATE)m_oleDtBirthdate;
42: if (m_bSendCard == TRUE)
43: pRs->m_bSendCard = VARIANT_TRUE;
44: else
45: pRs->m_bSendCard = VARIANT_FALSE;
46: if (m_strNotes != pRs->m_szNotes)
47: strcpy(pRs->m_szNotes, (LPCTSTR)m_strNotes);
48: }

```

Durch den Recordset navigieren

Die Navigation durch den Recordset unterstützen Sie in Ihrer Anwendung mit einer Reihe von Menüs für die vier grundlegenden Navigationsbefehle: Erster, Vorheriger, Nächster und Letzter. Da das Recordset-Objekt und die Schnittstellenzeiger zur Datensatzbindung im Dokumentobjekt angelegt sind, muß man die Nachrichten für diese Menüs an die Dokumentklasse weiterreichen, um den aktuellen Datensatz zu aktualisieren und dann zum ausgewählten Datensatz weiterzuschalten. Allerdings muß die Ansichtsklasse die Nachrichten zuerst empfangen, da sie die geänderten Werte aus den Steuerelementen auf dem Formular zurückkopieren muß, bevor der Recordset aktualisiert wird. Ist die Navigation beendet, muß die Ansicht wiederum das Formular mit den neuen Spaltenwerten des Datensatzes in Übereinstimmung bringen. Wenn man die Sequenz berücksichtigt, wie die Nachrichten zu übergeben sind, ist es am sinnvollsten, die Behandlungsroutine in der Ansichtsklasse unterzubringen und von hier aus die Behandlungsroutine für die Dokumentklasse aufzurufen.

Um die beschriebene Funktionalität in der Anwendung zu realisieren, fügen Sie vier Menübefehle und die korrespondierenden Symbolleistenschaltflächen hinzu. Mit dem Klassen-Assistenten erstellen Sie in der Ansichtsklasse Behandlungsroutinen für die Nachrichten der vier Menübefehle. In die Behandlungsroutine für den Menübefehl Erster nehmen Sie den Code aus Listing 15.9 auf.

Listing 15.9: Die Funktion OnDataFirst der Klasse CDbAdoView

```
1: void CDbAdoView::OnDataFirst()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4: // Aktuellen Datensatz aktualisieren
5: UpdateBoundData();
6: // Zum ersten Datensatz gehen
7: GetDocument()->MoveFirst();
8: // Formular mit Daten des neuen Datensatzes aktualisieren
9: RefreshBoundData();
10: }
```

Als nächstes fügen Sie die Funktion `MoveFirst` (zum ersten Datensatz gehen) in die Dokumentklasse ein und realisieren die eigentliche Funktionalität für den Recordset für diese Funktion. Nehmen Sie dazu eine Member-Funktion in die Dokumentklasse Ihrer Anwendung auf. Legen Sie den Funktionstyp als `void`, die Deklaration mit `MoveFirst` und den Zugriff als `Public` fest. In die Funktion schreiben Sie den Code aus Listing 15.10.

Listing 15.10: Die Funktion MoveFirst der Klasse CDbAdoDoc

```
1: void CDbAdoDoc::MoveFirst()
2: {
3: try
4: {
5: // Aktuellen Datensatz aktualisieren
6: m_piAdoRecordBinding->Update(&m_rsRecSet);
7: // Zum ersten Datensatz gehen
8: m_pRs->MoveFirst();
9: }
10: // Fehler?
11: catch (_com_error &e)
12: {
13: // Fehlermeldung generieren
14: GenerateError(e.Error(), e.Description());
15: }
16: }
```

Für die ADO-Funktionen `MovePrevious` (zum vorherigen Datensatz), `MoveNext` (zum nächsten) und `MoveLast` (zum letzten) fügen Sie entsprechende Funktionspaare in die Ansichts- und Dokumentklasse hinzu. Wenn Sie diese Schritte abgeschlossen haben, können Sie Ihre Anwendung kompilieren und ausführen. Mit der Anwendung ist es jetzt möglich, die Datenbanktabelle Adressen zu öffnen und die einzelnen Datensätze anzuzeigen, zu bearbeiten und zu aktualisieren (siehe Abbildung 15.7).

Unbenannt - DbAdo

Datei Bearbeiten Daten Ansicht ?

Adressen-Nr: 1

Email-Adresse: nicoled@irgendwo.co

Vorname: Nicole

Telefon/privat: (040)5559857

Nachname: Dauster

Telefon/beruflich: (040)5559922

Name des Ehepartners: Paul

Durchwahl Büro:

Adresse: Hofstr. 1
App. 2a

Faxnummer: (040)5557722

Ort: Hamburg

Geburtsdatum: 04.08.72

Bundesland:

☐ Karte senden

Postleitzahl: 21079

Anmerkungen:

Land: Deutschland

Bereit NUM

Abbildung 15.7:
Die laufende Anwendung

Neue Datensätze hinzufügen

In der momentanen Entwicklungsphase der Anwendung können Sie Datensätze aus der Datenbanktabelle abrufen und in der Ergebnismenge - dem Recordset - navigieren. Es wäre wünschenswert, wenn man auch neue Datensätze in die Tabelle aufnehmen könnte. Diese Funktionalität lässt sich in genau der gleichen Weise umsetzen, wie Sie es für die Navigation vorgenommen haben. Zu diesem Zweck fügen Sie einen Menübefehl hinzu, lösen über diesen Menübefehl eine Behandlungsroutine in der Ansichtsklasse aus, schreiben die aktuellen Werte des Datensatzes zurück in den Recordset, rufen eine Funktion in der Dokumentklasse auf und bringen den aktuellen Datensatz aus dem Recordset auf den neuesten Stand. In bezug auf das Menü und die Ansichtsklasse ändert sich gegenüber den Menübefehlen und Funktionen zur Navigation lediglich die ID des Menüs und der Name der aufgerufenen Funktion, praktisch genau wie bei den verschiedenen Navigationsfunktionen. In der Funktion der Dokumentklasse liegen die eigentlichen funktionellen Unterschiede.

Damit sich ein neuer Datensatz hinzufügen lässt, stellen Sie sicher, daß nach dem Aktualisieren des aktuellen Datensatzes in der Dokumentklasse das Hinzufügen eines neuen Datensatzes möglich ist. Wenn diese Möglichkeit besteht, bauen Sie einen leeren Datensatz zusammen und fügen ihn in den Recordset ein. Nachdem Sie den leeren Datensatz aufgenommen haben, navigieren Sie zum letzten

Datensatz in der Ergebnismenge, da dieser den neuen Datensatz repräsentiert. Jetzt können Sie die Funktion verlassen und der Ansichtsklasse die Aufgabe übertragen, das Formular mit den Datenwerten aus dem neuen - leeren - Datensatz zu aktualisieren.

Um diese Funktionalität in der Anwendung zu realisieren, fügen Sie einen neuen Menübefehl für das Hinzufügen eines neuen Datensatzes ein. In die Ansichtsklasse nehmen Sie eine Behandlungsroutine für den neuen Menübefehl auf. In diese Funktion schreiben Sie den gleichen Code wie für die Navigationsfunktionen, rufen aber die Funktion `AddNew` in der Dokumentklasse auf. Als nächstes erstellen Sie die Funktion `AddNew` in der Dokumentklasse. Nehmen Sie dazu mit dem Klassen-Assistenten eine neue Member-Funktion in die Dokumentklasse auf. Legen Sie den Typ als `void`, die Deklaration mit `AddNew` und den Zugriff als `Public` fest. In die Funktion schreiben Sie den Code aus Listing 15.11.

Listing 15.11: Die Funktion `AddNew` der Klasse `CDbAdoDoc`

```

1: void CDbAdoDoc::AddNew()
2: {
3: try
4: {
5: // Aktuellen Datensatz aktualisieren
6: m_piAdoRecordBinding->Update(&m_rsRecSet);
7: // Kann ein neuer Datensatz hinzugefügt werden?
8: if (m_pRs->Supports(adAddNew))
9: {
10: // Leeren Datensatz erzeugen
11: CreateBlankRecord();
12: // Leeren Datensatz hinzufügen
13: m_piAdoRecordBinding->AddNew(&m_rsRecSet);
14: // Zum letzten Datensatz gehen
15: m_pRs->MoveLast();
16: }
17: }
18: // Fehler?
19: catch (_com_error &e)
20: {
21: // Fehlermeldung generieren
22: GenerateError(e.Error(), e.Description());
23: }
24: }

```

Als nächstes fügen Sie die Funktion hinzu, die den leeren Datensatz erzeugt. In dieser Funktion setzen Sie alle Feldvariablen in der Datensatzklasse auf einen fast leeren String (der lediglich ein Leerzeichen enthält). Um diese Funktion in die Klasse aufzunehmen, fügen Sie eine neue Member-Funktion in die Dokumentklasse hinzu. Legen Sie den Typ als `void`, die Deklaration mit `CreateBlankRecord` und den Zugriff als `Privat` fest. Übernehmen Sie in diese Funktion den Code aus Listing 15.12.

Listing 15.12: Die Funktion `CreateBlankRecord`

```

1: void CDbAdoDoc::CreateBlankRecord()
2: {
3: // Als leer verwendete Werte erzeugen

```

```

4: CString strBlank = " ";
5: COleDateTime dtBlank;
6:
7: // Die einzelnen Werte im Datensatzobjekt setzen
8: m_rsRecSet.m_lAddressID = 0;
9: strcpy(m_rsRecSet.m_szFirstName, (LPCTSTR)strBlank);
10: strcpy(m_rsRecSet.m_szLastName, (LPCTSTR)strBlank);
11: strcpy(m_rsRecSet.m_szSpouseName, (LPCTSTR)strBlank);
12: strcpy(m_rsRecSet.m_szAddress, (LPCTSTR)strBlank);
13: strcpy(m_rsRecSet.m_szCity, (LPCTSTR)strBlank);
14: strcpy(m_rsRecSet.m_szStateOrProvince, (LPCTSTR)strBlank);
15: strcpy(m_rsRecSet.m_szPostalCode, (LPCTSTR)strBlank);
16: strcpy(m_rsRecSet.m_szCountry, (LPCTSTR)strBlank);
17: strcpy(m_rsRecSet.m_szEmailAddress, (LPCTSTR)strBlank);
18: strcpy(m_rsRecSet.m_szHomePhone, (LPCTSTR)strBlank);
19: strcpy(m_rsRecSet.m_szWorkPhone, (LPCTSTR)strBlank);
20: strcpy(m_rsRecSet.m_szWorkExtension, (LPCTSTR)strBlank);
21: strcpy(m_rsRecSet.m_szFaxNumber, (LPCTSTR)strBlank);
22: m_rsRecSet.m_dtBirthdate = (DATE)dtBlank;
23: m_rsRecSet.m_bSendCard = VARIANT_FALSE;
24: strcpy(m_rsRecSet.m_szNotes, (LPCTSTR)strBlank);
25: }

```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, können Sie neue Datensätze in die Datenbanktabelle einfügen und bearbeiten.

Datensätze löschen

Als krönenden Abschluß der heutigen Beispielanwendung realisieren wir das Löschen des aktuellen Datensatzes aus dem Recordset. Diese Funktion entspricht weitgehend den Funktionen zur Navigation und zum Hinzufügen. Die Anwendung erhält einen neuen Menübefehl, der eine Behandlungsroutine in der Ansichtsklasse auslöst. Die Funktion in der Ansichtsklasse realisiert die gleichen Aufgaben wie die vorherigen Funktionen: sie aktualisiert den aktuellen Datensatz, ruft die korrespondierende Funktion in der Dokumentklasse auf und bringt dann den aktuellen Datensatz im Formular auf den neuesten Stand.

Die Funktion der Dokumentklasse verfolgt beim Löschen des Datensatzes den gleichen Weg wie beim Hinzufügen. Sie aktualisiert den aktuellen Datensatz, prüft, ob der aktuelle Datensatz zum Löschen freigegeben ist, holt vom Benutzer eine Bestätigung ein, ob der Datensatz wirklich gelöscht werden soll, ruft dann die Funktion **Delete** auf und navigiert zu einem anderen Datensatz im Recordset.

Um diese Funktionalität in Ihrer Anwendung zu realisieren, fügen Sie einen neuen Menübefehl für die Löschen-Funktion hinzu und verbinden ihn mit einer Behandlungsfunktion in der Ansichtsklasse. In die Funktion schreiben Sie den gleichen Code wie bei den Funktionen zum Navigieren und Hinzufügen, rufen aber die Funktion **Delete** in der Dokumentklasse auf. In die Dokumentklasse nehmen Sie deshalb eine neue Member-Funktion auf, für die Sie den Typ als **void**, die Deklaration mit **Delete** und den Zugriff als **Public** festlegen. In die Funktion schreiben Sie den Code aus Listing 15.13.

Listing 15.13: Die Funktion Delete der Klasse CDbAdoDoc

```

1: void CDbAdoDoc::Delete()
2: {
3: try
4: {
5: // Aktuellen Datensatz aktualisieren
6: m_piAdoRecordBinding->Update(&m_rsRecSet);
7: // Kann ein Datensatz gelöscht werden?
8: if (m_pRs->Supports(adDelete))
9: {
10: // Nachfragen, ob Benutzer wirklich diesen Datensatz löschen
    will
11: if (AfxMessageBox("Diesen Datensatz wirklich löschen?",
12: MB_YESNO | MB_ICONQUESTION) == IDYES)
13: {
14: // Datensatz löschen
15: m_pRs->Delete(adAffectCurrent);
16: // Zum vorherigen Datensatz gehen
17: m_pRs->MovePrevious();
18: }
19: }
20: }
21: // Fehler?
22: catch (_com_error &e)
23: {
24: // Fehlermeldung generieren
25: GenerateError(e.Error(), e.Description());
26: }
27: }

```

Jetzt können Sie Ihre Anwendung kompilieren und ausführen. Das Löschen beliebiger Datensätze aus dem Recordset sollte nun funktionieren.

Zusammenfassung

Heute haben Sie die neueste Technologie für den Datenbankzugriff von Microsoft kennengelernt: ActiveX Data Objects (ADO). Es wurde gezeigt, wie man ADO als einfaches ActiveX-Steuer-element einsetzt, um Datenbankzugriff über datenbezogene Steuerelemente ohne zusätzliche Programmierung bereitzustellen. Sie haben die DLL importiert, die eine umfangreiche Palette von Funktionen für den Datenzugriff bereitstellt, und haben erfahren, wie man diese Funktionalität in eigenen Anwendungen verwendet und steuert. Mit der heutigen Beispielanwendung ist es möglich, Datensätze abzurufen, die Datensätze im Recordset (der Ergebnismenge) zu manipulieren und die Änderungen in die Datenbank zu übernehmen. Für den Zugriff und die Aktualisierung der Datenwerte in einem Datensatz des Recordsets haben Sie zwei verschiedene Wege kennengelernt. Vor allem wurde gezeigt, wie man mit etwas mehr Aufwand zu Beginn einen beträchtlichen Teil der Arbeit bei der Anwendungsentwicklung auf lange Sicht einspart.

Fragen und Antworten

Frage:

Visual C++ bietet für ADO keine Unterstützung in Form von Assistenten. Warum sollte ich dennoch auf ADO zurückgreifen?

Antwort:

Mit ADO gibt Microsoft den Trend für die Technologien des Datenbankzugriffs vor. Momentan haben wir es noch mit frühen Entwicklungsstufen dieser Technologie zu tun. Zukünftig wird diese Technologie jedoch in allen Programmiersprachen und Anwendungen zu finden sein.

Frage:

Wenn ADO den Zugriff auf meine Datenbank über ODBC realisiert, warum geht man dann nicht gleich den direkten Weg über die ODBC-Schnittstelle, um auf die Datenbank zuzugreifen?

Antwort:

ADO kann ODBC nutzen, um auf diejenigen Datenbanken zuzugreifen, die keine native OLE DB-Schnittstelle haben. Wenn Sie mit Datenbanken von Microsoft SQL Server oder von Oracle arbeiten, sind OLE DB-Schnittstellen vorhanden. In diesen Fällen greift ADO nicht über ODBC auf die Datenbank zu und bringt für die Anwendung ein besseres Leistungsverhalten. Mit zukünftigen Betriebssystemen von Microsoft bringt der Einsatz von ADO wahrscheinlich Zugriffsmöglichkeiten, die weit über konventionelle Datenbanken hinausgehen werden. ADO ist eine neue Technologie, deren Verbreitung in den kommenden Jahren zunehmen wird. Aus diesem Grund empfiehlt es sich, bereits jetzt mit ADO Bekanntschaft zu schließen, damit Sie zum richtigen Zeitpunkt darauf vorbereitet sind.

Workshop

Kontrollfragen

1. Was bezeichnet die Abkürzung ADO?
2. Was verwendet ADO für den Datenbankzugriff?
3. Welche Objekte sind in ADO vorhanden?
4. Wie initialisiert man die COM-Umgebung?
5. Wie verbindet man ein `Connection`-Objekt mit einem `Command`-Objekt?
6. Wie verbindet man ein `Recordset`-Objekt mit einem `Command`-Objekt und füllt den `Recordset` mit Daten?

Übung

Aktivieren und deaktivieren Sie die Menübefehle und Symbolleistenflächen zur Navigation, je nachdem, ob der `Recordset` am Beginn der Datei (`BOF`) oder am Ende der Datei (`EOF`, umbenannt zu `EndOfFile`) steht.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 3**Tag 16****Eigene Klassen und Module**

Manchmal ist eine Gruppe von Funktionen zu erstellen, die in einer Anwendung, an der ein anderer Programmierer arbeitet, zum Einsatz kommen. Es kann sein, daß die Funktionen in einer ganzen Reihe von Anwendungen benötigt werden. Weiterhin ist es möglich, daß Sie eine bestimmte Funktionsgruppe aus organisatorischen Gründen von der übrigen Anwendung abtrennen möchten. Diese funktionellen Einheiten können Sie dann separat entwickeln und den Code an die betreffenden Mitarbeiter weitergeben. Das hat allerdings den Nachteil, daß Änderungen, die Sie an der Funktionsgruppe vornehmen, auch noch in den bereits weitergegebenen Code eingebunden werden müssen. Es wäre praktischer, wenn man eine kompilierte Version der Funktionsgruppe an den anderen Programmierer weitergeben könnte, so daß Sie bei einer Aktualisierung auf Ihrer Seite lediglich die neu kompilierte Datei aushändigen müßten. Die neue Datei könnte einfach die vorherige Version ersetzen, ohne daß der andere Programmierer seinen bisher erstellten Code ändern muß.

Es ist natürlich möglich, einen ganzen Satz von Funktionalität in eine eigenständig kompilierte Datei zu verpacken, diese zur Anwendung eines anderen Programmierers zu linken und damit das Hinzufügen irgendwelcher neuer Dateien zur fertiggestellten Anwendung zu vermeiden. Heute lernen Sie, wie man ...

- eigene Klassen entwirft,
- kompilierte Module erzeugt, die sich zu anderen Anwendungen linken lassen,
- diese Module in eine Anwendung einbindet.

Klassen entwerfen

In den vergangenen Tagen haben Sie bereits eigene Klassen entworfen und erstellt, so daß dieses Thema nicht grundsätzlich neu für Sie ist. Warum haben Sie diese Klassen erzeugt? Die neu erzeugten Klassen haben alle eine Gruppe von Funktionalität verkapselt, die als selbständige Einheit agiert. Diese Einheiten bestehen sowohl aus Daten als auch als Funktionen, die in ihrer Gesamtheit das Objekt definieren.

Kapselung

Der objektorientierte Softwareentwurf verfolgt das Konzept, nach dem alles in der uns umgebenden Welt aufgebaut ist. Betrachten Sie zum Beispiel ein Auto, das aus einer Sammlung von Objekten besteht: Motor, Karosserie, Radaufhängung usw. Jedes dieser Objekte besteht aus einem Bündel anderer Objekte. Beispielsweise enthält der Motor entweder den Vergaser oder die Einspritzventile, die

Verbrennungskammer und die Kolben, den Anlasser, die Lichtmaschine, den Keilriemen usw. Auch hier bestehen die einzelnen Objekte wiederum aus noch mehr Objekten.

Alle diese Objekte haben eine Funktion, die sie ausführen. Jedes Objekt weiß, wie die eigenen Funktionen auszuführen sind, ohne oder nur mit geringen Kenntnissen, wie die anderen Objekte ihre Funktionen wahrnehmen. Jedes Objekt weiß, wie es mit den anderen Objekten in Wechselwirkung treten kann und wie es mit den anderen Objekten verbunden ist, mehr ist aber nicht über die anderen Objekte bekannt. Wie jedes dieser Objekte intern arbeitet, bleibt vor den anderen Objekten verborgen. Die Bremsen im Auto wissen zum Beispiel nichts darüber, wie das Getriebe funktioniert, bei einem Automatikgetriebe wissen aber die Bremsen, wie sie dem Getriebe mitteilen, daß sie aktiv sind, und das Getriebe entscheidet, wie auf diese Informationen zu reagieren ist.

Auf die gleiche Weise gehen Sie den Entwurf neuer Klassen für Ihre Anwendungen an. Die übrigen Anwendungsobjekte brauchen nicht zu wissen, wie Ihre Objekte arbeiten, sie müssen nur wissen, wie sie mit Ihren Objekten in Wechselwirkung treten. Diese sogenannte *Kapselung* gehört zu den Grundprinzipien der objektorientierten Software.

Vererbung

Ein weiteres Schlüsselprinzip des objektorientierten Softwareentwurfs ist die *Vererbung*. Ein Objekt kann von einem anderen Objekt geerbt sein. Das abgeleitete Objekt erbt die gesamte Funktionalität des Basisobjekts. Damit läßt sich das abgeleitete Objekt in Form von Änderungen gegenüber dem Basisobjekt definieren.

Sehen wir uns dieses Konzept am Beispiel eines Thermostaten an. Nehmen wir einen einfachen Thermostaten an, den Sie in nahezu jeder Einstellung verwenden können. Es läßt sich die Temperatur vorgeben, die der Thermostat regeln soll. Zu diesem Zweck schaltet er die Heizung oder Klimaanlage ein. Jetzt wollen wir einen Thermostat für einen Gefrierschrank erstellen. Man könnte nun einen eigens entwickelten Thermostaten von Grund auf neu aufbauen - oder man nimmt den vorhandenen Thermostaten und legt fest, wie sich die Gefrierschrankversion vom Original unterscheidet. Zu diesen Unterschieden kann gehören, daß der Thermostat nur noch die Klimaanlage einschalten muß und die Heizung überhaupt nicht mehr. Vielleicht möchten Sie auch den Temperaturbereich streng eingrenzen, auf den sich der Thermostat einstellen läßt, beispielsweise auf Temperaturen unterhalb 0° Celsius. Wenn Sie analog dazu einen Thermostat für eine Büroheizung brauchen, grenzen Sie wahrscheinlich den Temperaturbereich auf die üblichen Raumtemperaturen ein und erlauben keine Werte für extreme Kälte oder Hitze.

Das gleiche Prinzip verfolgen Sie, wenn Sie eigene Klassen per Vererbung erstellen. Nach Möglichkeit sollten Sie mit einer vorhandenen C++-Klasse, die bereits die erforderliche Basisfunktionalität mitbringt, beginnen. Dann programmieren Sie, wie sich Ihre Klasse von der Basisklasse, von der Sie erben, unterscheidet. Dabei können Sie neue Datenelemente hinzufügen, die vorhandene Funktionalität erweitern oder vorhandene Funktionen überschreiben, wie es gerade in Ihr Konzept paßt.

Klassentypen in Visual C++

Wenn Sie eine neue Klasse erzeugen, stehen Ihnen in den meisten Anwendungsprojekten verschiedene Optionen zum Typ der zu erzeugenden Klasse zur Verfügung. Zu diesen Optionen gehören:

- Allgemeine Klasse
- MFC-Klasse
- Formularklasse

Von welchem Typ Sie Ihre Klasse erzeugen, hängt von Ihren Anforderungen und den Aufgaben der Klasse ab. Weiterhin ist zu beachten, ob die Klasse von einer MFC-Klasse abzuleiten ist.

Allgemeine Klassen

Aus einer allgemeinen (oder generischen) Klasse erstellen Sie eine Klasse, die von einer bereits erstellten Klasse abgeleitet ist. Mit diesem Klassentyp erzeugen Sie in erster Linie Klassen, die von keiner MFC-Klasse abgeleitet sind (obwohl Sie bereits gesehen haben, wo man mit diesem Typ Klassen erstellt, die auf MFC-Klassen basieren). Wenn Sie eine speziellere Version der Klasse `CLine` erstellen möchten, zum Beispiel eine Klasse `CRedLine`, die nur in Rot zeichnet, erzeugen Sie eine allgemeine Klasse, weil sie von einer anderen Klasse, die Sie erzeugt haben, abgeleitet ist.

Wenn Sie eine allgemeine Klasse erzeugen, versucht der *neue Klassen-Assistent* (nicht zu verwechseln mit dem Klassen-Assistenten, der auf MFC-Klassen beschränkt ist) die Deklaration der Basisklasse - die Header-Datei mit der deklarierten Klasse - zu lokalisieren. Wenn der Assistent die entsprechende Header-Datei nicht finden kann, teilt er Ihnen in einem Meldungsfeld mit, daß Sie die Header-Datei mit der Definition der Basisklasse selbst in das Projekt einbinden müssen. Wenn es sich bei der Basisklasse um eine MFC-Klasse handelt, die nicht als MFC-Klasse zugänglich ist (wie etwa `CObject`), dann können Sie diese Warnung ignorieren, da die korrekte Header-Datei bereits Teil des Projekts ist.

MFC-Klassen

Wenn Sie eine wiederverwendbare Klasse erstellen möchten, die auf einer vorhandenen MFC-Klasse aufbaut, wie etwa ein Eingabefeld, das automatisch Zahlen als Währung formatiert, erstellen Sie eine MFC-Klasse. Die Option *MFC-Klasse* dient der Erstellung neuer Klassen, die von existierenden MFC-Klassen abgeleitet sind.

Formularklassen

Die Formularklasse ist ein spezialisierter Typ von MFC-Klasse. Diesen Klassentyp müssen Sie erzeugen, wenn Sie ein neues Formularfenster erstellen. Dabei kann es sich um eine Ansichtsklasse für ein Dialogfeld, ein Formular oder eine Datenbank handeln. Diese neue Klasse wird mit einer Dokumentklasse verbunden, die zusammen mit der Ansichtsklasse zum Einsatz kommt. Wenn Sie eine Datenbankanwendung erstellen, werden Sie wahrscheinlich eine Reihe derartiger Klassen erzeugen.

Bibliotheksmodule erstellen

Wenn Sie neue Klassen für Ihre Anwendung erstellen, können diese Klassen auch in anderen Anwendungen nützlich sein. Oftmals lassen sich die zu erzeugenden Klassen mit etwas Nachdenken und geringem Mehraufwand so flexibel gestalten, daß man sie in anderen Anwendungen einsetzen kann. In diesem Fall brauchen Sie eine Möglichkeit, um die Klassen für andere Anwendungen zusammenzupacken, ohne dabei den Quellcode aus der Hand geben zu müssen. Für diesen Zweck sind Bibliotheksmodule prädestiniert. Hiermit können Sie Ihre Klassen und Module in eine kompilierte Objektcodebibliothek kompilieren, die sich mit jeder anderen Visual C++-Anwendung linken läßt.

Bibliotheksmodule waren als eines der ersten Mittel verfügbar, um anderen Programmierern für deren Anwendungen kompilierten Code bereitzustellen. Der Linker kombiniert den Code mit dem Code der übrigen Anwendung im letzten Schritt beim Erstellungsvorgang. Bibliotheksmodule sind auch heute noch ein probates Mittel, um Module mit anderen Entwicklern gemeinsam nutzen zu können. Der Entwickler benötigt lediglich die Bibliotheksdatei (mit der Erweiterung `.lib`) und die entsprechenden Header-Dateien, die alle freigelegten Klassen, Methoden, Funktionen und Variablen zeigen, auf die der

andere Programmierer zugreifen kann. Am einfachsten läßt sich das bewerkstelligen, indem man dieselbe Header-Datei, die man für das Erstellen der Bibliotheksdatei verwendet, bereitstellt. Man kann die Header-Datei aber auch bearbeiten, damit nur die von anderen Programmierern benötigten Teile eingebunden sind.

Wenn Sie auf Bibliotheksdateien zurückgreifen, um Module mit anderen Programmierern gemeinsam zu nutzen, ist praktisch vereinbart, daß Ihr Teil der Anwendung in derselben ausführbaren Datei wie die übrige Anwendung eingebunden ist. Ihre Module sind nicht in einer separaten Datei enthalten, wie etwa bei einer DLL oder einem ActiveX-Steuererelement. Dadurch ist eine Datei weniger mit der Anwendung zu vertreiben. Es bedeutet auch, daß bei allen Änderungen am Modul - bei der Beseitigung von Fehlern oder der Erweiterung der Funktionalität - die Anwendung, die das Modul verwendet, neu gelinkt werden muß. Gegenüber DLLs sind Bibliotheksdateien etwas im Nachteil. Eine neue DLL können Sie ohne weiteres vertreiben, ohne daß irgendwelche Änderungen an der Anwendung erforderlich sind. Aber darauf gehen wir morgen ein.

Bibliotheksmodule einsetzen

Um eine Vorstellung davon zu bekommen, wie man Bibliotheksmodule verwendet, erstellen Sie heute ein Bibliotheksmodul, verwenden es in einer anderen Anwendung und nehmen dann einige Modifikationen am Bibliotheksmodul vor. Für die heutige Beispielanwendung erstellen Sie ein Modul, das eine Zufallszeichnung im angegebenen Fensterbereich generiert. Es wird möglich sein, alle diese Zeichnungen zu speichern und wiederherzustellen. Dann verwenden Sie dieses Modul in einer SDI-Anwendung, wo immer dann, wenn man ein neues Dokument anlegt, eine neue Zeichnung generiert wird. Das anfängliche Modul verwendet nur acht Farben und erzeugt nur eine begrenzte Zahl von Linienfolgen. Später modifizieren Sie das Modul, so daß es eine beliebige Anzahl von Farben verwendet und eine größere Anzahl von Liniensequenzen generiert.

Das Bibliotheksmodul erstellen

Um das Projekt eines Bibliotheksmoduls zu erstellen, wählen Sie auf der Registerkarte **Projekte** des Dialogfelds **Neu** den Eintrag *Win32-Bibliothek (statische)*, wie es Abbildung 16.1 zeigt. Damit teilen Sie Visual C++ mit, daß das Kompilat des Projekts ein Bibliotheksmodul und keine ausführbare Anwendung ist. Von da an brauchen Sie nur noch die Klassen zu definieren und den Code hinzuzufügen. Im einzigen Dialogfeld des Projektassistenten haben Sie die Wahl, Unterstützung für MFC einzubinden und vorkompilierte Header in Ihrem Projekt zu verwenden (siehe Abbildung 16.2).

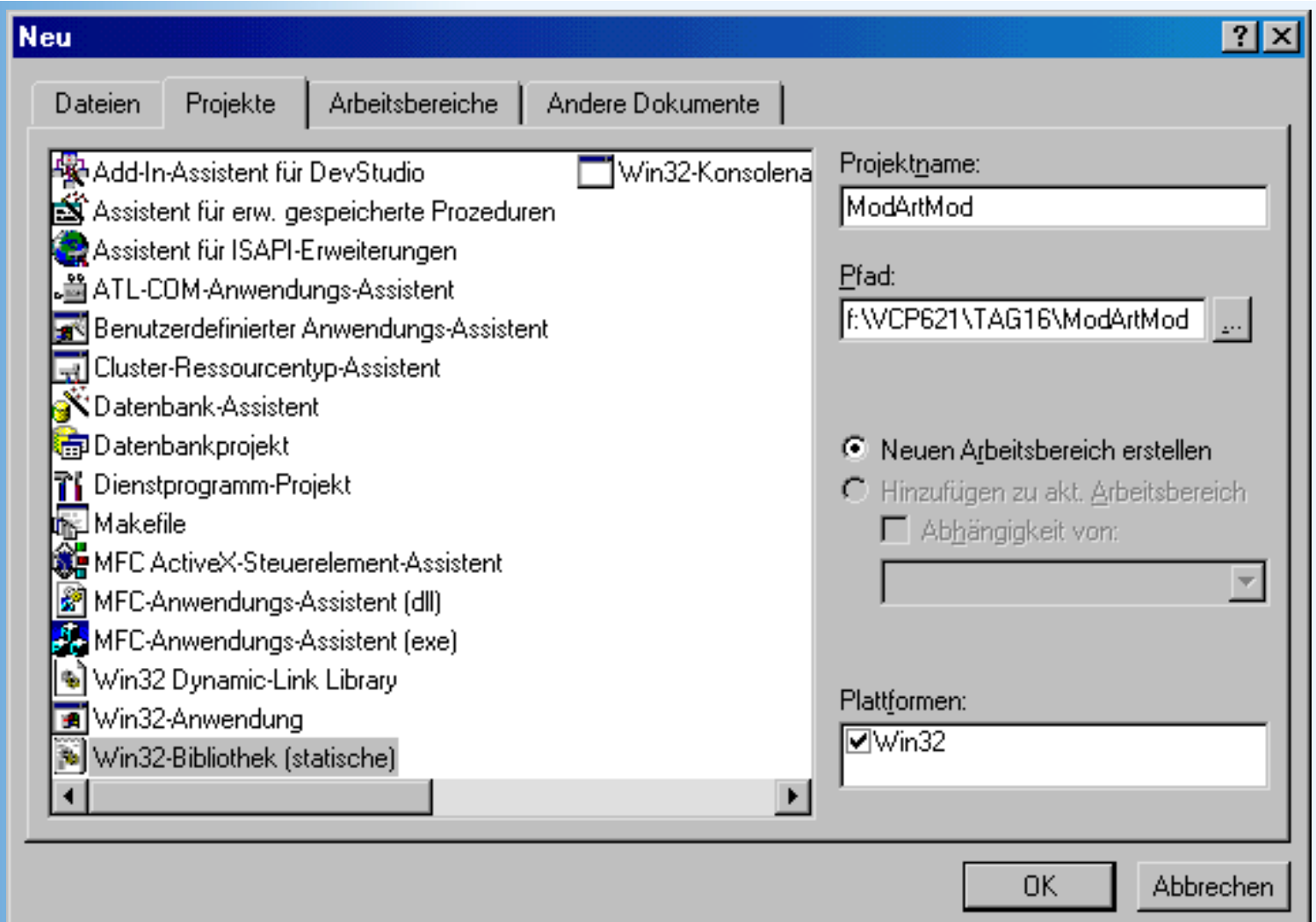


Abbildung 16.1:
Das Projekt eines Bibliotheksmoduls spezifizieren

Die Bibliothek, die Sie für die heutige Beispielanwendung erstellen, besteht aus zwei Klassen. Die erste Klasse ist die Klasse `CLine`, die Sie bereits am Tag 10 erstellt haben. Die zweite Klasse erzeugt zufällige Zeichnungen im Zeichenbereich. Diese Klasse enthält ein Objektarray von `CLine`-Objekten, die die Klasse erzeugt und mit jedem neuen Zeichenversuch füllt. Die zweite Klasse muß auch über die Möglichkeit verfügen, die Zeichnung zu speichern und wiederherzustellen sowie die vorhandene Zeichnung zu löschen, damit eine neue Zeichnung begonnen werden kann. Die Klasse muß die Abmessungen des Zeichenbereichs kennen, damit sie eine Zeichnung generieren kann, die in den Zeichenbereich paßt. Nachdem Sie dieses Modul erstellt haben, sehen Sie sich an, wie es sich in einem Anwendungsprojekt einsetzen läßt.

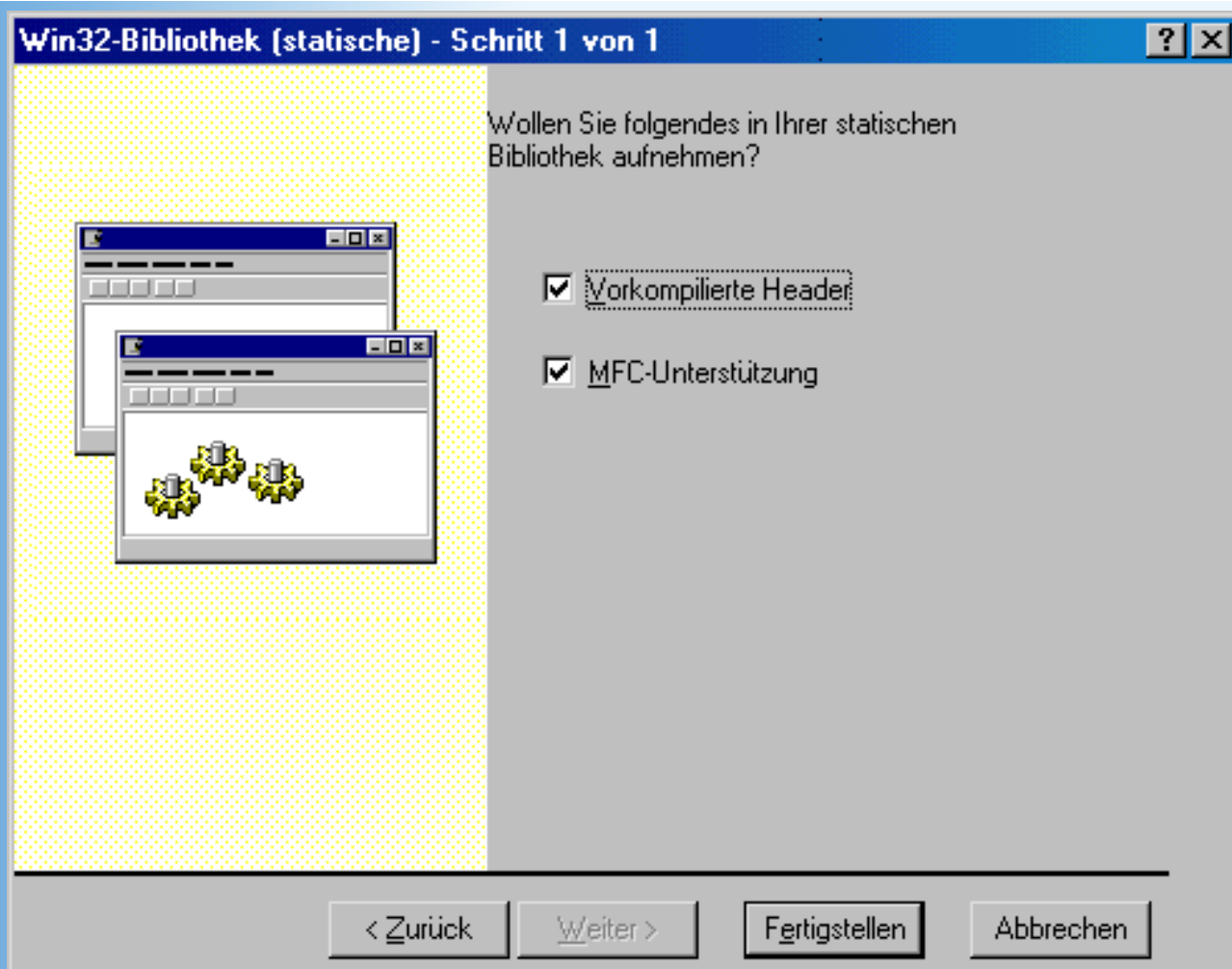


Abbildung 16.2:
Unterstützungsoptionen für das Projekt festlegen

Ein Bibliotheksprojekt erstellen

Um das Bibliotheksprojekt für das heutige Beispiel zu beginnen, legen Sie ein neues Projekt vom Typ *Win32-Bibliothek (statische)* an. Geben Sie dem Projekt einen passenden Namen, und klicken Sie auf **OK**, um das Projekt zu erzeugen.

Im ersten und einzigen Schritt des Assistenten legen Sie fest, daß sowohl MFC-Unterstützung als auch vorkompilierte Header einzubinden sind. Obwohl die Unterstützung für vorkompilierte Header nicht erforderlich ist, beschleunigt das in der Regel den Erstellungsvorgang des Moduls.

Nach diesen Schritten haben Sie ein Projekt vor sich, das keinerlei Klassen enthält. Sie haben einen Rohdiamanten, aus dem Sie nach Belieben die verschiedensten Modultypen erzeugen können.

Da Sie bereits die Klasse `CLine` erstellt haben, kopieren Sie sie für das heutige Beispielpjekt aus dem Projektverzeichnis von Tag 10 in das Projektverzeichnis für das heutige Projekt. Nehmen Sie sowohl die Header-Datei als auch die Quellcodedatei in das heutige Projekt auf. Wählen Sie dazu **Projekt / Dem Projekt hinzufügen / Dateien**. Wenn Sie beide Dateien dem Projekt hinzugefügt haben, sollte die Klasse `CLine` in der Klassenansicht des Projekts erscheinen.

Klassen definieren

Nachdem Sie nun über ein einsatzbereites Gerüst eines Bibliotheksmoduls verfügen, ist das Modul noch mit Leben zu erfüllen. Mit der Klasse `CLine` bietet sich eine gute Möglichkeit, eine bereits mit anderen Vorgaben realisierte Funktionalität wiederzuverwenden. Allerdings soll die eigentliche Funktionalität

des Moduls darin bestehen, zufällige Zeichnungen - oder Squiggles - zu erzeugen. Für diese Funktionalität benötigen Sie eine neue Klasse.

Um eine neue Klasse in das Projekt einzufügen, klicken Sie mit der rechten Maustaste auf der Registerkarte **Klassen** und wählen **Neue Klasse** aus dem Kontextmenü. Als erstes fällt im Dialogfeld **Neue Klasse** auf, daß als Klassentyp lediglich *Allgemeine Klasse* zur Verfügung steht. Da Sie eine statische Bibliothek erzeugen, die mit der Anwendung gelinkt wird, trifft Visual C++ einige Annahmen über den Klassentyp, den Sie erstellen möchten. Da es sich nicht um ein MFC-Projekt handelt, auch wenn Sie die MFC-Unterstützung eingebunden haben, läßt sich keine neue MFC- oder Formulkasse erzeugen. Wollen Sie eine neue Klasse von einer MFC-Klasse ableiten, müssen Sie sie so hinzufügen, als ob es eine allgemeine Klasse wäre.

Über das Dialogfeld **Neue Klasse** legen Sie Ihre neue Klasse an. Geben Sie der Klasse einen Namen, der ihre Funktionalität widerspiegelt, beispielsweise `CModArt`, und legen Sie fest, daß sie von der Klasse `CObjekt` als `public` abgeleitet wird. Sie erhalten nun die gleiche Warnung, daß der neue Klassen-Assistent die Header-Dateien der Basisklasse nicht finden konnte. Weil Sie aber die MFC-Unterstützung eingebunden haben, können Sie diese Meldung ignorieren.

Nachdem Sie Ihre Klasse angelegt haben, müssen Sie noch ein paar Variablen in die Klasse aufnehmen. Zuerst ist dafür Sorge zu tragen, daß alle Linien, aus denen die Zeichnung besteht, gespeichert werden können. Zu diesem Zweck legen Sie ein Objektarray an. Zweitens müssen Sie die Abmessungen des Zeichenbereichs kennen. Die entsprechenden Angaben speichern Sie in einer `CRect`-Struktur. Fügen Sie beide Variablen in Ihre neue Klasse ein, und verwenden Sie dabei die Einstellungen gemäß Tabelle 16.1.

Tabelle 16.1: Die Variablen von *CModArt*

Typ	Name	Zugriff
static const COLORREF	m_crColors[8]	Public
CRect	m_rDrawArea	Privat
CObArray	m_oaLines	Privat

Den Zeichenbereich einrichten

Bevor man überhaupt etwas zeichnen kann, muß man den verfügbaren Zeichenbereich kennen. Dazu fügen Sie in Ihre Klasse eine öffentliche Funktion ein, die die übergebene `CRect`-Struktur in die Elementvariable `CRect` kopiert. Für diese Funktion legen Sie den Typ als `void`, die Deklaration mit `SetRect(CRect rDrawArea)` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code gemäß Listing 16.1.

Listing 16.1: Die Funktion `SetRect` der Klasse *CModArt*

```

1: void CModArt::SetRect(CRect rDrawArea)
2: {
3:     // Rechteck für Zeichenbereich festlegen
4:     m_rDrawArea = rDrawArea;
5: }
```

Eine neue Zeichnung erstellen

Eines der Schlüsselmerkmale dieses Moduls ist die Fähigkeit, zufällige Squiggles zu generieren, die im Zeichenbereich erscheinen. Durch das Generieren einer ganzen Folge dieser Squiggles kann das Modul eine vollständige Zeichnung erzeugen. Beginnend mit einem einzelnen Squiggle können Sie eine Funktion entwerfen, die ein Squiggle generiert, und dann diese Funktion mehrmals aufrufen, um die gesamte Zeichnung zu erstellen.

Die erste Funktion, der Squiggle-Generator, muß ermitteln, wie viele Linien zu einem Squiggle gehören. Außerdem sind die Farbe und Breite des Zeichenstifts für das Squiggle zu bestimmen. Weiterhin braucht die Funktion den Anfangspunkt des Squiggles. Von diesem Punkt aus kann die Funktion in einer Schleife die entsprechende Anzahl von Linien durchlaufen und jeweils einen neuen Zielpunkt generieren, um das Squiggle vom vorherigen Zielpunkt fortzusetzen.

Um diese Funktionalität in Ihrem Projekt zu realisieren, fügen Sie eine neue Member- Funktion in die Zeichenklasse ein. Legen Sie den Funktionstyp als `void` und die Definition mit `NewLine` fest. Als Zugriffsstatus wählen Sie die Option *Privat*, da die Funktion nur durch die Hauptschleife aufgerufen wird, die bestimmt, wie viele Squiggles in der endgültigen Zeichnung enthalten sein sollen. In die neue Funktion übernehmen Sie den Code aus Listing 16.2.

Listing 16.2: Die Funktion `NewLine` der Klasse `CModArt`

```

1: void CModArt::NewLine()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:     int nCurColor;
6:     UINT nCurWidth;
7:     CPoint pTo;
8:     CPoint pFrom;
9:
10:    // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
11:    m_rDrawArea.NormalizeRect();
12:    // Breite und Höhe des Zeichenbereichs ermitteln
13:    int lWidth = m_rDrawArea.Width();
14:    int lHeight = m_rDrawArea.Height();
15:
16:    // Anzahl der Teile dieses Squiggles bestimmen
17:    lNumLines = rand() % 100;
18:    // Umfaßt das Squiggle mindestens ein Teil?
19:    if (lNumLines > 0)
20:    {
21:        // Farbe bestimmen
22:        nCurColor = rand() % 8;
23:        // Stiftbreite bestimmen
24:        nCurWidth = (rand() % 8) + 1;
25:        // Anfangspunkt für Squiggle bestimmen
26:        pFrom.x = (rand() % lWidth) + m_rDrawArea.left;
27:        pFrom.y = (rand() % lHeight) + m_rDrawArea.top;
28:        // Schleife durch Anzahl der Segmente
29:        for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
30:        {
31:            // Endpunkt des Segments bestimmen
32:            pTo.x = ((rand() % 20) - 10) + pFrom.x;

```

```

33: pTo.y = ((rand() % 20) - 10) + pFrom.y;
34: // Neues CLine-Objekt erzeugen
35: CLine *pLine = new CLine(pFrom, pTo, m_crColors[nCurColor],
    ÂnCurWidth);
36: try
37: {
38: // Neue Linie in das Objektarray hinzufügen
39: m_oaLines.Add(pLine);
40: }
41: // Speicherausnahme?
42: catch (CMemoryException* perr)
43: {
44: // Meldung an Benutzer mit schlechten
45: // Neuigkeiten
46: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
47: // Wurde ein Linienobjekt erzeugt?
48: if (pLine)
49: {
50: // Löschen
51: delete pLine;
52: pLine = NULL;
53: }
54: // Ausnahmeobjekt löschen
55: perr->Delete();
56: }
57: // Anfangspunkt auf Endpunkt setzen
58: pFrom = pTo;
59: }
60: }
61: }

```

Die Funktion ermittelt zuerst mit den folgenden drei Zeilen den verfügbaren Zeichenbereich:

```

m_rDrawArea.NormalizeRect();
int lWidth = m_rDrawArea.Width();
int lHeight = m_rDrawArea.Height();

```

Die erste Zeile normalisiert das Rechteck, damit garantiert ist, daß die in den nächsten beiden Zeilen zurückgegebenen Werte für Breite und Höhe positiv sind. Aufgrund des in Windows verwendeten Koordinatensystems kann das Ermitteln der Breite durch Subtraktion der linksseitigen Position von der rechtsseitigen Position zu negativen Zahlen führen. Das gleiche trifft auf die Höhe zu. Durch die Normalisierung des Rechtecks ist sichergestellt, daß man für die beiden Werte positive Ergebnisse erhält.

Nachdem der Zeichenbereich ermittelt ist, bestimmt die Funktion die Anzahl der Linienabschnitte, die im Squiggle zu verwenden sind:

```

lNumLines = rand() % 100;

```

Die Funktion `rand` kann Zahlen in einem weiten Bereich zurückgeben. Durch die Modulo-Division mit 100 stellt man sicher, daß die resultierende Zahl zwischen 0 und 100 liegt. Dieses Verfahren wendet man im allgemeinen an, um Zufallszahlen für einen bestimmten Bereich zu erzeugen, wobei man in der Modulo-Funktion die Obergrenze des Wertebereichs angibt. (Ist die Untergrenze ungleich 0, gibt man in

der Modulo-Funktion das Ergebnis aus Untergrenze minus der Obergrenze an und addiert den Wert der Untergrenze auf das Ergebnis der Modulo-Rechnung.) Nach dem gleichen Verfahren bestimmen Sie die Farbe, Breite und Startposition für das Squiggle:

```
nCurColor = rand() % 8;
nCurWidth = (rand() % 8) + 1;
pFrom.x = (rand() % lWidth) + m_rDrawArea.left;
pFrom.y = (rand() % lHeight) + m_rDrawArea.top;
```

Beim Berechnen der Startposition wird die linke und obere Position des Zeichenbereichs zur generierten Position addiert. Damit ist sichergestellt, daß der Startpunkt innerhalb des Zeichenbereichs liegt. Sobald die Funktion in die Schleife eingetreten ist und alle Linienabschnitte im Squiggle generiert, wird der verfügbare Bereich für den nächsten Zielpunkt auf 10 von der aktuellen Position aus begrenzt:

```
pTo.x = ((rand() % 20) - 10) + pFrom.x;
pTo.y = ((rand() % 20) - 10) + pFrom.y;
CLine *pLine = new CLine(pFrom, pTo, m_crColors[nCurColor],
nCurWidth);
m_oaLines.Add(pLine);
```

Diesen Abstand können Sie in einfacher Weise erhöhen, um die Zeichnung weiträumiger zu gestalten. Nachdem die Funktion das nächste Liniensegment generiert hat, erzeugt sie das Linienobjekt und fügt es in das Objektarray ein. Schließlich setzt die Funktion den Anfangspunkt auf den Endpunkt des eben generierten Liniensegments:

```
pFrom = pTo;
```

Nun kann die Funktion die Schleife erneut durchlaufen und das nächste Liniensegment generieren, bis alle Liniensegmente in diesem Squiggle generiert sind.

Nachdem Sie nun ein einzelnes Squiggle erzeugen können, ist der verbleibende Rest einfach zu erledigen. Zuerst ermittelt man, aus wie vielen Squiggles die Zeichnung bestehen soll. Als nächstes durchläuft man eine Schleife für die Anzahl der zu generierenden Squiggles und ruft in jedem Durchlauf die Funktion `NewLine` einmal für jedes Squiggle auf. Um diese Funktionalität im Beispielprojekt zu realisieren, fügen Sie in die Zeichenklasse eine neue Member-Funktion ein. Legen Sie den Typ als `void`, die Deklaration mit `NewDrawing` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code aus Listing 16.3.

Listing 16.3: Die Funktion `NewDrawing` der Klasse `CModArt`

```
1: void CModArt::NewDrawing()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:
6:     // Anzahl der zu erzeugenden Linien bestimmen
7:     lNumLines = rand() % 10;
8:     // Sind Linien zu erzeugen?
9:     if (lNumLines > 0)
10:    {
11:        // Schleife durch Anzahl der Linien
12:        for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
13:        {
14:            // Die neue Linie erzeugen
15:            NewLine();
```

```

16: }
17: }
18: }

```

Die Zeichnung anzeigen

Um die Gruppe der Squiggles im Zeichenbereich darzustellen, können Sie eine Funktion hinzufügen, die das Objektarray in einer Schleife durchläuft und dabei die Funktion `Draw` für jedes Liniensegment im Array aufruft. Diese Funktion muß den Gerätekontext als einziges Argument erhalten und es an die einzelnen Liniensegmente weiterreichen. Fügen Sie dazu der Zeichenklasse eine neue Funktion hinzu. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `Draw(CDC *pDC)` und den Zugriff als *Public* fest. In die Funktion übernehmen Sie den Code aus Listing 16.4.

Listing 16.4: Die Funktion Draw der Klasse CModArt

```

1: void CModArt::Draw(CDC *pDC)
2: {
3: // Anzahl der Linien im Objektarray ermitteln
4: int liCount = m_oaLines.GetSize();
5: int liPos;
6:
7: // Enthält das Array Objekte?
8: if (liCount)
9: {
10: // Schleife durch Array, dabei jedes Objekt zeichnen
11: for (liPos = 0; liPos < liCount; liPos++)
12: ((CLine*)m_oaLines[liPos])->Draw(pDC);
13: }
14: }

```

Die Zeichnung serialisieren

Da Sie auf die Liniensegmentklasse zurückgreifen, die Sie bereits an früherer Stelle erzeugt und serialisierbar gemacht haben, brauchen Sie keine Makros für die Serialisierung in die Zeichenklasse einzubinden. Allerdings müssen Sie eine `Serialize`-Funktion hinzufügen, die das Archivobjekt an das Objektarray weiterreicht und dem Objektarray sowie den Liniensegmentobjekten die Serialisierung überträgt. Zu diesem Zweck fügen Sie eine neue Member-Funktion in die Zeichenklasse ein. Legen Sie den Funktionstyp mit `void`, die Deklaration als `Serialize(CArchive &ar)` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code gemäß Listing 16.5.

Listing 16.5: Die Funktion Serialize der Klasse CModArt

```

1: void CModArt::Serialize(CArchive &ar)
2: {
3: // Archivobjekt an Array übergeben
4: m_oaLines.Serialize(ar);
5: }

```

Die Zeichnung löschen

Um die volle Funktionalität bereitzustellen, müssen Sie in der Lage sein, die Zeichnung aus der Zeichenklasse zu löschen, so daß sich eine neue Zeichnung generieren oder eine vorhandene Zeichnung laden läßt. Dazu durchläuft man einfach das Objektarray, zerstört alle Liniensegmentobjekte und setzt

dann das Objektarray zurück. Nehmen Sie also eine weitere Member-Funktion in das Projekt auf. Spezifizieren Sie den Typ als `void`, die Deklaration als `ClearDrawing` und den Zugriff als *Public*. Übernehmen Sie in diese Funktion den Code aus Listing 16.6.

Listing 16.6: Die Funktion `ClearDrawing` der Klasse `CModArt`

```

1: void CModArt::ClearDrawing()
2: {
3: // Anzahl der Linien im Objektarray bestimmen
4: int liCount = m_oaLines.GetSize();
5: int liPos;
6:
7: // Enthält das Array Objekte?
8: if (liCount)
9: {
10: // Schleife durch das Array, dabei alle Objekte löschen
11: for (liPos = 0; liPos < liCount; liPos++)
12: delete m_oaLines[liPos];
13: // Array zurücksetzen
14: m_oaLines.RemoveAll();
15: }
16: }
```

Die Klasse fertigstellen

Um die Zeichenklasse zu komplettieren, ist noch der Zufallszahlengenerator zu initialisieren. Die als Zufallszahlengenerator arbeitende Funktion `rand` generiert eine statistisch zufällige Zahlenfolge, die auf einer Reihe von mathematischen Berechnungen beruht. Wenn man den Zufallszahlengenerator jedesmal mit derselben Zahl startet, erhält man immer wieder die gleiche Zahlenfolge. Damit der Zufallszahlengenerator bei jedem Start der Anwendung eine anderen Zufallszahlenfolge liefert, muß man ihm einen Anfangswert übergeben, der jedesmal unterschiedlich ist. Normalerweise übergibt man zu diesem Zweck die Systemzeit an die Funktion `srand`, die den Zufallszahlengenerator bei jedem Start der Anwendung mit einer anderen Zeit initiiert. Den Zufallszahlengenerator muß man nur einmal während der Ausführung einer Anwendung initialisieren, so daß man diese Funktionalität im Konstruktor der Zeichenklasse mit dem Code gemäß Listing 16.7 unterbringen kann.

Listing 16.7: Der Konstruktor der Klasse `CModArt`

```

1: CModArt::CModArt()
2: {
3: // Zufallszahlengenerator initialisieren
4: srand((unsigned)time(NULL));
5: }
```

Um die Klasse zu vervollständigen, sind noch die erforderlichen Header-Dateien für die Funktionalität, die Sie in diese Klasse aufgenommen haben, einzubinden. Der Zufallszahlengenerator braucht die Header-Dateien `stdlib.h` und `time.h`, das Objektarray benötigt die Header-Datei für die Klasse `CLine`. Weiterhin müssen Sie die Farbtabelle füllen, die zur Generierung der Squiggles erforderlich ist. Diese letzten Feinheiten fügen Sie hinzu, indem Sie an den Beginn der Quellcodedatei für die Zeichenklasse gehen und die Zeilen 5, 6, 9 und 12 bis 21 aus Listing 16.8 einfügen.

Listing 16.8: Die Include-Anweisungen und die Farbtabelle der Klasse `CModArt`

```

1: // ModArt.cpp: Implementierung der Klasse CModArt.
```

```

2: //
3:
////////////////////////////////////
4:
5: #include <stdlib.h>
6: #include <time.h>
7:
8: #include "stdafx.h"
9: #include "Line.h"
10: #include "ModArt.h"
11:
12: const COLORREF CModArt::m_crColors[8] = {
13: RGB( 0, 0, 0), // Schwarz
14: RGB( 0, 0, 255), // Blau
15: RGB( 0, 255, 0), // Grün
16: RGB( 0, 255, 255), // Cyan
17: RGB( 255, 0, 0), // Rot
18: RGB( 255, 0, 255), // Magenta
19: RGB( 255, 255, 0), // Gelb
20: RGB( 255, 255, 255) // Weiß
21: };

```

Das Bibliotheksmodul ist damit fertiggestellt. Bevor Sie weitergehen können, müssen Sie Ihr Projekt kompilieren. Nach diesem Schritt können Sie noch nichts starten, da Sie erst eine Anwendung erstellen müssen, die Ihr Bibliotheksmodul verwendet. Nur so läßt sich der Code ausführen und testen. Um diese Testanwendung zu erstellen, schließen Sie zunächst den gesamten Arbeitsbereich, damit Sie einen leeren Arbeitsbereich für die Testanwendung vorfinden.

Eine Testanwendung erstellen

Damit Sie Ihr Modul testen können, brauchen Sie eine Anwendung, die das Modul verwendet. Diese einfache Anwendung muß nur soviel Funktionalität bieten, daß sich das Modul ausreichend testen läßt. Wir wollen lediglich die gesamte Funktionalität des Moduls ausprobieren und können auf eine ausgewachsene Anwendung verzichten.

Wenn Sie die Testanwendung erstellen, müssen Sie die Header-Datei für die Zeichenklasse in die relevanten Klassen in Ihrer Anwendung einbinden. Bei einer typischen SDI- oder MDI-Anwendung bedeutet das, daß man die Header-Datei zumindest in die Quelldateien der Dokumentklasse und gegebenenfalls noch in die der Ansichtsklasse und der Anwendungsklasse einbindet. Weiterhin ist die Bibliotheksdatei, die Ihr Modul erzeugt hat, in das Anwendungsprojekt einzubinden, damit sie zur Anwendung gelinkt wird.

Das Gerüst der Testanwendung

Der Rahmen für einen Test läßt sich auf einfache Weise mit einem normalen SDI- oder MDI-Anwendungsgerüst realisieren. Um die Beispielanwendung so einfach wie möglich zu halten, empfiehlt es sich, eine SDI-Anwendung zu wählen. Wenn Sie allerdings bestimmte Funktionen in Ihrem Modul vorgesehen haben, die sich speziell auf eine MDI-Anwendung beziehen, dann liegt es auf der Hand, daß Sie mit dem Gerüst einer MDI-Anwendung arbeiten.

Erstellen Sie als Testanwendung für das Beispielm modul das Gerüst einer normalen SDI-Anwendung mit

dem Klassen-Assistenten. Benennen Sie das Projekt etwa mit **TestApp** oder einem ähnlichen Namen. Klicken Sie im vierten Schritt des Assistenten auf die Schaltfläche **Weitere Optionen**, und legen Sie eine Dateierweiterung fest. Für die übrigen Einstellungen können Sie die Vorgaben übernehmen.

Nachdem Sie das Anwendungsgerüst erstellt haben, müssen Sie das Bibliotheksmodul in das Projekt einbinden. Wählen Sie dazu **Projekt / Dem Projekt hinzufügen / Dateien**. Im Dialogfeld **Dateien in Projekt einfügen** legen Sie als Dateityp *Bibliothekdateien* fest, wie es Abbildung 16.3 zeigt. Gehen Sie in das Debug-Verzeichnis des Modulprojekts, um das Bibliotheksmodul zu lokalisieren, das Sie im vorherigen Projekt erstellt haben. Dazu müssen Sie normalerweise eine Verzeichnisebene höher gehen, das Projektverzeichnis für das Modul aufsuchen und in diesem Verzeichnis nach unten zum Debug-Verzeichnis wechseln. (Wenn Sie die Release-Version des Moduls und der Anwendung erstellen, müssen Sie sinngemäß in die entsprechenden Release-Verzeichnisse wechseln.) Die Bibliotheksdatei für das von Ihnen erstellte Modul sollten Sie entsprechend Abbildung 16.4 lokalisieren können. Markieren Sie das Modul, und klicken Sie auf **OK**, um es in das Projekt einzufügen.

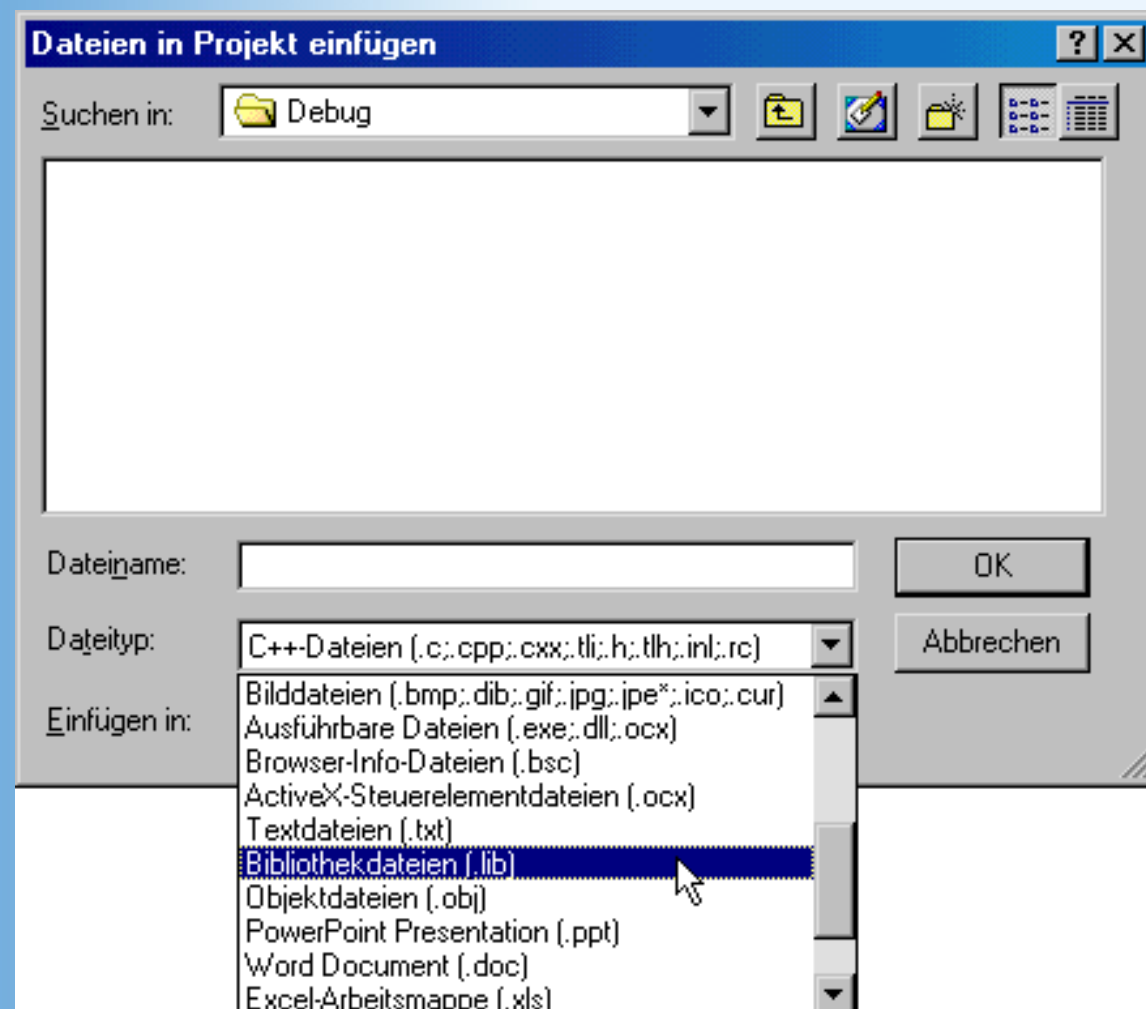


Abbildung 16.3:
Bibliothekdateien spezifizieren

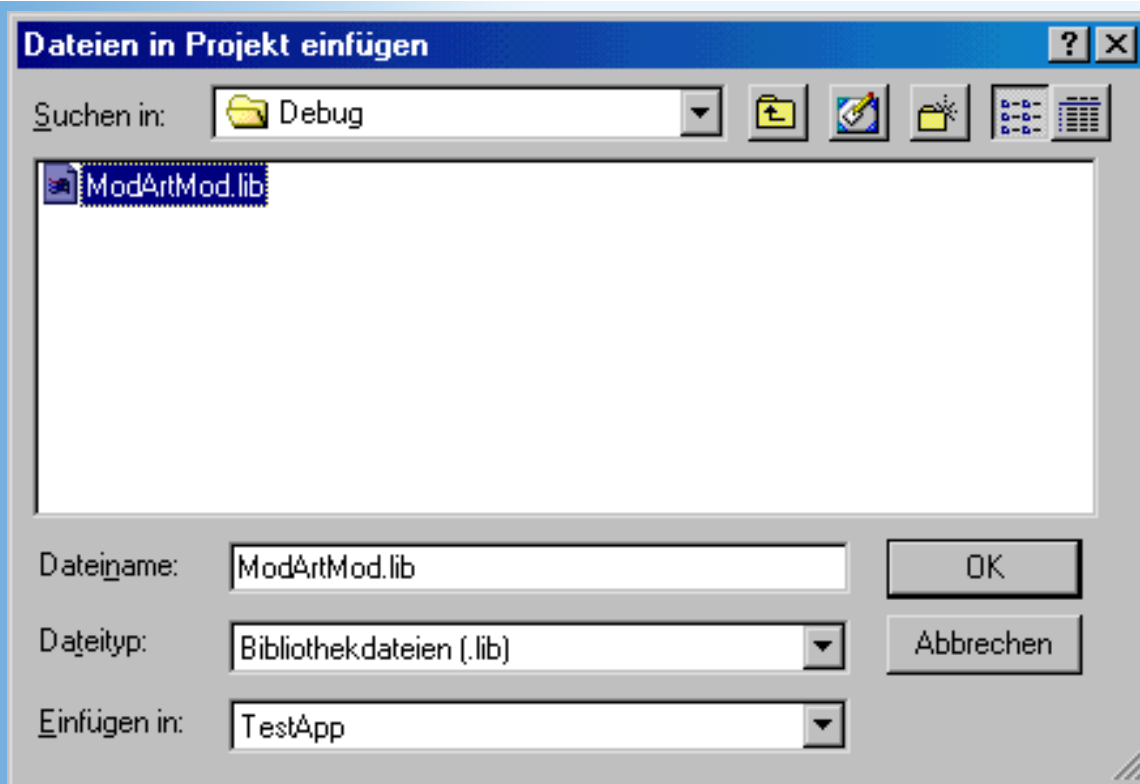


Abbildung 16.4:
Eine Bibliotheksdatei in das Projekt aufnehmen

Nachdem Sie eine Bibliotheksdatei in das Projekt aufgenommen haben, müssen Sie noch die Header-Dateien für alle Klassen im Modul, die in den entsprechenden Quellcodedateien der Anwendung zum Einsatz kommen, einbinden. Für die zu erstellende Testanwendung ist dazu Zeile 7 von Listing 16.9 hinzuzufügen. Die gleiche Zeile nehmen Sie auch in die Include-Abschnitte der Quellcodedateien für die Ansichts- und Anwendungsklasse auf.

Listing 16.9: Die Include-Anweisungen der Klasse CTestAppDoc

```

1: // TestAppDoc.cpp : Implementierung der Klasse CTestAppDoc
2: //
3:
4: #include "stdafx.h"
5: #include "TestApp.h"
6:
7: #include "..\ModArtMod\ModArt.h"
8: #include "TestAppDoc.h"

```

Um die Vorbereitungen am Anwendungsgerüst abzuschließen, nehmen Sie als letztes noch eine Variable für alle Klassen aus dem Bibliotheksmodul auf, die in irgendwelche Klassen der Anwendung eingebunden werden. Für die Testanwendung ist das eine Variable in der Dokumentklasse der Zeichenklasse, die Sie im Projekt des Bibliotheksmoduls erzeugt haben. Um diese Variable in Ihre Anwendung einzufügen, nehmen Sie eine neue Member-Variable in die Dokumentklasse auf. Legen Sie den Variablentyp als Zeichenklasse aus dem Bibliotheksmodul (in diesem Fall `CModArt`), den Namen mit `m_maDrawing` und den Zugriff als *Privat* fest.

Eine neue Zeichnung erstellen

Auf die Funktionalität des Moduls greifen Sie zuerst beim Erzeugen eines neuen Dokuments zurück. Das ist der Moment, wo eine neue Zeichnung zu generieren ist. Letztendlich sind zwei Dinge zu

realisieren. Erstens ist der Zeichenbereich der Ansichtsklasse zu ermitteln und an das Zeichenobjekt zu übergeben. Zweitens ist das Zeichenobjekt anzuweisen, eine neue Zeichnung zu generieren. Das ist alles ziemlich unkompliziert. Um diese Funktionalität in Ihre Anwendung aufzunehmen, bearbeiten Sie die Funktion `OnNewDocument` in der Dokumentklasse und fügen die Zeilen 9 bis 23 aus Listing 16.10 hinzu.

Listing 16.10: Die Funktion `OnNewDocument` der Klasse `CTestAppDoc`

```

1: BOOL CTestAppDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: // Position der Ansicht ermitteln
10: POSITION pos = GetFirstViewPosition();
11: // Ist Position gültig?
12: if (pos != NULL)
13: {
14: // Zeiger auf die Ansicht holen
15: CView* pView = GetNextView(pos);
16: RECT lWndRect;
17: // Rechteck des Anzeigebereichs holen
18: pView->GetClientRect(&lWndRect);
19: // Zeichenbereich festlegen
20: m_maDrawing.SetRect(lWndRect);
21: // Neue Zeichnung erzeugen
22: m_maDrawing.NewDrawing();
23: }
24:
25: return TRUE;
26: }

```

Eine Zeichnung speichern und löschen

In die Dokumentklasse sind noch die Funktionen einzubauen, um die Zeichnung zu speichern und wiederherzustellen sowie zu löschen. Diese Aufgaben sind die letzten der dokumentbezogenen Funktionalität Ihres Bibliotheksmoduls.

Um die Funktionalität zum Speichern und Wiederherstellen in der Anwendung zu realisieren, bearbeiten Sie die Funktion `Serialize` in der Dokumentklasse. Löschen Sie den gesamten Inhalt der Funktion, und ersetzen Sie ihn durch einen Aufruf der `Serialize`-Funktion des Zeichenobjekts, wie es Listing 16.11 zeigt.

Listing 16.11: Die Funktion `Serialize` der Klasse `CTestAppDoc`

```

1: void CTestAppDoc::Serialize(CArchive& ar)
2: {
3: // Zeichnung serialisieren
4: m_maDrawing.Serialize(ar);
5: }

```

Zum Löschen der Zeichnung - damit sich eine neue Zeichnung generieren oder eine gespeicherte Zeichnung laden läßt - brauchen Sie eine Behandlungsroutine für die Funktion `DeleteContents` der Dokumentklasse. In dieser Funktion rufen Sie die Funktion `ClearDrawing` des Zeichenobjekts auf. Um diese Funktionalität in die Anwendung aufzunehmen, fügen Sie mit dem Klassen-Assistenten die Behandlungsroutine für das Ereignis `DeleteContents` in die Dokumentklasse ein. Ergänzen Sie die Funktion mit Zeile 5 von Listing 16.12.

Listing 16.12: Die Funktion `DeleteContents` der Klasse `CTestAppDoc`

```
1: void CTestAppDoc::DeleteContents()
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
   aufrufen
4: // Zeichnung löschen
5: m_maDrawing.ClearDrawing();
6:
7: CDocument::DeleteContents();
8: }
```

Eine Zeichnung anzeigen

In die Testanwendung ist noch eine letzte Funktionsgruppe aufzunehmen, bevor Sie Ihr Bibliotheksmodul testen können: die Zeichenfunktionalität für die Anwendung. Diese Funktionalität gehört in die Ansichtsklasse, da ja das Objekt weiß, wann es sich selbst neu zeichnen muß. Bevor Sie die Funktionalität in der Ansichtsklasse realisieren können, müssen Sie eine Möglichkeit schaffen, damit die Ansichtsklasse auf das Zeichenobjekt zugreifen kann. Am einfachsten läßt sich das mit einer weiteren Funktion in der Dokumentklasse erreichen, die als Rückgabewert einen Zeiger auf das Zeichenobjekt liefert. Nachdem die Ansicht über diesen Zeiger verfügt, kann sie die zum Zeichenobjekt gehörende `Draw`-Funktion aufrufen.

Fügen Sie zu diesem Zweck eine neue Member-Funktion in die Dokumentklasse ein. Spezifizieren Sie den Funktionstyp als Zeiger auf das Zeichenobjekt, in diesem Fall `CModArt*`, die Deklaration als `GetDrawing` und den Zugriff als `Public`. In die Funktion übernehmen Sie den Code aus Listing 16.13.

Listing 16.13: Die Funktion `GetDrawing` der Klasse `CTestAppDoc`

```
1: CModArt* CTestAppDoc::GetDrawing()
2: {
3: // Das Zeichenobjekt zurückgeben
4: return &m_maDrawing;
5: }
```

Um die Zeichenfunktionalität der Ansichtsklasse zu realisieren, bearbeiten Sie einfach die Funktion `OnDraw` in der Ansichtsklasse. Die Funktion holt einen Zeiger auf das Zeichenobjekt und ruft dann dessen `Draw`-Funktion auf, wie es Listing 16.14 zeigt.

Listing 16.14: Die Funktion `OnDraw` der Klasse `CTestAppView`

```
1: CModArt* CTestAppDoc::GetDrawing()
2: {
3: // Das Zeichenobjekt zurückgeben
4: return &m_maDrawing;
5: }
```

Nachdem Sie die gesamte Funktionalität hinzugefügt haben, können Sie die Anwendung kompilieren und ausführen, um die Funktionalität des Bibliotheksmoduls zu testen. Immer, wenn Sie **Datei / Neu** aus dem Menü der Anwendung wählen, wird eine neue Zeichnung wie in Abbildung 16.5 erzeugt.

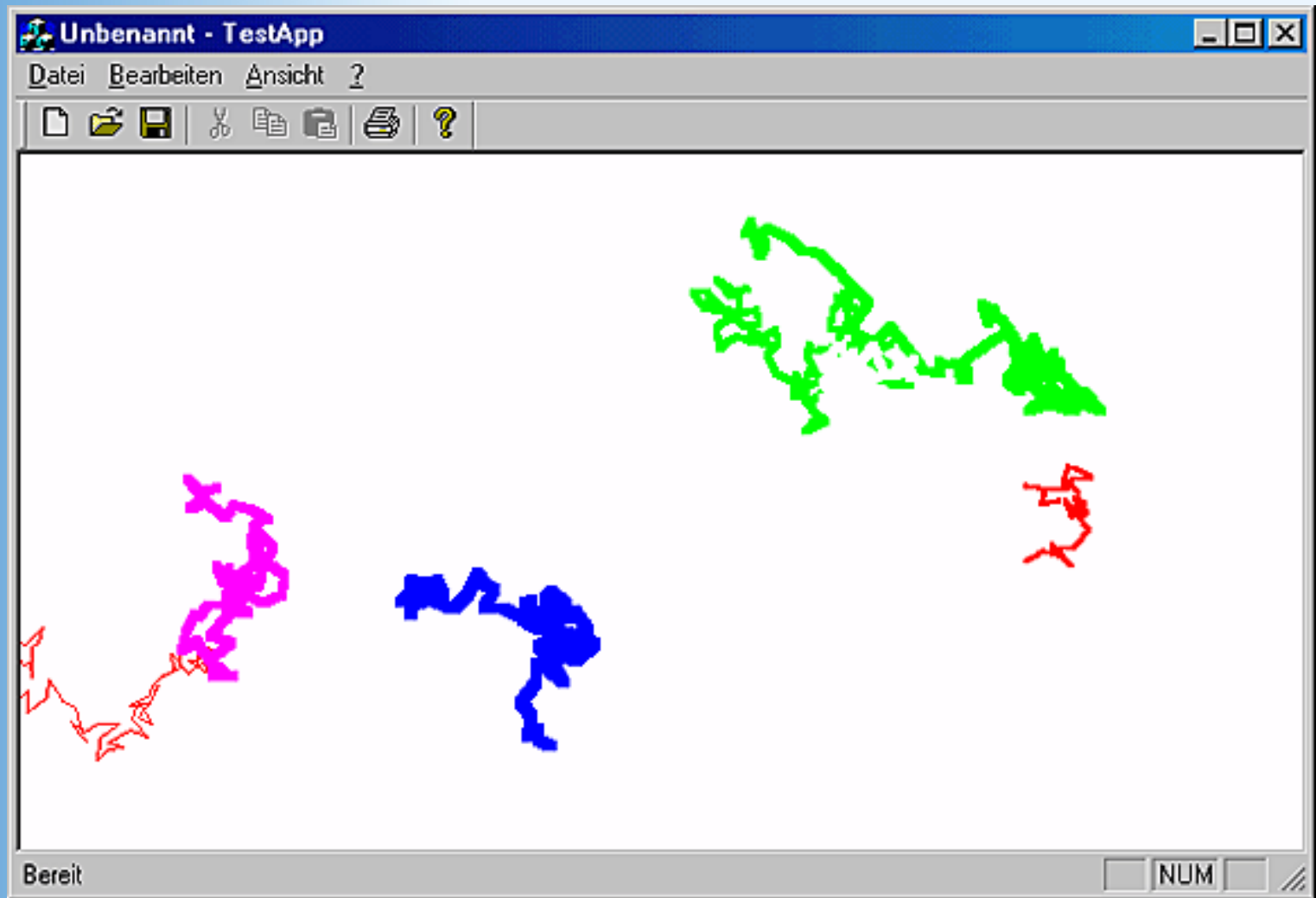


Abbildung 16.5:
Zufällige Schnörkelzeichnungen erzeugen

Das Bibliotheksmodul überarbeiten

Nachdem Sie nun über eine funktionsfähige Anwendung verfügen, gehen wir zurück ins Bibliotheksmodul und nehmen einige Änderungen vor. Wenn sich Änderungen - und seien es nur die geringsten - am Code des Bibliotheksmoduls erforderlich machen, müssen Sie alle Anwendungen neu linkern, die auf dieses Modul zurückgreifen, damit die Überarbeitungen in diesen Anwendungen wirksam werden. Das hängt damit zusammen, daß das Bibliotheksmodul zur ausführbaren Datei der Anwendung gelinkt wird und nicht in einer separaten Datei abgelegt ist.

Um diese Arbeitsweise zu untersuchen, öffnen Sie das Projekt des Bibliotheksmoduls erneut. An diesem Modul nehmen Sie drei Änderungen vor. Erstens erhöhen Sie die Anzahl der Squiggles, die in einer einzelnen Zeichnung enthalten sein können. Zweitens vergrößern Sie die Zahl der Linienabschnitte, aus denen sich ein Squiggle aufbaut. Drittens erzeugen Sie zufällige Farben, und zwar mehr als nur die acht aus der Farbtabelle. Nach diesen Änderungen kompilieren Sie das Bibliotheksmodul neu und linkern es zur Anwendung hinzu, damit Sie die Änderungen in die Anwendung einbinden können.

Für die erste Änderung des Moduls (Erhöhen der Anzahl Squiggles in einer Zeichnung) bearbeiten Sie die Funktion `NewDrawing` in der Zeichenklasse. Hier vergrößern Sie den Modulo-Wert in Zeile 7 der Funktion, wie es aus Listing 16.15 hervorgeht. Damit erhöht sich die mögliche Anzahl der Squiggles in

einer Zeichnung von einem Maximalwert von 10 auf das neue Maximum von 50. Gelegentlich können noch Zeichnungen entstehen, die überhaupt keine Squiggles aufweisen, aber diesen Fall ignorieren wir vorerst.

Listing 16.15: Die Funktion `NewDrawing` der Klasse `CModArt`

```

1: void CModArt::NewDrawing()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:
6:     // Anzahl der zu erzeugenden Linien bestimmen
7:     lNumLines = rand() % 50;
8:     // Sind Linien zu erzeugen?
9:     if (lNumLines > 0)
10:    {
11:        // Schleife durch Anzahl der Linien
12:        for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
13:        {
14:            // Die neue Linie erzeugen
15:            NewLine();
16:        }
17:    }
18: }
```

Nach dieser Änderung vergrößern Sie noch die Anzahl der Liniensegmente, aus denen sich ein Squiggle zusammensetzt. Dazu bearbeiten Sie die Funktion `NewLine` und erhöhen den Modulo-Wert von 100 auf 200, wie es Zeile 20 von Listing 16.16 zeigt. In dieser Funktion können Sie gleich noch die Anzahl der Farben vergrößern, die für jede Zeichnung generiert werden. Deklarieren Sie als erstes drei Integer-Variablen für die drei Grundfarben Rot, Grün und Blau (siehe die Zeilen 9 bis 11 in Listing 16.16). Als nächstes erzeugen Sie zufällige Werte zwischen 0 und 255 für diese Integer-Variablen (in den Zeilen 26 bis 28). Schließlich übergeben Sie diese Werte beim Erzeugen des `CLine`-Objekts über die Funktion `RGB`, um die eigentliche Farbe für die Zeichnung zu erzeugen, wie es Zeile 41 von Listing 16.16 zeigt.

Listing 16.16: Die modifizierte Funktion `NewLine` der Klasse `CModArt`

```

1: void CModArt::NewLine()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:     // int nCurColor;
6:     UINT nCurWidth;
7:     CPoint pTo;
8:     CPoint pFrom;
9:     int cRed;
10:    int cBlue;
11:    int cGreen;
12:
13:    // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
14:    m_rDrawArea.NormalizeRect();
15:    // Breite und Höhe des Zeichenbereichs ermitteln
```



```
16: int lWidth = m_rDrawArea.Width();
17: int lHeight = m_rDrawArea.Height();
18:
19: // Anzahl der Teile dieses Squiggles bestimmen
20: lNumLines = rand() % 200;
21: // Umfaßt das Squiggle mindestens ein Teil?
22: if (lNumLines > 0)
23: {
24: // Farbe bestimmen
25: // nCurColor = rand() % 8;
26: cRed = rand() % 256;
27: cBlue = rand() % 256;
28: cGreen = rand() % 256;
29: // Stiftbreite bestimmen
30: nCurWidth = (rand() % 8) + 1;
31: // Anfangspunkt für Squiggle bestimmen
32: pFrom.x = (rand() % lWidth) + m_rDrawArea.left;
33: pFrom.y = (rand() % lHeight) + m_rDrawArea.top;
34: // Schleife durch Anzahl der Segmente
35: for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
36: {
37: // Endpunkt des Segments bestimmen
38: pTo.x = ((rand() % 20) - 10) + pFrom.x;
39: pTo.y = ((rand() % 20) - 10) + pFrom.y;
40: // Neues CLine-Objekt erzeugen
41: CLine *pLine = new CLine(pFrom, pTo, RGB(cRed, cGreen, cBlue),
    nCurWidth);
42: try
43: {
44: // Neue Linie in das Objektarray hinzufügen
45: m_oaLines.Add(pLine);
46: }
47: // Speicherausnahme?
48: catch (CMemoryException* perr)
49: {
50: // Meldung an Benutzer mit schlechten
51: // Neuigkeiten
52: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
53: // Wurde ein Linienobjekt erzeugt?
54: if (pLine)
55: {
56: // Löschen
57: delete pLine;
58: pLine = NULL;
59: }
60: // Ausnahmeobjekt löschen
61: perr->Delete();
62: }
63: // Anfangspunkt auf Endpunkt setzen
64: pFrom = pTo;
```

```
65: }  
66: }  
67: }
```

Nachdem Sie alle erforderlichen Änderungen am Bibliotheksmodul vorgenommen haben, kompilieren Sie das Modul, um es für den Einsatz in der Testanwendung vorzubereiten. Wenn Sie die Testanwendung über den Befehl **Start / Ausführen** der Taskleiste starten (siehe Abbildung 16.6), ist noch kein Unterschied im Verhalten der Anwendung festzustellen, weil Sie Anwendung noch nicht auf den neuesten Stand gebracht haben. Die Anwendung greift immer noch auf die alte Version des Bibliotheksmoduls zurück. Um die neue Version in die Anwendung einzubauen, öffnen Sie das Projekt der Testanwendung in Visual C++. Erstellen Sie das Projekt neu. Dabei passiert nichts weiter, als ein erneutes Linken des Projekts. Führen Sie dann die Anwendung aus. Jetzt ist ein deutlicher Unterschied bei den Zeichnungen zu bemerken, die Ihre Anwendung generiert (siehe Abbildung 16.7).

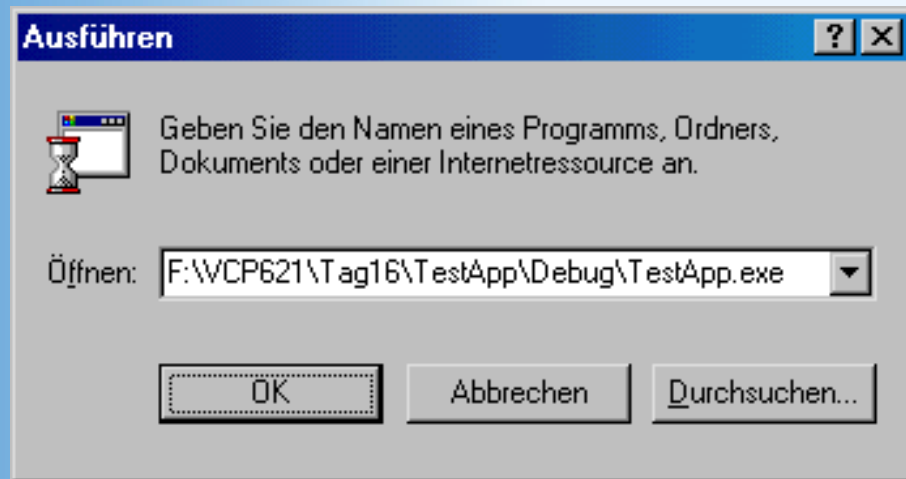


Abbildung 16.6:
Die Testanwendung über das Menü Start der Taskleiste ausführen

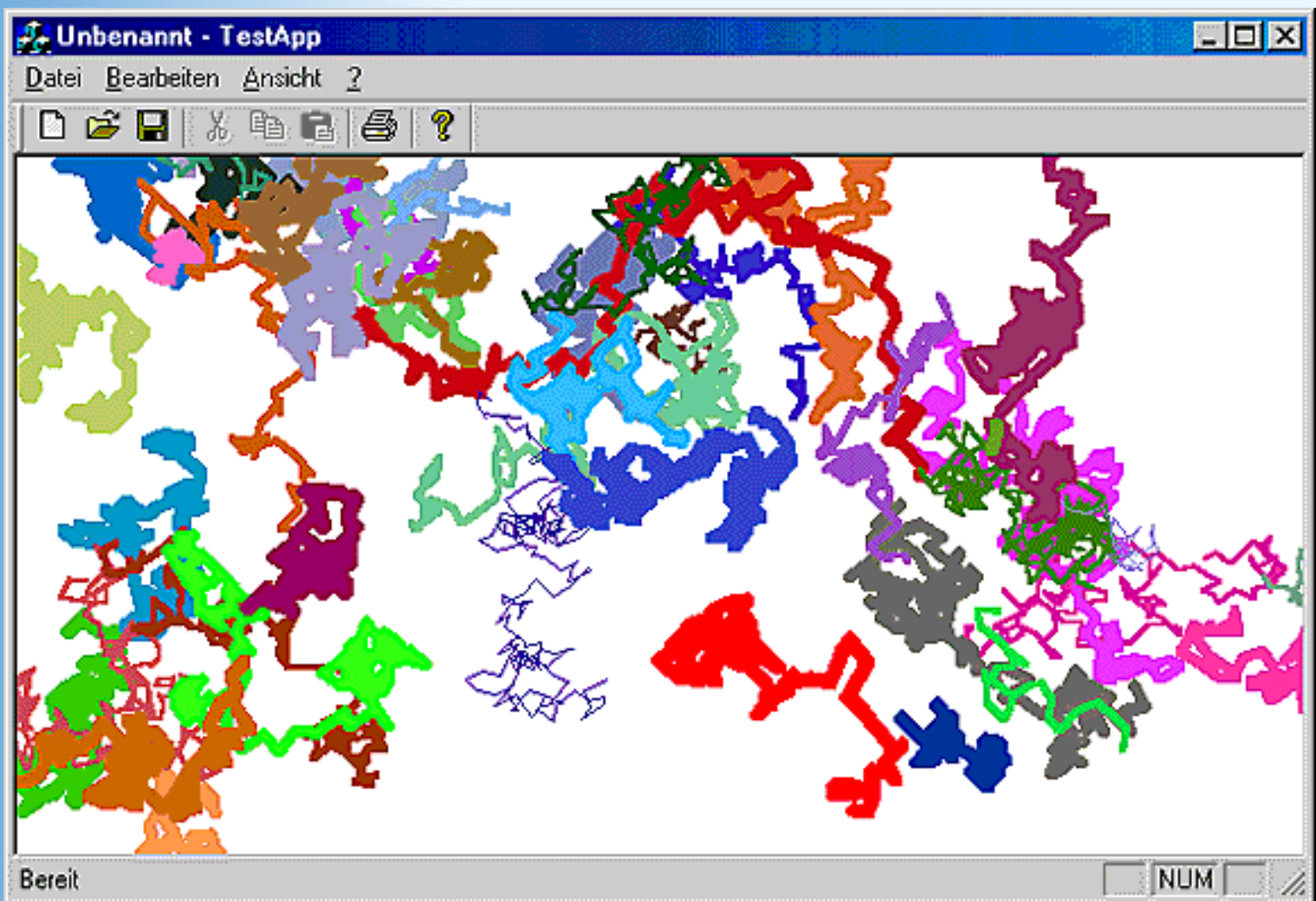


Abbildung 16.7:
Die aktualisierte Testanwendung

Zusammenfassung

Heute haben Sie neue Klassen für eigene Anwendungen entworfen und erstellt. Sie haben den Unterschied zwischen den verschiedenen Klassentypen kennengelernt, die Ihnen der neue Klassen-Assistent in Visual C++ bietet. Weiterhin wurde gezeigt, wie man ein Bibliotheksmodul mit einer Gruppen von Funktionen erzeugt, um es anderen Programmierern für deren Anwendungen bereitzustellen. Sie haben gesehen, wie dieses Modul in die eigentliche Anwendung gelinkt wird, wodurch sich keine getrennte Datei für den Vertrieb der Anwendung erforderlich macht.

Morgen lernen Sie eine andere Lösung kennen, um wiederverwendbare Funktionalität an andere Programmierer weitergeben zu können. Es wird gezeigt, wie man DLLs mit Visual C++ erzeugt, welche Unterschiede zwischen Bibliotheksmodulen und DLLs bestehen und wie man an jede Aufgabe herangeht.

Fragen und Antworten

Frage:

Ist heutzutage nicht der größte Teil der Funktionalität in DLLs verpackt? Warum soll ich überhaupt Bibliotheksmodule statt DLLs erstellen?

Antwort:

In den letzten Jahren zeichnet sich in der Tat der Trend ab, die Funktionalität nicht in Bibliotheksmodulen, sondern in DLLs zu verpacken. Allerdings gibt es noch Situationen, in denen Bibliotheksmodule vorzuziehen sind. Wenn Sie zum Beispiel ein Modul mit proprietären Funktionen erstellen, die Sie Dritten nicht zugänglich machen wollen, trotzdem aber in mehreren Anwendungen einsetzen, an denen Sie oder ein anderer Programmierer Ihrer Firma arbeiten, empfiehlt es sich, die gesamte Funktionalität in einem Bibliotheksmodul zu verpacken, damit sie intern zur Anwendung bleibt. Auf diese Weise erschweren Sie es Ihren Konkurrenten ungemein, die Anwendung zu disassemblieren, um den Code zu untersuchen.

Frage:
Warum muß ich die Header-Datei in die Anwendung, die auf mein Bibliotheksmodul zurückgreift, einbinden?

Antwort:
Die Anwendung muß die Objekte kennen, die in der Bibliotheksdatei enthalten sind. In der Beispielanwendung war es nicht erforderlich, die Header-Datei für die Klasse CLine einzubinden, weil die Anwendung nicht direkt auf die Klasse CLine zugreift oder diese referenziert. Allerdings verwendet die Anwendung das Zeichenobjekt aus dem Bibliotheksmodul, so daß sie dieses Objekt kennen muß, wie es definiert ist und welche Funktionen dafür verfügbar sind. Wenn Sie anderen Programmierern die interne Struktur Ihrer Klassen nicht offenlegen wollen, erzeugen Sie eine weitere Header-Datei, die Sie zusammen mit Ihrem Bibliotheksmodul vertreiben. Dieser Header enthält alle Definitionen der Klassen im Bibliotheksmodul, legt aber nur die öffentlichen Funktionen und Variablen frei, auf die andere Programmierer tatsächlich zugreifen können.

Workshop

Kontrollfragen

1. Wann bietet es sich an, eine neue MFC-Klasse zu erstellen?
2. Wenn man Änderungen an einer Bibliotheksdatei vornimmt, was ist dann mit der Anwendung zu tun, die auf die Bibliotheksdatei zurückgreift?
3. Welche verschiedenen Klassentypen können Sie erzeugen?
4. Wenn Sie bestimmte Funktionen in einer Bibliotheksdatei verpacken, was müssen Sie dann anderen Programmierern übergeben, die Ihr Bibliotheksmodul einsetzen möchten?
5. Nennen Sie zwei der grundlegenden Prinzipien des objektorientierten Softwareentwurfs.

Übung

Spalten Sie die Klasse CLine in ein separates Bibliotheksmodul von der Zeichenklasse ab, so daß Sie dann zwei Bibliotheksmodule und nicht nur ein Modul haben. Linken Sie die Module zur Testanwendung.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 3**Tag 17****DLLs**

Lektion 16 brachte Ihnen bei, wie man eine Funktionsgruppe erstellen kann, die sich in mehreren Anwendungen einsetzen läßt, und wie man sie in einer Bibliotheksdatei verpackt, die man mit diesen Anwendungen linkt. Heute wiederholen Sie das Ganze, allerdings mit einer weitaus flexibleren Lösung.

Oftmals weist eine Anwendungsfamilie gleiche Funktionen auf. Wenn Sie diese gemeinsam genutzte Funktionalität in DLLs statt in Bibliotheksmodulen unterbringen, können alle Anwendungen auf die gleiche Funktionalität zugreifen, wozu nur ein Exemplar des Funktionspakets in Form von DLLs zu vertreiben ist, statt die gleiche Funktionalität mehrfach für die einzelnen Anwendungen bereitzustellen. Dieses Verfahren spart Platz auf allen Systemen, wo die Anwendungen installiert sind.

Die heutige Lektion zeigt, ...

- welche Typen von DLLs Sie mit Visual C++ erstellen können und wie Sie den am besten geeigneten Typ auswählen,
- wie man zwei dieser DLL-Typen erstellt und wie sich die Lösungen bei diesen Typen unterscheiden,
- wie man die Funktionalität für beide DLL-Typen in einer Visual C++-Anwendung einsetzt,
- wie man bestimmt, wann eine Anwendung bei Änderungen einer DLL, auf die eine Anwendung zurückgreift, neu zu linkt ist.

Warum DLLs erstellen?

Microsoft hat bereits in den frühen Tagen von Windows sogenannte dynamische Linkbibliotheken (DLLs - Dynamic Link Libraries) eingeführt. DLLs stimmen mit Bibliotheksmodulen darin überein, daß sie beide eine Funktionsgruppe verpacken, auf die Anwendungen zurückgreifen können. In der Bindung der Bibliothek zur Anwendung unterscheiden sich beide Typen. Wenn Sie eine Anwendung mit einem Bibliotheksmodul (LIB) erstellen, wird die Funktionalität der Bibliothek zur Anwendung gelinkt. Die in der Bibliotheksdatei enthaltene Funktionalität wird Teil der ausführbaren Anwendungsdatei. Bei einer DLL dagegen bindet die Anwendung die in der Bibliotheksdatei enthaltene Funktionalität zur Laufzeit der Anwendung. Die Bibliotheksdatei bleibt eine separate Datei, die von der Anwendung referenziert und aufgerufen wird.

Es gibt mehrere Gründe, warum man DLLs anstelle von Bibliotheksmodulen erstellen sollte. Erstens

läßt sich die Größe der ausführbaren Anwendungsdateien verringern, indem man die von mehreren Anwendungen gemeinsam genutzte Funktionalität in DLLs unterbringt. Die Funktionalität in den DLLs kann man aktualisieren und modifizieren, ohne die ausführbare Anwendung auf den neuesten Stand bringen zu müssen (vorausgesetzt, daß die von der DLL exportierte Schnittstelle unverändert bleibt). Schließlich kann man DLLs mit nahezu jeder anderen Programmiersprache unter Windows verwenden, wodurch die von Ihnen erstellte Funktionalität einer breiteren Masse von Programmierern zur Verfügung steht und nicht nur den Anhängern von Visual C++.

DLLs erstellen und einsetzen

DLLs sind Bibliotheksdateien mit kompiliertem Code, auf den andere Anwendungen zurückgreifen können. Die DLLs legen bestimmte Funktionen und Klassen für diese Anwendungen frei, indem sie die Funktion exportieren. Eine exportierte Funktion wird in eine Tabelle eingetragen, die zur DLL gehört. Diese Tabelle enthält den Standort aller exportierten Funktionen, die in der DLL vorhanden sind. Alle nicht exportierten Funktionen erscheinen auch nicht in der Tabelle, und eine Anwendung außerhalb der DLL kann diese weder sehen noch aufrufen.

Eine Anwendung kann die Funktionen in der DLL nach zwei Verfahren aufrufen. Bei der komplizierteren Methode sucht die Anwendung nach dem Standort der gewünschten Funktion in der DLL und holt einen Zeiger auf diese Funktion. Über den Zeiger findet dann der Aufruf der Funktion statt.

Der andere und wesentlich einfachere Weg (und der einzige Weg, den Sie in allen Beispielen in diesem Buch gehen) ist, die Anwendung mit der LIB-Datei, die mit der DLL erzeugt wird, zu verknüpfen. Diese LIB-Datei behandelt der Linker als normale Bibliotheksdatei - genau wie die gestern erstellte. Allerdings enthält diese LIB-Datei Funktionsrümpfe für alle exportierten Funktionen der DLL. Diese Pseudofunktionen haben den gleichen Namen und die gleiche Argumentliste wie die echten Funktionen, enthalten aber im Inneren nur einen winzigen Codeabschnitt, der die echte Funktion in der DLL aufruft und alle an die Pseudofunktion übergebenen Parameter an die echte Funktion weiterreicht. Damit können Sie die Funktionen in der DLL genauso behandeln, als wären sie Teil des Anwendungscode und nicht einer separaten Datei.



Die LIB-Datei für eine DLL wird automatisch beim Kompilieren der DLL erzeugt. Auf Ihrer Seite sind keine weiteren Schritte dafür erforderlich.



Es ist nicht nur leichter, Anwendungen mit Hilfe von LIB-Dateien für alle verwendeten DLLs zu erstellen, sondern auch sicherer, wenn Sie die Anwendung ausführen. Wenn Sie mit LIB-Dateien arbeiten, werden alle DLLs, auf die Sie in Ihrer Anwendung zurückgreifen, zu dem Zeitpunkt in den Arbeitsspeicher geladen, zu dem die Anwendung startet. Sollte irgendeine DLL fehlen, informiert Windows den Benutzer automatisch über dieses Problem, und Ihre Anwendung startet nicht. Wenn Sie dagegen nicht mit den LIB-Dateien arbeiten, müssen Sie die DLL selbst in den Speicher laden und sind auch dafür verantwortlich, alle Fehler zu behandeln, die bei einer fehlenden DLL auftreten können.

Zwei DLL-Typen lassen sich mit Visual C++ sehr einfach erstellen. Dabei handelt es sich um erweiterte MFC-DLLs und Standard-DLLs.



Mit Visual C++ lassen sich auch andere DLL-Typen erstellen. Diese umfassen aber einen beträchtlichen Teil der ActiveX-Funktionalität, so daß wir im Rahmen dieses Buches nicht darauf eingehen können. Wenn Sie mit Active Technology sogenannte prozeßinterne Server in einer DLL unterbringen möchten oder andere Arten von ActiveX-DLLs erstellen wollen, sollten Sie sich nach einem weiterführenden Buch zu Visual C++ umsehen, das sich speziell diesen Themen widmet.

Erweiterte MFC-DLLs

MFC-DLLs lassen sich am einfachsten kodieren und erstellen, da man sie genauso behandeln kann wie jede andere Sammlung von Klassen. Bei allen Klassen, die Sie aus einer DLL exportieren wollen, brauchen Sie lediglich das Makro `AFX_EXT_CLASS` wie folgt in die Klassendeklaration einzufügen:

```
class AFX_EXT_CLASS CMyClass
{
.
.
.
};
```

Dieses Makro exportiert die Klasse und macht sie damit für Visual-C++-Anwendungen sichtbar. Das Makro ist in die Header-Datei einzubinden, die von den Anwendungen, die auf die DLL zurückgreifen, verwendet wird. Damit importieren Sie die Klasse aus der DLL, so daß sie sich verwenden läßt.

Ein Nachteil bei der Erstellung von erweiterten MFC-DLLs besteht darin, daß sie sich nicht in anderen Programmiersprachen nutzen lassen, sondern nur mit anderen C++-Compilern, solange diese Compiler MFC unterstützen (wie das bei den Compilern von Borland - bzw. Inprise - und Symantec der Fall ist).

Standard-DLLs

Standard-DLLs exportieren Standardfunktionen aus der DLL und keine C++-Klassen. Im Ergebnis kann dieser DLL-Typ etwas mehr Nachdenken und Planung erfordern als eine erweiterte MFC-DLL. Innerhalb der DLL selbst können Sie allerdings alle Klassen wie gewünscht verwenden, für die externen Anwendungen müssen Sie jedoch reine Funktionsaufrufe vorsehen.

Um eine Funktion zu exportieren, ist sie als Exportfunktion zu deklarieren. Dazu versehen Sie den Funktionsnamen mit folgendem Vorsatz:

```
extern "C" <Funktionstyp> PASCAL EXPORT <Funktionsdeklaration>
```

Diese zusätzlichen Angaben sind wohl in der Header-Datei für den Prototyp der Funktion als auch im eigentlichen Quellcode erforderlich. Der Abschnitt `extern "C"` deklariert, daß es sich um einen normalen Funktionsaufruf in C handelt, so daß bei der C++-Namensanalyse nicht der Funktionsname zerrissen wird. `PASCAL` weist den Compiler an, daß alle Funktionsargumente in der `PASCAL`-Reihenfolge zu übergeben sind, bei der die Argumente auf dem Stack in umgekehrter Reihenfolge gegenüber der Schreibweise plazierte werden. Schließlich teilt `EXPORT` dem Compiler mit, daß diese Funktion aus der DLL zu exportieren ist und von außerhalb der DLL aufgerufen werden kann.

Beim Exportieren von Funktionen aus der DLL sind darüber hinaus alle exportierten Funktionsnamen in die DEF-Datei für das DLL-Projekt einzutragen. Diese Datei ist für das Erstellen der LIB-Datei mit den Pseudofunktionen sowie der Exporttabelle in der DLL erforderlich. Sie enthält den Namen der DLL oder Bibliothek, eine kurze Beschreibung der DLL und die Namen aller Funktionen, die exportiert werden. Für die Datei ist ein spezielles Format vorgeschrieben, so daß Sie die vom DLL-Assistenten automatisch erzeugte Standard-DEF-Datei nur insofern verändern sollten, als daß Sie exportierte Funktionsnamen hinzufügen. Eine typische DEF-Datei sieht folgendermaßen aus:

```
LIBRARY "mydll"
DESCRIPTION 'mydll Windows Dynamic Link Library'

EXPORTS
; Explizite Exporte können hier eingefügt werden
MyFunc1
MyFunc2
```

Wenn Sie MFC-Klassen in Ihren normalen DLLs verwenden, müssen Sie das Makro `AFX_MANAGE_STATE` als erste Codezeile in allen exportierten Funktionen aufrufen. Das ist erforderlich, um die exportierten Funktionen thread-sicher zu machen. Damit können Ihre Klassenfunktionen gleichzeitig durch mehrere Programme (oder Threads) aufgerufen werden. Das Makro `AFX_MANAGE_STATE` übernimmt als einziges Argument einen Zeiger auf eine `AFX_MODULE_STATE`-Struktur, die sich durch Aufruf der Funktion `AfxGetStaticModuleState` abrufen läßt. Eine typische exportierte Funktion, die mit MFC arbeitet, sieht wie folgt aus:

```
extern "C" void PASCAL EXPORT MyFunc(...)
{
AFX_MANAGE_STATE(AfxGetStaticModuleState());
// Hier normaler Funktionsrumpf
.
.
.
}
```

DLLs entwerfen

Wenn Sie Ihre DLLs entwerfen, sollten Sie sich dessen bewußt sein, daß alle Funktionen in den DLLs gleichzeitig durch mehrere Anwendungen, die zur gleichen Zeit laufen, aufgerufen werden können. Daraus folgt, daß die gesamte Funktionalität in allen von Ihnen erstellten DLLs thread-sicher sein muß.

Alle Variablen, die Werte über eine separate Funktion hinaus aufnehmen, sind durch die Anwendung und nicht von der DLL zu speichern und zu verwalten. Alle Anwendungsvariablen, die von der DLL manipuliert werden müssen, sind an die DLL als Funktionsargumente zu übergeben. Alle globalen Variablen, die innerhalb der DLL verwaltet werden, können bei laufender Funktion mit Variablen aus anderen Anwendungsprozessen ausgetauscht werden, was zu unvorhersagbaren Ergebnissen führt.

Erweiterte MFC-DLLs erstellen und einsetzen

Damit Sie sich davon überzeugen können, wie einfach es ist, eine erweiterte MFC- DLL zu erstellen und einzusetzen, konvertieren Sie heute das gestern erzeugte Bibliotheksmodul in eine erweiterte MFC-DLL. Nachdem Sie wissen, welche Arten von Änderungen Sie vornehmen müssen, um die DLL zu nutzen, implementieren Sie die gleiche Funktionalität wie bei einer normalen DLL erneut, so daß Sie am Ende die verschiedenen Lösungswege kennen, die bei den beiden DLL-Stilen erforderlich sind.

Die erweiterte MFC-DLL erstellen

Um das gestern erstellte Bibliotheksmodul in eine erweiterte MFC-DLL umzuwandeln, erzeugen Sie ein neues Projekt mit dem *MFC-Anwendungs-Assistenten (dll)* und spezifizieren das Projekt als *Erweiterungs-MFC-DLL*. Kopieren Sie den Quellcode und die Header-Dateien für die Linien- und Zeichenklassen in das Projektverzeichnis. Laden Sie die Dateien für die Linien- und Zeichenklassen in das aktuelle Projekt. Fügen Sie in die Zeichenklasse das Makro `AFX_EXT_CLASS` hinzu. Schließlich verschieben Sie die Farbtabelle von einer globalen statischen Tabelle in eine lokale Variable in die Funktion, die die Squiggles generiert.

Beginnen Sie ein neues Projekt, um die DLL zu erstellen. Das Projekt benennen Sie beispielsweise mit `ModArtDll`. Auf der Registerkarte **Projekte** des Dialogfelds **Neu** wählen Sie den Eintrag *MFC-Anwendungs-Assistent (dll)*, wie es Abbildung 17.1 zeigt. Im ersten Schritt des MFC-Anwendungs-Assistenten wählen Sie die Option *Erweiterungs-MFC-DLL* (siehe Abbildung 17.2).

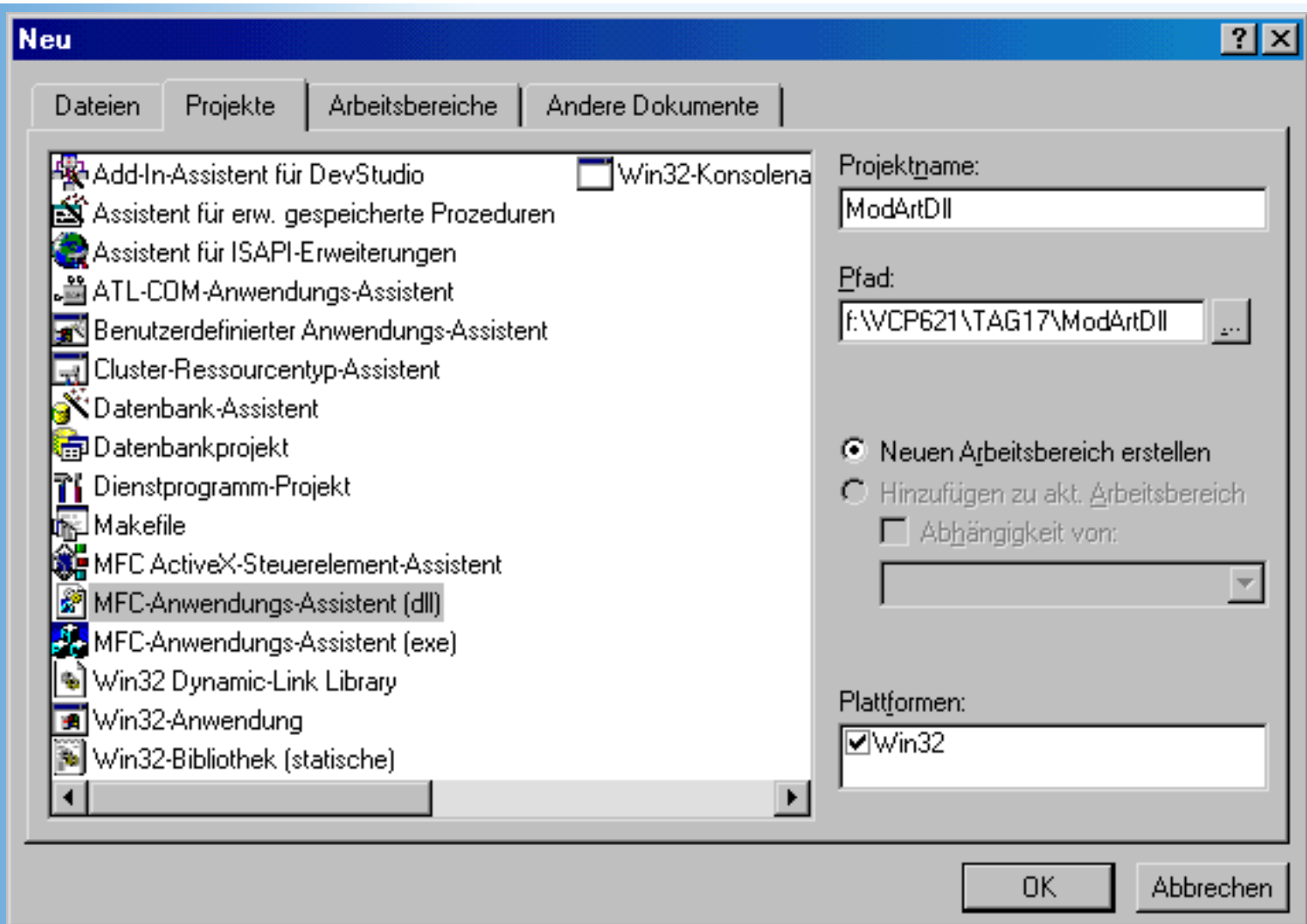


Abbildung 17.1:
Den Typ der DLL festlegen

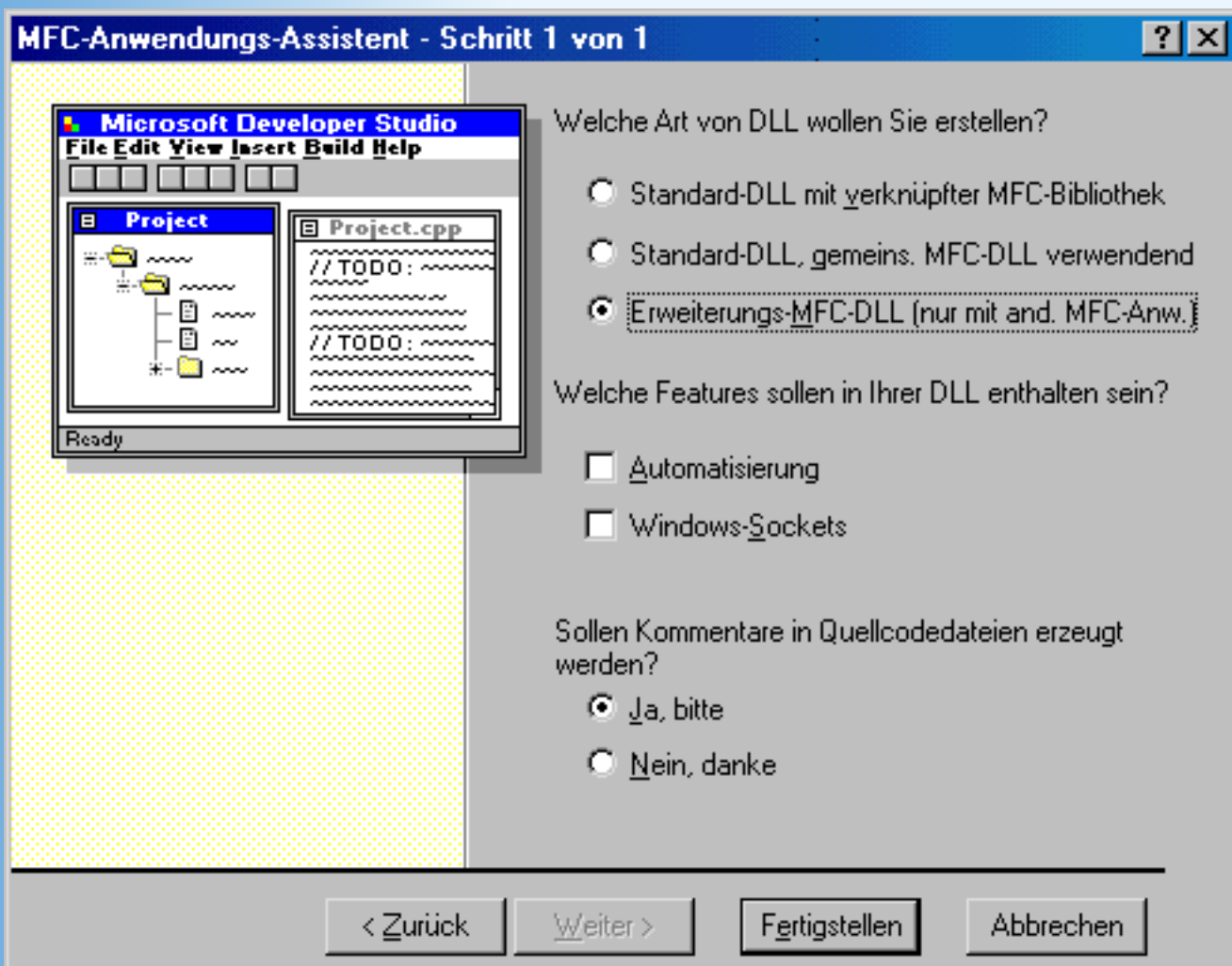


Abbildung 17.2:
Den MFC-Anwendungs-Assistenten (dll) auswählen

Nachdem Sie das Gerüst der DLL erstellt haben, öffnen Sie den Explorer und kopieren die Quellcode- und die Header-Dateien für die Linien- und Zeichenklassen (`line.cpp`, `line.h`, `ModArt.cpp` und `ModArt.h`) aus dem gestern erstellten Projekt des Bibliotheksmoduls in das eben erstellte Projektverzeichnis. Fügen Sie alle vier Dateien in das Projekt ein. Auf der Registerkarte **Klassen** des Arbeitsbereichs sollten nun beide Klassen erscheinen.

Öffnen Sie die Header-Datei, die die Definition der Zeichenklasse enthält. Fügen Sie das Makro `AFX_EXT_CLASS` in die Klassendeklaration gemäß Listing 17.1 ein. Entfernen Sie außerdem die Variable der Farbtabelle aus der Klassendeklaration.

Listing 17.1: Die modifizierte Deklaration der Klasse CModArt

```

1: class AFX_EXT_CLASS CModArt : public CObject
2: {
3: public:
4: void NewDrawing();
5: virtual void Serialize(CArchive &ar);
6: void Draw(CDC *pDC);
7: void ClearDrawing();
8: void SetRect(CRect rDrawArea);
9: CModArt();
10: virtual ~CModArt();
11:

```



```

12: private:
13: void NewLine();
14: CRect m_rDrawArea;
15: CObArray m_oaLines;
16: };

```

In einer DLLs sollten keine öffentlichen, statischen Variablen vorkommen, so daß Sie die Farbtabelle nicht wie im gestrigen Projekt als öffentliches, statisches Element der Zeichenklasse deklarieren können. Aus diesem Grund verschieben Sie die Variable in eine lokale Variable in der Elementfunktion `NewLine`. In die Funktion `NewLine` nehmen Sie die lokale Variable auf und setzen das Verhalten der Funktion auf die ursprüngliche Realisierung zurück, wie es aus Listing 17.2 hervorgeht.

Listing 17.2: Die Funktion `NewLine` der Klasse `CModArt`

```

1: void CModArt::NewLine()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:     int nCurColor;
6:     UINT nCurWidth;
7:     CPoint pTo;
8:     CPoint pFrom;
9:
10:    // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
11:    m_rDrawArea.NormalizeRect();
12:    // Breite und Höhe des Zeichenbereichs ermitteln
13:    int lWidth = m_rDrawArea.Width();
14:    int lHeight = m_rDrawArea.Height();
15:
16:    COLORREF crColors[8] = {
17:        RGB( 0, 0, 0), // Schwarz
18:        RGB( 0, 0, 255), // Blau
19:        RGB( 0, 255, 0), // Grün
20:        RGB( 0, 255, 255), // Cyan
21:        RGB( 255, 0, 0), // Rot
22:        RGB( 255, 0, 255), // Magenta
23:        RGB( 255, 255, 0), // Gelb
24:        RGB( 255, 255, 255) // Weiß
25:    };
26:
27:    // Anzahl der Teile dieses Squiggles bestimmen
28:    lNumLines = rand() % 100;
29:    // Umfaßt das Squiggle mindestens ein Teil?
30:    if (lNumLines > 0)
31:    {
32:        // Farbe bestimmen
33:        nCurColor = rand() % 8;
34:        // Stiftbreite bestimmen
35:        nCurWidth = (rand() % 8) + 1;

```

```

36: // Anfangspunkt für Squiggle bestimmen
37: pFrom.x = (rand() % lWidth) + m_rDrawArea.left;
38: pFrom.y = (rand() % lHeight) + m_rDrawArea.top;
39: // Schleife durch Anzahl der Segmente
40: for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
41: {
42: // Endpunkt des Segments bestimmen
43: pTo.x = ((rand() % 20) - 10) + pFrom.x;
44: pTo.y = ((rand() % 20) - 10) + pFrom.y;
45: // Neues CLine-Objekt erzeugen
46: CLine *pLine = new CLine(pFrom, pTo, crColors[nCurColor],
  ÂnCurWidth);
47: try
48: {
49: // Neue Linie in das Objektarray hinzufügen
50: m_oaLines.Add(pLine);
51: }
52: // Speicherausnahme?
53: catch (CMemoryException* perr)
54: {
55: // Meldung an Benutzer mit schlechten
56: // Neuigkeiten
57: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
58: // Wurde ein Linienobjekt erzeugt?
59: if (pLine)
60: {
61: // Löschen
62: delete pLine;
63: pLine = NULL;
64: }
65: // Ausnahmeobjekt löschen
66: perr->Delete();
67: }
68: // Anfangspunkt auf Endpunkt setzen
69: pFrom = pTo;
70: }
71: }
72: }

```

Nachdem Sie diese Änderungen an der Zeichenklasse vorgenommen haben, können Sie die DLL kompilieren. Wechseln Sie dann in den Explorer, suchen Sie die DLL im Debug-Unterverzeichnis unterhalb des Projektverzeichnisses, und kopieren Sie die DLL in das Debug-Verzeichnis des Projekts für die Testanwendung.

Die Testanwendung anpassen

Um die Testanwendung für die Zusammenarbeit mit der DLL anzupassen, öffnen Sie das gestern erstellte Projekt der Testanwendung. In diesem Projekt löschen Sie das gestern erstellte Bibliotheksmodul und fügen die LIB-Datei, die der Anwendungs-Assistent mit der DLL erzeugt hat,

hinzu. Weiterhin ändern Sie die Header-Datei, die Sie für die Zeichenklasse eingebunden haben. Nach diesen beiden Änderungen ist die Testanwendung für den Einsatz der DLL vorbereitet.

Gehen Sie zunächst auf die Registerkarte **Dateien** im Arbeitsbereich, um das Bibliotheksmodul aus dem Projekt zu löschen. Markieren Sie die LIB-Datei in der Liste der Projektdateien, und drücken Sie die (Entf)-Taste. Anschließend wählen Sie **Projekt / Dem Projekt hinzufügen / Dateien**. Im Dialogfeld **Dateien in Projekt einfügen** wählen Sie als Dateityp *Bibliothekdateien (.lib)*. Gehen Sie dann ins Debug-Verzeichnis des DLL-Projekts. Markieren Sie die LIB-Datei, die mit der DLL erzeugt wurde, im Beispiel `ModArtDll.lib`. Klicken Sie auf **OK**, um die Datei in das Projekt aufzunehmen.

Im nächsten Schritt bearbeiten Sie die Quellcodedateien für die Dokument-, die Ansicht- und die Anwendungsklasse. Dabei passen Sie die Include-Anweisungen für die Zeichenklasse wie in Zeile 7 von Listing 17.3 auf den Pfad für das Projektverzeichnis der DLL an.

Listing 17.3: Die Include-Anweisungen der Klasse CTestAppDoc

```
1: // TestAppDoc.cpp : Implementierung der Klasse CTestAppDoc
2: //
3:
4: #include "stdafx.h"
5: #include "TestApp.h"
6:
7: #include "..\ModArtDll\ModArt.h"
8: #include "TestAppDoc.h"
```

Nachdem Sie diese Änderung in allen drei Quellcodedateien vorgenommen haben, können Sie die Testanwendung kompilieren und ausführen. Die Testanwendung sollte genau wie gestern laufen, nur daß sie kürzere Squiggles erzeugt und lediglich acht Farben in der Farbtabelle verwendet.

Die DLL ändern

Da die Testanwendung nunmehr mit der DLL lauffähig ist, nehmen Sie an der DLL die gleichen Änderungen vor, die Sie gestern im Bibliotheksmodul durchgeführt haben. Erhöhen Sie die Anzahl der Squiggles, die in einer Zeichnung erscheinen können, vergrößern Sie die mögliche Länge jedes Squiggles, und generieren Sie eine beliebige Anzahl von Farben für das Squiggle.

Wechseln Sie zurück in das DLL-Projekt, um diese Änderungen vorzunehmen. In der Elementfunktion `NewDrawing` der Zeichenklasse erhöhen Sie die Anzahl der Linien, die generiert werden können. In der Elementfunktion `NewLine` vergrößern Sie die mögliche Länge der Squiggles und fügen wieder die zufälligen Farben ein (siehe Listing 17.4).

Listing 17.4: Die modifizierte Funktion NewLine der Klasse CModArt

```
1: void CModArt::NewLine()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:     // int nCurColor;
6:     UINT nCurWidth;
7:     CPoint pTo;
8:     CPoint pFrom;
```

```

9: int cRed;
10: int cBlue;
11: int cGreen;
12:
13: // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
14: m_rDrawArea.NormalizeRect();
15: // Breite und Höhe des Zeichenbereichs ermitteln
16: int lWidth = m_rDrawArea.Width();
17: int lHeight = m_rDrawArea.Height();
18:
19: // COLORREF crColors[8] = {
20: // RGB( 0, 0, 0), // Schwarz
21: // RGB( 0, 0, 255), // Blau
22: // RGB( 0, 255, 0), // Grün
23: // RGB( 0, 255, 255), // Cyan
24: // RGB( 255, 0, 0), // Rot
25: // RGB( 255, 0, 255), // Magenta
26: // RGB( 255, 255, 0), // Gelb
27: // RGB( 255, 255, 255) // Weiß
28: // };
29:
30: // Anzahl der Teile dieses Squiggles bestimmen
31: lNumLines = rand() % 200;
32: // Umfaßt das Squiggle mindestens ein Teil?
33: if (lNumLines > 0)
34: {
35: // Farbe bestimmen
36: // nCurColor = rand() % 8;
37: cRed = rand() % 256;
38: cBlue = rand() % 256;
39: cGreen = rand() % 256;
40: // Stiftbreite bestimmen
41: nCurWidth = (rand() % 8) + 1;
42: // Anfangspunkt für Squiggle bestimmen
43: pFrom.x = (rand() % lWidth) + m_rDrawArea.left;
44: pFrom.y = (rand() % lHeight) + m_rDrawArea.top;
45: // Schleife durch Anzahl der Segmente
46: for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
47: {
48: // Endpunkt des Segments bestimmen
49: pTo.x = ((rand() % 20) - 10) + pFrom.x;
50: pTo.y = ((rand() % 20) - 10) + pFrom.y;
51: // Neues CLine-Objekt erzeugen
52: CLine *pLine = new CLine(pFrom, pTo, RGB(cRed, cGreen, cBlue),
53: // nCurWidth);
54: {
55: // Neue Linie in das Objektarray hinzufügen
56: m_oaLines.Add(pLine);

```

```

57: }
58: // Speicherausnahme?
59: catch (CMemoryException* perr)
60: {
61: // Meldung an Benutzer mit schlechten
62: // Neuigkeiten
63: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
64: // Wurde ein Linienobjekt erzeugt?
65: if (pLine)
66: {
67: // Löschen
68: delete pLine;
69: pLine = NULL;
70: }
71: // Ausnahmeobjekt löschen
72: perr->Delete();
73: }
74: // Anfangspunkt auf Endpunkt setzen
75: pFrom = pTo;
76: }
77: }
78: }

```

Wenn Sie die Änderungen ausgeführt haben, kompilieren Sie die DLL erneut. Wechseln Sie anschließend in den Explorer, und kopieren Sie wieder die DLL in das Debug- Verzeichnis der Testanwendung. Dann starten Sie die Testanwendung über **Start / Ausführen**, wie es Abbildung 17.3 zeigt. Die Anwendung spiegelt nun den neuesten Stand wider und sollte jetzt sowohl mehr Squiggles als auch viele unterschiedliche Farben zeigen.

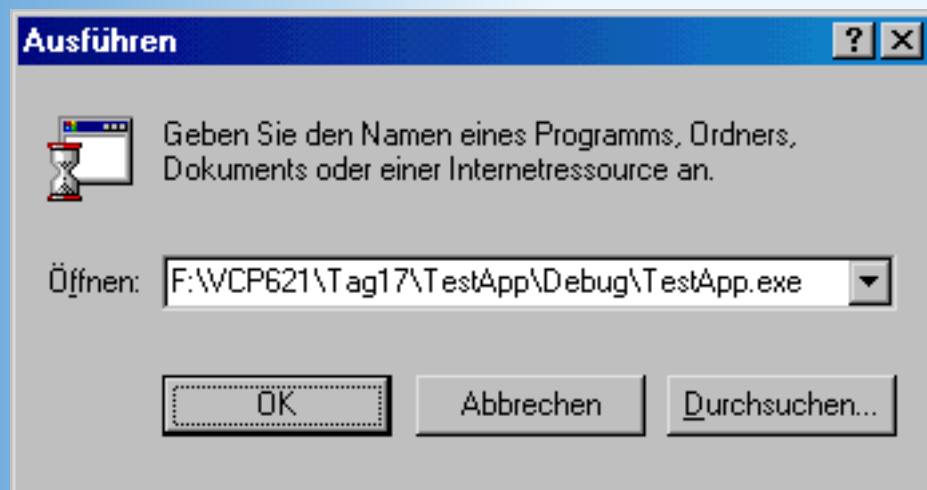


Abbildung 17.3:
Die Beispielanwendung starten

Eine Standard-DLL erstellen und einsetzen

Vielleicht kommt es Ihnen so vor, daß Sie die Verwendungsregeln der nicht von der Anwendung besessenen Variablen in der DLL verletzen, als Sie die erweiterte MFC- DLL erzeugt und erstellt haben. Das ist allerdings nicht der Fall. Die Instanz der Zeichenklasse ist ein Element der

Dokumentklasse in der Testanwendung. Sie wird von der Anwendung und nicht von der DLL erzeugt und verwaltet. Wenn Sie jetzt darangehen, die gleiche Funktionalität in einer normalen DLL zu implementieren, wird das deutlicher.

Um die erweiterte MFC-DLL in eine Standard-DLL umzuwandeln, müssen Sie die Zeichenklasse in eine Folge von normalen Funktionsaufrufen konvertieren. Das Objektarray muß im Zuge dieser Umwandlung zu einer Elementvariablen der Dokumentklasse der Anwendung werden und ist als Argument an jede exportierte Funktion in der DLL zu übergeben.

Die Standard-DLL erstellen

Beginnen Sie mit einem neuen Projekt, um die erweiterte MFC-DLL in eine Standard- DLL zu konvertieren. Visual C++ muß ein Projekt erstellen, das dem Compiler mitteilt, welcher Dateityp erstellt wird. Dieses neue Projekt können Sie in den gleichen Schritten anlegen, wie Sie das Projekt der erweiterten MFC-DLL erstellt haben. Allerdings wählen Sie im ersten Dialogfeld des DLL-Assistenten die Option *Standard- DLL*. (Damit können Sie die Standardeinstellungen des Assistenten übernehmen.) Nachdem Sie das Projekt angelegt haben, können Sie die Quellcode- und Header- Dateien der Linien- und Zeichenklasse in das Projektverzeichnis kopieren und diese Dateien in das Projekt einfügen. Anschließend beginnen Sie, die Zeichenklasse in eine Folge einfacher Funktionsaufrufe umzuwandeln.

Die Header-Datei abändern

Zunächst einmal müssen Sie die Header-Datei für die Zeichenklasse radikal ändern, damit sie sich für eine Standard-DLL eignet. Dabei müssen Sie alle Spuren der eigentlichen Klasse aus der Header-Datei eliminieren und nur die Funktionsaufrufe beibehalten. Allen Funktionen sind die jeweils erforderlichen Objekte zu übergeben. (Ein Argument jeder Funktion ist das Objektarray.) Als nächstes modifizieren Sie alle Funktionsnamen etwas, so daß der Compiler nicht durcheinanderkommt und irrtümlich eine Elementfunktion irgendeiner Klasse aufruft (wie es zum Beispiel bei der Funktion `Serialize` der Fall ist). Schließlich sind alle öffentlichen Funktionen als exportierbare Funktionen zu deklarieren. Wenn Sie diese Änderungen an der Header-Datei vornehmen, ersetzen Sie letztendlich die gesamte Klassendeklaration durch die Funktionsprototypen wie in Listing 17.5.

Listing 17.5: Die modifizierte Header-Datei `ModArt`

```
1: extern "C" void PASCAL EXPORT ModArtNewDrawing(CRect pRect,
CObArray &*poaLines);
2: extern "C" void PASCAL EXPORT ModArtSerialize(CArchive &ar,
CObArray &*poaLines);
3: extern "C" void PASCAL EXPORT ModArtDraw(CDC *pDC, CObArray
*poaLines);
4: extern "C" void PASCAL EXPORT ModArtClearDrawing(CObArray
*poaLines);
5: void NewLine(CRect pRect, CObArray *poaLines);
```



Beachten Sie, daß das Objektarray an alle diese Funktionen immer als Zeiger zu übergeben ist. Da die Funktionen Objekte in das Array einfügen und daraus löschen, müssen sie mit dem eigentlichen Array und nicht mit einer Kopie arbeiten.

Die Funktionen zum Generieren der Zeichnung anpassen

Bei der Umwandlung der Quellcodedateien sind zahlreiche kleine und doch signifikante Änderungen an diesen Funktionen vorzunehmen. Beginnen wir bei der Funktion `NewDrawing`. Hier müssen Sie das `CRect`-Objekt übergeben, um den Zeichenbereich zu erhalten. Die Funktion für das Festlegen des Zeichenbereichs löschen Sie, da es keine lokalen Variablen gibt, in denen Sie dieses Objekt aufnehmen könnten. Letztendlich sind Sie besser dran, wenn Sie es an die Funktionen zur Generierung der Zeichnung übergeben. Die andere Änderung betrifft die Stelle, wo Sie das Objektarray als weiteres Argument an die Funktion übergeben. Mit diesen Argumenten unternehmen Sie in dieser Funktion nichts weiter, sondern reichen sie lediglich an die Funktion, die das Squiggle generiert, durch. Darüber hinaus ist in dieser Funktion das Makro `AFX_MANAGE_STATE` als erste Zeile im Rumpf der Funktion hinzuzufügen. Nachdem Sie diese Änderungen vorgenommen haben, sollte die Funktion `NewDrawing` wie in Listing 17.6 aussehen.

Listing 17.6: Die Funktion `ModArtNewDrawing`

```

1: extern "C" void PASCAL EXPORT ModArtNewDrawing(CRect pRect,
CObArray &poaLines)
2: {
3:  AFX_MANAGE_STATE(AfxGetStaticModuleState());
4:  // Hier normaler Funktionsrumpf
5:  int lNumLines;
6:  int lCurLine;
7:
8:  // Zufallszahlengenerator initialisieren
9:  srand((unsigned)time(NULL));
10: // Anzahl der zu erzeugenden Linien bestimmen
11: lNumLines = rand() % 50;
12: // Sind Linien zu erzeugen?
13: if (lNumLines > 0)
14: {
15: // Schleife durch Anzahl der Linien
16: for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
17: {
18: // Die neue Linie erzeugen
19: NewLine(pRect, poaLines);
20: }
21: }
22: }

```

In der Funktion `NewDrawing` ist eine weitere Änderung hinzugekommen: die Initialisierung des Zufallszahlengenerators in Zeile 9. Da es keinen Klassenkonstruktor mehr gibt, können Sie ihm auch keinen Startwert für den Zufallszahlengenerator übergeben. Demzufolge kommt als nächster logischer Platz die Funktion `NewDrawing` in Frage, bevor irgendwelche Zufallszahlen erzeugt werden.

In der Funktion `NewLine` machen sich umfangreichere Änderungen erforderlich. Zuerst sind das `CRect`-Objekt und das Objektarray als Argumente zu übergeben. Da es sich nicht um eine exportierte Funktion handelt, brauchen Sie auch nicht das Makro `AFX_MANAGE_STATE` hinzufügen. Im dritten Schritt ersetzen Sie alle Stellen, wo die Elementvariable `CRect` vorkommt, durch das `CRect`-Objekt, das als Argument an die Funktion übergeben wird. Schließlich ist die Aufnahme von Objekten in das Objektarray so zu ändern, daß die als Argument übergebenen Objektarrayzeiger verwendet werden. Nach diesen Änderungen sollte die Funktion `NewLine` Listing 17.7 entsprechen.

Listing 17.7: Die Funktion NewLine

```

1: void NewLine(CRect pRect, CObArray *poaLines)
2: {
3:     int lNumLines;
4:     int lCurLine;
5:     // int nCurColor;
6:     UINT nCurWidth;
7:     CPoint pTo;
8:     CPoint pFrom;
9:     int cRed;
10:    int cBlue;
11:    int cGreen;
12:
13:    // Rechteck normalisieren, dann erst Breite und Höhe bestimmen
14:    pRect.NormalizeRect();
15:    // Breite und Höhe des Zeichenbereichs ermitteln
16:    int lWidth = pRect.Width();
17:    int lHeight = pRect.Height();
18:
19:    // COLORREF crColors[8] = {
20:    //     RGB( 0, 0, 0), // Schwarz
21:    //     RGB( 0, 0, 255), // Blau
22:    //     RGB( 0, 255, 0), // Grün
23:    //     RGB( 0, 255, 255), // Cyan
24:    //     RGB( 255, 0, 0), // Rot
25:    //     RGB( 255, 0, 255), // Magenta
26:    //     RGB( 255, 255, 0), // Gelb
27:    //     RGB( 255, 255, 255) // Weiß
28:    // };
29:
30:    // Anzahl der Teile dieses Squiggles bestimmen
31:    lNumLines = rand() % 200;
32:    // Umfaßt das Squiggle mindestens ein Teil?
33:    if (lNumLines > 0)
34:    {
35:        // Farbe bestimmen
36:        // nCurColor = rand() % 8;
37:        cRed = rand() % 256;
38:        cBlue = rand() % 256;
39:        cGreen = rand() % 256;
40:        // Stiftbreite bestimmen
41:        nCurWidth = (rand() % 8) + 1;
42:        // Anfangspunkt für Squiggle bestimmen
43:        pFrom.x = (rand() % lWidth) + pRect.left;
44:        pFrom.y = (rand() % lHeight) + pRect.top;
45:        // Schleife durch Anzahl der Segmente
46:        for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
47:        {

```

```

48: // Endpunkt des Segments bestimmen
49: pTo.x = ((rand() % 20) - 10) + pFrom.x;
50: pTo.y = ((rand() % 20) - 10) + pFrom.y;
51: // Neues CLine-Objekt erzeugen
52: CLine *pLine = new CLine(pFrom, pTo, RGB(cRed, cGreen, cBlue),
ÂnCurWidth);
53: try
54: {
55: // Neue Linie in das Objektarray hinzufügen
56: poaLines->Add(pLine);
57: }
58: // Speicherausnahme?
59: catch (CMemoryException* perr)
60: {
61: // Meldung an Benutzer mit schlechten
62: // Neuigkeiten
63: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
64: // Wurde ein Linienobjekt erzeugt?
65: if (pLine)
66: {
67: // Löschen
68: delete pLine;
69: pLine = NULL;
70: }
71: // Ausnahmeobjekt löschen
72: perr->Delete();
73: }
74: // Anfangspunkt auf Endpunkt setzen
75: pFrom = pTo;
76: }
77: }
78: }

```

Die übrigen Funktionen anpassen

Die erforderlichen Änderungen an den anderen Funktionen sind weniger kompliziert als die Änderungen an den Funktionen zur Generierung der Zeichnung. Bei den übrigen Funktionen ist ein Zeiger auf das Objektarray als Funktionsargument hinzuzufügen und dann die Verwendung des Arrays auf diese Zeiger anstelle der nicht mehr existierenden Elementvariablen umzustellen. Außerdem müssen Sie das Makro `AFX_MANAGE_STATE` als erste Zeilen in allen verbleibenden Funktionen hinzufügen. Damit erhalten Sie die Funktionen entsprechend der Listings 17.8 bis 17.10.

Listing 17.8: Die Funktion `ModArtDraw`

```

1: extern "C" void PASCAL EXPORT ModArtDraw(CDC *pDC, CObArray
*poaLines)
2: {
3: AFX_MANAGE_STATE(AfxGetStaticModuleState());
4: // Hier normaler Funktionsrumpf
5: // Anzahl der Linien im Objektarray ermitteln

```

```

6: int liCount = poaLines->GetSize();
7: int liPos;
8:
9: // Enthält das Array Objekte?
10: if (liCount)
11: {
12: // Schleife durch Array, dabei jedes Objekt zeichnen
13: for (liPos = 0; liPos < liCount; liPos++)
14: ((CLine*)poaLines->GetAt(liPos))->Draw(pDC);
15: }
16: }

```

Listing 17.9: Die Funktion ModArtSerialize

```

1: extern "C" void PASCAL EXPORT ModArtSerialize(CArchive &ar,
CObArray *poaLines)
2: {
3: AFX_MANAGE_STATE(AfxGetStaticModuleState());
4: // Hier normaler Funktionsrumpf
5: // Archivobjekt an Array übergeben
6: poaLines->Serialize(ar);
7: }

```

Listing 17.10: Die Funktion ModArtClearDrawing

```

1: extern "C" void PASCAL EXPORT ModArtClearDrawing(CObArray
*poaLines)
2: {
3: AFX_MANAGE_STATE(AfxGetStaticModuleState());
4: // Hier normaler Funktionsrumpf
5: // Anzahl der Linien im Objektarray bestimmen
6: int liCount = poaLines->GetSize();
7: int liPos;
8:
9: // Enthält das Array Objekte?
10: if (liCount)
11: {
12: // Schleife durch das Array, dabei alle Objekte löschen
13: for (liPos = 0; liPos < liCount; liPos++)
14: delete poaLines->GetAt(liPos)
15: // Array zurücksetzen
16: poaLines->RemoveAll();
17: }
18: }

```

Nachdem Sie die Änderungen an diesen Funktionen ausgeführt haben, ist nur noch der gesamte Code für den Konstruktor und Destruktor der Klasse sowie der Code für die Funktion `SetRect` zu löschen.

Die Moduldefinitionsdatei erstellen

Bevor Sie die DLL kompilieren, müssen Sie alle Funktionsnamen in die Moduldefinitionsdatei aufnehmen. Diese Datei finden Sie in der Liste der Quellcodedateien auf der Registerkarte **Dateien** des Arbeitsbereichs. Wenn Sie diese Datei öffnen, sehen Sie, daß sie kurz das zu erstellende Modul in allgemeiner Form beschreibt. Am Ende der Datei finden Sie eine Stelle, wo Sie die Exportanweisungen für die DLL unterbringen können. Bearbeiten Sie diese Datei, indem Sie die exportierbaren Funktionsnamen wie in Listing 17.11 hinzufügen.

Listing 17.11: Die Moduldefinitionsdatei der DLL

```
1: ; ModArtRDll.def : Deklariert die Modul-Parameter für die DLL.
2:
3: LIBRARY "ModArtRDll"
4: DESCRIPTION 'ModArtRDll Windows Dynamic Link Library'
5:
6: EXPORTS
7: ; Explizite Exporte können hier eingefügt werden
8: ModArtNewDrawing
9: ModArtSerialize
10: ModArtDraw
11: ModArtClearDrawing
```

Jetzt können Sie Ihre Standard-DLL kompilieren. Anschließend kopieren Sie sie in das Debug-Verzeichnis der Testanwendung.

Die Testanwendung anpassen

Um die Testanwendung an den Einsatz der neuen DLL, die Sie gerade erstellt haben, anzupassen, sind eine Reihe von Änderungen erforderlich. Erstens müssen Sie die Elementvariable der Dokumentklasse von einer Instanz der Zeichenklasse in das Objektarray ändern. Als nächstes sind die Include-Anweisungen im Quellcode des Dokuments und der Ansicht anzupassen, um die Header der neuen DLL anstelle der alten DLL einzubinden. (In der Quellcodedatei der Anwendung können Sie sämtliche Include-Anweisungen löschen.) Löschen Sie die DLL-LIB-Datei, und fügen Sie die LIB-Datei für die neue DLL in das Projekt ein. Ändern Sie alle Funktionsaufrufe der Zeichenklasse in Aufrufe von Funktionen in der neuen DLL. Schließlich ändern Sie die Funktion `GetDrawing` in der Dokumentklasse, so daß sie einen Zeiger auf das Objektarray statt auf das Zeichenobjekt zurückgibt.

Löschen Sie als erstes die LIB-Datei aus dem Projekt der Testanwendung. Fügen Sie anschließend die LIB-Datei für die neue DLL mit **Projekt / Dem Projekt hinzufügen / Dateien** in das Projekt ein.

Nachdem Sie die LIB-Dateien im Projekt ausgetauscht haben, bearbeiten Sie den Quellcode für die Dokument- und Ansichtsklassen, um die Include-Anweisung an das Projektverzeichnis für die neue DLL anzupassen. In der Quellcodedatei der Anwendung können Sie die Include-Anweisungen löschen. Da Sie keine Instanz der Zeichenklasse erzeugen, muß die Anwendungsdatei überhaupt nichts über die DLL wissen.

Nach diesen Änderungen öffnen Sie die Header-Datei für die Dokumentklasse. Bearbeiten Sie die Deklaration der Dokumentklasse wie folgt: Ändern Sie den Typ der Funktion `GetDrawing`, so daß die Funktion einen Zeiger auf ein Objektarray zurückgibt, entfernen Sie die Variable der Zeichenklasse, und fügen Sie eine Objektarrayvariable hinzu, wie es aus Listing 17.12 hervorgeht. Nehmen Sie nur diese drei Änderungen vor, alles andere in der Klassendeklaration bleibt erhalten.

Listing 17.12: Die Deklaration der Klasse CTestAppDoc


```

1: class CTestAppDoc : public CDocument
2: {
3: protected: // Nur aus Serialisierung erzeugen
4: CTestAppDoc();
5: DECLARE_DYNCREATE(CTestAppDoc)
6: .
7: .
8: .
9: // Implementierung
10: public:
11: CObArray* GetDrawing();
12: virtual ~CTestAppDoc();
13: .
14: .
15: .
16: private:
17: CObArray m_oaLines;
18: };

```

Die Funktionen des Dokuments modifizieren

Nachdem Sie die allgemeinen Änderungen an der Testanwendung vorgenommen haben, sind nun die Änderungen in der Funktionalität an der Reihe. Alle Aufrufe von Klassenmethoden des Zeichenobjekts sind in die entsprechenden Aufrufe von Funktionen in der neuen DLL zu ändern.

In der Funktion `OnNewDocument` löschen Sie den Funktionsaufruf, der das `CRect`-Objekt an das Zeichenobjekt übergibt, und ersetzen den Aufruf der Funktion `NewDocument` durch die neue DLL-Funktion - in diesem Fall `ModArtNewDrawing`, wie es Zeile 20 in Listing 17.13 zeigt.

Listing 17.13: Die Funktion `OnNewDocument` der Klasse `CTestAppDoc`

```

1: BOOL CTestAppDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: // Position der Ansicht ermitteln
10: POSITION pos = GetFirstViewPosition();
11: // Ist Position gültig?
12: if (pos != NULL)
13: {
14: // Zeiger auf die Ansicht holen
15: CView* pView = GetNextView(pos);
16: RECT lwndRect;
17: // Rechteck des Anzeigebereichs holen
18: pView->GetClientRect(&lwndRect);
19: // Neue Zeichnung erzeugen

```



```

20: ModArtNewDrawing(lWndRect, &m_oaLines);
21: }
22:
23: return TRUE;
24: }

```

In der Funktion `Serialize` ändern Sie den Aufruf der `Serialize`-Funktion des Zeichenobjekts in die neue Serialisierungsfunktion der DLL - in diesem Fall `ModArtSerialize` , wie in Listing 17.14 angegeben.

Listing 17.14: Die Funktion Serialize der Klasse CTestAppDoc

```

1: void CTestAppDoc::Serialize(CArchive& ar)
2: {
3: // Zeichnung serialisieren
4: ModArtSerialize(ar, &m_oaLines);
5: }

```

In der Funktion `DeleteContents` ist der Aufruf der Funktion `ClearDrawing` durch den Aufruf der neuen DLL-Funktion `ModArtClearDrawing` entsprechend Zeile 5 von Listing 17.15 zu ersetzen.

Listing 17.15: Die Funktion DeleteContents der Klasse CTestAppDoc

```

1: void CTestAppDoc::DeleteContents()
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
aufrufen
4: // Zeichnung löschen
5: ModArtClearDrawing(&m_oaLines);
6:
7: CDocument::DeleteContents();
8: }

```

Schließlich müssen Sie für die Funktion `GetDrawing` die Funktionsdeklaration ändern, um zu kennzeichnen, daß sie einen Zeiger auf ein Objektarray zurückgibt, genau wie Sie es in der Header-Datei vorgenommen haben. Als nächstes ändern Sie die Variable, die zurückgegeben wird, in die Objektarrayvariable, die Sie in die Header-Datei aufgenommen haben (siehe Listing 17.16).

Listing 17.16: Die Funktion GetDrawing der Klasse CTestAppDoc

```

1: CObArray* CTestAppDoc::GetDrawing()
2: {
3: // Das Zeichenobjekt zurückgeben
4: return &m_oaLines;
5: }

```

Die Funktionen der Ansicht modifizieren

In der Ansichtsklasse ist eine einfache Änderung an der Funktion `OnDraw` vorzunehmen. In dieser Funktion ist der Typ des Zeigers, der aus der Funktion `GetDrawing` abgerufen wird, von einem Zeichenobjekt in ein Objektarrayobjekt zu ändern, wie es Zeile 9 von Listing 17.17 zeigt. Unmittelbar danach - in Zeile 11 von Listing 17.17 - rufen Sie die DLL-Funktion `ModArtDraw` auf, um das

Zeichnen im Fenster auszuführen.

Listing 17.17: Die Funktion OnDraw der Klasse CTestAppView

```

1: void CTestAppView::OnDraw(CDC* pDC)
2: {
3: CModTestAppDoc* pDoc = GetDocument();
4: ASSERT_VALID(pDoc);
5:
6: // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen Daten
  Ähinzufügen
7:
8: // Zeichenobjekt holen
9: CObArray* m_oaLines = pDoc->GetDrawing();
10: // Zeichnung ausgeben
11: ModArtDraw(pDC, m_oaLines);
12: }

```

Nachdem Sie alle Änderungen an der Testanwendung vorgenommen haben, können Sie die Anwendung kompilieren und testen. Die Anwendung sollte genauso arbeiten wie mit der vorherigen DLL. Sie können auch damit experimentieren, indem Sie zurückgehen und die DLL ändern, die neue DLL in das Debug-Verzeichnis für die Testanwendung kopieren und verfolgen, wie sich die Änderungen im Verhalten der Testanwendung widerspiegeln.



Die im heutigen Beispiel entwickelte Version einer normalen DLL ist noch nicht mit anderen Programmiersprachen einsetzbar. Der Grund dafür liegt darin, daß Sie MFC-Klassen als Argumente für alle Funktionen der DLL übergeben. Damit ist die Einsatzbreite auf Anwendungen beschränkt, die mittels MFC erstellt wurden. Um diese DLL tatsächlich portierbar zu machen, müssen Sie die nackten Strukturen anstelle der Klassen übergeben (beispielsweise die **RECT**-Struktur anstelle der **CRect**-Klasse) und dann die Strukturen innerhalb der DLL in Klassen konvertieren.

Zusammenfassung

Heute haben Sie zwei Möglichkeiten kennengelernt, wie Sie Ihre Funktionalität für andere Programmierer verpacken können. Es wurde dargestellt, wie leicht sich Klassen in einer erweiterten MFC-DLL unterbringen lassen und wie einfach man sie in einer Visual-C++-Anwendung einsetzen kann. Sie haben gesehen, wie man Änderungen an der DLL vornehmen kann, ohne die darauf zugreifenden Anwendungen erneut kompilieren zu müssen. Weiterhin haben Sie gelernt, was zum Erstellen einer Standard- DLL gehört, die sich mit anderen - nicht mit Visual C++ erstellten - Anwendungen einsetzen läßt. Sie haben gesehen, wie Sie die exportierten Klassen aus der DLL in normale Funktionen nach C-Konventionen konvertieren mußten und was erforderlich ist, um eine Anwendung, die diese Art von DLL verwendet, anzupassen.

Fragen und Antworten

Frage:

Wie kann ich die Standard-DLL konvertieren, so daß sie sich in Anwendungen einsetzen läßt, die nicht mit Visual C++ erstellt wurden?

Antwort:

Zuerst sind alle Argumente an die Funktionen mit reinen Strukturen anstelle von MFC-Klassen zu übergeben. Um zum Beispiel die Funktion `ModArtNewDrawing` umzuwandeln, ändern Sie sie dahingehend, daß sie die `RECT`-Struktur anstelle der `CRect`-Klasse sowie einen generischen Zeiger anstelle eines Zeigers auf ein Objektarray erhält. Die Umwandlungen sind in den entsprechenden Klassen in der DLL vorzunehmen, wie es die Zeilen 4 bis 9 von Listing 17.18 zeigen.

Listing 17.18: Die Funktion `ModArtNewDrawing`

```
1: extern "C" void PASCAL EXPORT ModArtNewDrawing(RECT spRect,
LPVOID lpoaLines)
2: {
3:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
4:   CRect pRect;
5:   pRect.top = spRect.top;
6:   pRect.left = spRect.left;
7:   pRect.right = spRect.right;
8:   pRect.bottom = spRect.bottom;
9:   CObArray* poaLines = (CObArray*)lpoaLines;
10:  // Hier normaler Funktionsrumpf
11:  int m_lNumLines;
12:  int m_lCurLine;
13:
14:  // Zufallszahlengenerator initialisieren
15:  srand((unsigned)time(NULL));
16:  // Anzahl der zu erzeugenden Linien bestimmen
17:  m_lNumLines = rand() % 50;
18:  // Sind Linien zu erzeugen?
19:  if (m_lNumLines > 0)
20:  {
21:    // Schleife durch Anzahl der Linien
22:    for (m_lCurLine = 0; m_lCurLine < m_lNumLines; m_lCurLine++)
23:    {
24:      // Die neue Linie erzeugen
25:      NewLine(pRect, poaLines);
26:    }
27:  }
28: }
```

Darüber hinaus müssen Sie bei der Anwendung, die das Objektarray als generischen Zeiger speichert, Funktionen hinzufügen, um das Objektarray zu erzeugen und zu zerstören, wie es aus Listing 17.19 hervorgeht.

Listing 17.19: Die Funktion `ModArtInit`

```

1: extern "C" LPVOID PASCAL EXPORT ModArtInit()
2: {
3:   AFX_MANAGE_STATE(AfxGetStaticModuleState());
4:   // Objektarray erzeugen
5:   return (LPVOID)new CObArray;
6: }

```

Frage:

Wann muß ich die Anwendungen, die meine DLL verwenden, neu kompilieren?

Antwort:

Immer dann, wenn Sie die Aufrufe von exportierten Funktionen ändern. Wenn Sie Argumente in diesen Funktionen ändern, hinzufügen oder entfernen, müssen Sie die Anwendungen, die auf die DLL zurückgreifen, neu kompilieren. Arbeiten Sie mit einer erweiterten MFC-DLL, ist die Anwendung, die die DLL verwendet, neu zu kompilieren, wenn sich die öffentliche Schnittstelle für die exportierten Klassen ändert oder eine neue Funktion oder Variable hinzugefügt oder entfernt wird. Es spielt keine Rolle, wenn die Anwendung gar nicht auf die geänderten Funktionen zurückgreift. Allerdings ist es übliche Praxis, die Anwendungen neu zu kompilieren, einfach um sicher zu sein.

Workshop

Kontrollfragen

1. Welche Art von DLL müssen Sie erstellen, um Klassen in der DLL für die Anwendungen verfügbar zu machen?
2. Was müssen Sie in die Klasse einfügen, um sie aus einer DLL zu exportieren?
3. Welche Art von DLL läßt sich mit anderen Programmiersprachen nutzen?
4. Müssen Sie Anwendungen, die auf eine DLL zurückgreifen, neu kompilieren, wenn Sie die betreffende DLL verändern?
5. Welche Funktion stellt die LIB-Datei für eine DLL bereit?

Übungen

1. Spalten Sie die Linienklasse in ihre eigene erweiterte MFC-DLL auf, und verwenden Sie sie mit der zweiten (Standard-) DLL.
2. Ändern Sie die DLL für die Linienklasse, so daß sie eine einheitliche Linienbreite für alle Linien verwendet.

Woche 3**Tag 18****Multitasking**

Manchmal ist es angebracht, Anwendungen mehrere Dinge gleichzeitig ausführen zu lassen. Die Anwendung könnte eine Sicherungsdatei schreiben oder im Hintergrund drucken, während der Benutzer am selben Dokument weiterarbeitet. Vielleicht führt Ihre Anwendung Berechnungen aus, während der Benutzer neue Daten eingibt, oder zeichnet mehrere Bilder gleichzeitig. Es gibt viele verschiedene Gründe, warum man diese Fähigkeit, das sogenannte Multitasking, in Anwendungen realisieren möchte. Windows stellt verschiedene Einrichtungen bereit, um derartige Fähigkeiten in eine Anwendung einbauen zu können.

Heute lernen Sie, wie ...

- sich Tasks ausführen lassen, während eine Anwendung im Leerlauf arbeitet,
- man Tasks unabhängig von der übrigen Anwendung starten kann,
- man den Zugriff auf die Ressourcen koordiniert, die von mehreren unabhängigen Tasks gemeinsam genutzt werden,
- man unabhängig laufende Tasks startet und stoppt.

Was ist Multitasking?

In den Tagen von Windows 3.x waren alle Windows-Anwendungen Singlethreading- Anwendungen, die zu einem beliebigen Zeitpunkt nur einen Ausführungspfad aufwiesen. Die in Windows 3.x realisierte Form des Multitasking bezeichnet man als kooperatives Multitasking. Der Schlüssel zu dieser Art des Multitasking liegt darin, daß jede einzelne Anwendung darüber entscheidet, wann sie den Prozessor für eine andere Anwendung freigibt, damit diese die anstehenden Arbeiten erledigen kann. Im Ergebnis war Windows 3.x anfällig für eine sich schlecht verhaltende Anwendung, die andere Anwendungen gefangenhält, während sie einen länger dauernden Prozeß ausführt oder sich sogar selbst in irgendeiner Schleife verfängt.

Mit Windows NT und Windows 95 hat sich die Natur des Betriebssystems geändert. Das neue präemptive Multitasking hat das kooperative Multitasking abgelöst. Beim präemptiven Multitasking entscheidet das Betriebssystem, wann es der einen Anwendung den Prozessor entzieht und einer anderen Anwendung, die bereits darauf wartet, zuteilt. Es spielt keine Rolle, ob die Anwendung, die momentan den Prozessor für sich in Anspruch nimmt, bereit ist, den Prozessor abzugeben. Das Betriebssystem entzieht der Anwendung den Prozessor, ohne die Anwendung dazu um Erlaubnis zu fragen. Auf diese Weise trägt das Betriebssystem dazu bei, daß mehrere Anwendungen rechenintensive Aufgaben erledigen können und trotzdem alle Anwendungen in ihren eigenen Verarbeitungen vorwärts kommen.

Indem man diese Fähigkeit dem Betriebssystem überträgt, verhindert man, daß eine einzige Anwendung alle anderen Anwendungen blockiert, während sie den Prozessor beansprucht.



Bei der 16/32-Bit-Struktur von Windows 95 ist es trotzdem noch möglich, daß eine sich schlecht verhaltende 16-Bit-Anwendung das System blockiert, da ein großer Teil von 16-Bit-Code weiterhin zum Kern des Betriebssystems gehört. Der 16-Bit-Code von Windows 95 stellt immer noch eine kooperative Multitasking-Umgebung dar, so daß nur jeweils eine Anwendung 16-Bit-Code zu einem bestimmten Zeitpunkt ausführen kann. Da alle USER- Funktionen und ein guter Teil der GDI-Funktionen auf die 16-Bit-Versionen zurückgreifen, kann eine einzige 16-Bit-Anwendung das gesamte System blockieren.

Wenn in Windows NT alle 16-Bit-Anwendungen in einem gemeinsam genutzten Speicherraum laufen, kann eine sich schlecht verhaltende Anwendung zwar alle 16-Bit-Anwendungen blockieren, was aber ohne Auswirkung auf irgendwelche 32-Bit-Anwendungen bleibt.

Mehrere Aufgaben gleichzeitig ausführen

Zusätzlich zur Fähigkeit, mehrere Anwendungen simultan ausführen zu können, kommt die Fähigkeit, daß eine einzelne Anwendung mehrere Threads zu ein und demselben Zeitpunkt ausführt. Ein Thread (Programmfaden) ist für eine Anwendung das, was eine Anwendung für das Betriebssystem darstellt. Wenn eine Anwendung mehrere Threads laufen hat, führt sie im Prinzip mehrere Anwendungen innerhalb der Anwendung als Ganzes aus und kann damit mehrere Dinge gleichzeitig erledigen. Beispielsweise prüft Microsoft Word die Rechtschreibung zum gleichen Zeitpunkt, zu dem man Eingaben in das Dokument vornimmt.

Threads zur Leerlaufverarbeitung

Einer der einfachsten Wege, um die Anwendung mehrere Verarbeitungen gleichzeitig ausführen zu lassen, ist die Realisierung der Leerlaufverarbeitung. Eine Leerlaufverarbeitung kommt zum Zuge, wenn sich eine Anwendung im Leerlaufzustand befindet. Im wahrsten Sinne des Wortes wird eine Funktion in der Anwendungsklasse aufgerufen, wenn keine Nachrichten in der Nachrichtenwarteschlange der Anwendung stehen. Der Gedanke hinter dieser Funktion ist der, daß eine Anwendung im Leerlaufzustand Aufgaben wie Speicherbereinigung (die sogenannte Garbage Collection) oder Schreiben in einen Drucker-Spooler ausführen kann.

Die Funktion `OnIdle` ist ein Überbleibsel aus den Tagen von Windows 3.x. Sie gehört zur Klasse `CWinApp`, von der Ihre Anwendung abgeleitet ist. Per Vorgabe fügt der Anwendungs-Assistent keinerlei Verarbeitungen in diese Funktion ein. Wenn Sie demzufolge die Funktion in Ihrer Anwendung nutzen möchten, müssen Sie sie in Ihre Anwendungsklasse mit dem Klassen-Assistenten aufnehmen. (`OnIdle` ist eine der verfügbaren Nachrichten für die Anwendungsklasse in Ihren Anwendungen.)

Die Funktion `OnIdle` übernimmt ein Argument, das die Anzahl der Aufrufe dieser Funktion angibt, die seit der letzten von der Anwendung verarbeiteten Nachricht stattgefunden haben. Anhand dieses Wertes können Sie ermitteln, wie lange die Anwendung im Leerlauf gearbeitet hat und wann eine Aktion auszulösen ist, falls die Anwendung eine bestimmte Zeitbegrenzung im Leerlauf überschritten hat.

Einer der wichtigsten Punkte bei der `OnIdle`-Verarbeitung in Ihren Anwendungen ist der, daß der Umfang der Funktionalität klein sein und die Steuerung schnell an den Benutzer zurückgehen muß. Wenn eine Anwendung eine `OnIdle`-Verarbeitung durchführt, kann der Benutzer nicht mit der

Anwendung weiterarbeiten, bis die `OnIdle`-Verarbeitung abgeschlossen ist und der Benutzer die Steuerung wiedererlangt. Wenn Sie eine längere, ausgedehnte Aufgabe in der Funktion `OnIdle` ausführen müssen, sollten Sie sie in viele kleine und schnelle Fragmente aufteilen, damit der Benutzer die Steuerung zwischenzeitlich zurückerhält. Sobald die Nachrichtenwarteschlange wieder leer ist, können Sie Ihre `OnIdle`-Verarbeitung fortsetzen. Das bedeutet, daß Sie auch den Fortschritt der Anwendung in der `OnIdle`-Verarbeitung verfolgen müssen, damit die Anwendung beim nächsten Aufruf der Funktion `OnIdle` die Verarbeitung an der Stelle wieder aufnehmen kann, wo sie verlassen wurde.

Unabhängige Threads aufspannen

Wenn Sie tatsächlich eine längere Hintergrundverarbeitung ausführen müssen, die der Benutzer nicht unterbrechen soll, müssen Sie einen unabhängigen Thread erzeugen. Ein Thread verhält sich wie eine andere Anwendung, die innerhalb Ihrer Anwendung läuft. Der Thread muß nicht darauf warten, daß die Anwendung in den Leerlauf gelangt, um seine Aufgaben auszuführen, und er bewirkt auch nicht, daß der Benutzer warten muß, bis der Thread die Steuerung zwischenzeitlich abgibt.

Die beiden Methoden zum Erzeugen eines unabhängigen Threads verwenden die gleiche Funktion, um den Thread zu erzeugen und zu starten. Um einen unabhängigen Thread zu erzeugen und zu starten, ruft man die Funktion `AfxBeginThread` auf. Dabei kann man wählen, ob man ihr eine aufzurufende Funktion für die Ausführung der Thread-Verarbeitung oder einen Zeiger auf die Laufzeitklasse eines Objekts, das von der Klasse `CWinThread` abgeleitet ist, übergibt. Beide Versionen der Funktion liefern einen Zeiger auf ein `CWinThread`-Objekt zurück, das als unabhängiger Thread läuft.

In der ersten Version der Funktion `AfxBeginThread` ist das erste Argument ein Zeiger auf die Hauptfunktion, um den Thread zu starten. Diese Funktion ist das Äquivalent der Funktion `main` in einem C/C++-Programm. Sie steuert die Ausführung auf höchster Ebene für den Thread. Diese Funktion ist als `UINT`-Funktion mit einem einzelnen `LPVOID`-Argument zu definieren:

```
UINT MyThreadFunction( LPVOID pParam);
```

Diese Version von `AfxBeginThread` erfordert auch ein zweites Argument, das als einziges Argument an die Hauptfunktion des Threads weitergereicht wird. Dieses Argument kann ein Zeiger auf eine Struktur sein, die alle Informationen enthält, die der Thread für die korrekte Ausführung seiner Aufgabe kennen muß.

Das erste Argument an die zweite Version der Funktion `AfxBeginThread` ist ein Zeiger auf die Laufzeitklasse eines Objekts, das von der Klasse `CWinThread` abgeleitet ist. Einen Zeiger auf die Laufzeitklasse Ihrer `CWinThread`-Klasse können Sie mit dem Makro `RUNTIME_CLASS` ermitteln, wobei Sie Ihre Klasse als einziges Argument übergeben.

Nach diesen ersten Argumenten sind die restlichen Argumente an die Funktion `AfxBeginThread` bei beiden Versionen gleich und insgesamt optional. Das erste dieser Argumente bezeichnet die gewünschte Priorität des Threads, wobei der Wert `THREAD_PRIORITY_NORMAL` vorgegeben ist. Tabelle 18.1 listet die verfügbaren Thread- Prioritäten auf.

Tabelle 18.1: Prioritäten von Threads

Priorität	Beschreibung
0	Der Thread erbt die Thread-Priorität der Anwendung, die den Thread erzeugt.
<code>THREAD_PRIORITY_NORMAL</code>	Eine normale (Standard-) Priorität.

THREAD_PRIORITY_ABOVE_NORMAL	1 Punkt über der normalen Priorität.
THREAD_PRIORITY_BELOW_NORMAL	1 Punkt unterhalb der normalen Priorität.
THREAD_PRIORITY_HIGHEST	2 Punkte über der normalen Priorität.
THREAD_PRIORITY_LOWEST	2 Punkte unterhalb der normalen Priorität.
THREAD_PRIORITY_IDLE	Prioritätsebene 1 für die meisten Threads (alle Threads ohne Echtzeitverarbeitung).
THREAD_PRIORITY_TIME_CRITICAL	Prioritätsebene 15 für die meisten Threads (alle Threads ohne Echtzeitverarbeitung).



Die Thread-Priorität steuert, wieviel CPU-Zeit der Thread in Relation zu den anderen Threads und Prozessen, die auf dem Computer laufen, erhält. Wenn ein Thread keinerlei Verarbeitungen ausführt, die schnell fertigzustellen sind, dann sollten Sie dem Thread beim Erstellen eine niedrigere Priorität geben. Es empfiehlt sich nicht, einem Thread eine Priorität höher als normal zu geben, solange es nicht unbedingt notwendig ist, daß der Thread seine Verarbeitung schneller als andere Prozesse auf dem Computer ausführen muß. Je höher die Priorität eines Threads ist, desto mehr CPU-Zeit erhält er und desto weniger CPU-Zeit steht für alle anderen Prozesse und Threads auf dem Computer zur Verfügung.

Das nächste Argument an die Funktion `AfxBeginThread` ist die Stack-Größe, die für den neuen Thread bereitzustellen ist. Der Standardwert für dieses Argument ist 0. Damit erhält der Thread die gleiche Stack-Größe wie die Hauptanwendung.

Als nächstes Argument an die Funktion `AfxBeginThread` übergibt man das Flag zum Erzeugen des Threads. Dieses Flag kann einen von zwei Werten enthalten. Es steuert, wie der Thread gestartet wird. Wenn man für dieses Argument den Wert `CREATE_SUSPENDED` übergibt, wird der Thread in einem angehaltenen Zustand erzeugt. Der Thread startet erst dann, wenn die Funktion `ResumeThread` für den Thread aufgerufen wird. Übergibt man für dieses Argument den Wert 0, d.h. den Standardwert, beginnt der Thread die Ausführung in dem Moment, zu dem er erzeugt wird.

Das letzte Argument an die Funktion `AfxBeginThread` ist ein Zeiger auf die Sicherheitsattribute für den Thread. Der Standardwert für dieses Argument ist `NULL`. Damit wird der Thread mit dem gleichen Sicherheitsprofil wie die Anwendung erzeugt. Solange Sie keine Anwendungen für Windows NT erstellen und einen Thread mit einem bestimmten Sicherheitsprofil bereitstellen müssen, sollten Sie für dieses Argument immer den Standardwert einsetzen.

Strukturen erstellen

Stellen Sie sich eine Anwendung vor, die zwei Threads ausführt, wobei jeder seinen eigenen Satz von Variablen zur gleichen Zeit abarbeitet. Stellen wir uns weiterhin vor, daß die Anwendung ein globales Objektfeld verwendet, um diese Variablen aufzunehmen. Wenn die Methode zur Reservierung und Größenänderung des Arrays die aktuelle Größe prüft und eine Position am Ende des Feldes anfügt, bauen Ihre beiden Threads vielleicht ein Feld auf, das etwa gemäß Abbildung 18.1 gefüllt ist. Hier sind die durch den ersten Thread gefüllten Positionen mit denjenigen vermischt, die der zweite Thread erzeugt hat. Die beiden Threads können dabei leicht durcheinanderkommen, wenn sie Werte aus dem Feld abrufen und weiterverarbeiten wollen, da jeder Thread mit hoher Wahrscheinlichkeit einen Wert abrufen, der eigentlich zum anderen Thread gehört. Das führt dazu, daß die Threads auf den falschen

Daten arbeiten und falsche Ergebnisse zurückliefern.

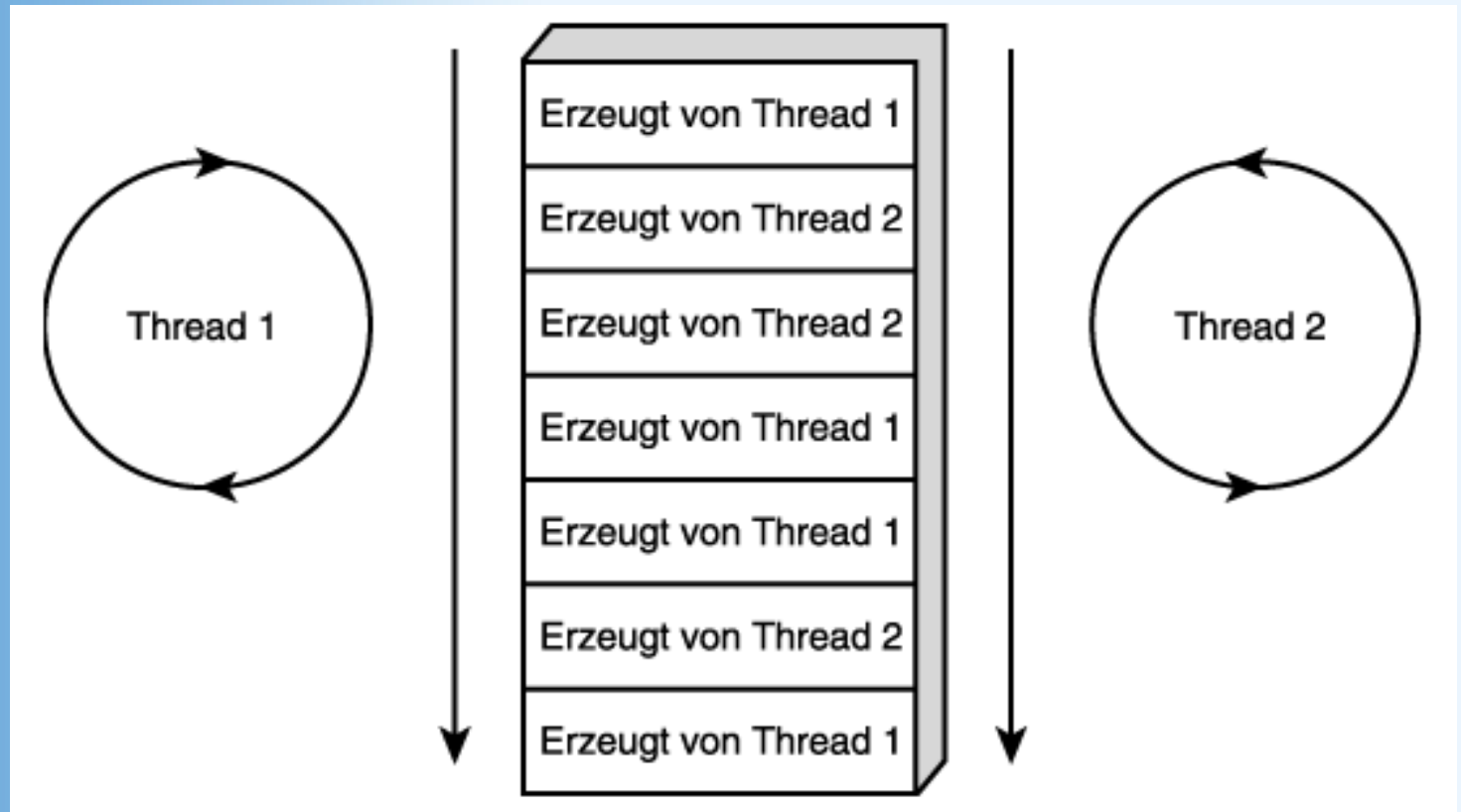


Abbildung 18.1:
Zwei Threads, die ein gemeinsames Datenfeld füllen

Wenn die Anwendung die Felder als lokal und nicht als global erstellt, kann sie den Zugriff auf die Felder nur auf den Thread einschränken, der das Feld erstellt. Im Beispiel von Abbildung 18.2 tritt keine Vermischung von Daten aus mehreren Threads auf. Wenn Sie mit dieser Lösung bei Feldern und anderen Speicherstrukturen arbeiten, kann jeder Thread seine Aufgaben erledigen und die Ergebnisse an den Client zurückgeben. Dabei ist sichergestellt, daß es sich um die richtigen Ergebnisse handelt, da die Berechnungen auf eigenständigen Daten ablaufen.

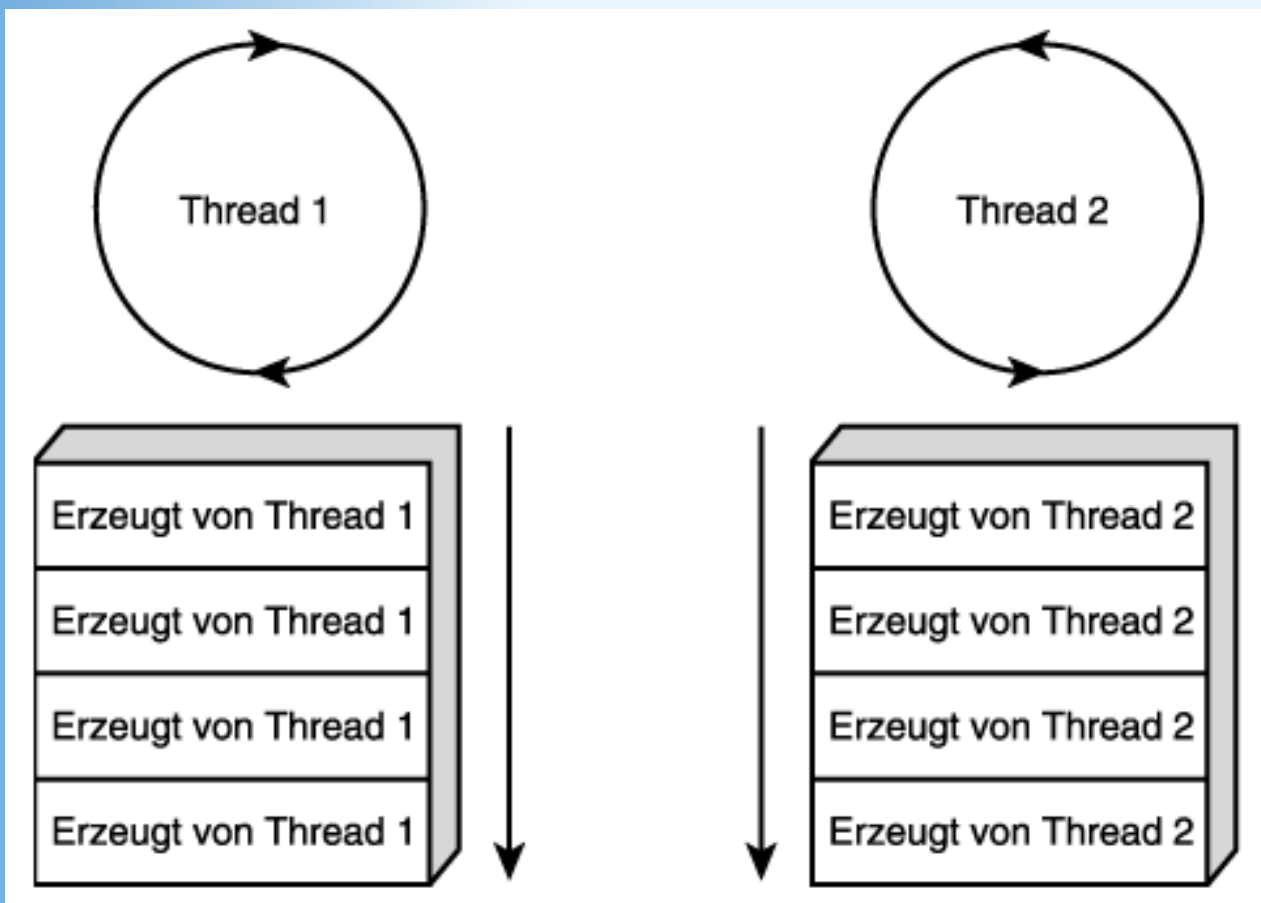


Abbildung 18.2:
Zwei Threads, die lokal angelegte Datenfelder füllen

Zugriff auf gemeinsam genutzte Ressourcen verwalten

Nicht alle Variablen lassen sich in einer lokalen Form bereitstellen, und es ist oft erforderlich, bestimmte Ressourcen zwischen allen Threads, die in Ihren Anwendungen laufen, gemeinsam zu nutzen. Diese gemeinsame Nutzung führt zu einem Problem bei Anwendungen mit mehreren Threads. Nehmen wir an, daß drei Threads mit einem einzigen Zähler arbeiten, der eindeutige Zahlen erzeugt. Da man nicht weiß, wann die Steuerung des Prozessors von einem Thread zum nächsten übergeht, kann die Anwendung doppelte »eindeutige« Werte erzeugen, wie es Abbildung 18.3 zeigt.

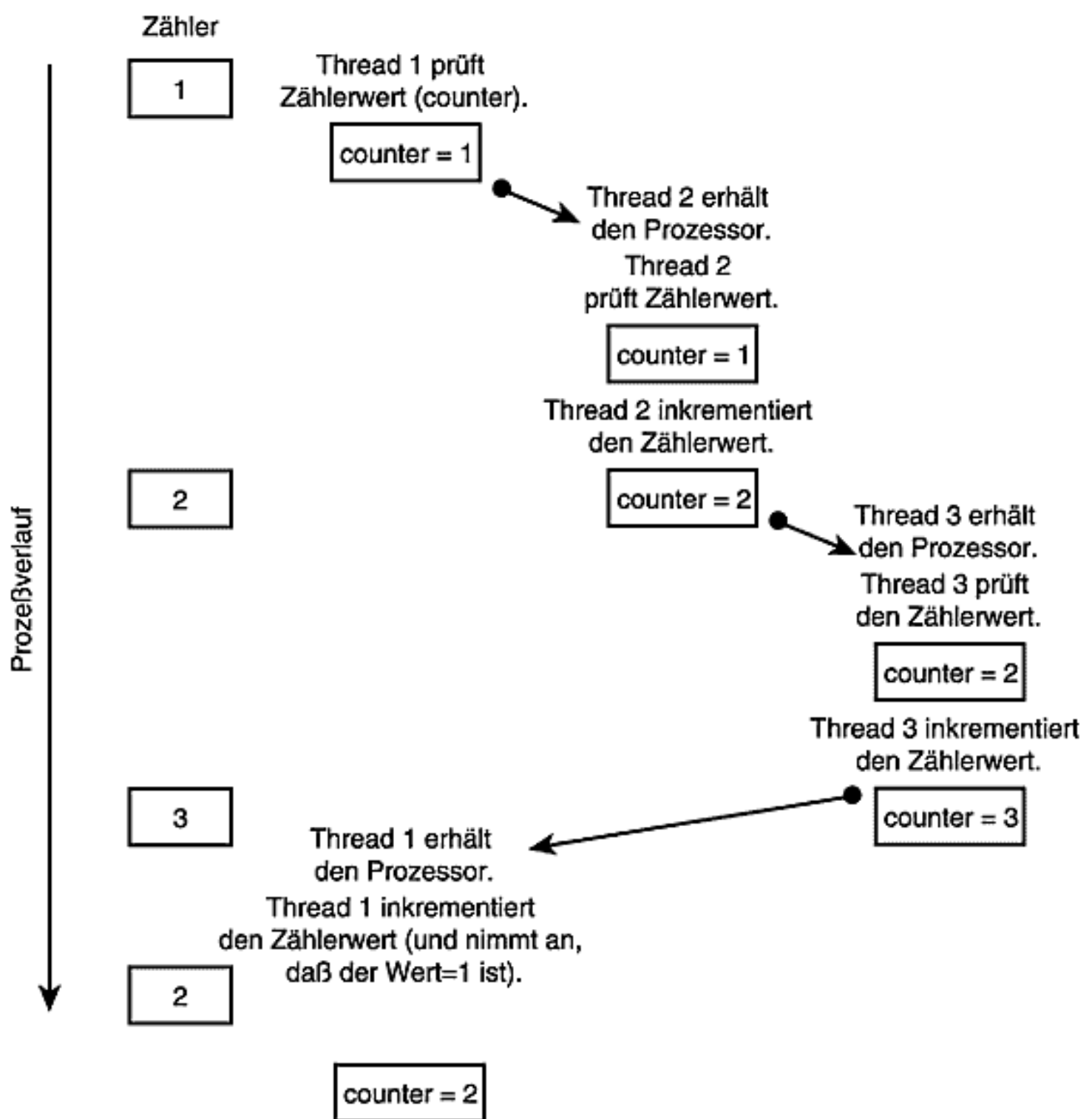


Abbildung 18.3:
Drei Threads, die ein und denselben Zähler gemeinsam nutzen

Wie man sieht, funktioniert diese gemeinsame Nutzung in einer Anwendung mit mehreren Threads nicht besonders gut. Man braucht eine Möglichkeit, um den Zugriff auf eine gemeinsam genutzte Ressource zu einem bestimmten Zeitpunkt auf nur einen Thread einzuschränken. In der Tat existieren vier Mechanismen, mit denen man den Zugriff auf gemeinsame Ressourcen begrenzen und die Verarbeitung zwischen Threads synchronisieren kann.

Diese Mechanismen arbeiten nach verschiedenen Methoden. Welche davon geeignet ist, hängt von den konkreten Umständen ab. Die vier Mechanismen heißen:

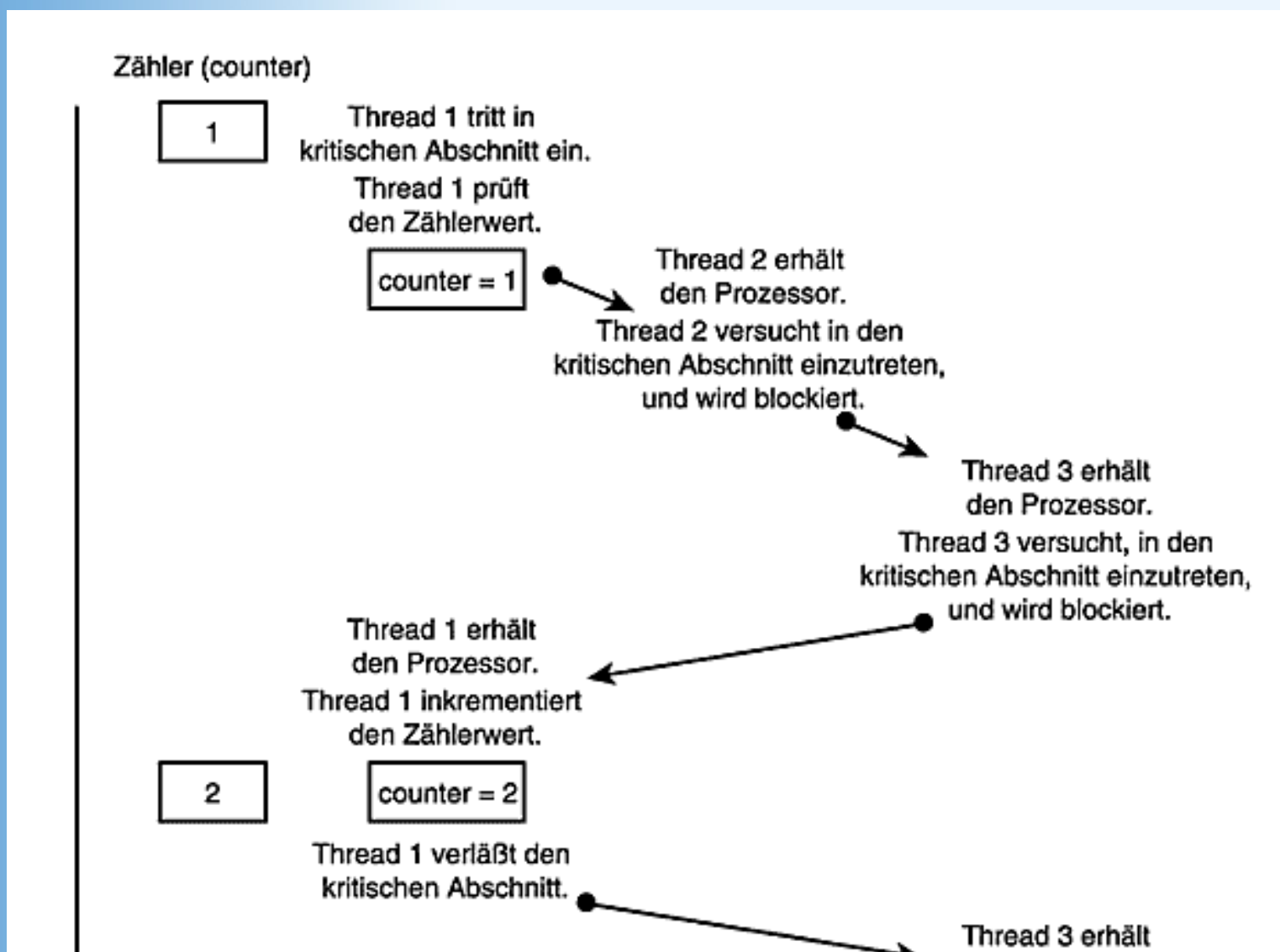
- Kritische Abschnitte

- Mutexe
- Semaphore
- Ereignisse

Kritische Abschnitte

Ein kritischer Abschnitt ist ein Mechanismus, der den Zugriff auf eine bestimmte Ressource auf einen einzelnen Thread innerhalb einer Anwendung beschränkt. Ein Thread tritt in den kritischen Abschnitt ein, bevor er mit der angegebenen gemeinsam genutzten Ressource arbeiten muß, und verläßt den kritischen Abschnitt, nachdem der Zugriff auf die Ressource abgeschlossen ist. Wenn ein anderer Thread versucht, in den kritischen Abschnitt einzutreten, bevor der erste Thread den kritischen Abschnitt verlassen hat, wird der zweite Thread blockiert und erhält keine Prozessorzeit, bis der erste Thread den kritischen Abschnitt verläßt und damit dem zweiten Thread den Eintritt ermöglicht. Mit kritischen Abschnitten markiert man Codebereiche, die nur ein Thread zu einem bestimmten Zeitpunkt ausführen sollte. Damit verhindert man nicht, daß der Prozessor von diesem Thread zu einem anderen umschaltet. Es wird nur unterbunden, daß zwei oder mehr Threads in denselben Codeabschnitt eintreten.

Wenn Sie einen kritischen Abschnitt mit dem Zähler gemäß Abbildung 18.3 verwenden, können Sie erzwingen, daß jeder Thread in einen kritischen Abschnitt eintritt, bevor der Thread den aktuellen Wert des Zählers prüft. Wenn der Thread den kritischen Abschnitt erst dann verläßt, wenn er den Zähler inkrementiert und aktualisiert hat, ist garantiert, daß - unabhängig davon wie viele Threads ausgeführt werden und in welcher Reihenfolge die Ausführung erfolgt - wirklich eindeutige Zahlen generiert werden, wie es aus Abbildung 18.4 hervorgeht.



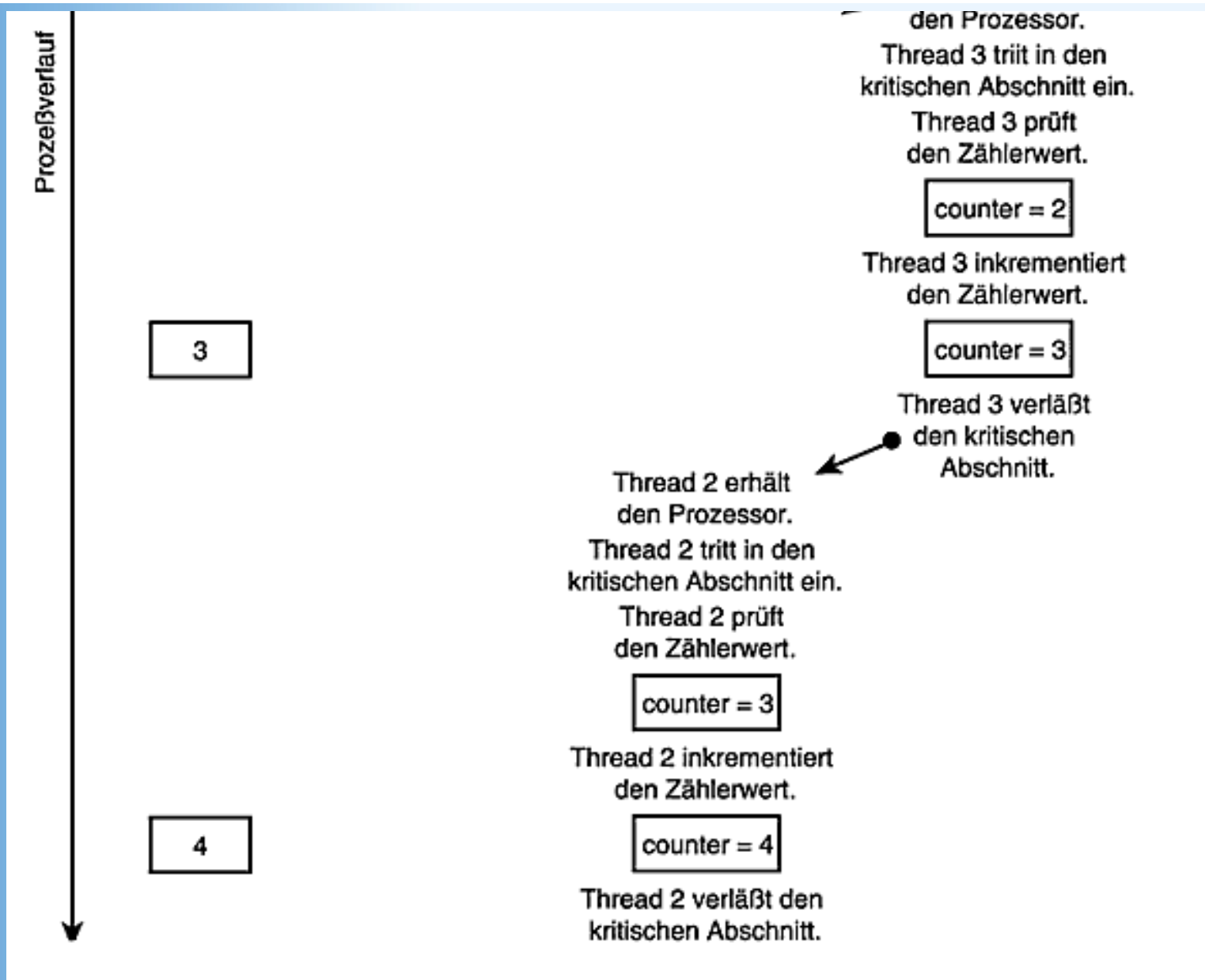


Abbildung 18.4:
Drei Threads, die denselben Zähler verwenden, wobei der Zähler durch einen kritischen Abschnitt geschützt ist.

Wenn Sie ein Objekt mit einem kritischen Abschnitt in Ihrer Anwendung benötigen, erzeugen Sie eine Instanz der Klasse `CCriticalSection`. Dieses Objekt enthält die beiden Methoden `Lock` und `Unlock`, mit denen Sie die Steuerung des kritischen Abschnitts an sich nehmen bzw. freigeben.

Mutexe

Mutexe arbeiten grundsätzlich in der gleichen Weise wie kritische Abschnitte. Allerdings setzt man Mutexe ein, wenn man Ressourcen zwischen mehreren Anwendungen gemeinsam nutzen will. Mit einem Mutex läßt sich garantieren, daß keine zwei Threads, die in einer beliebigen Anzahl von Anwendungen laufen, auf dieselbe Ressource zum selben Zeitpunkt zugreifen.

Aufgrund ihrer systemweiten Verfügbarkeit bringen Mutexe wesentlich mehr Overhead mit als kritische Abschnitte. Die Lebensdauer eines Mutexes endet nicht, wenn die Anwendung, die ihn erzeugt hat, beendet wird. Der Mutex kann weiterhin von anderen Anwendungen verwendet werden. Das Betriebssystem muß demnach verfolgen, welche Anwendungen einen Mutex verwenden, und muß den Mutex zerstören, sobald er nicht mehr benötigt wird. Im Gegensatz dazu bringen kritische Abschnitte nur einen geringen Verwaltungsaufwand mit sich, da sie nicht außerhalb der Anwendung existieren, die

sie erzeugt und verwendet. Nachdem die Anwendung beendet wird, ist der kritische Abschnitt verschwunden.

Wenn Sie einen Mutex in Ihren Anwendungen verwenden müssen, erzeugen Sie eine Instanz der Klasse `CMutex`. Der Konstruktor der Klasse `CMutex` weist drei verfügbare Argumente auf. Das erste Argument ist ein boolescher Wert, der festlegt, ob der Thread, der das `CMutex`-Objekt erzeugt, der anfängliche Besitzer des Mutex ist. In diesem Fall muß dieser Thread den Mutex freigeben, bevor irgendwelche anderen Threads darauf zugreifen können.

Das zweite Argument stellt den Namen für den Mutex dar. Alle Anwendungen, die gemeinsam auf den Mutex zurückgreifen müssen, können ihn nach diesem textuellen Namen identifizieren. Das dritte und letzte Argument an den `CMutex`-Konstruktor ist ein Zeiger auf die Sicherheitsattribute für das Mutex-Objekt.

Übergibt man für diesen Zeiger den Wert `NULL`, verwendet das Mutex-Objekt die Sicherheitsattribute des Threads, der den Mutex erzeugt hat.

Nachdem Sie ein `CMutex`-Objekt erzeugt haben, können Sie es mit den Elementfunktionen `Lock` und `Unlock` sperren bzw. freigeben. Damit haben Sie die Möglichkeit, den Zugriff auf gemeinsam genutzte Ressourcen zwischen mehreren Threads in mehreren Anwendungen zu steuern.

Semaphoren

Die Arbeitsweise von Semaphoren unterscheidet sich grundsätzlich von kritischen Abschnitten und Mutexen. Semaphoren setzt man bei Ressourcen ein, die nicht auf einen einzelnen Thread zu einem Zeitpunkt beschränkt sind - eine Ressource, die auf eine bestimmte Anzahl von Threads beschränkt ist. Vom Prinzip her ist ein Semaphor eine Art Zähler, den die Threads inkrementieren oder dekrementieren können. Der Trick bei Semaphoren besteht darin, daß sie nicht kleiner als Null werden können. Wenn demzufolge ein Thread versucht, einen Semaphor zu dekrementieren, der bereits Null ist, wird dieser Thread blockiert, bis ein anderer Thread den Semaphor inkrementiert.

Nehmen wir an, daß eine Warteschlange von mehreren Threads gefüllt wird und ein Thread die Elemente aus der Warteschlange entfernt und weiterverarbeitet. Wenn die Warteschlange leer ist, hat der Thread, der Elemente entfernt und weiterverarbeitet, nichts zu tun. Dieser Thread kann in eine Leerlaufschleife eintreten, in der die Warteschlange ständig daraufhin untersucht wird, ob sich irgend etwas darin befindet. Das Problem bei diesem Szenario besteht darin, daß der Thread Verarbeitungszeit verbraucht, um eigentlich überhaupt nichts zu tun. Diese Prozessorzeiten könnte man für andere Threads vergeben, die wirklich etwas ausführen müssen. Wenn Sie eine Warteschlange mit Semaphoren steuern, kann jeder Thread, der Elemente in die Warteschlange stellt, den Semaphor für jedes plazierte Element inkrementieren, und der Thread, der die Elemente aus der Warteschlange entfernt, kann den Semaphor dekrementieren, bevor er ein Element aus der Warteschlange nimmt. Wenn die Warteschlange leer ist, hat der Semaphor den Wert Null, und der Thread, der Elemente entfernt, wird beim Aufruf zum Dekrementieren der Warteschlange blockiert. Der Thread verbraucht damit keine Prozessorzeit, bis einer der anderen Threads den Semaphor inkrementiert, um anzuzeigen, daß er ein Element in die Warteschlange gestellt hat. Zu diesem Zeitpunkt wird die Blockierung für den Thread, der Elemente entfernt, unverzüglich aufgehoben, und der Thread kann das eben plazierte Element aus der Warteschlange nehmen und weiterverarbeiten, wie es Abbildung 18.5 zeigt.

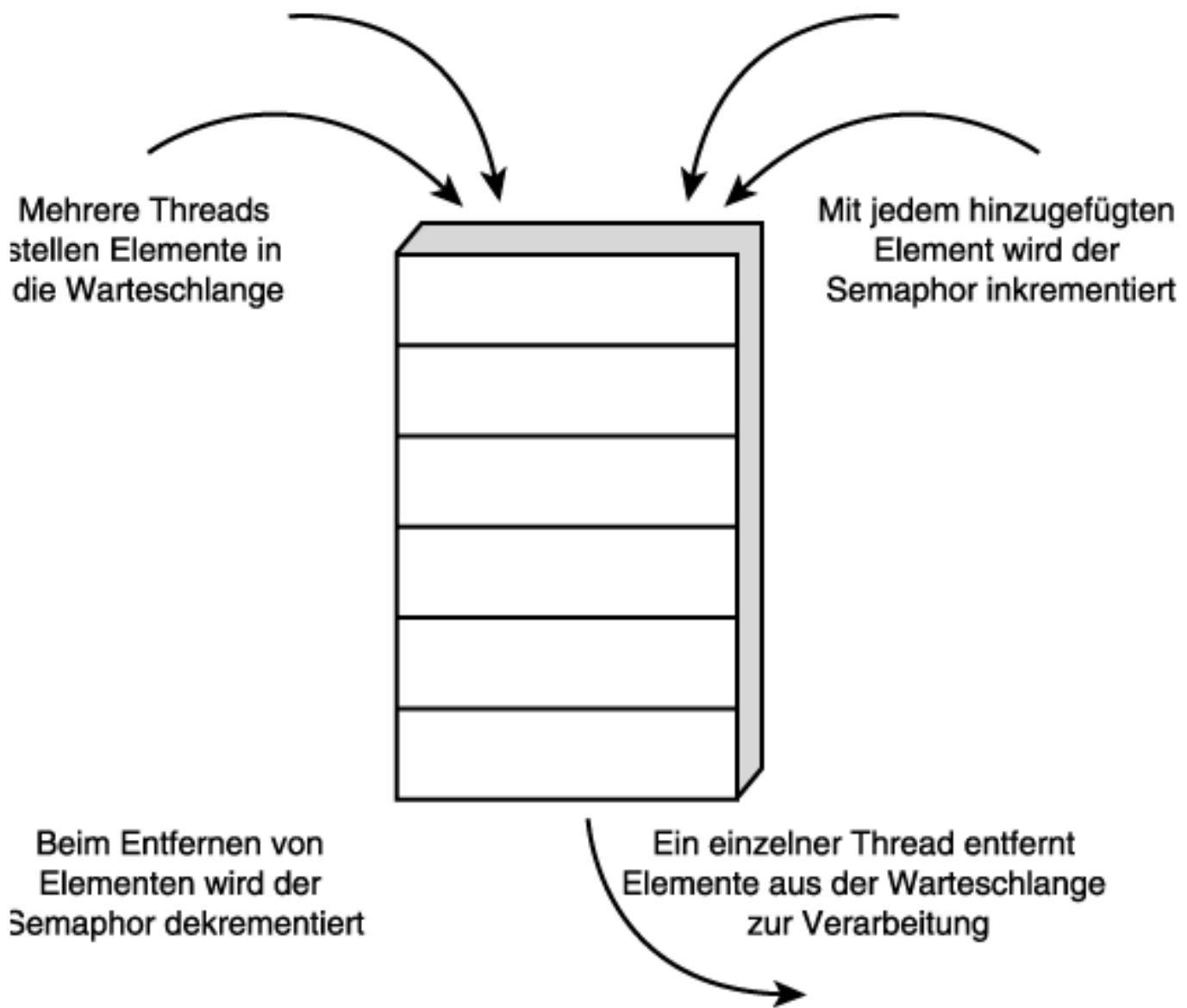


Abbildung 18.5:
Mehrere Threads, die Objekte in eine Warteschlange stellen

Wenn Sie einen Semaphore in Ihrer Anwendung einsetzen müssen, können Sie eine Instanz der Klasse `CSemaphore` erzeugen. Diese Klasse hat vier Argumente, die man an den Konstruktor der Klasse übergeben kann. Das erste Argument gibt den anfänglichen Zählerstand für den Semaphore an. Im zweiten Argument legt man den maximalen Zählerstand für den Semaphore fest. Mit diesen beiden Argumenten läßt sich steuern, wie viele Threads und Prozesse gleichzeitig auf eine gemeinsam genutzte Ressource zugreifen können. Das dritte Argument spezifiziert den Namen für den Semaphore. Dieser Name identifiziert den Semaphore gegenüber allen Anwendungen, die im System laufen, analog zur Klasse `CMutex`. Das letzte Argument ist ein Zeiger auf die Sicherheitsattribute für den Semaphore.

Beim `CSemaphore`-Objekt kann man mit den Elementfunktionen `Lock` und `Unlock` die Steuerung des Semaphors erlangen oder freigeben. Ruft man die Funktion `Lock` auf, wenn der Verwendungszähler des Semaphors größer als Null ist, wird der Verwendungszähler dekrementiert, und Ihr Programm kann weiterlaufen. Ist der Verwendungszähler bereits Null, dann wartet die Funktion `Lock`, bis der Verwendungszähler inkrementiert wird, damit Ihr Prozeß den Zugriff auf die gemeinsam genutzte Ressource erlangen kann. Wenn Sie die Funktion `Unlock` aufrufen, wird der Verwendungszähler des Semaphors inkrementiert.

Ereignisse

In dem Maße wie die Synchronisierungsmechanismen für Threads dafür ausgelegt sind, den Zugriff auf begrenzte Ressourcen zu steuern, sind sie auch dafür vorgesehen, unnötige Prozessorzeit in Threads zu unterbinden. Je mehr Threads gleichzeitig laufen, desto langsamer führt jeder einzelne Thread seine Aufgaben aus. Wenn ein Thread also nichts zu tun hat, blockieren Sie ihn, und belassen Sie ihn im Leerlauf. Damit erhalten andere Threads mehr Prozessorzeit und laufen deshalb schneller, bis die Bedingungen erfüllt sind, daß der im Leerlauf arbeitende Thread etwas zu tun bekommt.

Aus diesem Grund verwendet man Ereignisse - um Threads den Leerlauf zu ermöglichen, bis die Bedingungen erfüllt sind, daß sie etwas zu tun bekommen. Ereignisse sind dem Namen nach mit den Ereignissen verwandt, die die meisten Windows-Anwendungen treiben, nur daß das Ganze etwas komplizierter ist. Die Ereignisse zur Synchronisierung von Threads arbeiten nicht nach den Mechanismen der Ereigniswarteschlangen und -behandlung. Statt eine Nummer zu erhalten und dann darauf zu warten, daß diese Nummer an die Behandlungsroutine von Windows übergeben wird, sind Ereignisse der Thread-Synchronisierung eigentliche Objekte, die sich im Speicher befinden. Jeder Thread, der auf ein Ereignis warten muß, teilt dem Ereignis mit, daß er auf dessen Auslösung wartet, und nimmt dann einen Ruhezustand ein. Wird das Ereignis ausgelöst, geht ein Signal an jeden Thread, der dem Ereignis seinen Wartezustand bekanntgegeben hat. Die Threads nehmen die Verarbeitung genau an dem Punkt auf, wo sie dem Ereignis die Warteabsicht mitgeteilt haben.

Wenn Sie in Ihrer Anwendung mit Ereignissen arbeiten, können Sie ein **CEvent**-Objekt einsetzen. Das **CEvent**-Objekt ist zu erzeugen, wenn Sie auf das Ereignis zugreifen und darauf warten müssen. Sobald der **CEvent**-Konstruktor zurückkehrt, ist das Ereignis aufgetreten, und Ihr Thread kann seine Arbeit weiter fortsetzen.

Der Konstruktor für die **CEvent**-Klasse kann vier Argumente übernehmen. Das erste Argument ist ein boolesches Flag. Es gibt an, ob der Thread das von ihm erzeugte Ereignis anfänglich besitzt. Diesen Wert sollte man auf **TRUE** setzen, wenn der das **CEvent**-Objekt erzeugende Thread auch ermittelt, wann das Ereignis auftritt.

Das zweite Argument an den **CEvent**-Konstruktor legt fest, ob es sich um ein automatisches oder manuelles Ereignis handelt. Ein manuelles Ereignis verbleibt im signalisierten oder nicht signalisierten Zustand, bis es der Thread, der das Ereignisobjekt besitzt, explizit in den anderen Zustand versetzt. Ein automatisches Ereignis verbleibt die meiste Zeit im nicht signalisierten Zustand. Wurde es in den signalisierten Zustand gesetzt und mindestens ein Thread wird freigegeben und setzt seine Arbeit im Ausführungspfad fort, kehrt das Ereignis in den nicht signalisierten Zustand zurück.

Das dritte Argument an den Ereigniskonstruktor ist der Name für das Ereignis. Alle Threads, die auf das Ereignis zugreifen müssen, identifizieren es anhand dieses Namens. Das vierte und letzte Argument ist ein Zeiger auf die Sicherheitsattribute für das Ereignisobjekt.

Die Klasse **CEvent** verfügt über mehrere Member-Funktionen, mit denen Sie den Zustand des Ereignisses steuern können. Tabelle 18.2 listet diese Funktionen auf.

Tabelle 18.2: Member-Funktionen der Klasse CEvent

Funktion	Beschreibung
SetEvent	Versetzt das Ereignis in den signalisierten Zustand.
PulseEvent	Versetzt das Ereignis in den signalisierten Zustand und dann wieder zurück in den nicht signalisierten.
ResetEvent	Versetzt das Ereignis in den nicht signalisierten Zustand.

Eine Multitasking-Anwendung erstellen

Um zu zeigen, wie Sie eigene Multitasking-Anwendungen erstellen können, erzeugen Sie eine Anwendung mit vier sich drehenden Farbfächern, von denen jeder in seinem eigenen Thread läuft. Zwei der Fächer verwenden die Funktion `OnIdle`, während die beiden anderen als unabhängige Threads laufen. Mit dieser Einrichtung können Sie die Unterschiede zwischen beiden Typen des Threadings studieren und lernen, wie man jeden Typ einsetzt. Das Anwendungsfenster erhält vier Kontrollkästchen, um jeden Thread zu starten und anzuhalten. Auf diese Weise läßt sich demonstrieren, wieviel Verarbeitungsleistung das System erbringen muß, wenn jeder Thread allein oder in Verbindung mit den anderen läuft.

Das Anwendungsgerüst

Für die heute zu erstellende Anwendung ist das Gerüst einer SDI-Anwendung erforderlich, bei dem die Ansichtsklasse von der Klasse `CFormView` abgeleitet ist, so daß Sie mit dem Dialog-Editor das Layout der wenigen Steuerelemente im Fenster gestalten können. In der Dokumentklasse sind die Fächer und die unabhängigen Threads realisiert. Die Ansicht enthält die Kontrollkästchen und Variablen, die Aktivität oder Leerlauf der Threads steuern.

Erzeugen Sie mit dem MFC-Anwendungs-Assistenten einen neuen Projekt-Arbeitsbereich. Geben Sie der Anwendung einen passenden Projektnamen wie etwa `Tasking`.

Im Anwendungs-Assistenten wählen Sie die Option **Einzelnes Dokument (SDI)**. Die Vorgaben der meisten anderen Dialogfelder des Anwendungs-Assistenten können Sie übernehmen, obwohl eigentlich keine Unterstützung für ActiveX-Steuerelemente, andockbare Symbolleisten, Statusleiste zu Beginn sowie Drucken und Seitenansicht erforderlich ist. Schalten Sie die betreffenden Kontrollkästchen einfach aus. Im letzten Dialogfeld des Assistenten legen Sie für Ihre Ansichtsklasse (`CTaskingView`) die Basisklasse `CFormView` fest.

Im fertigen Anwendungsgerüst löschen Sie das **ZU ERLEDIGEN**-Textfeld aus dem Hauptfenster der Anwendung und fügen vier Kontrollkästchen etwa in der oberen linken Ecke jedes Quadranten hinzu, wie es Abbildung 18.6 zeigt. Legen Sie die Eigenschaften der Kontrollkästchen gemäß Tabelle 18.3 fest.

Tabelle 18.3: Entwurf des Hauptfensters

Objekt	Eigenschaft	Einstellung
Kontrollkästchen	ID	IDC_CBONIDLE1
	Titel	On &Idle Thread 1
Kontrollkästchen	ID	IDC_CBTHREAD1
	Titel	Thread &1
Kontrollkästchen	ID	IDC_CBONIDLE2
	Titel	On Idle &Thread 2

Kontrollkästchen	ID	IDC_CBTHREAD2
	Titel	Thread &2

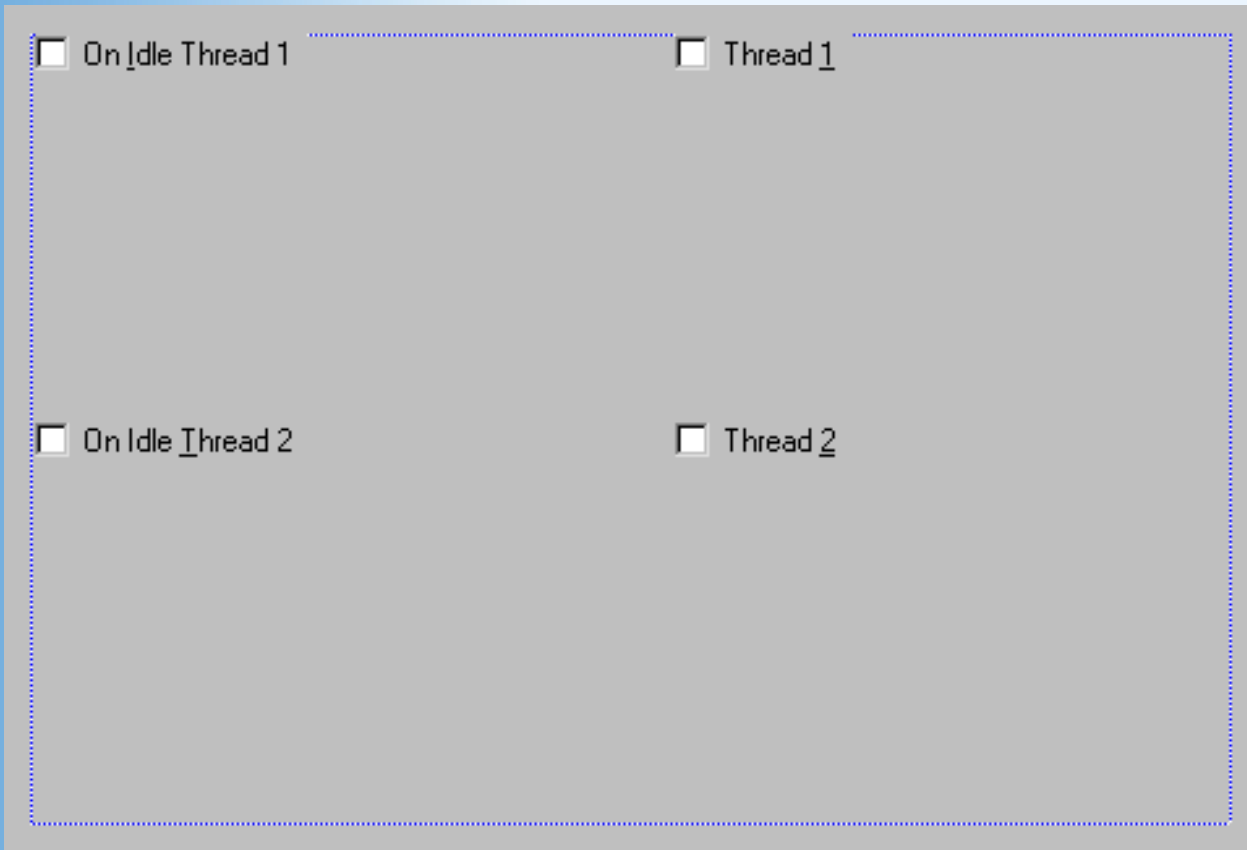


Abbildung 18.6:
Der Entwurf des Hauptfensters

Nachdem Sie die Kontrollkästchen in das Fenster aufgenommen und die Eigenschaften konfiguriert haben, fügen Sie mit dem Klassen-Assistenten für jedes Kontrollkästchen eine Variable hinzu. Namen und Datentyp der Variablen sind in Tabelle 18.4 zusammengefaßt.

Tabelle 18.4: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_CBONIDLE1	m_bOnIdle1	Wert	BOOL
IDC_CBONIDLE2	m_bOnIdle2	Wert	BOOL
IDC_CBTHREAD1	m_bThread1	Wert	BOOL
IDC_CBTHREAD2	m_bThread2	Wert	BOOL

Fächer entwerfen

Bevor Sie Threads in Ihre Anwendung hinzufügen können, erstellen Sie den drehenden Farbfächer, auf dem die Threads operieren. Da sich alle vier Farbfächer unabhängig voneinander drehen, ist es sinnvoll, die gesamte Funktionalität in einer einzigen Klasse zu verkapseln. Diese Klasse verfolgt, welche Farbe gezeichnet wird, wo im Fächer die nächste Linie zu zeichnen ist, welche Größe der Fächer hat und wo sich der Fächer im Anwendungsfenster befindet. Außerdem braucht die Klasse einen Zeiger auf die

Ansichtsklasse, um den Gerätekontext zu holen und sich selbst darin zu zeichnen. Für die unabhängigen Fächer benötigt die Klasse einen Zeiger auf das Flag, das steuert, ob sich der Fächer drehen soll.

Für die Fächerklasse erzeugen Sie eine neue allgemeine Klasse, die von der Basisklasse `CObject` abgeleitet ist. Das Beispiel verwendet für den Namen der neuen Klasse `CSpinner`.

Die Fächervariablen festlegen

Der neu erstellten Klasse für Ihr Fächerobjekt fügen Sie nun einige Variablen hinzu. Gemäß der objektorientierten Richtlinien legen Sie diese Variablen als »Privat« fest und nehmen Methoden in die Klasse auf, um die Werte der Variablen zu setzen und abzurufen.

Die Variablen sind für die folgenden Werte vorgesehen:

- aktuelle Farbe
- aktuelle Position in der Drehung des Farbfächers
- Größe des Farbfächers
- Position des Farbfächers im Anwendungsfenster
- die Farbtabelle, aus der die Farben für das Zeichnen im Farbfächer ausgewählt werden
- ein Zeiger auf das Ansichtsobjekt, damit der Fächer den Gerätekontext für das Zeichnen im Fenster holen kann
- ein Zeiger auf die Variable des Kontrollkästchens, das festlegt, ob der jeweilige Thread laufen soll

Tabelle 18.5 gibt die Namen und Datentypen dieser Variablen an, die Sie alle in die Fächerklasse einfügen.

Tabelle 18.5: Die Variablen der Klasse CSpinner

Name	Typ	Beschreibung
m_crColor	int	Die aktuelle Farbe aus der Farbtabelle.
m_nMinute	int	Die Position bei der Drehung um die Achse.
m_iRadius	int	Der Radius (die Größe) des Fächers.
m_pCenter	CPoint	Der Mittelpunkt des Fächers.
m_crColors[8]	static COLORREF	Die Farbtabelle mit allen Farben, die im Farbfächer zu zeichnen sind.
m_pViewWnd	CWnd*	Ein Zeiger auf das Ansichtsobjekt.
m_bContinue	BOOL*	Ein Zeiger auf die Variable des Kontrollkästchens, die festlegt, ob dieser Thread laufen soll.

Wenn Sie alle erforderlichen Variablen hinzugefügt haben, müssen Sie sicherstellen, daß Ihre Klasse die Variablen entweder initialisiert oder ein geeignetes Instrument bereitstellt, um die Werte der Variablen zu setzen und abzurufen. Die Integer-Variablen können den Anfangswert 0 erhalten. Die Zeiger sollten Sie mit NULL initialisieren. Die Initialisierungen lassen sich insgesamt im Konstruktor der Klasse erledigen, wie es Listing 18.1 zeigt.

Listing 18.1: Der Konstruktor von CSpinner

```
1: CSpinner::CSpinner()
```

```

2: {
3: // Position, Größe und Farbe initialisieren
4: m_iRadius = 0;
5: m_nMinute = 0;
6: m_crColor = 0;
7: // Zeiger auf NULL setzen
8: m_pViewWnd = NULL;
9: m_bContinue = NULL;
10: }

```

Für diejenigen Variablen, die Sie setzen und abrufen müssen, ist Ihre Fächerklasse einfach genug, daß Sie alle diesbezüglichen Funktionen als Inline-Funktionen in der Deklaration der Klasse schreiben können. Farbe und Position werden automatisch durch das Fächerobjekt berechnet, so daß für diese beiden Variablen keine Funktionen für das Setzen erforderlich sind. Die übrigen Variablen (nicht mitgezählt die Farbtabelle) müssen sich aber setzen lassen. Die einzigen Variablen, die Sie aus dem Fächerobjekt abrufen müssen, sind die Zeiger auf die Ansichtsclass und die Variable des Kontrollkästchens. Die betreffenden Funktionen können Sie in die Deklaration der Klasse `CSpinner` aufnehmen, indem Sie die Header-Datei öffnen und die Inline-Funktionen gemäß Listing 18.2 einfügen.

Listing 18.2: Die Klassendeklaration von `CSpinner`

```

1: class CSpinner : public CObject
2: {
3: public:
4: BOOL* GetContinue() {return m_bContinue;}
5: void SetContinue(BOOL* bContinue) { m_bContinue = bContinue;}
6: CWnd* GetViewWnd() { return m_pViewWnd;}
7: void SetViewWnd(CWnd* pWnd) { m_pViewWnd = pWnd;}
8: void SetLength(int iLength) { m_iRadius = iLength;}
9: void SetPoint(CPoint pPoint) { m_pCenter = pPoint;}
10: CSpinner();
11: virtual ~CSpinner();
12:
13: private:
14: BOOL* m_bContinue;
15: CWnd* m_pViewWnd;
16: static COLORREF m_crColors[8];
17: int m_iRadius;
18: CPoint m_pCenter;
19: int m_nMinute;
20: int m_crColor;
21: };

```

Nachdem Sie nun alle Unterstützungsfunktionen für das Setzen und Abrufen der erforderlichen Variablen hinzugefügt haben, müssen Sie die Farbtabelle deklarieren und füllen. Das entspricht in etwa der Definition für die Farbtabelle, die Sie in die Anwendung von Tag 10 aufgenommen haben. Die Farbtabelle besteht aus acht RGB- Werten, wobei jeder Wert entweder 0 oder 255 sein kann und jede Kombination dieser beiden Einstellungen möglich ist. Der beste Platz für diese Tabellendeklaration ist in der Quellcodedatei des Fächers unmittelbar vor dem Konstruktor der Klasse, wie es Listing 18.3 angibt.

Listing 18.3: Die Farbtabelle von `CSpinner`

```

1: static char THIS_FILE[]=__FILE__;
2: #define new DEBUG_NEW
3: #endif
4:
5: COLORREF CSpinner::m_crColors[8] = {
6: RGB( 0, 0, 0), // Schwarz
7: RGB( 0, 0, 255), // Blau
8: RGB( 0, 255, 0), // Grün
9: RGB( 0, 255, 255), // Cyan
10: RGB( 255, 0, 0), // Rot
11: RGB( 255, 0, 255), // Magenta
12: RGB( 255, 255, 0), // Gelb
13: RGB( 255, 255, 255) // Weiß
14: };
15:
16:
17: //////////////////////////////////////
18: // Konstruktion/Destruktion
19: //////////////////////////////////////
20: CSpinner::CSpinner()
21: {
22: // Position, Größe und Farbe initialisieren
23: m_iRadius = 0;
24: .
25: .
26: .

```

Den Fächer zeichnen

Nun kommt der angenehme Teil: das Fächerobjekt tatsächlich in Drehung versetzen. Um dies zu erreichen, berechnen Sie die neue Position für den Start- und Endpunkt jeder Linie, setzen den Ursprungspunkt des Viewports, wählen die Zeichenfarbe und erzeugen einen Zeichenstift in dieser Farbe. Anschließend zeichnen Sie die Linie vom Startpunkt zum Endpunkt. Danach stellen Sie den ursprünglichen Stift - der vor dem Zeichnen der Linie gültig war - wieder her. Vor dem Verlassen der Funktion berechnen Sie noch die Position der nächsten Linie.

Um diese Funktionalität in das Fächerobjekt aufzunehmen, fügen Sie eine Member- Funktion in die Klasse CSpinner ein. Legen Sie den Typ als `void`, den Namen mit `Draw` und den Zugriff als `Public` fest. In die Funktion schreiben Sie den Code aus Listing 18.4.

Listing 18.4: Die Funktion Draw der Klasse CSpinner

```

1: void CSpinner::Draw()
2: {
3: // Zeiger auf Gerätekontext holen
4: CDC *pDC = m_pViewWnd->GetDC();
5: // Abbildungsmodus festlegen
6: pDC->SetMapMode (MM_LOENGLISH);
7: // Mittelpunkt des Fächers kopieren

```

```
8: CPoint org = m_pCenter;
9: CPoint pStartPoint;
10: // Anfangspunkt festlegen
11: pStartPoint.x = (m_iRadius / 2);
12: pStartPoint.y = (m_iRadius / 2);
13: // Ursprung festlegen
14: org.x = m_pCenter.x + (m_iRadius / 2);
15: org.y = m_pCenter.y + m_iRadius;
16: // Ursprung des View-Ports festlegen
17: pDC->SetViewportOrg(org.x, org.y);
18:
19: CPoint pEndPoint;
20: // Winkel zur nächsten Linie berechnen
21: double nRadians = (double) (m_nMinute * 6) * 0.017453292;
22: // Endpunkt der Linie festlegen
23: pEndPoint.x = (int) (m_iRadius * sin(nRadians));
24: pEndPoint.y = (int) (m_iRadius * cos(nRadians));
25:
26:
27: // Zeichenstift erzeugen
28: CPen pen(PS_SOLID, 0, m_crColors[m_crColor]);
29: // Stift auswählen
30: CPen* pOldPen = pDC->SelectObject(&pen);
31:
32: // Zum Anfangspunkt gehen
33: pDC->MoveTo (pEndPoint);
34: // Linie zum Endpunkt ziehen
35: pDC->LineTo (pStartPoint);
36:
37: // Ursprünglichen Stift wiederherstellen
38: pDC->SelectObject(&pOldPen);
39:
40: // Gerätekontext freigeben
41: m_pViewWnd->ReleaseDC(pDC);
42:
43: // Minute (Winkel) inkrementieren
44: if (++m_nMinute == 60)
45: {
46: // Nach vollem Umlauf Minute auf 0 zurücksetzen
47: m_nMinute = 0;
48: // Farbe inkrementieren
49: if (++m_crColor == 8)
50: // Wenn alle Farben durch, von vorn beginnen
51: m_crColor = 0;
52: }
53: }
```

Eine ganze Menge Code! Was bewirkt er? Um zu verstehen, was diese Funktion realisiert und wie der sich drehende Farbfächer in das Fenster gelangt, sehen wir uns den Code näher an.

Damit die verschiedenen Threads den Fächer effektiv nutzen können, zeichnet die Funktion bei jedem Aufruf nur jeweils eine Linie. Für einen vollständigen Kreis ist diese Funktion 60mal aufzurufen, einmal für jede »Minute« bei Drehung im Uhrzeigersinn. Nach jeder kompletten Umdrehung schaltet der Fächer zur nächsten Farbe in der Farbtabelle.

Bevor man überhaupt in das Fenster zeichnen kann, muß man zunächst den Gerätekontext des Fensters holen. Dazu ruft man die Funktion `GetDC` auf dem Zeiger des Ansichtsspekts auf:

```
CDC *pDC = m_pViewWnd->GetDC();
```

Die Funktion liefert einen Zeiger auf ein CDC-Objekt zurück, d.h. auf eine MFC-Klasse, die den Gerätekontext verkapselt.

Nachdem Sie über einen Zeiger auf den Gerätekontext verfügen, können Sie die Member-Funktion `SetMapMode` aufrufen, um den Abbildungsmodus festzulegen:

```
pDC->SetMapMode (MM_LOENGLISH);
```

Der Abbildungsmodus bestimmt, wie die x- und y-Koordinaten in Positionen auf dem Bildschirm übersetzt werden. Der Modus `MM_LOENGLISH` bildet eine logische Einheit auf 0,01 Zoll auf dem Bildschirm ab. Es sind verschiedene Abbildungsmodi verfügbar, die jeweils logische Einheiten in bestimmte Maßeinheiten auf dem Bildschirm konvertieren.

Jetzt wenden Sie sich den Vorbereitungen zu, um die aktuelle Linie für den Farbfächer zu zeichnen. Zuerst berechnen Sie den Anfangspunkt für die zu zeichnende Linie. Dieser Punkt ist für alle vom Fächerobjekt gezeichneten Linien gleich. Anschließend berechnen Sie die Position des Viewports. Der Viewport dient als Ausgangspunkt für das Koordinatensystem, in dem die Zeichnung liegt.



Der Viewport ist der rechteckige Bereich eines Formulars, das im Container des Formulars angezeigt wird - mit anderen Worten ein Grafikfenster.



Der Anfangspunkt der zu zeichnenden Linie wird für eine Position außerhalb des Mittelpunktes berechnet. Wenn der Ursprung für die Linien genau im Mittelpunkt des Fächers liegen soll, setzen Sie sowohl die x- als auch die y-Koordinate des Anfangspunktes auf 0.

Nachdem der Ursprung des Viewports berechnet ist, setzen Sie den Viewport mit der Funktion `SetViewportOrg`:

```
pDC->SetViewportOrg(org.x, org.y);
```

Der Zeichenbereich und der Startpunkt für die zu zeichnende Linie sind nun definiert. Als nächstes ist der Endpunkt der Linie zu bestimmen. Diese Berechnung steht in den folgenden drei Codezeilen:

```
double nRadians = (double) (m_nMinute * 6) * 0.017453292;
pEndPoint.x = (int) (m_iRadius * sin(nRadians));
pEndPoint.y = (int) (m_iRadius * cos(nRadians));
```

Die erste Zeile konvertiert die Minuten (nicht zu verwechseln mit Bogenminuten) in Grad. Das Ergebnis läßt sich dann in die Sinus- und Kosinusfunktionen einsetzen, um die x- und y-Koordinaten für das Zeichnen eines Kreises festzulegen. Daraus ergibt sich der Endpunkt der zu zeichnenden Linie.

Nunmehr verfügen Sie über den Anfangs- und Endpunkt der Linie und können einen Stift erzeugen, der die Linie zeichnet:


```
CPen pen(PS_SOLID, 0, m_crColors[m_crColor]);
```

Diese Anweisung legt fest, daß der Stift durchgehend und dünn sein soll und die aktuelle Farbe aus der Farbtabelle auszuwählen ist. Nachdem der zu verwendende Stift erzeugt wurde, selektieren Sie den Stift zum Zeichnen. Dabei übernehmen Sie den aktuellen Stift als Rückgabewert vom Gerätekontextobjekt:

```
CPen* pOldPen = pDC->SelectObject(&pen);
```

Jetzt läßt sich die Linie zeichnen. Dazu kommen die Funktionen `MoveTo` und `LineTo` zum Einsatz, die Ihnen mittlerweile vertraut sein dürften. Nach dem Zeichnen der Linie geben Sie den Gerätekontext wieder frei, damit kein Ressourcenmangel in der Anwendung auftritt:

```
m_pViewWnd->ReleaseDC(pDC);
```

Damit ist die Linie fertiggestellt. Jetzt inkrementieren Sie noch den Minutenzähler. Falls der Kreis geschlossen ist, setzen Sie den Zähler auf 0 zurück. Nach jedem vollständigen Kreis inkrementieren Sie den Farbenzähler und setzen ihn nach der achten Farbe wieder auf 0.

Damit Sie in dieser Funktion auf die trigonometrischen Funktionen zurückgreifen können, binden Sie die Header-Datei `math.h` in die Quelldatei der Fächerklasse (`Spinner.cpp`) ein. Gehen Sie dazu an den Beginn der Quelldatei, und nehmen Sie die `#include`-Zeile mit der Header-Datei `math.h` gemäß Listing 18.5 auf.

Listing 18.5: Die Quelldatei von CSpinner

```
1: // Spinner.cpp: Implementierung der Klasse CSpinner.
2: //
3:
4: //////////////////////////////////////
5: #include "stdafx.h"
6: #include <math.h>
7: #include "Tasking.h"
8: #include "Spinner.h"
```

Die Fächer unterstützen

Sie haben nun die Fächerklasse erzeugt, um den sich drehenden Farbfächer im Fenster zu zeichnen. Für die Fächer sind noch verschiedene Unterstützungsroutinen erforderlich. Man kann zwar der Dokumentklasse ein Feld hinzufügen, das die vier Fächer aufnimmt, trotzdem muß man noch berechnen, wo jeder Fächer im Anwendungsfenster erscheinen soll. Außerdem sind die Variablen für jeden Fächer festzulegen.

Diesen Code bringen Sie in der Dokumentklasse unter, wobei Sie mit dem Feld der Fächer beginnen. Fügen Sie der Dokumentklasse (im Beispiel `CTaskingDoc`) eine Member-Variable hinzu, für die Sie den Typ mit `CSpinner`, den Namen als `m_cSpin[4]` und den Zugriff als *Privat* festlegen. Dann öffnen Sie den Quellcode der Dokumentklasse und binden die Header-Datei für den Fächer ein, wie es Listing 18.6 zeigt.

Listing 18.6: Die Quelldatei von CTaskingDoc

```
1: // TaskingDoc.cpp : Implementierung der Klasse CTaskingDoc
2: //
3:
4: #include "stdafx.h"
```



```

5: #include "Tasking.h"
6:
7: #include "Spinner.h"
8: #include "TaskingDoc.h"
9: #include "TaskingView.h"
10: .
11: .
12: .

```

Die Fächerpositionen berechnen

Zu den vorbereitenden Maßnahmen in der Initialisierungsphase der Anwendung gehört es, die Positionen aller vier Fächer zu bestimmen. Das Fenster ist durch die Kontrollkästchen, die die Fächer-Threads ein- und ausschalten, in vier etwa gleich große Quadranten unterteilt, so daß es naheliegt, den Fensterbereich in vier Rechtecke aufzuteilen und in jedem Rechteck einen Fächer unterzubringen.

Die Lage der einzelnen Fächer läßt sich am einfachsten berechnen, indem man eine Funktion erzeugt, die die Position für einen Fächer berechnet und den Fächer im Quadranten der jeweiligen Fächernummer plaziert. Übergibt man der Funktion einen Zeiger auf das Fächerobjekt, kann sie das Fächerobjekt direkt an der Position aktualisieren.

Fügen Sie zu diesem Zweck eine neue Elementfunktion in die Dokumentklasse (im Beispiel die Klasse `CTaskingDoc`) ein. Legen Sie den Funktionstyp als `void`, die Deklaration mit `CalcPoint(int nID, CSpinner *pSpin)` und den Zugriff mit *Privat* fest. Nehmen Sie in die Funktion den Code aus Listing 18.7 auf.

Listing 18.7: Die Funktion `CalcPoint` der Klasse `CTaskingDoc`

```

1: void CTaskingDoc::CalcPoint(int nID, CSpinner *pSpin)
2: {
3:     RECT lWndRect;
4:     CPoint pPos;
5:     int iLength;
6:     CTaskingView *pWnd;
7:
8:     // Zeiger auf das Anzeigefenster holen
9:     pWnd = (CTaskingView*)pSpin->GetViewWnd();
10:    // Rechteck des Anzeigebereichs ermitteln
11:    pWnd->GetClientRect(&lWndRect);
12:    // Größe des Fächers berechnen
13:    iLength = lWndRect.right / 6;
14:    // Welcher Fächer wird plaziert?
15:    switch (nID)
16:    {
17:    case 0: // Position des ersten Fächers
18:        pPos.x = (lWndRect.right / 4) - iLength;
19:        pPos.y = (lWndRect.bottom / 4) - iLength;
20:        break;
21:    case 1: // Position des zweiten Fächers
22:        pPos.x = ((lWndRect.right / 4) * 3) - iLength;
23:        pPos.y = (lWndRect.bottom / 4) - iLength;

```

```

24: break;
25: case 2: // Position des dritten Fächers
26: pPos.x = (lWndRect.right / 4) - iLength;
27: pPos.y = ((lWndRect.bottom / 4) * 3) - (iLength * 1.25);
28: break;
29: case 3: // Position des vierten Fächers
30: pPos.x = ((lWndRect.right / 4) * 3) - iLength;
31: pPos.y = ((lWndRect.bottom / 4) * 3) - (iLength * 1.25);
32: break;
33: }
34: // Größe des Fächers setzen
35: pSpin->SetLength(iLength);
36: // Lage des Fächers setzen
37: pSpin->SetPoint(pPos);
38: }

```

Diese Funktion ermittelt zuerst den Zeiger auf das Ansichtsfenster vom Fächerobjekt durch den Aufruf der Funktion `GetViewWnd`:

```
pWnd = (CTaskingView*)pSpin->GetViewWnd();
```

Indem man den Zeiger direkt vom Fächerobjekt ermittelt, spart man ein paar Schritte ein, um zu diesen Informationen zu gelangen.

Nachdem der Zeiger auf das Ansichtsobjekt vorhanden ist, kann man die Funktion `GetClientRect` des Fensters aufrufen, um die Größe des verfügbaren Zeichenbereichs zu ermitteln:

```
pWnd->GetClientRect(&lWndRect);
```

Aus der Größe des verfügbaren Zeichenbereichs lässt sich eine sinnvolle Größe für den Farbfächer berechnen, indem man die Länge des Zeichenbereichs durch 6 teilt:

```
iLength = lWndRect.right / 6;
```

Teilt man den Zeichenbereich durch 4, erhält man eine Position in der Mitte des oberen linken Quadranten. Von diesem Punkt zieht man die Größe des Kreises ab, und man erhält die obere linke Ecke des Zeichenbereichs für den ersten Fächer:

```

pPos.x = (lWndRect.right / 4) - iLength;
pPos.y = (lWndRect.bottom / 4) - iLength;

```

Von diesem Punkt ausgehend erhält man die anderen Positionen, indem man den Mittelpunkt des Quadranten mit 3 multipliziert, um ihn in die Mitte des rechts daneben oder darunterliegenden Quadranten zu verschieben. Die Positionen der anderen drei Fächer lassen sich analog berechnen.

Nachdem die Länge und die Lage für den Fächer feststehen, übergibt man mit den Funktionen `SetLength` und `SetPoint` die ermittelten Werte an den Fächer, für den sie berechnet wurden:

```

pSpin->SetLength(iLength);
pSpin->SetPoint(pPos);

```

Die Fächer initialisieren

Da Sie die obige Funktion, mit der Sie die Lage jedes Fächers im Fenster berechnen, geschrieben haben, um bei einem Aufruf der Funktion nur mit jeweils einem Fächer zu arbeiten, braucht man eine Routine, die jeden Fächer initialisiert und dabei die obige Funktion einmal für jeden Fächer aufruft. Diese

Funktion ist erforderlich, um einen Zeiger auf das Ansichtsobjekt zu ermitteln und den Zeiger dann an den Fächer weiterzureichen. Außerdem sind Zeiger auf die Variablen der Kontrollkästchen für die Fächer zu ermitteln, die von den unabhängig laufenden Threads verwendet werden. Der Code durchläuft dazu eine Schleife durch das Fächerfeld, legt dabei die Zeiger für jeden Fächer fest und übergibt dann den Fächer an die obige Funktion.

Fügen Sie zu diesem Zweck eine neue Member-Funktion in die Dokumentklasse (CTaskingDoc) ein. Legen Sie den Typ als `void` und die Deklaration mit `InitSpinners` fest. Als Zugriffsstatus wählen Sie die Option *Privat*, da die Funktion nur einmal aufzurufen ist, wenn die Anwendung startet. Nehmen Sie in die neue Funktion den Code aus Listing 18.8 auf.

Listing 18.8: Die Funktion `InitSpinners` der Klasse `CTaskingDoc`

```

1: void CTaskingDoc::InitSpinners()
2: {
3:     int i;
4:
5:     // Position der Ansicht holen
6:     POSITION pos = GetFirstViewPosition();
7:     // Ist Position gültig?
8:     if (pos != NULL)
9:     {
10:        // Zeiger auf die Ansicht holen
11:        CView* pView = GetNextView(pos);
12:
13:        // Schleife durch die Fächer
14:        for (i = 0; i < 4; i++)
15:        {
16:            // Zeiger auf die Ansicht setzen
17:            m_cSpin[i].SetViewWnd(pView);
18:            // Zeiger auf Fortsetzungskennzeichner initialisieren
19:            m_cSpin[i].SetContinue(NULL);
20:            switch (i)
21:            {
22:                case 1: // Zeiger auf Fortsetzungskennzeichner
23:                    // des ersten Threads setzen
24:                    m_cSpin[i].SetContinue(&((CTaskingView*)pView)->m_bThread1);
25:                    break;
26:                case 3: // Zeiger auf zweiten Fortsetzungs-
27:                    // kennzeichner setzen
28:                    m_cSpin[i].SetContinue(&((CTaskingView*)pView)->m_bThread2);
29:                    break;
30:            }
31:            // Lage des Fächers berechnen
32:            CalcPoint(i, &m_cSpin[i]);
33:        }
34:    }
35: }

```

Die Funktion durchläuft zuerst die Schritte, um einen Zeiger auf die Ansichtsclass vom Dokument zu ermitteln, wie Sie es zu Beginn von Tag 10 kennengelernt haben. Nachdem Sie einen gültigen Zeiger auf die Ansicht besitzen, starten Sie eine Schleife, um alle Fächer im Feld zu initialisieren. Sie rufen die

Fächerfunktion `SetViewWnd` auf, um den Zeiger des Fächers auf das Ansichtsfenster zu setzen, und initialisieren dann den Zeiger auf die Variable für das Kontrollkästchen des Fächers mit `NULL` für alle Fächer. Gehört der Fächer zu den beiden, die unabhängige Threads verwenden, übergeben Sie einen Zeiger auf die entsprechende Variable des Kontrollkästchens. Nachdem das alles eingerichtet ist, rufen Sie die Funktion `CalcPoint` auf, die Sie gerade eben erstellt haben, um die Position des Fächers im Ansichtsfenster zu berechnen.



Auch wenn Sie schon mehrere Beispiele zu Zeigern gesehen haben, verdient die Art und Weise der Übergabe eines Zeigers auf die Variable des Kontrollkästchens für den Fächer eine genauere Betrachtung:

```
m_cSpin[i].SetContinue(&((CTaskingView*)pView)->m_bThread1);
```

In dieser Anweisung nehmen Sie den Zeiger auf das Ansichtsobjekt `pView`, das einen Zeiger auf ein `CView`-Objekt darstellt, und führen eine Typumwandlung in einen Zeiger auf die spezielle Ansichtsklasse durch, die Sie in Ihrer Anwendung erstellt haben:

```
(CTaskingView*)pView
```

Da Sie nun den Zeiger auf das Ansichtsobjekt als `CTaskingView`-Objekt behandeln, können Sie zur Variablen `m_bThread1` des Kontrollkästchens gelangen, die ein öffentliches Element der Klasse `CTaskingView` ist:

```
((CTaskingView*)pView)->m_bThread1
```

Nunmehr haben Sie Zugriff auf die Variable `m_bThread1` und können die Adresse dieser Variablen ermitteln, indem Sie ein kaufmännisches Und-Zeichen vor die gesamte Anweisung stellen:

```
&((CTaskingView*)pView)->m_bThread1
```

Wenn Sie diese Adresse für die Variable `m_bThread1` an die Funktion `SetContinue` übergeben, liefern Sie praktisch einen Zeiger auf die Variable `m_bThread1`. Damit kann man schließlich den Zeiger auf diese Variable, die das Fächerobjekt enthält, setzen.

Sie haben nun die Routinen erstellt, um alle Fächer zu initialisieren. Jetzt müssen Sie gewährleisten, daß diese Routine beim Start der Anwendung aufgerufen wird. Es liegt auf der Hand, daß man diese Logik in der Funktion `OnNewDocument` in der Dokumentklasse unterbringt. Diese Funktion wird aufgerufen, wenn die Anwendung startet. Somit ist es logisch, die Initialisierung der Fächerobjekte von dieser Stelle auszulösen. Der Code, den Sie in die Funktion `OnNewDocument` in der Dokumentklasse einfügen, ist in Listing 18.9 wiedergegeben.

Listing 18.9: Die Funktion `OnNewDocument` der Klasse `CTaskingDoc`

```
1: BOOL CTaskingDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
```

```

7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: //////////////////////////////////
10: // EIGENER CODE, ANFANG
11: //////////////////////////////////
12:
13: // Die Fächer initialisieren
14: InitSpinners();
15:
16: //////////////////////////////////
17: // EIGENER CODE, ENDE
18: //////////////////////////////////
19:
20: return TRUE;
21: }

```

Den Fächer drehen

Eines der letzten Dinge, die Sie fürs erste in die Dokumentklasse aufnehmen, ist ein Instrument, das den Aufruf der Funktion `Draw` für einen bestimmten Fächer von außerhalb der Dokumentklasse erlaubt. Da das Fächerfeld als `private` Variable deklariert ist, können Objekte von außerhalb nicht auf die Fächer zugreifen. Daher sind Mechanismen für derartige Zugriffe vorzusehen. Mit einer neuen Member-Funktion in der Dokumentklasse kann man diesen Zugriff bereitstellen. Legen Sie für die Funktion den Typ `void` fest, spezifizieren Sie die Funktionsdeklaration mit einem Namen und einem einzelnen Integer-Argument für die Fächernummer wie etwa `DoSpin(int nIndex)`, und wählen Sie als Zugriffsstatus die Option *Public*. In den Funktionsrumpf schreiben Sie den Code aus Listing 18.10, um den eigentlichen Aufruf des angegebenen Fächers zu realisieren.

Listing 18.10: Die Funktion `DoSpin` der Klasse `CTaskingDoc`

```

1: void CTaskingDoc::DoSpin(int nIndex)
2: {
3: // Den Fächer drehen
4: m_cSpin[nIndex].Draw();
5: }

```

Die `OnIdle`-Tasks hinzufügen

Nachdem Sie nun die unterstützende Funktionalität unter Dach und Fach haben, ist es an der Zeit, Ihre Aufmerksamkeit den verschiedenen Threads zu widmen, die für die Drehung der einzelnen Fächer verantwortlich sind. Als erstes fügen Sie die Threads hinzu, die im Leerlauf der Anwendung ausgeführt werden. Für die zwei Kontrollkästchen sehen Sie eine Behandlungsroutine für das Klickereignis vor, so daß die Variablen für diese beiden Kontrollkästchen mit dem Fenster synchron bleiben. Weiterhin schreiben Sie Code in die `OnIdle`-Funktion der Anwendung, um diese beiden Fächer zu starten, wenn sich die Anwendung im Leerlauf befindet und die Kontrollkästchen für diese beiden Fächer-Threads eingeschaltet sind.



Der Begriff *Thread* in den vorstehenden Erläuterungen ist etwas irreführend. Die Funktionalität, die Sie in die `OnIdle`-Funktion aufnehmen, läuft im Haupt-Thread der Anwendung ab. Die gesamte `OnIdle`-Verarbeitung, die Sie der Beispielanwendung hinzufügen, läuft nicht als unabhängiger Thread, sondern besteht eigentlich nur aus Funktionen, die sich über den Haupt-Thread aufrufen lassen.

Die OnIdle-Tasks starten und anhalten

Die Funktion `OnIdle` prüft die Werte der beiden Variablen für die Kontrollkästchen, die festlegen, ob die jeweilige Task laufen soll. Die Anwendung muß also nur dafür Sorge tragen, daß die Variablen im Ansichtsobjekt mit den Steuerelementen im Fenster synchronisiert werden, wenn man auf die Kontrollkästchen klickt. Dazu ist lediglich die Funktion `UpdateData` aufzurufen. Um die `OnIdle`-Tasks starten und anhalten zu können, muß man nur eine einzige Behandlungsroutine für beide *On Idle Thread*- Kontrollkästchen hinzufügen und dann die Funktion `UpdateData` in dieser Behandlungsroutine aufrufen.

Öffnen Sie dazu den Klassen-Assistenten, und wählen Sie die Ansichtsklasse (in diesem Fall `CTaskingView`) aus. Markieren Sie eines der *On Idle*-Kontrollkästchen, und fügen Sie eine Funktion für das Ereignis `BN_CLICKED` hinzu. Ändern Sie den vorgeschlagenen Namen der Funktion in `OnCbonidle`, und klicken Sie auf OK. Wiederholen Sie das gleiche für das andere *On Idle*-Kontrollkästchen. Damit haben Sie festgelegt, daß beide Ereignisse denselben Code verwenden. Klicken Sie auf **Code bearbeiten**, und fügen Sie den Code aus Listing 18.11 in die Funktion ein.

Listing 18.11: Die Funktion `OnCbonidle` der Klasse `CTaskingView`

```
1: void CTaskingView::OnCbonidle()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Variablen mit Dialogfeld synchronisieren
10: UpdateData(TRUE);
11:
12: //////////////////////////////////////
13: // EIGENER CODE, ENDE
14: //////////////////////////////////////
15: }
```

Die OnIdle-Threads erstellen

Wenn Sie den Quellcode der Anwendungsklasse (`CTaskingApp`) untersuchen, fällt Ihnen sicherlich auf, daß es hier keine `OnIdle`-Funktion gibt. Die ganze Funktionalität, die die `OnIdle`-Funktion per Vorgabe ausführen muß, befindet sich in der Basisklasse der Anwendungsklasse, die für Ihr Projekt erzeugt wurde. Der einzige Grund, eine `OnIdle`-Funktion in Ihre Anwendungsklasse aufzunehmen, besteht darin, daß Ihre Anwendung eine bestimmte Funktionalität während dieses Ereignisses realisieren muß. Im Ergebnis müssen Sie diese Behandlungsroutine explizit in Ihre Anwendung mit Hilfe des

Klassen-Assistenten einbinden.

Nachdem Sie die `OnIdle`-Funktion in Ihre Anwendungsklasse aufgenommen haben, was muß sie tun? Zuerst ist ein Zeiger auf die Ansicht zu ermitteln, damit sich der Zustand der Variablen für die Kontrollkästchen prüfen läßt. Als nächstes muß die Funktion einen Zeiger auf die Dokumentklasse holen, damit der Aufruf der Funktion `DoSpin` möglich ist, um das entsprechende Fächerobjekt auszulösen. Der Schlüssel für beide Aktionen liegt darin, Zeiger auf jedes dieser Objekte zu ermitteln. Wenn Sie sich damit beschäftigen, auf welche Weise diese Zeiger zu erhalten sind, stellen Sie fest, daß die angegebene Reihenfolge umzukehren ist. Sie müssen einen Zeiger auf das Dokumentobjekt holen, damit Sie einen Zeiger auf die Ansicht erhalten. Um jedoch einen Zeiger auf das Dokument zu erhalten, muß man durch die Dokumentvorlage gehen und einen Zeiger auf die Vorlage holen, bevor man einen Zeiger auf das Dokument erhalten kann. Jeder dieser Schritte erfordert die gleiche Folge von Ereignissen - zuerst die Position des ersten Objekts holen und dann einen Zeiger auf das Objekt auf dieser Position ermitteln. Es ist also folgendes zu erledigen: Die Position der ersten Dokumentvorlage ermitteln und dann einen Zeiger auf die Dokumentvorlage an dieser Position holen. Als nächstes nehmen Sie die Dokumentvorlage, um die Position des ersten Dokuments zu ermitteln, und dann verwenden Sie die Dokumentvorlage, um einen Zeiger auf das Dokument an dieser ersten Position zu holen. Schließlich verwenden Sie das Dokument, um die Position der ersten Ansicht zu ermitteln, und verwenden dann erneut das Dokument, um einen Zeiger auf die Ansicht an der angegebenen Position zu holen. Nachdem Sie einen Zeiger auf die Ansicht besitzen, können Sie den Wert der Kontrollkästchen testen und den betreffenden Fächer aufrufen.

Um diese Funktionalität in Ihre Anwendung einzubauen, nehmen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `OnIdle` in die Anwendungsklasse (in diesem Fall `CTaskingApp`) auf. Anschließend klicken Sie auf die Schaltfläche **Code bearbeiten** und fügen den Code von Listing 18.12 ein.

Listing 18.12: Die Funktion `OnIdle` der Klasse `CTaskingApp`

```

1: BOOL CTaskingApp::OnIdle(LONG lCount)
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
aufrufen
4:
5: // Position der ersten Dokumentvorlage holen
6: POSITION pos = GetFirstDocTemplatePosition();
7: // Ist Position der Dokumentvorlage gültig?
8: if (pos)
9: {
10: // Zeiger auf die Dokumentvorlage ermitteln
11: CDocTemplate* pDocTemp = GetNextDocTemplate(pos);
12: // Ist Zeiger gültig?
13: if (pDocTemp)
14: {
15: // Position des ersten Dokuments holen
16: POSITION dPos = pDocTemp->GetFirstDocPosition();
17: // Ist Position des Dokuments gültig?
18: if (dPos)
19: {
20: // Zeiger auf das Dokument ermitteln
21: CTaskingDoc* pDocWnd =
22: (CTaskingDoc*)pDocTemp->GetNextDoc(dPos);

```

```
23: // Ist Zeiger gültig?
24: if (pDocWnd)
25: {
26: // Position der Ansicht ermitteln
27: POSITION vPos = pDocWnd->GetFirstViewPosition();
28: // Ist Ansichtsposition gültig?
29: if (vPos)
30: {
31: // Zeiger auf die Ansicht holen
32: CTaskingView* pView = (CTaskingView*)pDocWnd->
    ^GetNextView(vPos);
33: // Ist Zeiger gültig?
34: if (pView)
35: {
36: // Fächer für ersten Leerlauf-Thread drehen?
37: if (pView->m_bOnIdle1)
38: // Ersten Leerlauf-Thread drehen
39: pDocWnd->DoSpin(0);
40: // Zweiten Leerlauf-Thread drehen?
41: if (pView->m_bOnIdle2)
42: // Zweiten Leerlauf-Thread drehen
43: pDocWnd->DoSpin(2);
44: }
45: }
46: }
47: }
48: }
49: }
50:
51: // Leerlaufverarbeitung der Basisklasse aufrufen
52: return CWinApp::OnIdle(lCount);
53: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollten Sie die Kontrollkästchen `On Idle Thread` einschalten und Farbfächer wie in Abbildung 18.7 zeichnen können, solange Sie die Maus bewegen. In dem Moment aber, in dem Sie die Anwendung im Leerlauf - ohne Mausbewegung oder irgend etwas anderes - arbeiten lassen, dreht sich der Fächer nicht mehr.

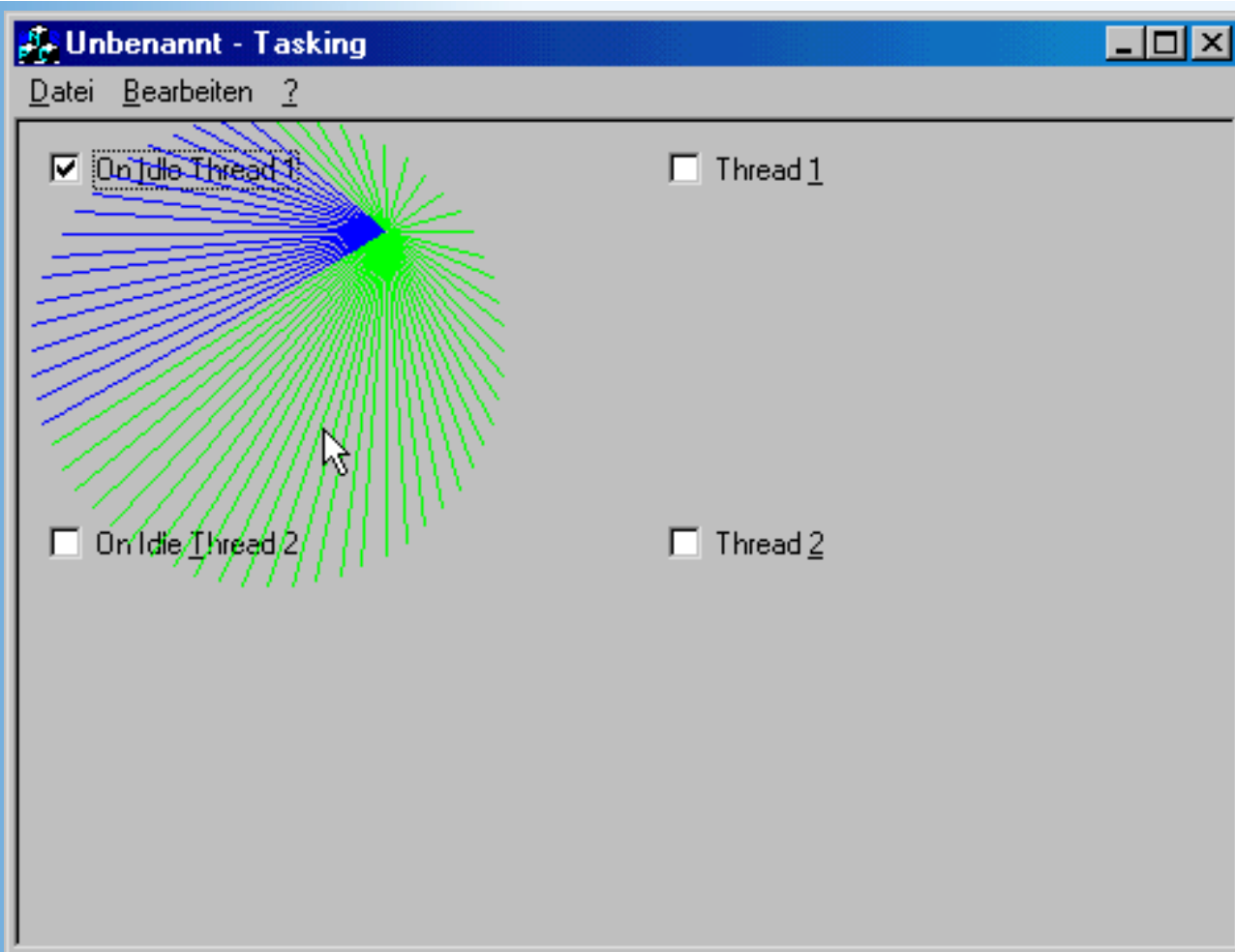


Abbildung 18.7:
Ein OnIdle-Thread, der einen Farbfächer zeichnet

Fortlaufend arbeitende OnIdle-Tasks

Es ist nicht gerade sehr praktisch, die Maus in Bewegung zu halten, damit die Anwendung die Aufgaben, die eigentlich im Leerlauf stattfinden sollen, fortlaufend ausführt. Es muß eine Möglichkeit geben, die Funktion `OnIdle` aufzurufen, solange sich die Anwendung im Leerlauf befindet. Natürlich gibt es die. In der letzten Zeile der `OnIdle`-Funktion ist zu sehen, daß die Funktion `OnIdle` den Ergebniswert der `OnIdle`-Funktion aus der Basisklasse zurückgibt. Und diese Funktion liefert den Wert `FALSE`, sobald keine Leerlaufaktivitäten mehr auszuführen sind.

Man muß also erreichen, daß die Funktion `OnIdle` immer den Wert `TRUE` zurückliefert. Damit wird die Funktion `OnIdle` fortlaufend aufgerufen, wann immer die Anwendung im Leerlauf arbeitet. Wenn Sie den Aufruf der `OnIdle`-Funktion der Basisklasse - wie in Listing 18.13 angegeben - in den ersten Teil der Funktion verschieben und dann `TRUE` zurückgeben, dreht sich der Fächer ununterbrochen, unabhängig davon, wie lange die Anwendung im Leerlauf arbeitet.

Listing 18.13: Die modifizierte Funktion `OnIdle` der Klasse `CTaskingApp`

```
1: BOOL CTaskingApp::OnIdle(LONG lCount)
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
aufrufen
4:
5: // Leerlaufverarbeitung der Basisklasse aufrufen
6: CWinApp::OnIdle(lCount);
```

The screenshot shows a window titled 'Unbenannt - Tasking'. The menu bar includes 'Datei' and 'Bearbeiten'. A button labeled 'Info über Tasking...' is visible. The main area displays a tasking diagram with three threads: Thread 1 (blue), Thread 2 (green), and Thread 3 (red). Thread 1 and Thread 2 are active, while Thread 3 is idle. The diagram shows a complex network of lines representing task dependencies and execution flow. A mouse cursor is hovering over the 'Info über Tasking...' button.

<http://www.mut.com/lesecke/buecher/vcplus6/kap18.htm> (30 von 41) [19.04.2000 14:23:48]

Unabhängige Threads hinzufügen

Nachdem Sie sich damit bekannt gemacht haben, was alles zu einer `OnIdle`-Task gehört, geht es nun darum, unabhängige Threads in einer Anwendung zu realisieren. Als erstes fügen Sie eine Hauptfunktion für die Threads hinzu. Weiterhin ist der Code zum Starten und Anhalten der Threads aufzunehmen. Schließlich kommt noch der Code für die Kontrollkästchen der unabhängigen Threads hinzu, um diese Threads starten und stoppen zu können.

Die Hauptfunktion der Threads

Bevor Sie irgendeinen unabhängigen Thread starten können, muß der Thread wissen, was zu tun ist. Dazu erzeugen Sie eine Thread-Hauptfunktion, die der Thread ausführt, wenn er startet. Der Thread endet, sobald die Funktion endet. Demzufolge muß diese Funktion als Hauptsteuerung des Threads agieren und den Thread so lange laufen lassen, solange es irgend etwas für den Thread zu tun gibt. Ist die Arbeit des Threads erledigt, muß die Funktion zurückkehren.

Wenn man eine Funktion als Hauptfunktion für einen Thread erzeugt, kann man ihr einen einzelnen Parameter übergeben. Dieser Parameter ist ein Zeiger auf Informationen, die der Thread für die Ausführung seiner Aufgaben benötigt. Bei der Anwendung, die Sie in diesem Kapitel erstellen, kann der Parameter ein Zeiger auf den Fächer sein, auf dem der Thread operiert. Alles andere, was der Thread benötigt, läßt sich aus dem Fächerobjekt herausziehen.

Nachdem der Thread einen Zeiger auf seinen Fächer hat, kann er einen Zeiger auf die Variable für das Kontrollkästchen ermitteln, das darüber Auskunft gibt, ob die Drehung weiterlaufen oder sich der Thread selbst anhalten soll. Solange die Variable `TRUE` ist, soll der Thread die Drehung weiter ausführen.

Fügen Sie zu diesem Zweck der Dokumentklasse Ihrer Anwendung eine neue Member-Funktion hinzu. Legen Sie den Typ der Funktion mit `UINT`, die Funktionsdeklaration mit `ThreadFunc(LPVOID pParam)` und den Zugriff als *Privat* fest. Den Thread starten Sie aus der Dokumentklasse heraus, so daß diese Funktion für keine andere Klasse sichtbar sein muß. In die Funktion übernehmen Sie den Code aus Listing 18.14.

Listing 18.14: Die Funktion `ThreadFunc` der Klasse `CTaskingDoc`

```

1:  UINT CTaskingDoc::ThreadFunc(LPVOID pParam)
2:  {
3:      // Argument in einen Zeiger auf den Fächer
4:      // für diesen Thread konvertieren
5:      CSpinner* lpSpin = (CSpinner*)pParam;
6:      // Zeiger auf Fortsetzungsflag holen
7:      BOOL* pbContinue = lpSpin->GetContinue();
8:
9:      // Schleife, solange Fortsetzungsflag TRUE ist
10:     while (*pbContinue)
11:     // Fächer drehen
12:     lpSpin->Draw();
13:     return 0;
14: }
```

Die Threads starten und anhalten

Mit der Funktion `ThreadFunc` können Sie nun die unabhängigen Threads aufrufen. Jetzt brauchen Sie noch eine Möglichkeit, die Threads zu steuern, d.h. die Threads zu starten und anzuhalten. Dafür müssen Sie eine Reihe von Zeigern für `CWinThread`-Objekte, die die Threads verkapseln, speichern können. Die Zeiger fügen Sie als Variablen in das Dokumentobjekt ein und verwenden sie, um den Rückgabewert der Funktion `AfxBeginThread`, mit der Sie beide Threads starten, aufzunehmen.

Fügen Sie als erstes eine neue Member-Variable in die Dokumentklasse ein. Legen Sie den Variablentyp mit `CWinThread*`, den Variablennamen als `m_pSpinThread[2]` und den Zugriff auf die Variable mit `Privat` fest. Auf diese Weise erhalten Sie ein zweielementiges Feld für die Aufnahme der oben genannten Variablen.

Sie verfügen jetzt über einen Platz, an dem Sie die Zeiger auf die beiden Threads speichern können. Als nächstes schreiben Sie den Code, um die Threads zu starten. Für beide Threads können Sie ein und dieselbe Funktion vorsehen. Läuft der Thread noch nicht, starten Sie ihn, ansonsten warten Sie, bis sich der Thread selbst anhält. Die Funktion muß wissen, auf welchen Thread sich der Aufruf bezieht und ob der Thread zu starten oder anzuhalten ist.

Nehmen Sie dazu eine neue Member-Funktion in die Dokumentklasse auf. Legen Sie den Funktionstyp mit `void`, die Funktionsdeklaration als `SuspendSpinner(int nIndex, BOOL bSuspend)` und den Zugriff auf die Funktion mit `Public` fest. Schalten Sie außerdem das Kontrollkästchen **Statisch** ein. In die Funktion schreiben Sie den Code gemäß Listing 18.15.

Listing 18.15: Die Funktion `SuspendSpinner` der Klasse `CTaskingDoc`

```
1: void CTaskingDoc::SuspendSpinner(int nIndex, BOOL bSuspend)
2: {
3: // Thread anhalten?
4: if (!bSuspend)
5: {
6: // Ist Zeiger für den Thread gültig?
7: if (m_pSpinThread[nIndex])
8: {
9: // Handle für den Thread holen
10: HANDLE hThread = m_pSpinThread[nIndex]->m_hThread;
11: // Auf Absterben des Threads warten
12: ::WaitForSingleObject (hThread, INFINITE);
13: }
14: }
15: else // Thread laufen lassen
16: {
17: int iSpnr;
18: // Welchen Fächer verwenden?
19: switch (nIndex)
20: {
21: case 0:
22: iSpnr = 1;
23: break;
24: case 1:
25: iSpnr = 3;
26: break;
27: }
28: // Thread starten, Zeiger auf Fächer übergeben
```



```

29: m_pSpinThread[nIndex] = AfxBeginThread(ThreadFunc,
30: (LPVOID)&m_cSpin[iSpnr]);
31: }
32: }

```

Als erstes prüft die Funktion, ob der Thread zu starten oder anzuhalten ist. Wird der Thread gestoppt, prüft die Funktion als nächstes, ob der Zeiger auf den Thread gültig ist. Bei einem gültigen Zeiger rufen Sie den Handle für den Thread ab, indem Sie den Wert der Handle-Eigenschaft der Klasse `CWinThread` lesen:

```
HANDLE hThread = m_pSpinThread[nIndex]->m_hThread;
```

Mit diesem Handle rufen Sie die Funktion `WaitForSingleObject` auf, um darauf zu warten, daß sich der Thread selbst stoppt.

```
::WaitForSingleObject (hThread, INFINITE);
```

Die Windows-API-Funktion `WaitForSingleObject` teilt dem Betriebssystem mit, daß Sie darauf warten, daß der per Handle bezeichnete Thread anhält. Das zweite Argument an diese Funktion gibt an, wie lange Sie warten wollen. Der Wert `INFINITE` bedeutet, daß Sie unbegrenzt lange warten wollen, bis dieser Thread stoppt. Wenn Sie eine Zeitüberschreitung spezifizieren und der Thread nicht innerhalb dieser Zeit anhält, liefert die Funktion einen Wert zurück, der angibt, ob der Thread gestoppt hat. Da hier `INFINITE` als Zeitüberschreitung festgelegt ist, brauchen Sie sich nicht um den Rückgabewert der Funktion zu kümmern, da die Funktion erst dann zurückkehrt, wenn der Thread stoppt.

Wenn der Thread noch nicht läuft, bestimmt die Funktion den zu verwendenden Fächer und startet dann den Thread durch Aufruf der Funktion `AfxBeginThread`.

```
m_pSpinThread[nIndex] = AfxBeginThread(ThreadFunc,
(LPVOID)&m_cSpin[iSpnr]);
```

Als erstes Argument übergeben Sie die für den Thread aufzurufende Hauptfunktion, als zweites Argument die Adresse des Fächers, den der Thread verwenden soll.

Die Threads vom Ansichtsbjekt auslösen

Sie verfügen nun über die Mechanismen, um unabhängige Threads zu starten und zu stoppen. Jetzt müssen Sie noch die Threads über die Kontrollkästchen im Fenster starten und anhalten können. Sind beide Kontrollkästchen eingeschaltet, starten Sie beide Threads. Wenn die Kontrollkästchen ausgeschaltet sind, müssen beide Threads gestoppt werden. Die zweite Aufgabe ist leicht: Solange die Variable, die an das Kontrollkästchen gebunden ist, mit dem Steuerelement synchronisiert ist, stoppt sich der Thread selbst, sobald Sie das Kontrollkästchen ausschalten. Wenn das Kontrollkästchen jedoch eingeschaltet ist, müssen Sie die eben erstellte Funktion des Dokumentobjekts (zum Starten der Threads) aufrufen.

Um die Funktionalität für das erste der beiden Thread-Kontrollkästchen zu realisieren, fügen Sie mit dem Klassen-Assistenten eine Funktion für das Ereignis `BN_CLICKED` des Kontrollkästchens hinzu und übernehmen den Code aus Listing 18.16 in die Funktion.

Listing 18.16: Die Funktion `OnCbthread1` der Klasse `CTaskingView`

```

1: void CTaskingView::OnCbthread1()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4:

```

```

5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Variablen mit Dialogfeld synchronisieren
10: UpdateData(TRUE);
11:
12: // Zeiger auf Dokument holen
13: CTaskingDoc* pDocWnd = (CTaskingDoc*)GetDocument();
14: // Ist Zeiger gültig?
15: ASSERT_VALID(pDocWnd);
16:
17: // Fächer-Thread anhalten oder starten
18: pDocWnd->SuspendSpinner(0, m_bThread1);
19:
20: //////////////////////////////////////
21: // EIGENER CODE, ENDE
22: //////////////////////////////////////
23: }

```

Die Funktion ruft zuerst `UpdateData` auf, um die Variablen mit den Steuerelementen im Fenster zu synchronisieren. Als nächstes wird ein Zeiger auf das Dokument abgerufen. Mit dem gültigen Zeiger rufen Sie die Funktion `SuspendSpinner` des Dokuments auf, wobei Sie den ersten Thread spezifizieren. Außerdem übergeben Sie den aktuellen Wert der mit dem Kontrollkästchen verbundenen Variablen, damit die Funktion weiß, ob der Thread zu starten oder zu stoppen ist.

Um die gleiche Funktionalität für das andere Thread-Kontrollkästchen zu realisieren, führen Sie die gleichen Schritte aus, fügen aber den Code aus Listing 18.17 ein.

Listing 18.17: Die Funktion `OnCbthread2` der Klasse `CTaskingView`

```

1: void CTaskingView::OnCbthread2()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Âbenachrichtigung hier einfügen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Variablen mit Dialogfeld synchronisieren
10: UpdateData(TRUE);
11:
12: // Zeiger auf Dokument holen
13: CTaskingDoc* pDocWnd = (CTaskingDoc*)GetDocument();
14: // Ist Zeiger gültig?
15: ASSERT_VALID(pDocWnd);
16:
17: // Fächer-Thread anhalten oder starten
18: pDocWnd->SuspendSpinner(1, m_bThread2);
19:
20: //////////////////////////////////////

```

```

21: // EIGENER CODE, ENDE
22: //////////////////////////////////////
23: }

```

Nachdem Sie nun die unabhängigen Threads starten und stoppen können, kompilieren und starten Sie die Anwendung. Überzeugen Sie sich davon, daß sich die unabhängigen Threads über die betreffenden Kontrollkästchen starten und stoppen lassen.

Wenn Sie in dieser Entwicklungsphase ein wenig mit der Anwendung experimentieren, stellen Sie einen kleinen Unterschied zwischen den beiden Arten von Threads fest. Wenn alle Threads laufen und Sie die Maus ausgiebig bewegen, ist zu bemerken, daß die `OnIdle`-Fächer langsamer drehen (außer wenn Sie mit einem sehr schnellen Computer arbeiten). Die unabhängigen Threads beanspruchen einen guten Teil der Prozessorzeit vom Haupt-Thread der Anwendung und lassen kaum Prozessorzeit für den Leerlauf übrig. Im Ergebnis ist es einfacher, die Anwendung beschäftigt zu halten. Wenn Sie Menüs aktivieren oder das Info-Fenster öffnen, ist weiterhin festzustellen, daß zwar die `OnIdle`-Tasks komplett stoppen, die unabhängigen Threads aber weiterlaufen, wie es Abbildung 18.9 verdeutlicht. Die beiden Threads sind vollkommen unabhängige Prozesse innerhalb Ihrer Anwendung, so daß sie nicht vom übrigen Teil der Anwendung beeinflußt werden.

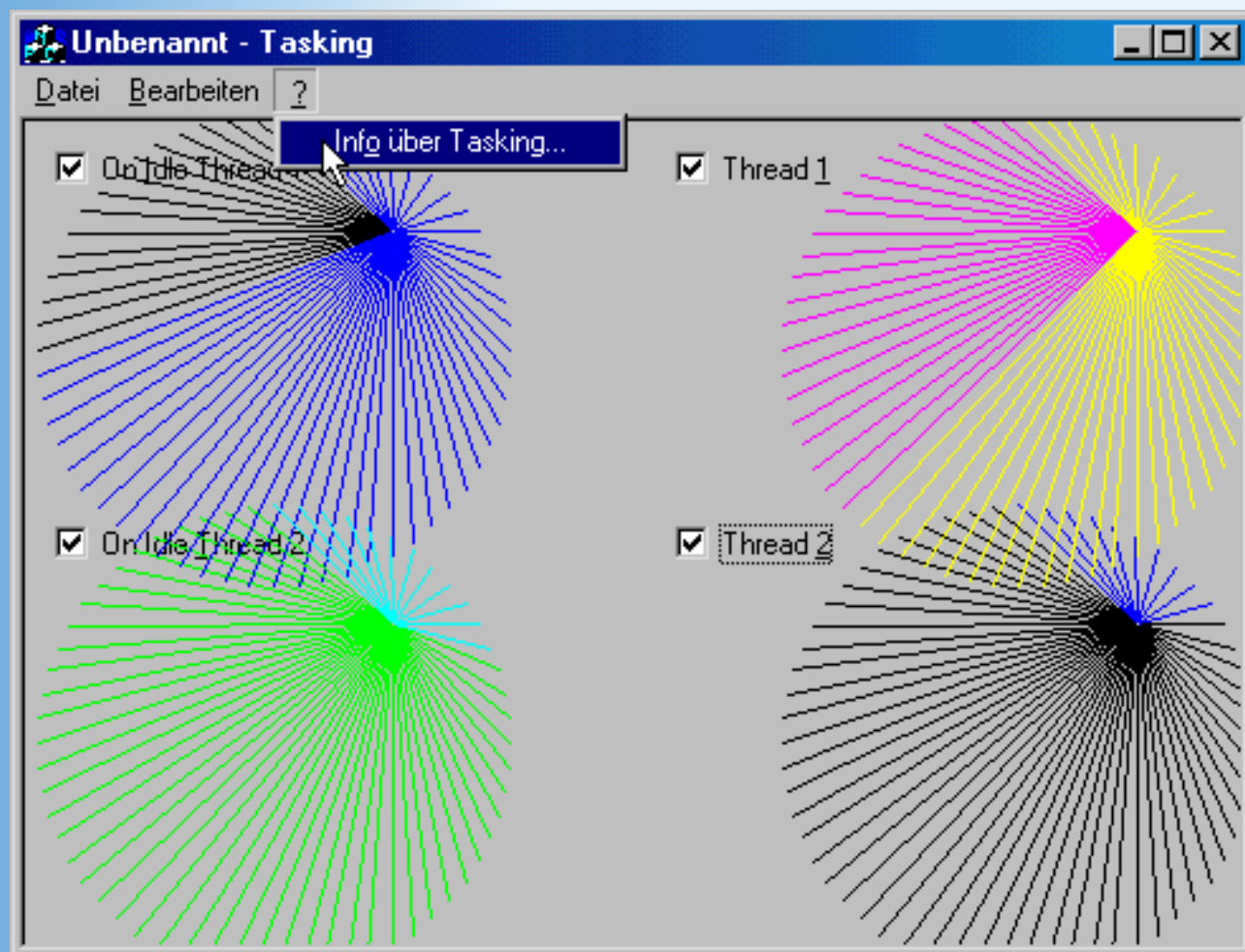


Abbildung 18.9:
Die Threads werden nicht durch das Menü beeinflußt.

Sauberes Herunterfahren

Vielleicht sind Sie der Meinung, daß die Anwendung fertiggestellt ist. Versuchen Sie aber nun, die Anwendung zu schließen, während ein oder beide unabhängigen Threads laufen. Leider erhalten Sie einen unwillkommenen Hinweis wie in Abbildung 18.10, so daß Sie noch etwas unternehmen müssen.

Es scheint, daß laufende Threads beim Schließen der Anwendung einen Absturz herbeiführen können.

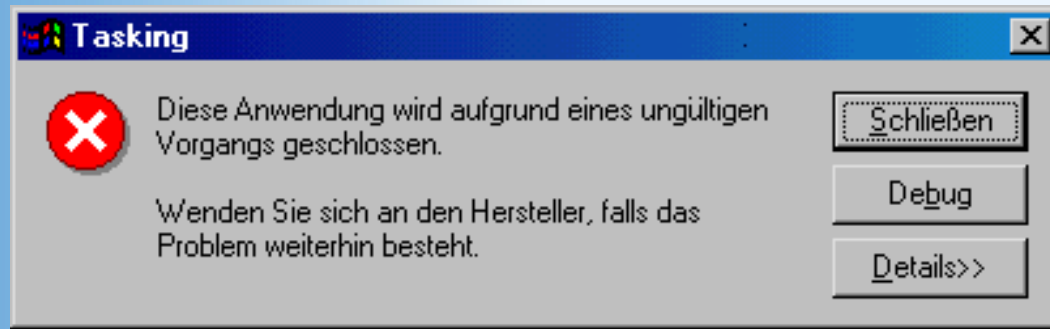


Abbildung 18.10:
Fehlermeldung der Anwendung

Auch wenn Sie die Anwendung schließen, laufen die Threads unbekümmert weiter. Sobald die Threads den Wert der Variablen testen, die angibt, ob der Thread weiterlaufen oder den zugehörigen Fächer drehen soll, greifen sie auf ein Speicherobjekt zu, das nicht mehr existiert.

Dieses Problem verursacht einen der grundlegendsten und schwerwiegendsten Speicherfehler in einer Anwendung, und Sie sollten diesen Fehler eliminieren, bevor Sie irgend jemandem die Anwendung zum Einsatz übergeben.

Um den Fehler zu unterbinden, stoppen Sie beide Threads, bevor Sie erlauben, die Anwendung zu schließen. Diese Aktion realisiert man am besten in der Behandlungsroutine für die Nachricht `WM_DESTROY` in der Ansichtsklasse. Mit dieser Nachricht wird der Ansichtsklasse mitgeteilt, daß vor dem Schließen der Anwendung noch alle erforderlichen Aufräumarbeiten zu erledigen sind. Mit dem Code in der Behandlungsfunktion setzen Sie die Variablen der beiden Kontrollkästchen auf `FALSE`, so daß sich die Threads selbst anhalten. Dann rufen Sie die Funktion `SuspendSpinner` für jeden Thread auf, um sich davon zu überzeugen, daß beide Threads gestoppt haben, bevor Sie der Anwendung das Schließen erlauben. Es ist kein Aufruf von `UpdateData` erforderlich, um die Variablen mit den Steuerelementen zu synchronisieren, da der Benutzer nicht mehr sehen muß, wenn Sie den Wert der Kontrollkästchen ändern.

Nehmen Sie zu diesem Zweck eine Behandlungsroutine für die Nachricht `WM_DESTROY` in die Ansichtsklasse Ihrer Anwendung auf. Der Anwendungs-Assistent erstellt diese Funktion normalerweise nicht für die Ansichtsklasse, so daß Sie sie bei Bedarf in der abgeleiteten Ansichtsklasse hinzufügen müssen. In die Funktion übernehmen Sie den Code von Listing 18.18.

Listing 18.18: Die Funktion `OnDestroy` der Klasse `CTaskingView`

```
1: void CTaskingView::OnDestroy()
2: {
3:     CFormView::OnDestroy();
4:
5:     // TODO: Code für die Behandlungsroutine für Nachrichten hier
    einfügen
6:
7:     ///////////////////////////////////
8:     // EIGENER CODE, ANFANG
9:     ///////////////////////////////////
10:
11:    // Läuft der erste Thread?
12:    if (m_bThread1)
13:    {
```



```

14: // Stoppen für ersten Thread veranlassen
15: m_bThread1 = FALSE;
16: // Zeiger auf das Dokument holen
17: CTaskingDoc* pDocWnd = (CTaskingDoc*)GetDocument();
18: // Ist der Zeiger gültig?
19: ASSERT_VALID(pDocWnd);
20:
21: // Fächer-Thread anhalten
22: pDocWnd->SuspendSpinner(0, m_bThread1);
23: }
24: // Läuft der zweite Thread?
25: if (m_bThread2)
26: {
27: // Stoppen für zweiten Thread veranlassen
28: m_bThread2 = FALSE;
29: // Zeiger auf das Dokument holen
30: CTaskingDoc* pDocWnd = (CTaskingDoc*)GetDocument();
31: // Ist der Zeiger gültig?
32: ASSERT_VALID(pDocWnd);
33:
34: // Fächer-Thread anhalten
35: pDocWnd->SuspendSpinner(1, m_bThread2);
36: }
37:
38: //////////////////////////////////////
39: // EIGENER CODE, ENDE
40: //////////////////////////////////////
41: }

```

Die Funktion führt genau das aus, was zu tun ist. Zuerst prüft sie die Variablen der Kontrollkästchen. Wenn eines der Kontrollkästchen den Wert **TRUE** aufweist, wird die entsprechende Variable auf **FALSE** gesetzt, ein Zeiger auf das Dokument ermittelt und die Funktion **SuspendSpinner** für den betreffenden Thread aufgerufen. Wenn Sie nun die Anwendung schließen, stürzt sie nicht mehr ab, auch wenn gerade unabhängige Threads laufen.

Zusammenfassung

Der heutige Tag hat eine Menge Stoff gebracht. Es ging um die verschiedenen Möglichkeiten, wie man mehrere Aufgaben (sogenannte Tasks) mit Anwendungen gleichzeitig ausführen kann. Sie haben wichtige Punkte kennengelernt, die zu beachten sind, wenn man die Anwendungen mit Multitasking-Fähigkeiten ausstatten möchte. Es wurde gezeigt, wie sich Tasks im Leerlauf einer Anwendung ausführen lassen. In diesem Zusammenhang wurden auch die Einschränkungen und Nachteile dieser Lösung behandelt. Weiterhin haben Sie gelernt, wie man unabhängige Threads in einer Anwendung erstellt. Die Threads arbeiten vollkommen unabhängig von der übrigen Anwendung. In der heutigen Beispielanwendung haben Sie die beiden Lösungsansätze verwendet, so daß Sie damit Erfahrung sammeln können.



Wenn Sie in Ihre Anwendungen die Fähigkeiten zum Multitasking einbauen wollen, sollten Sie sich darüber im Klaren sein, daß diese Aufgaben zu den komplizierteren Aspekten der Windows-Programmierung gehören. Sie müssen eine ganze Reihe von Faktoren verstehen und weit mehr in Betracht ziehen, als wir an einem einzigen Tag behandeln konnten. Wenn Sie Multitasking-Anwendungen erstellen möchten, sollten Sie sich mit einschlägigen Büchern zur Programmierung von Windows-Anwendungen mit MFC oder Visual C++ beschäftigen. Das Buch sollte profundes Wissen zum Multithreading mit MFC vermitteln und alle Synchronisierungsklassen detaillierter behandeln, als es an dieser Stelle möglich war. Denken Sie daran, daß Sie ein Buch brauchen, das sich auf MFC konzentriert und nicht auf die Entwicklungsumgebung von Visual C++. (MFC wird von den meisten kommerziellen C++-Entwicklungswerkzeugen für Windows-Anwendungen unterstützt. Dazu gehören auch die Compiler von Borland - jetzt Inprise - und Symantec. Eine Behandlung dieser Themen geht also über die Umgebung von Visual C++ hinaus.)

Fragen und Antworten

Frage:

Wie kann ich die andere Version von `AfxBeginThread` einsetzen, um einen Thread in einer benutzerdefinierten Klasse zu verkapseln?

Antwort:

Erstens dient die andere Version von `AfxBeginThread` hauptsächlich dazu, Threads für Benutzeroberflächen zu erzeugen. Die in der heutigen Beispielanwendung eingesetzte Version ist für sogenannte Worker-Threads vorgesehen, die sofort mit einer bestimmten Task beginnen. Wenn Sie einen Thread für eine Benutzeroberfläche erzeugen möchten, müssen Sie Ihre benutzerdefinierte Klasse von der Klasse `CWinThread` ableiten. Als nächstes sind mehrere Vorgängerfunktionen in der benutzerdefinierten Klasse zu überschreiben. Nachdem die Klasse für den Einsatz vorbereitet ist, holen Sie mit dem Makro `RUNTIME_CLASS` einen Zeiger auf die Laufzeitklasse Ihrer Klasse und übergeben diesen Zeiger folgendermaßen an die Funktion `AfxBeginThread`:

```
CWinThread* pMyThread =
AfxBeginThread(RUNTIME_CLASS(CMyThreadClass));
```

Frage:

Kann ich mit `SuspendThread` und `ResumeThread` meine unabhängigen Threads in der Beispielanwendung starten und stoppen?

Antwort:

Ja. Es sind aber ein paar wesentliche Änderungen an der Anwendung erforderlich. Erstens müssen Sie in der Funktion `OnNewDocument` die beiden Thread- Zeiger mit `NULL` initialisieren, wie es Listing 18.19 zeigt.

Listing 18.19: Die modifizierte Funktion `OnNewDocument` der Klasse `CTaskingDoc`

```
1: BOOL CTaskingDoc::OnNewDocument()
2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
```



```

8:
9: ///////////////////////////////////////////////////
10: // EIGENER CODE, ANFANG
11: ///////////////////////////////////////////////////
12:
13: // Die Fächer initialisieren
14: InitSpinners();
15:
16: // Die Thread-Zeiger initialisieren
17: m_pSpinThread[0] = NULL;
18: m_pSpinThread[1] = NULL;
19:
20: ///////////////////////////////////////////////////
21: // EIGENER CODE, ENDE
22: ///////////////////////////////////////////////////
23:
24: return TRUE;
25: }

```

Als nächstes modifizieren Sie die Thread-Funktion gemäß Listing 18.20, damit sich der Thread nicht selbst anhält, sondern weiter in der Schleife läuft, wenn die Variable des Kontrollkästchens den Wert FALSE aufweist.

Listing 18.20: Die modifizierte Funktion ThreadFunc der Klasse CTaskingDoc

```

1: UINT CTaskingDoc::ThreadFunc(LPVOID pParam)
2: {
3: // Argument in einen Zeiger auf den Fächer
4: // für diesen Thread konvertieren
5: CSpinner* lpSpin = (CSpinner*)pParam;
6: // Zeiger auf Fortsetzungsflag holen
7: BOOL* pbContinue = lpSpin->GetContinue();
8:
9: // Schleife, solange Fortsetzungsflag TRUE ist
10: while (TRUE)
11: // Fächer drehen
12: lpSpin->Draw();
13: return 0;
14: }

```

Schließlich ist die Funktion SuspendSpinner entsprechend Listing 18.21 zu modifizieren, so daß sie bei einem gültigen Thread-Zeiger die Funktion SuspendThread auf dem Thread-Zeiger aufruft, um den Thread zu stoppen, und die Funktion ResumeThread, um den Thread erneut zu starten.

Listing 18.21: Die modifizierte Funktion SuspendSpinner der Klasse CTaskingDoc

```

1: void CTaskingDoc::SuspendSpinner(int nIndex, BOOL bSuspend)
2: {
3: // Thread anhalten?
4: if (!bSuspend)
5: {
6: // Ist Zeiger für den Thread gültig?
7: if (m_pSpinThread[nIndex])

```

```

8: {
9: // Handle für den Thread holen
10: m_pSpinThread[nIndex]->SuspendThread();
11: }
12: }
13: else // Thread laufen lassen
14: {
15: // Ist Zeiger für den Thread gültig?
16: if (m_pSpinThread[nIndex])
17: {
18: // Thread fortsetzen
19: m_pSpinThread[nIndex]->ResumeThread();
20: }
21: else
22: {
23: int iSpnr;
24: // Welchen Fächer verwenden?
25: switch (nIndex)
26: {
27: case 0:
28: iSpnr = 1;
29: break;
30: case 1:
31: iSpnr = 3;
32: break;
33: }
34: // Thread starten, Zeiger auf Fächer übergeben
35: m_pSpinThread[nIndex] = AfxBeginThread(ThreadFunc,
36: (LPVOID)&m_cSpin[iSpnr]);
37: }
38: }
39: }

```

Workshop

Kontrollfragen

1. Wann wird die Funktion `OnIdle` aufgerufen?
2. Wie kann man den wiederholten Aufruf der Funktion `OnIdle` bewirken, während sich die Anwendung im Leerlauf befindet?
3. Worin besteht der Unterschied zwischen einer `OnIdle`-Task und einem Thread?
4. Wie heißen die vier Objekte zur Synchronisierung von Threads?
5. Warum sollten Sie keine höhere als die normale Priorität für die Threads in Ihrer Anwendung festlegen?

Übungen

1. Wenn Sie die heutige Beispielanwendung ausführen und die Auslastung Ihres Computers mit einem Systemmonitor überwachen, zeigt sich, daß die Prozessorauslastung sogar ohne laufende Threads bei 100 Prozent bleibt. Die Funktion `OnIdle` wird fortlaufend aufgerufen, selbst wenn nichts zu tun ist.

Modifizieren Sie die Funktion `OnIdle`, damit keine der `OnIdle`-Tasks aktiv ist, wenn es nichts zu tun gibt. Somit wird die Funktion `OnIdle` erst dann wieder fortlaufend aufgerufen, wenn einer der Threads aktiv wird. Dieser Zustand dauert nur so lange an, bis beide Threads wieder ausgeschaltet sind. Der Prozessor kann dann zu einer minimalen Auslastung zurückkehren.

2. Geben Sie beim Starten der unabhängigen Threads einem der Threads die Priorität `THREAD_PRIORITY_NORMAL` und dem anderen die Priorität `THREAD_PRIORITY_LOWEST` .

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](http://www.mut.com/lesecke/buecher/vcplus6/kap18.htm).

Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 3**Tag 19****Eigene ActiveX- Steuerelemente**

Die Softwareindustrie hat in den vergangenen Jahren eine Revolution erlebt. Die Art und Weise, wie man Software erstellt und im Paket zusammenschnürt, hat sich von einem Modell, bei dem alle Anwendungen große, monolithische Stücke von ausführbarem Code waren, zu einem Modell entwickelt, bei dem die meisten Anwendungen aus kleinen Bausteinen bestehen. Diese kleinen Bausteine - oftmals als Komponenten bezeichnet - lassen sich mit den unterschiedlichsten Sprachen erstellen und können die verschiedensten Formen annehmen. Eine der bekanntesten Komponenten ist das ActiveX-Steuerelement. Wenn Sie wissen, wie man eigene ActiveX-Steuerelemente erstellt, können Sie eigene Komponenten erzeugen und anderen Programmierern zur Verfügung stellen. Heute lernen Sie, wie man ...

- mit den Assistenten von Visual C++ ActiveX-Steuerelemente erstellt,
- Eigenschaften und Methoden mit dem Klassen-Assistenten in die Steuerelemente aufnimmt,
- das Steuerelement mit den Werkzeugen von Visual C++ testet.

Was ist ein ActiveX-Steuerelement?

Ein ActiveX-Steuerelement ist eine funktionelle Einheit, die in einem COM- (Component Object Model) Objekt verpackt ist. Dieses COM-Objekt ist selbständig, obwohl es nicht die Fähigkeit besitzt, eigenständig zu laufen. Ein ActiveX-Steuerelement läßt sich nur innerhalb eines ActiveX-Containers ausführen, beispielsweise in einer Visual C++- oder Visual Basic-Anwendung.

Wie Sie am Tag 9 gelernt haben, stellen ActiveX-Steuerelemente eine Reihe von Schnittstellen bereit, auf die die Container-Anwendung zurückgreift, um die verschiedenen funktionellen Einheiten, die das Steuerelement bietet, auszulösen. Viele dieser Schnittstellen dienen dazu, Ereignisse im Steuerelement oder in der Container-Anwendung auszulösen. Andere Schnittstellen sind dafür vorgesehen, die Eigenschaftsseite des Steuerelements zu spezifizieren oder um mitzuteilen, ob das Steuerelement aktiviert wurde. Alles in allem sind in den meisten ActiveX-Steuerelementen derartig viele Schnittstellen eingebaut, daß man eine ganze Zeit brauchen würde, um die Funktionalität für jede dieser Schnittstellen selbst zu codieren. Glücklicherweise fügen der Anwendungs-Assistent und der Klassen-Assistent von Visual C++ den größten Teil dieser Funktionalität automatisch hinzu und gestatten Ihnen damit, daß Sie sich auf die spezifische Funktionalität, die man vom Steuerelement erwartet, konzentrieren können.

Für das zu erstellende Steuerelement müssen Sie trotzdem selbst festlegen, welche Eigenschaften, Methoden und Ereignisse Sie für das Steuerelement freilegen möchten. Diese Elemente können Sie

dem Steuerelement über den Klassen-Assistenten hinzufügen. Wenn aber irgendwelche Eigenschaften oder Ereignisse speziellen Code von Ihrer Seite erfordern, müssen Sie ihn auch selbst hinzufügen. Wie bei allen Methoden, die Sie Ihren Steuerelementen hinzufügen, müssen Sie den gesamten Code bereitstellen. Der Klassen-Assistent baut automatisch die umgebende Struktur und den Code auf, um der Container-Anwendung zu erlauben, die Methode zu sehen und aufzurufen - genauso wie der Klassen-Assistent Ihrer Anwendung den erforderlichen Code hinzufügt, um beliebige Behandlungsroutinen aufzurufen.

Eigenschaften

Eigenschaften sind die Attribute von Steuerelementen, die für die Container-Anwendung sichtbar sind und sich oftmals auch durch diese modifizieren lassen. Die vier grundlegenden Typen der Eigenschaften sind *ambient*, *erweitert*, *vordefiniert* und *benutzerdefiniert*. Ambient-Eigenschaften - wie etwa Hintergrundfarbe oder Standardschrift - stellt die Container-Anwendung dem Steuerelement bereit, so daß das Steuerelement wie ein Teil der Container-Anwendung aussieht. Erweiterte Eigenschaften sind eigentlich keine Eigenschaften des Steuerelements, sondern werden von der Container-Anwendung bereitgestellt und implementiert, beispielsweise die Tabulator-Reihenfolge. Das Steuerelement kann diese Eigenschaften bis zu einem gewissen Grad erweitern. Wenn zum Beispiel das Steuerelement zwei oder mehr Standardsteuerelemente enthält, kann es die Tabulator-Reihenfolge im Steuerelement als Ganzes festlegen und die Kontrolle der Tabulator-Reihenfolge an die Anwendung zurückgeben, nachdem das Steuerelement die interne Tabulator-Reihenfolge durchlaufen hat. Vordefinierte Eigenschaften werden durch das ActiveX-Entwicklungskit implementiert, beispielsweise die Schrift oder die Hintergrundfarbe des Steuerelements. Mit dem letzten Typ, den benutzerdefinierten Eigenschaften, haben Sie am meisten zu tun, da diese Eigenschaften für Ihr Steuerelement spezifisch sind und sich direkt auf die Funktionalität des Steuerelements beziehen.

Über die Registerkarte **Automatisierung** können Sie alle Eigenschaften festlegen, die Sie in Ihrem Steuerelement brauchen. Wenn Sie eine neue Eigenschaft über den Klassen-Assistenten in das Steuerelement aufnehmen, spezifizieren Sie mehrere Aspekte der Eigenschaft.

Der erste Aspekt ist der externe Eigenschaftsname. Unter diesem Namen spricht auch die Container-Anwendung die Eigenschaft an. Ein weiterer Aspekt ist der, daß man den internen Variablennamen festlegen kann, den man selbst im Code verwendet, aber nur, wenn die Eigenschaft als Member-Variable implementiert ist. Weiterhin legen Sie den Variablentyp für die Eigenschaft fest.

Wenn Sie angeben, die Eigenschaft als Member-Variable zu implementieren (die Eigenschaft ist eine Member-Variable der Steuerelementklasse), dann können Sie den Namen der Benachrichtigungsfunktion spezifizieren, die aufzurufen ist, wenn die Eigenschaft durch die Container-Anwendung geändert wird. Ist die Eigenschaft keine Member-Variable der Steuerelementklasse, müssen Sie festlegen, daß sie über die Methoden **Get** und **Set** abgerufen und geändert wird, wobei die Container-Anwendung eine **Get**-Methode aufruft, um den aktuellen Wert der Eigenschaft zu holen, und mit einer **Set**-Methode den Wert der Eigenschaft ändert. Wenn die Eigenschaft über **Get**- und **Set**-Methoden verwaltet wird, können Sie die Namen dieser beiden Methoden festlegen.

Für alle diese Aspekte einer Eigenschaft schlägt das Dialogfeld **Eigenschaft hinzufügen** einen passenden Namen vor, nachdem Sie den externen Namen für die Eigenschaft eingegeben haben. Wenn Sie die Standardnamen übernehmen möchten, müssen Sie nur folgende Dinge festlegen: den externen Namen, den Typ und ob die Variable eine Member-Variable ist oder die Methoden **Get** und **Set** zu verwenden sind. Wenn Sie eine vordefinierte Eigenschaft aus der Liste der Standardeigenschaften auswählen, werden die übrigen Elemente automatisch spezifiziert. Nachdem Sie alle Informationen

angegeben haben, fügt der Klassen-Assistent den gesamten erforderlichen Code und die Variablen in Ihr Steuerelementprojekt ein.

Methoden

Methoden sind Funktionen im Steuerelement, die sich durch die Container-Anwendung aufrufen lassen. Die Funktionen werden über die (am Tag 9 behandelte) **IDispatch** -Schnittstelle anderen Anwendungen verfügbar gemacht. Aufgrund der Arbeitsweise von **IDispatch** beim Aufruf der Methoden in einem Steuerelement sind die an die Methode zu übergebenden Variablen in einer Struktur zu verpacken, die an das Steuerelement übergeben wird. Diese Struktur ist maschinenunabhängig, so daß es keine Rolle spielt, ob das Steuerelement unter Windows 95/98 auf einem Intel Pentium II oder unter Windows NT mit einem MIPS oder Alpha-Prozessor läuft. Die Struktur ist immer die gleiche. Jede Seite des Funktionsaufrufs ist selbst dafür verantwortlich, die Parameter nach Bedarf zu konvertieren, um sie korrekt in die Struktur einzutragen oder sie daraus zu extrahieren. Diese Verpackung der Methodenparameter bezeichnet man als **Marshaling**.

Wenn Sie Ihrem Steuerelement eine neue Methode über den Klassen-Assistenten auf der Registerkarte **Automatisierung** hinzufügen, fügt der Klassen-Assistent sowohl den gesamten erforderlichen Code ein, um das Marshaling der Parameter zu realisieren, als auch andere unterstützende Funktionalität, wozu auch der Aufbau der **IDispatch** -Schnittstelle und der Tabelle gehört.

Der Klassen-Assistent fordert Sie auf, den externen Namen für die Methode anzugeben, die durch die Container-Anwendung aufgerufen wird. Die Methode erhält einen vorgegebenen internen Namen, den Sie mit einem selbst gewählten internen Namen überschreiben können. Andere Aspekte Ihrer Steuerelementmethoden, die Sie festlegen müssen, sind der Rückgabetyt und die Parameter der Methode. Nachdem Sie diese Informationen eingegeben haben, fügt der Klassen-Assistent den gesamten erforderlichen Code automatisch ein.

Ereignisse

Ereignisse sind Benachrichtigungen, die vom Steuerelement an die Container-Anwendung gesendet werden. Sie sind dafür vorgesehen, die Anwendung zu informieren, daß ein bestimmtes Ereignis eingetreten ist, und die Anwendung kann bei Bedarf eine Aktion unternehmen. Es lassen sich zwei Ereignisarten vom Steuerelement auslösen: vordefinierte und benutzerdefinierte Ereignisse.

Vordefinierte Ereignisse werden vom ActiveX-Steuerelement-Entwicklungskit implementiert und sind als Funktionsaufrufe innerhalb des Steuerelements verfügbar. Über diese vordefinierten Ereignisse können Sie in der Container-Anwendung Ereignisse für Maus und Tastatur, Fehler oder Zustandsänderungen auslösen.

Neben den vordefinierten Ereignissen kann man eigene benutzerdefinierte Ereignisse hinzufügen, die in der Container-Anwendung ausgelöst werden. Diese Ereignisse sollten sich auf die spezifische Funktionalität Ihres Steuerelements beziehen. Man kann zusammen mit dem Ereignis an die Container-Anwendung Argumente übergeben, so daß die Anwendung die Daten erhält, die sie für die Reaktion auf die Nachricht benötigt.

Wenn Sie diese Ereignisse auslösen müssen, rufen Sie einfach die interne Ereignisfunktion auf, die das Ereignis initiiert, und übergeben alle erforderlichen Parameter an die Funktion. Der Klassen-Assistent fügt auch den gesamten Code hinzu, damit sich die Nachricht vom internen Funktionsaufruf auslösen läßt.

Ereignisse gehören zu den drei Elementen, die Sie für Ihr Steuerelement nicht auf der Registerkarte

Automatisierung im Klassen-Assistenten festlegen, sondern über die Registerkarte **ActiveX-Ereignisse** des Klassen-Assistenten.

Ein ActiveX-Steuerelement erstellen

Im heutigen Beispiel erstellen Sie ein ActiveX-Steuerelement für Freihandzeichnungen, das Ihnen bereits aus Lektion 16 als Bibliotheksmodul bzw. Lektion 17 als DLL bekannt ist. Dieses Modul wandeln Sie nun in ein ActiveX-Steuerelement um. Dabei legen Sie als Eigenschaften des Steuerelements die Anzahl der vom Steuerelement zu zeichnenden Squiggles und auch die maximale Länge der Squiggles frei. Diese Eigenschaften kann die Container-Anwendung setzen. Das Steuerelement ist so programmiert, daß es beim Anklicken eine neue Squiggle-Zeichnung erzeugt. Außerdem fügen Sie eine Methode hinzu, um eine Squiggle-Zeichnung in das Steuerelement zu laden, die mit den vorherigen Versionen des Squiggle-Moduls erstellt wurde. Schließlich lassen Sie das Steuerelement ein Ereignis auslösen, um der Container-Anwendung mitzuteilen, daß das Steuerelement die Zeichnung geladen hat.

Das Gerüst des Steuerelements erstellen

Sicherlich haben Sie mittlerweile bemerkt, daß im Dialogfeld **Neu** auf der Registerkarte **Projekte** die Option **MFC ActiveX-Steuerelement-Assistent** vorhanden ist. Es handelt sich um einen weiteren Projektassistenten analog zum Anwendungs-Assistenten, der Anwendungs- und DLL-Projekte erzeugt. Mit dem ActiveX-Steuerelement-Assistenten können Sie ein Gerüst für beliebige ActiveX-Steuerelemente erstellen. Der Assistent legt die erforderlichen Dateien an und konfiguriert das Projekt, so daß der Compiler ein ActiveX-Steuerelement erstellen kann.

Der Steuerelement-Assistent stellt im ersten Schritt ein paar Fragen über das Steuerelementprojekt, beispielsweise wie viele Steuerelemente das Projekt umfassen soll und ob die Steuerelemente mit Laufzeitlizenzen auszustatten sind.



Mit Laufzeitlizenzen kann man sicherstellen, daß der Benutzer des Steuerelements eine Lizenz erworben hat, um das Steuerelement einzusetzen. Steuerelemente, die für den Verkauf an Entwickler vorgesehen sind, haben oftmals Laufzeitlizenzen. Die Lizenz verhindert, daß ein Benutzer das Steuerelement kostenlos nutzt. Wenn Sie das Steuerelement in einer Anwendung einsetzen, wird die Laufzeitlizenz entweder durch die Installationsroutine in der Registrierung des Benutzers eingerichtet oder zusammen mit der Anwendung kompiliert. Auf diese Weise lassen sich mit dem Steuerelement keine neuen Anwendungen erstellen.

Das zweite Dialogfeld des Steuerelement-Assistenten stellt etwas tiefergehende Fragen, die aber dennoch leicht zu beantworten sind. In diesem Schritt können Sie auf die Schaltfläche **Namen bearbeiten** klicken, um dem Steuerelement einen aussagekräftigen Namen zu geben, mit dem der Benutzer etwas anfangen kann. Das Kombinationsfeld im unteren Teil des Assistenten listet eine Reihe von Fensterklassen auf, die von diesem Steuerelement als untergeordnete Klasse definiert werden können. Wenn Sie ein spezielles Eingabefeld erstellen möchten, das bestimmte Editorfunktionen für die Benutzereingaben erlaubt, wählen Sie **EDIT** aus der Liste der Fensterklassen im Dropdown-Teil des Kombinationsfelds. Über die Schaltfläche **Erweitert** gelangen Sie zu einem Dialogfeld, bei dem die Beantwortung der Fragen ein tiefergehendes Verständnis der ActiveX-Steuerelemente erfordert.

Legen Sie für das heutige Beispiel eines Steuerelements ein neues Projekt an, indem Sie im Dialogfeld

Neu auf der Registerkarte **Projekte** die Option **MFC ActiveX- Steuerelement-Assistent** wählen und dem Projekt einen passenden Namen wie etwa **Squiggle** geben (siehe Abbildung 19.1).

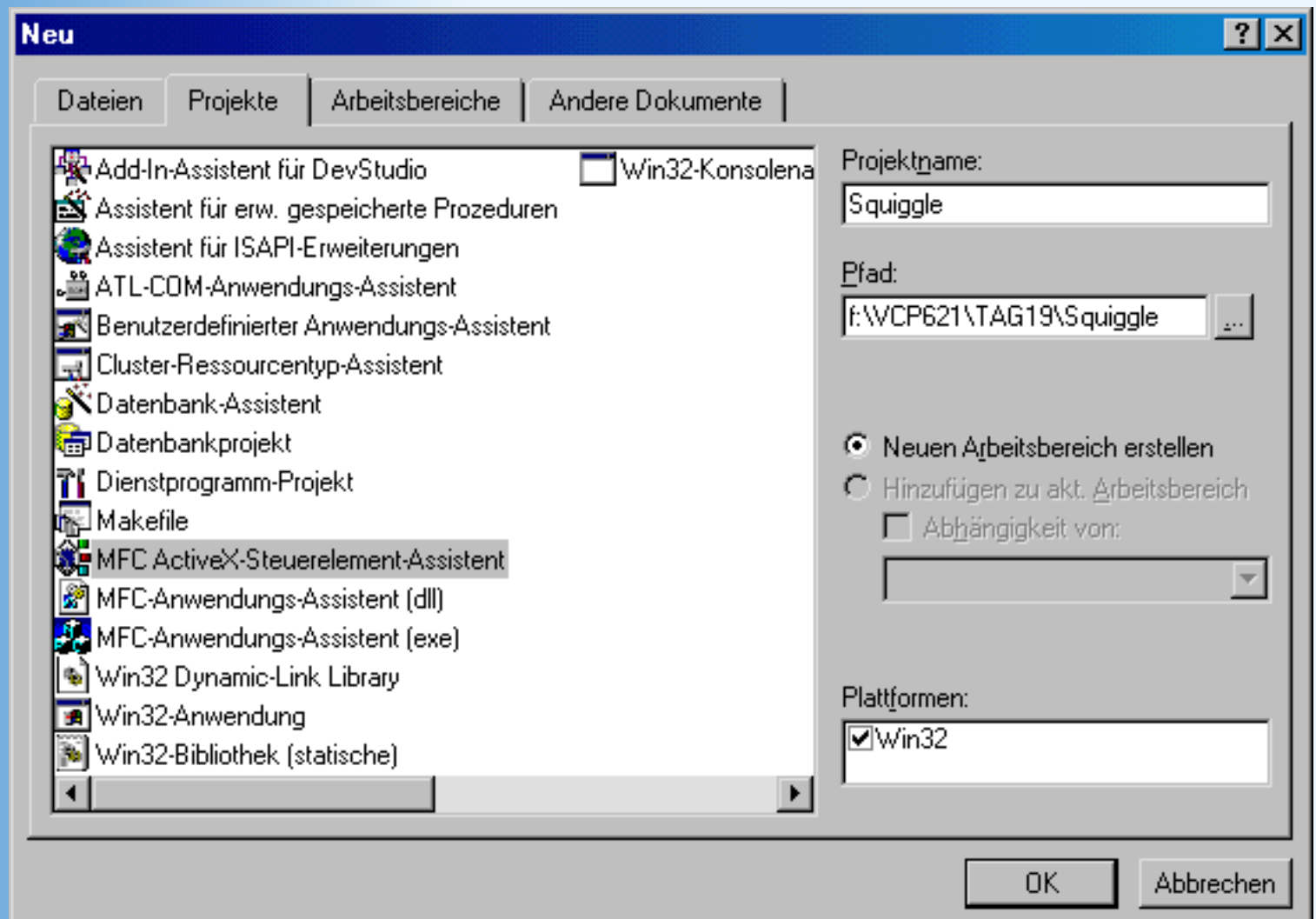


Abbildung 19.1:
Das Projekt eines ActiveX-Steuerelements beginnen

Im ersten Schritt des MFC ActiveX-Steuerelement-Assistenten übernehmen Sie die Standardeinstellungen, da Sie heute nur ein einzelnes Steuerelement erstellen und auch keine Laufzeitlizenz einbinden. Im zweiten Dialogfeld des Steuerelement-Assistenten klicken Sie auf die Schaltfläche **Namen bearbeiten**. Stellen Sie sicher, daß der Typ-Name das Steuerelement ausreichend charakterisiert. Klicken Sie auf OK, um die Namen zu akzeptieren und zum zweiten Dialogfeld des ActiveX-Steuerelement-Assistenten zurückzukehren. Wenn Sie im ersten Schritt festgelegt haben, daß Sie mehrere Steuerelemente erzeugen, können Sie im Dropdown-Listefeld neben der Schaltfläche **Namen bearbeiten** die einzelnen Steuerelemente auswählen und jeweils die Namen für das Projekt festlegen. Für die übrigen Optionen des Steuerelement-Assistenten können Sie im Beispielprojekt die Standardeinstellungen übernehmen.

Die Klasse CModArt modifizieren

Nachdem Sie das Gerüst des Steuerelements erstellt haben, kopieren Sie die Dateien **Line** und **ModArt** aus dem Projektverzeichnis des am Tag 16 angelegten Bibliotheksprojekts. Laden Sie alle vier Dateien in das Steuerelementprojekt, und fügen Sie dem Projekt die Klassen **CLine** und **CModArt** hinzu.

Die Änderungen, die Sie an der Klasse **CModArt** für Ihr Steuerelement vornehmen müssen, betrifft die

Variablen für die Maximalzahl und die Länge der Squiggels, die als Eigenschaften des Steuerelements freigelegt werden können. Zu diesem Zweck fügen Sie der Klasse `CModArt` zwei Member-Variablen hinzu. Mit der einen Variablen steuern Sie die Länge der Squiggles, mit der anderen die Anzahl. Fügen Sie diese beiden Variablen der Klasse `CModArt` gemäß Tabelle 19.1 hinzu.

Tabelle 19.1:
Elementvariablen für die Klasse CModArt

Name	Typ	Zugriff
m_iLength	int	privat
m_iSegments	int	privat

Es ist nun eine Möglichkeit zu schaffen, um die Variablen über die freigelegten Eigenschaften abrufen und aktualisieren zu können. Das bedeutet, daß Sie für beide Werte Funktionen brauchen, um den aktuellen Wert zu holen und den neuen Wert einzustellen. Für die Variable `m_iLength` nehmen Sie zu diesem Zweck eine Member-Funktion in die Klasse `CModArt` auf. Legen Sie den Funktionstyp als `int`, die Deklaration als `GetLength` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code aus Listing 19.1.

Listing 19.1: Die Funktion GetLength der Klasse CModArt

```
1: int CModArt::GetLength()
2: {
3: // Aktuellen Wert für Variable m_iLength zurückgeben
4: return m_iLength;
5: }
```

Als nächstes fügen Sie eine weitere Member-Funktion in die Klasse `CModArt` ein. Legen Sie den Funktionstyp als `void`, die Deklaration mit `SetLength(int iLength)` und den Zugriff als *Public* fest. In die Funktion übernehmen Sie den Code aus Listing 19.2.

Listing 19.2: Die Funktion SetLength der Klasse CModArt

```
1: void CModArt::SetLength(int iLength)
2: {
3: // Aktuellen Wert für Variable m_iLength setzen
4: m_iLength = iLength;
5: }
```

Analoge Funktionen fügen Sie für die Variable `m_iSegments` hinzu, so daß sich diese Variable ebenfalls als Eigenschaft des Steuerelements freilegen läßt.

Nachdem Sie die beiden Eigenschaften für das Steuerelement verfügbar gemacht haben, stellen Sie sicher, daß sie auf vernünftige Werte initialisiert werden, bevor das Steuerelement zum Einsatz kommt. Um die Werte zu initialisieren, modifizieren Sie den Konstruktor von `CModArt` gemäß Listing 19.3.

Listing 19.3: Der modifizierte Konstruktor der Klasse CModArt

```
1: CModArt::CModArt()
2: {
3: // Zufallszahlengenerator initialisieren
4: srand((unsigned)time(NULL));
```

```

5: // Eigenschaftsvariablen initialisieren
6: m_iLength = 200;
7: m_iSegments = 50;
8: }

```

Schließlich modifizieren Sie die beiden Funktionen, die die Squiggle-Zeichnungen erstellen, so daß sie die Variablen anstelle der momentan fest kodierten Werte verwenden. Um die Funktion `NewDrawing` zu modifizieren, ersetzen Sie die Maximalzahl der Squiggles entsprechend Zeile 7 von Listing 19.4 durch die Variable `m_iSegments`.

Listing 19.4: Die modifizierte Funktion `NewDrawing` der Klasse `CModArt`

```

1: void CModArt::NewDrawing()
2: {
3:     int lNumLines;
4:     int lCurLine;
5:
6:     // Anzahl der zu erzeugenden Linien bestimmen
7:     lNumLines = rand() % m_iSegments;
8:     // Sind Linien zu erzeugen?
9:     if (lNumLines > 0)
10:    {
11:        // Schleife durch Anzahl der Linien
12:        for (lCurLine = 0; lCurLine < lNumLines; lCurLine++)
13:        {
14:            // Die neue Linie erzeugen
15:            NewLine();
16:        }
17:    }
18: }

```

Schließlich ersetzen Sie die Maximallänge der Squiggles in der Funktion `NewLine` gemäß Zeile 20 von Listing 19.5 durch die Variable `m_iLength`.

Listing 19.5: Die modifizierte Funktion `NewLine` der Klasse `CModArt`

```

1: void CModArt::NewLine()
2: {
3:     int lNumLines;
4:     .
5:     .
6:     .
7:
18:
19:     // Anzahl der Teile dieses Squiggles bestimmen
20:     lNumLines = rand() % m_iLength;
21:     // Umfaßt das Squiggle mindestens ein Teil?
22:     .
23:     .
24:     .
25:
67: }

```

An den Klassen `CModArt` und `CLine` haben Sie nun alle erforderlichen Änderungen für Ihr ActiveX-Steuerelement vorgenommen. Jetzt müssen Sie eine Instanz der Klasse `CModArt` in die

Steuerelementklasse als Member-Variable hinzufügen. Nehmen Sie eine neue Member-Variable in die Steuerelementklasse `CSquiggleCtrl` auf. Legen Sie den Variablentyp als `CModArt`, den Namen als `m_maDrawing` und den Zugriff als *Privat* fest. Weiterhin müssen Sie die Header-Datei für die Klasse `CModArt` in die Quellcodedatei der Steuerelementklasse aufnehmen. Öffnen Sie dazu die Datei, gehen Sie an den Beginn der Datei, und fügen Sie eine Include-Anweisung für die Datei `ModArt.h` gemäß Listing 19.6 hinzu.

Listing 19.6: Die Include-Anweisungen von `CSquiggleCtrl`

```
1: // SquiggleCtrl.cpp: Implementierung der CSquiggleCtrl-ActiveX-
   Steuerelementklasse.
2:
3: #include "stdafx.h"
4: #include "Squiggle.h"
5: #include "SquiggleCtrl.h"
6: #include "SquigglePpg.h"
7: #include "ModArt.h"
```

Eigenschaften hinzufügen

Da die beiden Variablen, die Sie in die Klasse `CModArt` aufgenommen haben, keine Variablen der Steuerelementklasse (`CSquiggleCtrl`) sind, sieht man normalerweise `Get`- und `Set`-Methoden vor, um den Eigenschaftswert abzurufen bzw. zu setzen. Wären diese Variablen Elemente der Steuerelementklasse, könnten Sie sie über den Klassen-Assistenten als Member-Variablen hinzufügen. Sie wüßten dann immer darüber Bescheid, wann und ob sich die Variablen geändert haben, da in der Steuerelementklasse auch eine Benachrichtigungsmethode enthalten wäre, wie bei Änderung des Eigenschaftswerts aufgerufen wird. Weil aber nun die Variablen Elemente einer internen Klasse sind, braucht man etwas mehr Kontrolle über deren Werte.



Selbst wenn die Variablen, die Sie freilegen wollen, Member-Variablen der Steuerelementklasse wären, würde man trotzdem `Get`- und `Set`-Methoden für den Zugriff auf die Variablen als Steuerelementeigenschaften vorsehen. Bei Einsatz von `Get`- und `Set`-Methoden kann man eine Gültigkeitsprüfung für die neuen Eigenschaftswerte vorsehen und damit testen, ob die Container-Anwendung den passenden Wert in die Eigenschaft schreibt.

Um diese Eigenschaften Ihrem Steuerelement hinzuzufügen, öffnen Sie den Klassen- Assistenten und gehen auf die Registerkarte **Automatisierung**, wie es Abbildung 19.2 zeigt. Klicken Sie auf die Schaltfläche **Eigenschaft hinzufügen**, um die erste Eigenschaft aufzunehmen. Im Dialogfeld **Eigenschaft hinzufügen** geben Sie den externen Namen ein, den die Eigenschaft haben soll, beispielsweise `SquiggleLength`, und legen den Typ als `short` fest (der Typ `int` ist nicht verfügbar, nur `short` und `long`). Klicken Sie auf das Optionsfeld **Get/Set-Methoden**. Daraufhin erscheinen im Dialogfeld die Namen für die beiden Methoden, wie es Abbildung 19.3 zeigt. Klicken Sie auf OK, um die Eigenschaft hinzuzufügen.

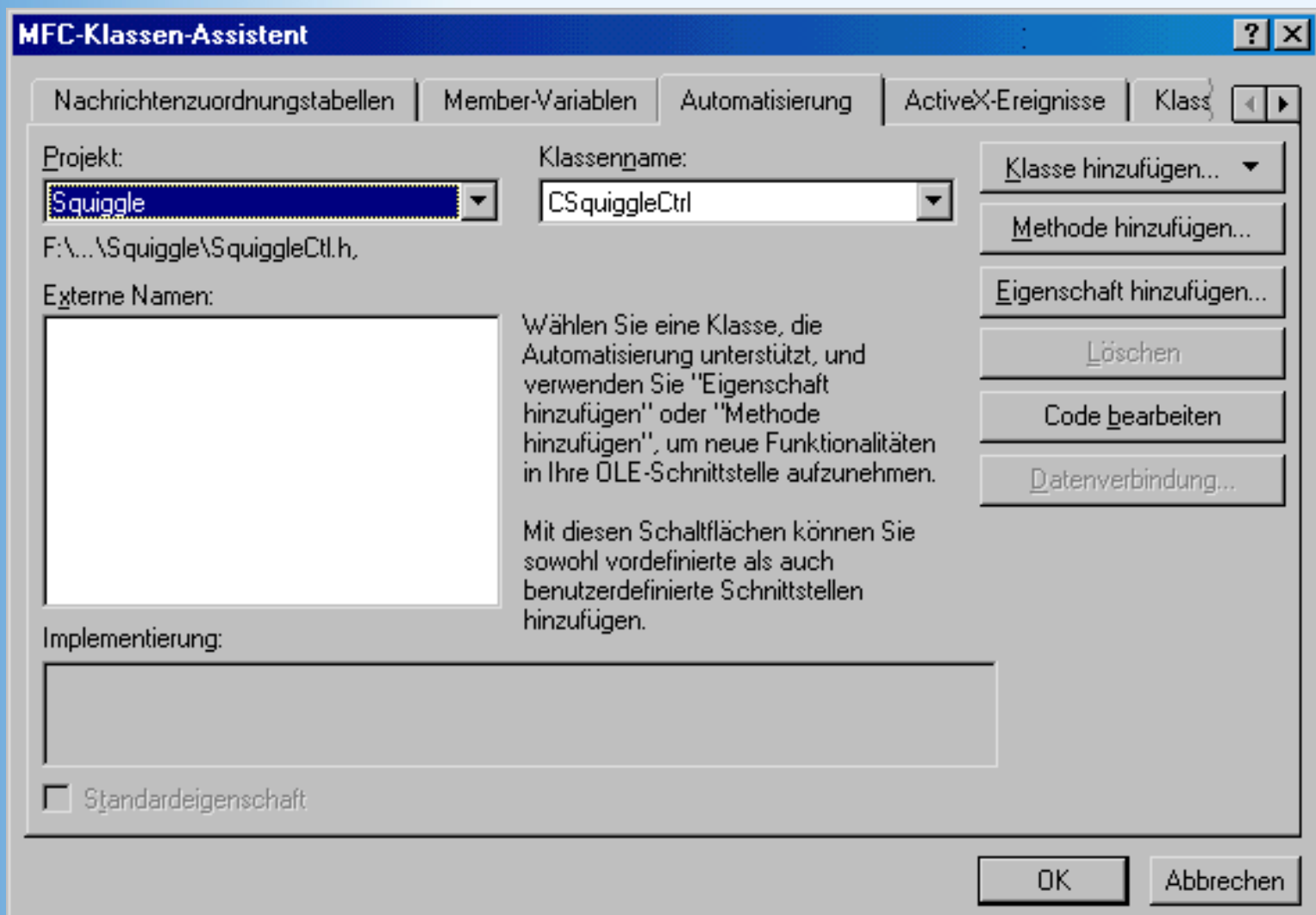


Abbildung 19.2:
Die Registerkarte Automatisierung des Klassen-Assistenten

Eigenschaft hinzufügen

Externer Name: SquiggleLength

Typ: short

Get-Funktion: GetSquiggleLength

Set-Funktion: SetSquiggleLength

Implementierung

☐ Vordefiniert ☐ Member_Variable ☒ Get/Set-Methoden

Parameterliste:

Name	Typ

Abbildung 19.3:
Das Dialogfeld Eigenschaften hinzufügen

Klicken Sie auf die Schaltfläche **Code bearbeiten**, um den Code für die **Get-** und **Set-** Methoden, die Sie der Klasse **CModArt** hinzugefügt haben, um den Zugriff auf die Längenvariable zu steuern, einzugeben. Schreiben Sie den Code aus Listing 19.7 in die beiden Methoden.

Listing 19.7: Die Funktionen Get/SetSquiggleLength

```

1: short CSquiggleCtrl::GetSquiggleLength()
2: {
3: // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
Eigenschaften hinzu
4: // Ergebnis der Funktion GetLength zurückgeben
5: return m_maDrawing.GetLength();
6: }
7:
8: void CSquiggleCtrl::SetSquiggleLength(short nNewValue)
9: {
10: // ZU ERLEDIGEN: Fügen Sie hier Ihre Behandlungsroutine für
Eigenschaften hinzu
11: // Neue Länge setzen
12: m_maDrawing.SetLength(nNewValue);
13: SetModifiedFlag();
14: }

```

Sehen Sie eine weitere Eigenschaft für die Anzahl der Squiggles in der Zeichnung vor, indem Sie die gleichen Schritte ausführen, dabei aber einen entsprechenden Eigenschaftsnamen wie etwa **NumberSquiggles** verwenden.

Als letzte Eigenschaft soll das Steuerelement eine boolesche Eigenschaft erhalten, über die die Container-Anwendung Einfluß auf das Erzeugen neuer Zeichnungen und die Sichtbarkeit der aktuellen Zeichnung hat. Nehmen Sie eine neue Eigenschaft über den Klassen-Assistenten auf, vergeben Sie einen geeigneten Namen wie etwa `KeepCurrentDrawing`, und legen Sie den Typ als `BOOL` fest. Übernehmen Sie die vorgeschlagene Implementierung als Member-Variable, und klicken Sie auf OK. Der Klassen-Assistent nimmt die Variable automatisch in die Steuerelementklasse auf und schreibt auch den erforderlichen Code, um die Variable zu verwalten.

Die Eigenschaftsseite entwerfen und erstellen

Für das Steuerelement ist eine Eigenschaftsseite bereitzustellen, auf die Entwickler zugreifen können, wenn sie mit Ihrem Steuerelement arbeiten. Diese Eigenschaftsseite liefert den Benutzern ein Mittel, um die Eigenschaften des Steuerelements zu setzen, selbst wenn ihre eigenen Entwicklungswerkzeuge keine Einrichtung bieten, um die Eigenschaften in einer anderen Form als per Code zugänglich zu machen.

Eine Eigenschaftsseite läßt sich dem Steuerelement in einfacher Weise hinzufügen. Wenn Sie die Registerkarte **Ressourcen** im Arbeitsbereich wählen und den Ordner **Dialog** erweitern, finden Sie hier bereits ein Dialogfeld für die Eigenschaften des Steuerelements (`IDD_PROPPAGE_SQUIGGLE`). Es handelt sich um ein normales Dialogfeld, das Sie mit den Standardsteuerelementen des Dialog-Editors entwerfen können. Um die Eigenschaftsseite für das Steuerelement des Beispiels zu gestalten, nehmen Sie den Entwurf des Dialogfelds wie in Abbildung 19.4 und mit den Eigenschaftseinstellungen gemäß Tabelle 19.2 vor.



Abbildung 19.4:
Das Layout der Eigenschaftsseite für das Steuerelement

Tabelle 19.2: Eigenschaftseinstellungen für das Steuerelement

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Größte Anzahl von Squiggles:
Eingabefeld	ID	IDC_ENBRSQUIG
Text	ID	IDC_STATIC
	Titel	Größte Länge der Squiggles:
Eingabefeld	ID	IDC_ELENSQUIG
Kontrollkästchen	ID	IDC_CMAINTDRAW
	Titel	Aktuelle Zeichnung verwalten

Nachdem Sie alle Steuerelemente platziert und die Eigenschaften festgelegt haben, öffnen Sie den Klassen-Assistenten, um den Steuerelementen Variablen zuzuordnen. Wenn Sie einem der Steuerelemente im Dialogfeld **Eigenschaften** eine Variable hinzufügen, erscheint ein zusätzliches Kombinationsfeld im Dialogfeld **Member-Variable** hinzufügen. Dieses Kombinationsfeld ist für den externen Namen der Eigenschaft vorgesehen, an die die Variable im Steuerelement gebunden werden soll. Die Dropdown- Liste in diesem Kombinationsfeld enthält alle Standardeigenschaften, die Sie mit der Eigenschaftsseite des Steuerelements verbinden können. Wenn Sie aber das Steuerelement mit einer benutzerdefinierten Eigenschaft verbinden möchten, müssen Sie den Eigenschaftsnamen selbst eingeben, wie es Abbildung 19.5 zeigt.

Abbildung 19.5:
Das Dialogfeld Member-Variable hinzufügen

Fügen Sie Variablen für die Steuerelemente auf der Eigenschaftsseite Ihres Steuerelements hinzu, und binden Sie sie an die Eigenschaften des Steuerelements, wie es Tabelle 19.3 angibt.

Tabelle 19.3: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ	Eigenschaft
IDC_CMAINTDRAW	m_bKeepDrawing	Wert	BOOL	KeepCurrentDrawing
IDC_ELENSQUIG	m_iLenSquig	Wert	int	SquiggleLength
IDC_ENBRSSQUIG	m_iNbrSquiggles	Wert	int	NumberSquiggles

Klicken Sie auf OK, um alle Variablen in die Klasse der Eigenschaftsseite für das Steuerelement aufzunehmen.

Grundlegende Steuerelementfunktionen realisieren

Als grundlegende Funktionalität soll Ihr Steuerelement auf Mausklicks hin eine neue Zeichnung generieren. Um dieses Verhalten zu steuern, nehmen Sie eine zweite boolesche Variable in die Steuerelementklasse auf, damit die Funktion `OnDraw` weiß, daß ein Mausklick ausgelöst wurde. Am einfachsten läßt sich der Zeichenbereich in der Funktion `OnDraw` ermitteln, so daß hier auch die Zeichnungen generiert werden. Soll nun das Steuerelement jedesmal eine neue Zeichnung hervorbringen, wenn der Benutzer die Anwendung mit Ihrem Steuerelement vor eine andere Anwendung verschiebt? Wahrscheinlich nicht. Höchstwahrscheinlich wollen Sie mehr Einfluß auf das Verhalten des Steuerelements ausüben, so daß sich diese zweite boolesche Variable anbietet. Nehmen Sie eine Member-Variable in die Steuerelementklasse (`CSquiggleCtrl`) auf. Legen Sie den Variablentyp als `BOOL`, den Variablennamen mit `m_bGenNewDrawing` und den Zugriff als *Privat* fest.

Bevor Sie den Code eingeben, der die verschiedenen Aufgaben realisiert, müssen Sie zunächst alle Member-Variablen in der Steuerelementklasse initialisieren. Dazu gehört die als Eigenschaft festgelegte Member-Variable `m_keepCurrentDrawing` und die Member-Variable, die Sie gerade hinzugefügt haben, `m_bGenNewDrawing`. Das Steuerelement soll eine neue Zeichnung aufs Geratewohl generieren, und Sie wollen sicherlich nicht irgendwelche Zeichnungen verwalten, außer wenn die Container-Anwendung explizit festlegt, daß eine Zeichnung zu verwalten ist. Die beiden Variablen setzen Sie entsprechend im Konstruktor der Steuerelementklasse, wie es Listing 19.8 zeigt.

Listing 19.8: Der Konstruktor der Klasse `CSquiggleCtrl`

```
1: CSquiggleCtrl::CSquiggleCtrl()
2: {
3:     InitializeIIDs(&IID_DSquiggle, &IID_DSquiggleEvents);
4:
5:     // ZU ERLEDIGEN: Daten der Steuerelementinstanz hier
    initialisieren.
6:     // Variablen initialisieren
7:     m_bGenNewDrawing = TRUE;
8:     m_keepCurrentDrawing = FALSE;
9: }
```

Als nächstes fügen Sie den Code hinzu, um die Squiggle-Zeichnungen zu generieren und anzuzeigen. Für diese Funktionalität bietet sich die Funktion `OnDraw` in der Steuerelementklasse an. Die Funktion wird immer dann aufgerufen, wenn sich das Steuerelement selbst neu zeichnen muß, ob es nun verborgen war oder ein anderes Ereignis über die Funktion `Invalidate` das Neuzeichnen des Steuerelements ausgelöst hat. In der Funktion `OnDraw` können Sie auch gleich ermitteln, ob eine neue Zeichnung zu generieren oder einfach die vorhandene Zeichnung neu darzustellen ist. Man sollte auch daran denken, daß man für das Zeichnen des gesamten Bereichs, den das Steuerelement einnimmt, verantwortlich ist. Das bedeutet, daß Sie den Hintergrund der Squiggle-Zeichnung erstellen müssen, da sonst die Squiggles auf dem erscheinen, was gerade in diesem Anzeigebereich vorhanden ist. (Wer weiß? Vielleicht ist es gerade der Effekt, den Sie erzielen wollen.) Um diese Funktionalität in Ihrem Steuerelement zu realisieren, bearbeiten Sie die Funktion `OnDraw` in der Steuerelementklasse und fügen den Code aus Listing 19.9 hinzu.

Listing 19.9: Die Funktion `OnDraw` der Klasse `CSquiggleCtrl`

```
1: void CSquiggleCtrl::OnDraw(
2: CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
```

```

3: {
4: // ZU ERLEDIGEN: Folgenden Code durch eigene Zeichenfunktion
ersetzen.
5: //pdc->FillRect(rcBounds,
  CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
6: //pdc->Ellipse(rcBounds);
7: // Muß neue Zeichnung generiert werden?
8: if (m_bGenNewDrawing)
9: {
10: // Zeichenbereich für neue Zeichnung festlegen
11: m_maDrawing.SetRect(rcBounds);
12: // Alte Zeichnung löschen
13: m_maDrawing.ClearDrawing();
14: // Neue Zeichnung generieren
15: m_maDrawing.NewDrawing();
16: // Steuerflag zurücksetzen
17: m_bGenNewDrawing = FALSE;
18: }
19: // Hintergrund füllen
20: pdc->FillRect(rcBounds,
  CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
21: // Squiggle-Zeichnung ausgeben
22: m_maDrawing.Draw(pdc);
23: }

```

Schließlich lösen Sie das Steuerelement aus, um eine neue Zeichnung zu generieren, sobald der Benutzer auf das Steuerelement klickt. Dafür ist eine Behandlungsroutine für das Ereignis `OnClick` des Steuerelements erforderlich. Zuerst nehmen Sie aber eine vordefinierte Methode in das Steuerelement auf, um sicherzustellen, daß es die Nachricht `OnClick` erhält. Öffnen Sie dazu den Klassen-Assistenten, und gehen Sie auf die Registerkarte **Automatisierung**. Nehmen Sie in die Steuerelementklasse eine neue Methode auf. Wählen Sie dafür die Methode `DoClick` aus der Dropdown-Liste der vordefinierten Methoden aus, wie es Abbildung 19.6 zeigt. Klicken Sie auf OK, um die Methode in Ihr Steuerelement aufzunehmen, und gehen Sie dann im Klassen-Assistenten auf die Registerkarte **Nachrichtenzuordnungstabellen**. Markieren Sie die Nachricht `OnClick` in der Liste der verfügbaren Nachrichten, und fügen Sie für diese Nachricht eine Behandlungsroutine hinzu. In die `OnClick`-Behandlungsroutine übernehmen Sie den Code aus Listing 19.10.

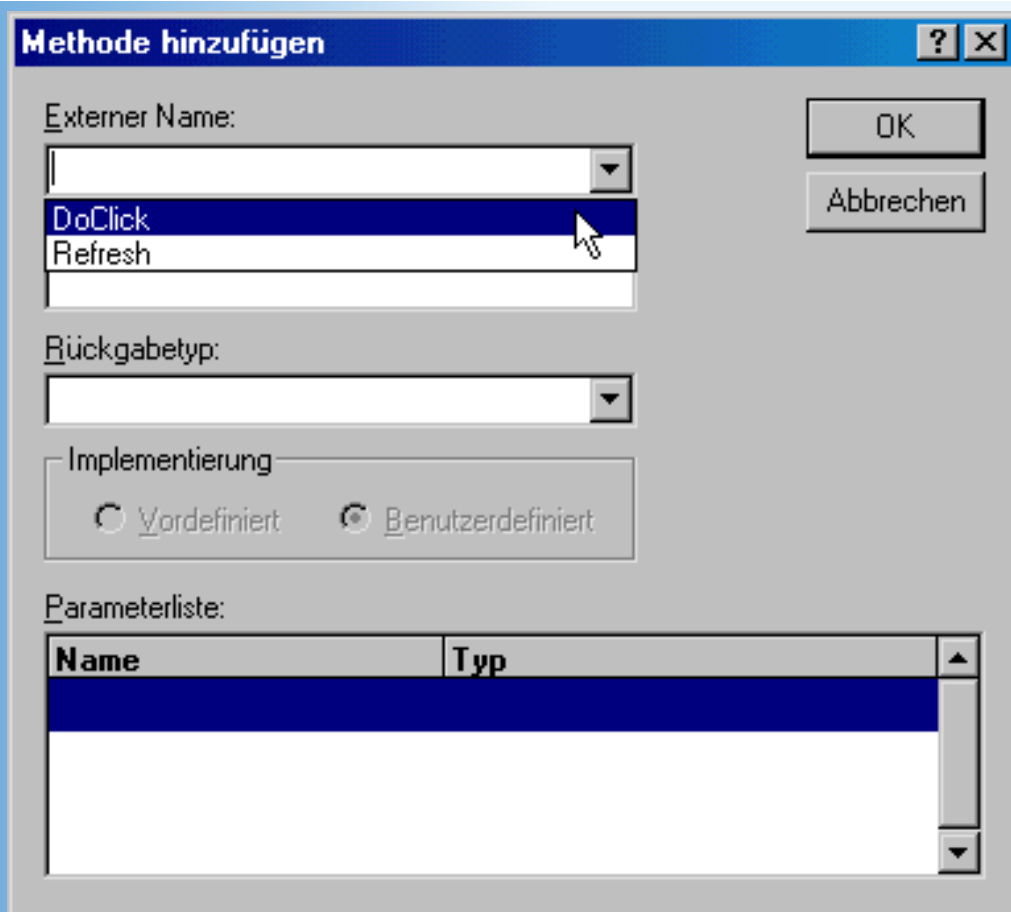


Abbildung 19.6:
Das Dialogfeld Methode hinzufügen

Listing 19.10: Die Funktion OnClick der Klasse CSquiggleCtrl

```

1: void CSquiggleCtrl::OnClick(USHORT iButton)
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
aufrufen
4: // Kann eine neue Zeichnung generiert werden?
5: if (!m_keepCurrentDrawing)
6: {
7: // Flag setzen, so daß neue Zeichnung generiert wird
8: m_bGenNewDrawing = TRUE;
9: // Steuerelement ungültig machen, um Funktion OnDraw auszulösen
10: Invalidate();
11: }
12: COleControl::OnClick(iButton);
13: }

```

Die Funktion `OnClick` prüft zuerst, ob eine neue Zeichnung zu generieren oder die aktuelle Zeichnung zu verwalten ist. Beim Generieren einer neuen Zeichnung wird das Flag `m_bGenNewDrawing` auf `TRUE` gesetzt und das Steuerelement mit der Funktion `Invalidate` ungültig gemacht. Das löst den Aufruf der Funktion `OnDraw` aus.

Methoden hinzufügen

Denken Sie noch einmal an die Funktionalität, die das Steuerelement erhalten soll: Eine der Funktionen lädt eine Squiggle-Zeichnung, die mit dem Squiggle-Modul in der Version von Tag 16 erstellt wurde. Zu diesem Zweck nehmen Sie eine Methode in das Steuerelement auf, die die Container-Anwendung aufrufen kann, um den Namen einer zu ladenden Datei zu übergeben. Der Anwendung haben Sie bereits eine Methode hinzugefügt, und zwar eine vordefinierte Methode. Eine benutzerdefinierte Methode läßt sich fast in der gleichen Weise hinzufügen, man muß aber ein paar mehr Angaben im Dialogfeld **Methode hinzufügen** ausfüllen.

In der Methode zum Laden einer vorhandenen Zeichnung erzeugen Sie ein `CFile`-Objekt für den als Parameter übergebenen Dateinamen. Der Konstruktor von `CFile` übernimmt den Dateinamen und das Flag `CFile::modeRead`, um dem Objekt mitzuteilen, daß die Datei schreibgeschützt zu öffnen ist. Nachdem Sie das `CFile`-Objekt erstellt haben, erzeugen Sie ein `CArchive`-Objekt, um die Datei zu lesen. Der Konstruktor von `CArchive` übernimmt das `CFile`-Objekt, das Sie gerade erzeugt haben, und das Flag `CArchive::load`, um dem Objekt mitzuteilen, daß die Datei zu laden ist. Jetzt können Sie das `CArchive`-Objekt an die Funktion `Serialize` des Zeichenobjekts übergeben und die Zeichnung lesen und laden lassen. Nachdem die Zeichnung geladen ist, zeigen Sie sie an, indem Sie das Steuerelement ungültig machen. Vorher setzen Sie noch das Flag `m_bGenNewDrawing` auf `FALSE`, damit die eben geladene Zeichnung nicht überschrieben wird.

Öffnen Sie den Klassen-Assistenten, und gehen Sie auf die Registerkarte **Automatisierung**. Klicken Sie auf die Schaltfläche **Methode hinzufügen**, um eine benutzerdefinierte Methode aufzunehmen. Geben Sie den externen Namen der Methode in das erste Kombinationsfeld ein, im Beispiel `LoadDrawing`. Der interne Name wird automatisch auf der Basis des eingegebenen externen Namens erzeugt. Legen Sie dann den Rückgabotyp als `BOOL` fest, um die Container-Anwendung darüber informieren zu können, ob das Steuerelement die Zeichnung laden konnte. Schließlich fügen Sie noch einen Parameter in die Parameterliste ein. Legen Sie den Namen als `SFileName` und den Typ als `LPCTSTR` fest (der Typ `CString` ist nicht verfügbar, aber der Typ `LPCTSTR` ist kompatibel), wie es Abbildung 19.7 zeigt. Klicken Sie auf OK, um die Methode in Ihr Steuerelement aufzunehmen. Anschließend klicken Sie auf die Schaltfläche **Code bearbeiten**, um den Code gemäß Listing 19.11 in die Methode zu übernehmen.

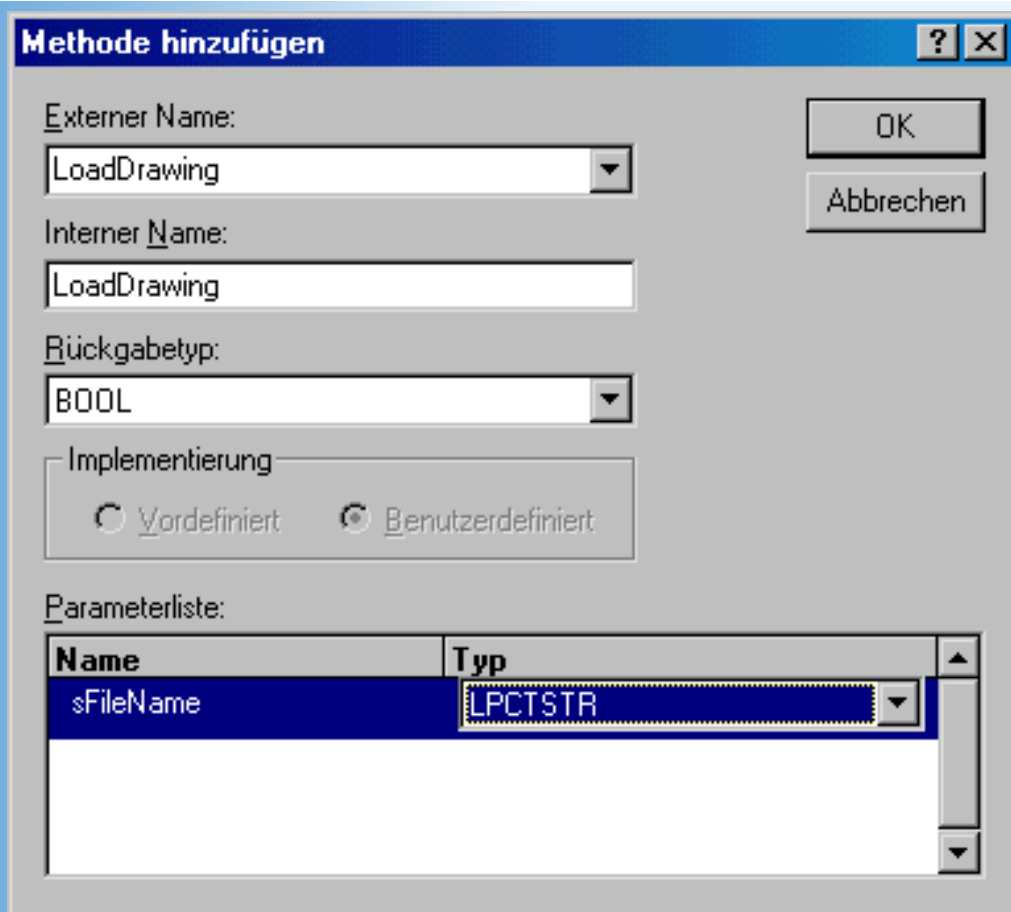


Abbildung 19.7:
Das Dialogfeld Methode hinzufügen für eine benutzerdefinierte Methode

Listing 19.11: Die Funktion LoadDrawing der Klasse CSquiggleCtrl

```

1: BOOL CSquiggleCtrl::LoadDrawing(LPCTSTR sFileName)
2: {
3: // ZU ERLEDIGEN: Fügen Sie hier den Code für Ihre Dispatch-
  Behandlungsroutine ein
4: try
5: {
6: // Ein CFile-Objekt erzeugen
7: CFile lFile(sFileName, CFile::modeRead);
8: // Ein CArchive-Objekt erzeugen, um die Datei zu laden
9: CArchive lArchive(&lFile, CArchive::load);
10: // Datei laden
11: m_maDrawing.Serialize(lArchive);
12: // Sicherstellen, daß geladene Datei nicht überschrieben wird
13: m_bGenNewDrawing = FALSE;
14: // Geladene Zeichnung anzeigen
15: Invalidate();
16: }
17: catch (CFileException err)
18: {
19: return FALSE;
20: }
21: return TRUE;
22: }

```

Ereignisse hinzufügen

Als letzten Schritt beim Erstellen des Steuerelements fügen Sie die Ereignisse hinzu, die das Steuerelement in der Container-Anwendung auslöst. Beim Einsatz des Steuerelements kann der Entwickler Code vorsehen, der bei diesen Ereignissen ausgelöst wird. Die Ereignisse nehmen Sie über die Registerkarte **ActiveX-Ereignisse** des Klassen-Assistenten auf. Wenn Ihr Steuerelement ein vordefiniertes Ereignis auslösen soll, klicken Sie einfach auf die Schaltfläche **Ereignis hinzufügen** und wählen ein vordefiniertes Ereignis aus der Dropdown-Liste der Ereignisse aus. Soll Ihr Steuerelement ein benutzerdefiniertes Ereignis erhalten, wählen Sie im Dialogfeld **Ereignis hinzufügen** kein vordefiniertes Ereignis aus, sondern geben den Namen Ihres benutzerdefinierten Ereignisses ein. In die Parameterliste im unteren Teil des Dialogfelds **Ereignis hinzufügen** tragen Sie die Parameter ein, die Sie mit dem Ereignis von Ihrem Steuerelement an die Container-Anwendung übergeben möchten.

Für das Steuerelement im heutigen Beispiel fügen Sie ein Ereignis hinzu, und zwar ein benutzerdefiniertes Ereignis, um die Anwendung darüber zu informieren, daß die angegebene Datei mit der Zeichnung geladen wurde. Öffnen Sie dazu den Klassen-Assistenten. Gehen Sie auf die Registerkarte **ActiveX-Ereignisse** (siehe Abbildung 19.8), und klicken Sie auf die Schaltfläche **Ereignis hinzufügen**. Geben Sie den Namen `FileLoaded` für das benutzerdefinierte Ereignis ein. Das Dialogfeld **Ereignis hinzufügen** erzeugt automatisch einen internen Namen für das Ereignis, im Beispiel `FireFileLoaded` (siehe Abbildung 19.9). Dieser interne Name ist der Name der Funktion, die Sie in Ihrem Code aufrufen müssen, wenn Sie das Ereignis auslösen möchten. Klicken Sie auf OK, um das Ereignis hinzuzufügen. Um ein vordefiniertes Ereignis hinzuzufügen, markieren Sie das Ereignis in der Dropdown-Liste der vordefinierten Ereignisse und klicken auf OK, um dieses zweite Ereignis hinzuzufügen.

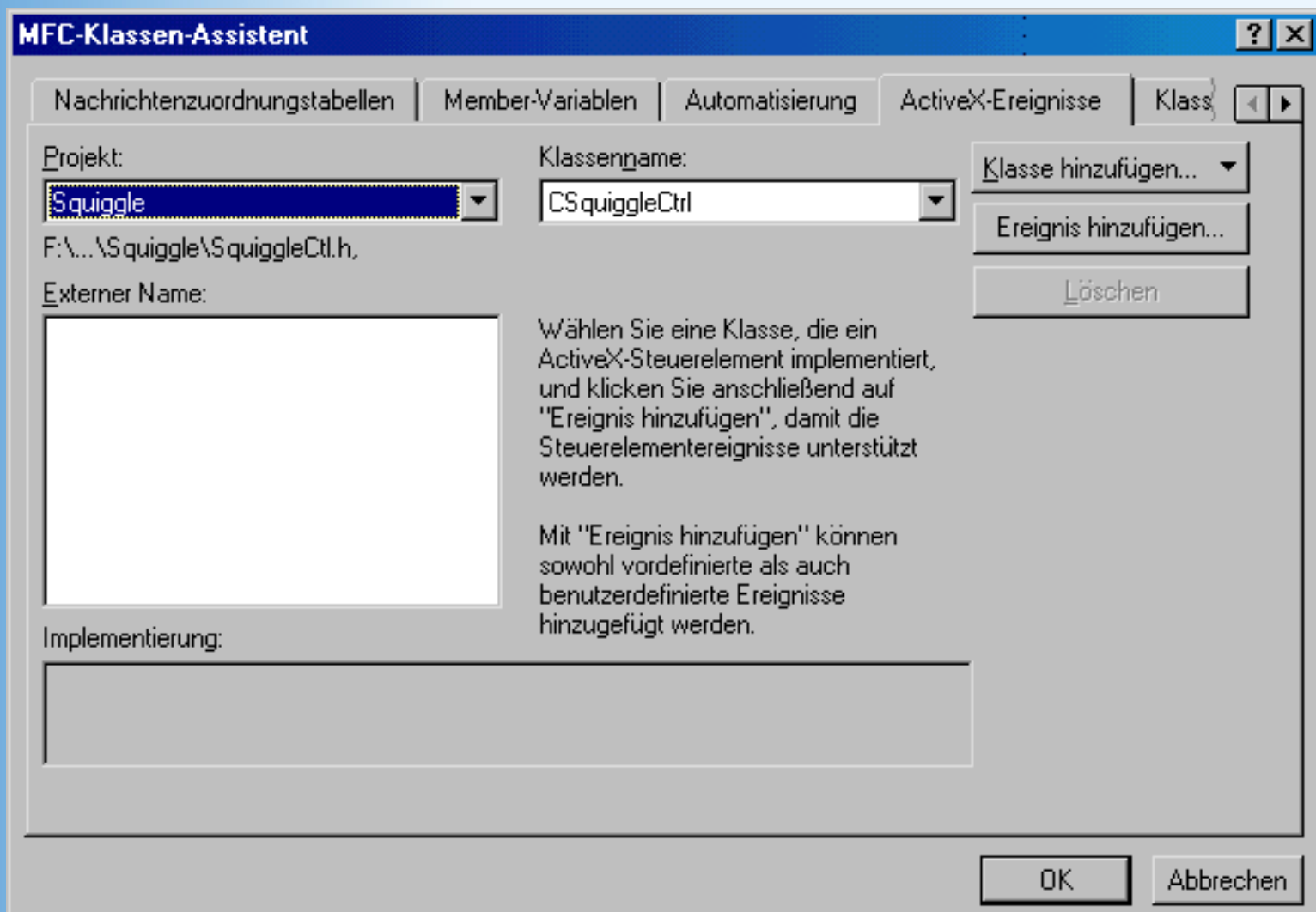


Abbildung 19.8:
Die Registerkarte ActiveX-Ereignisse des Klassen-Assistenten

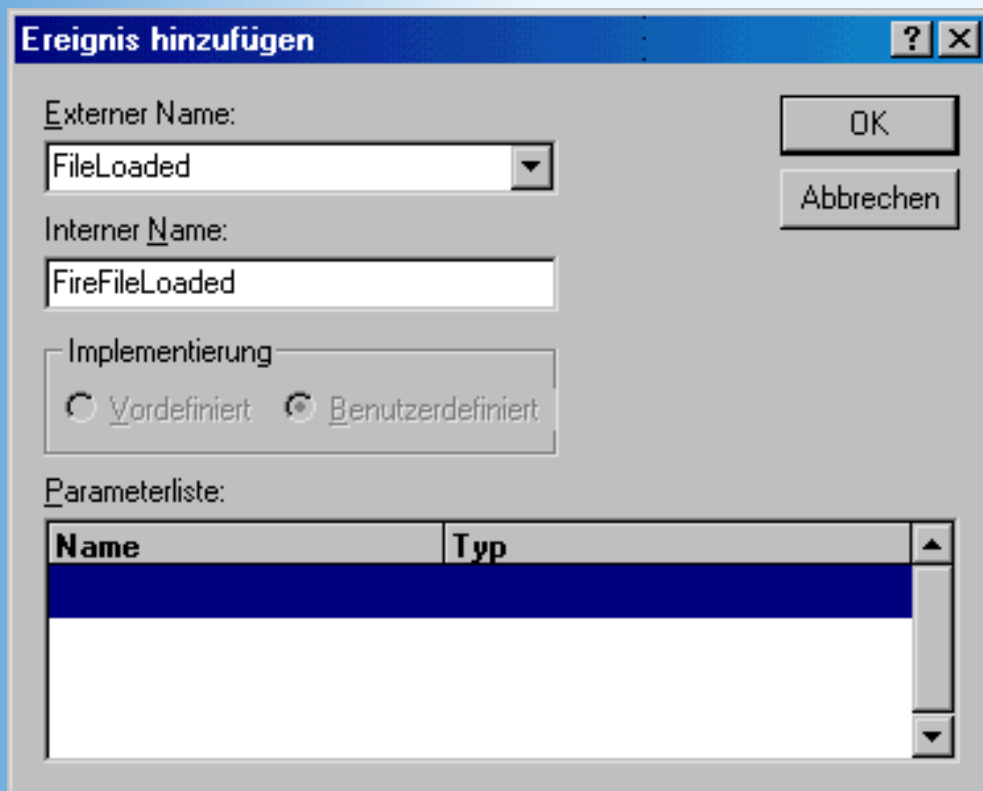


Abbildung 19.9:

Das Dialogfeld Ereignis hinzufügen

Nachdem Sie nun das Ereignis in das Steuerelement eingefügt haben, passen Sie noch den Code an, um das Ereignis an den geeigneten Stellen auszulösen. Das Ereignis lösen Sie am Ende der Funktion LoadDrawing aus, vorausgesetzt, daß die Zeichnung korrekt geladen wurde. Bauen Sie die zusätzliche Funktionalität gemäß Zeile 17 von Listing 19.12 in die Funktion LoadDrawing ein.

Listing 19.12: Die modifizierte Funktion LoadDrawing der Klasse CSquiggleCtrl

```

1: BOOL CSquiggleCtrl::LoadDrawing(LPCTSTR sFileName)
2: {
3: // ZU ERLEDIGEN: Fügen Sie hier den Code für Ihre Dispatch-
  ÂBehandlungsroutine ein
4: try
5: {
6: // Ein CFile-Objekt erzeugen
7: CFile lFile(sFileName, CFile::modeRead);
8: // Ein CArchive-Objekt erzeugen, um die Datei zu laden
9: CArchive lArchive(&lFile, CArchive::load);
10: // Datei laden
11: m_maDrawing.Serialize(lArchive);
12: // Sicherstellen, daß geladene Datei nicht überschrieben wird
13: m_bGenNewDrawing = FALSE;
14: // Geladene Zeichnung anzeigen
15: Invalidate();
16: // Das Ereignis FileLoaded auslösen
17: FireFileLoaded();
18: }
19: catch (CFileException err)
20: {
21: return FALSE;
22: }
23: return TRUE;
24: }
```

Das Steuerelement testen

Damit sind alle Vorbereitungen abgeschlossen. Sie können nun das Steuerelement kompilieren und testen. Sie brauchen aber nun nicht in den nächsten Computerladen rennen, um sich Visual Basic zuzulegen, sondern verfügen bereits über ein Werkzeug zum Testen von ActiveX-Steuerelementen. Im Menü **Extras** finden Sie den Befehl **Testcontainer für ActiveX-Steuerelemente**. Es handelt sich hierbei um ein Hilfsprogramm, das speziell dafür vorgesehen ist, selbst erstellte ActiveX-Steuerelemente auszuprobieren. Nachdem Sie das Steuerelement kompiliert haben, starten Sie den Testcontainer für ActiveX-Steuerelemente, um das Steuerelement auf Herz und Nieren zu prüfen.



Wenn Visual C++ das Steuerelement zwar kompilieren, nicht aber registrieren kann, müssen Sie das Steuerelement manuell registrieren. Wählen Sie **Extras / Steuerelement** registrieren, um das Steuerelement in der Registrierung von Windows anzumelden.

Beim Start des Testcontainers sehen Sie einen leeren Bereich, in dem Ihr Steuerelement erscheinen

wird. Um das Steuerelement in diesen Bereich einzufügen, wählen Sie **Bearbeiten / Neues Steuerelement** einfügen aus dem Menü des Testcontainers. Daraufhin öffnet sich das Dialogfeld **Steuerelement einfügen** (siehe Abbildung 19.10). Markieren Sie das Steuerelement in der Liste der verfügbaren Steuerelemente, und klicken Sie auf OK, um das Steuerelement in den Containerbereich einzufügen, wie es Abbildung 19.11 zeigt.

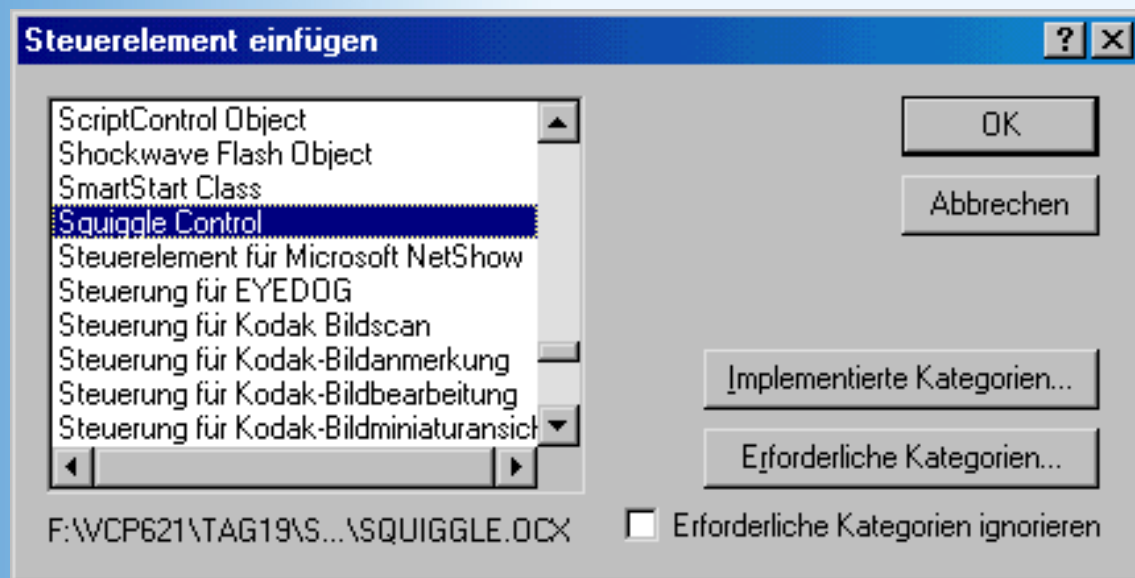


Abbildung 19.10:
Das Dialogfeld ActiveX-Steuerelement einfügen

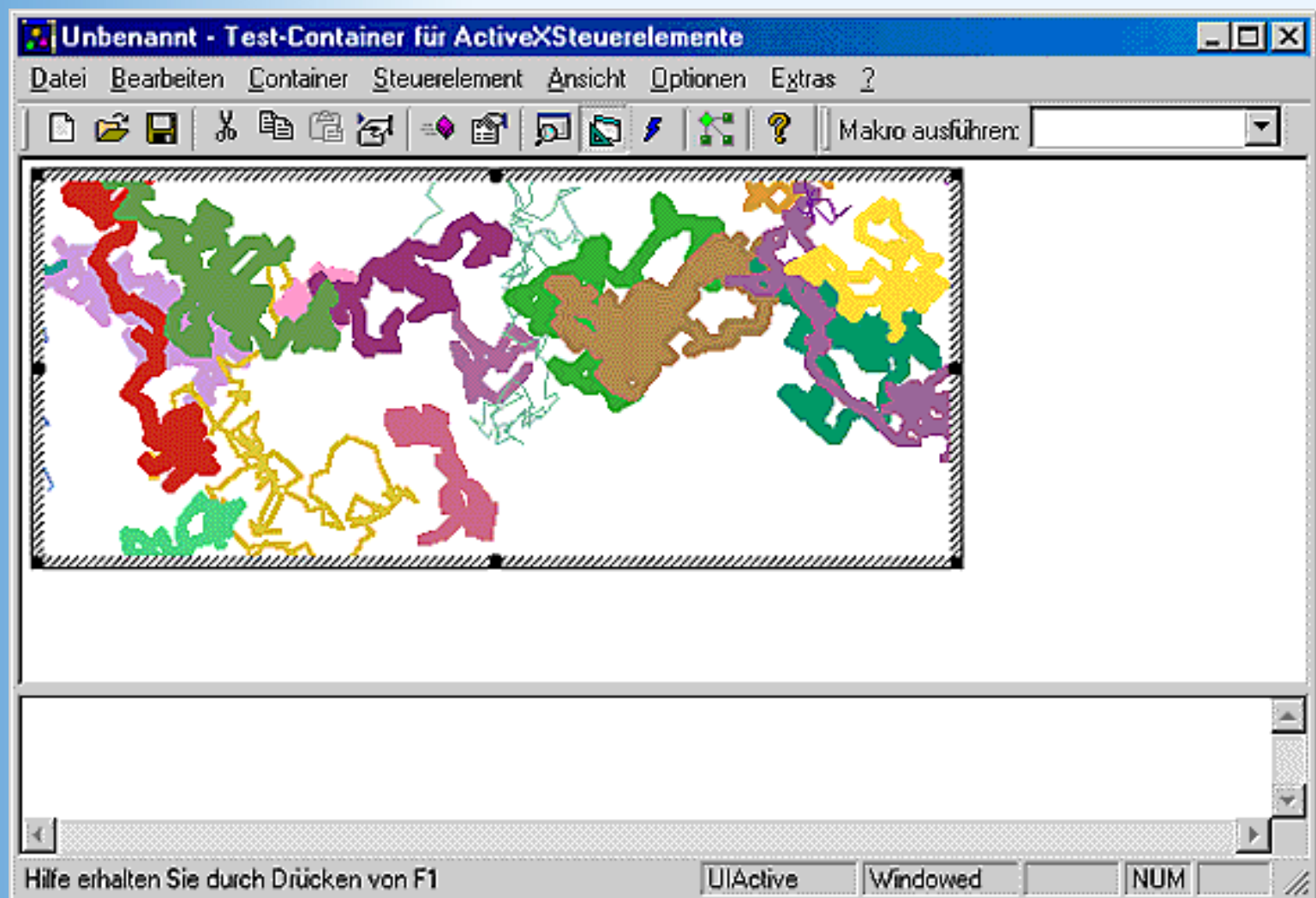


Abbildung 19.11:
Das Squiggle-Steuerelement im Testcontainer

Nachdem Sie nun das Steuerelement in den Testcontainer geladen haben, können Sie es ausprobieren, in der Größe ändern, darauf klicken und prüfen, wann es eine neue Zeichnung generiert und wann es lediglich die vorhandene Zeichnung neu darstellt. Wenn Sie irgendwelche Ereignisse für das Steuerelement auslösen, sehen Sie das vom Steuerelement ausgelöste Ereignis im unteren Fensterbereich des Testcontainers, so daß Sie alle ausgelösten Ereignisse, die Sie für das Steuerelement vorgesehen haben, verfolgen können.

Wenn Sie bei markiertem Steuerelement den Befehl **Bearbeiten / Eigenschaften** wählen, erscheint die Eigenschaftsseite, die Sie für Ihr Steuerelement erstellt haben. Hier können Sie die verschiedenen Eigenschaften des Steuerelements modifizieren, um sich davon zu überzeugen, daß sie korrekt arbeiten (siehe Abbildung 19.12).

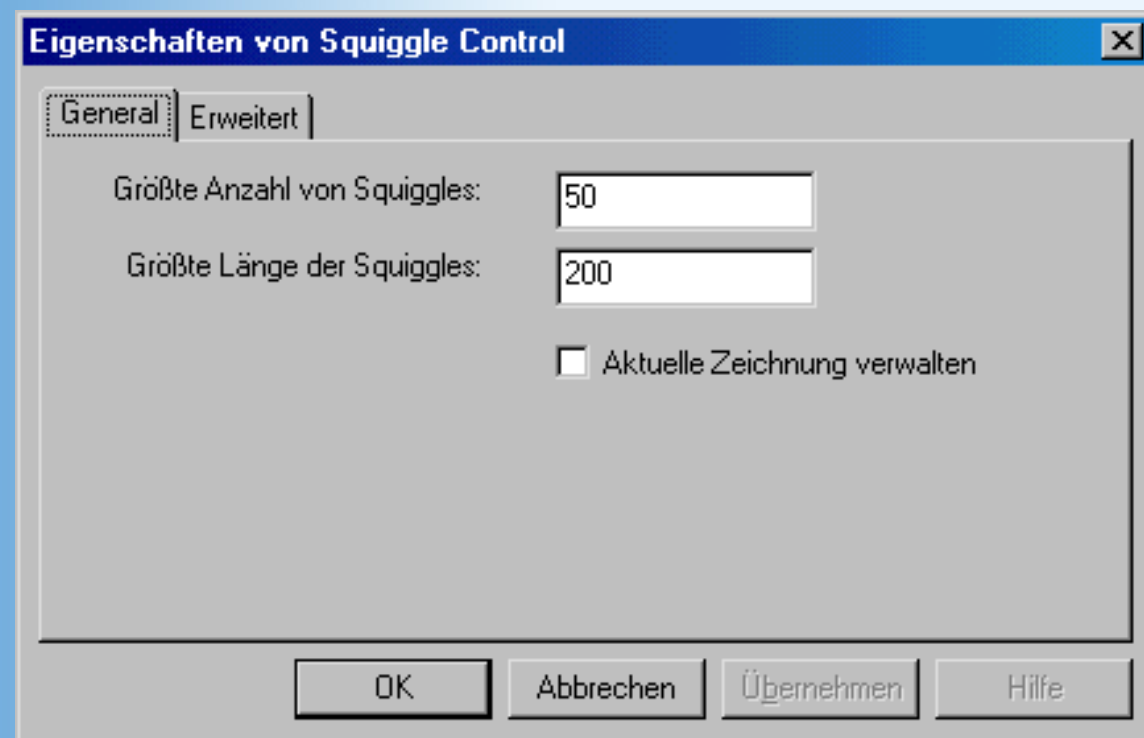


Abbildung 19.12:
Die Eigenschaftsseite des Squiggle-Steuerelements

Um schließlich die in das Steuerelement hinzugefügten Methoden zu testen, wählen Sie **Steuerelement / Methoden** aufrufen. Daraufhin erscheint das Dialogfeld **Methoden aufrufen**, wie in Abbildung 19.13 zu sehen. Hier können Sie aus der Liste der für Ihr Steuerelement verfügbaren Methoden auswählen, die erforderlichen Parameter für die betreffende Methode eingeben und dann auf die Schaltfläche **Aufrufen** klicken, um die betreffende Methode zu starten. Dabei können Sie den Aufruf der Methode und die Reaktion des Steuerelements überwachen.

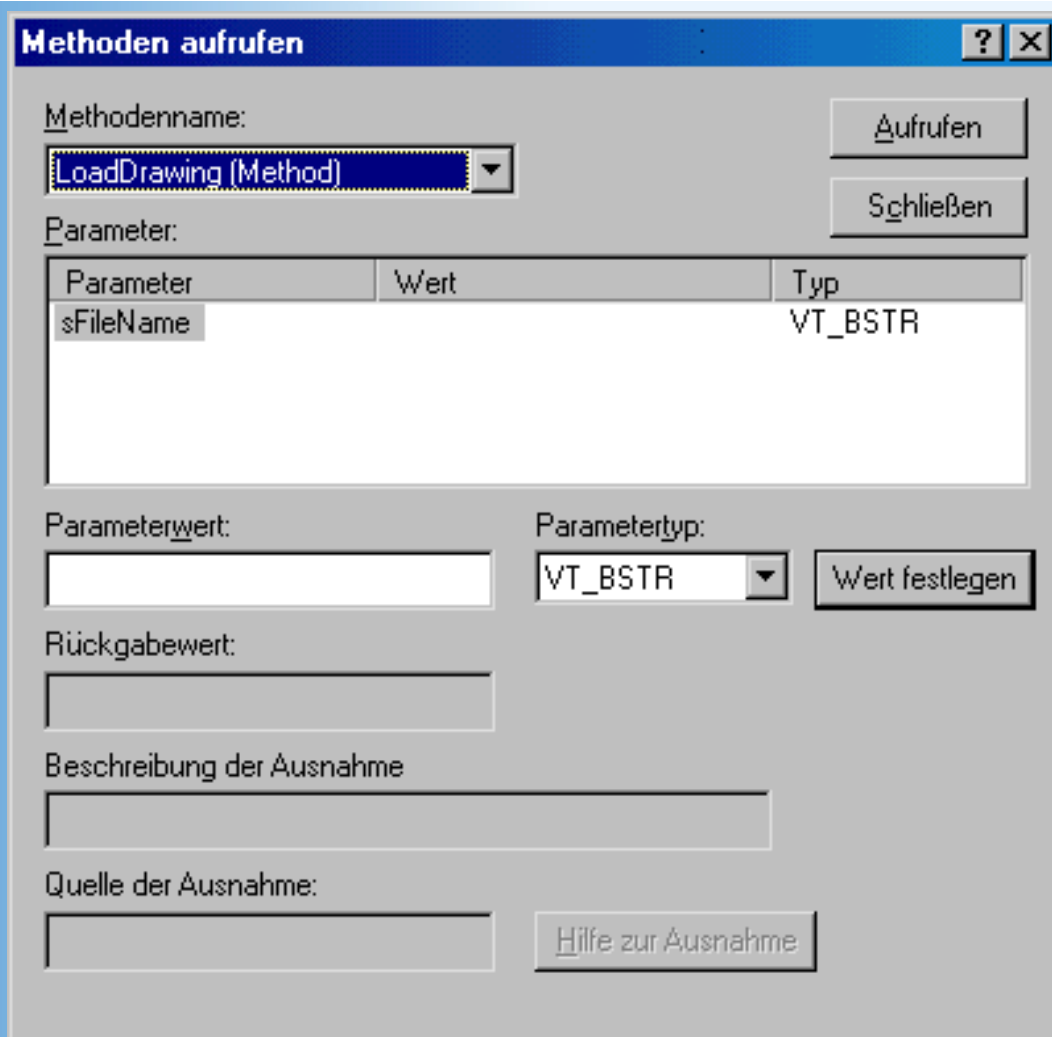


Abbildung 19.13:
Das Dialogfeld Aufrufen der Steuerelementmethode

Zusammenfassung

Heute haben Sie gelernt, wie man die Werkzeuge und Assistenten von Visual C++ einsetzt, um ActiveX-Steuerelemente mit geringem Aufwand von Ihrer Seite zu erstellen. Es wurde gezeigt, wie man das Gerüst des Steuerelementprojekts mit dem MFC- ActiveX-Steuerelement-Assistenten erstellt. Weiterhin haben Sie gesehen, wie man mit dem Klassen-Assistenten Eigenschaften, Methoden und Ereignisse in das Steuerelement aufnimmt. Sie haben eine Eigenschaftsseite für das Steuerelement angelegt und erfahren, wie man mit dem Klassen-Assistenten die Steuerelemente auf diesem Dialogfeld mit den Eigenschaften für das Steuerelement verbindet, ohne daß man dazu Code schreiben müßte. Schließlich haben Sie das Steuerelement im Testcontainer für ActiveX-Steuerelemente ausprobiert und mit den Werkzeugen dieses Programms die gesamte Funktionalität ausgelöst.

Fragen und Antworten

Frage:
Wie ändere ich das Symbol, das in der Toolbox für mein Steuerelement erscheint?

Antwort:
Öffnen Sie über die Registerkarte **Ressourcen** des Arbeitsbereichs den Ordner Bitmap, der ein einzelnes Bitmap enthalten sollte. Dieses Bild wird in der Toolbox für Ihr Steuerelement angezeigt,

wenn Sie es in ein Visual-C++- oder Visual-Basic-Projekt einbinden. Dieses Bitmap können Sie bearbeiten, so daß es Ihr Steuerelement in gewünschter Weise repräsentiert.

Frage:

Warum weist mein Steuerelement ein Dialogfeld Info auf?

Antwort:

Wenn Sie ActiveX-Steuerelemente erstellen, die für andere Entwickler vorgesehen sind, möchten Sie sicherlich irgendwo einen Hinweis anbringen, daß Sie das Steuerelement geschrieben haben und Sie oder Ihr Brötchengeber das Copyright besitzen. Damit haben Sie eine rechtliche Kennzeichnung des Steuerelements geschaffen, so daß niemand Ihr Steuerelement ohne Ihre Zustimmung als seine eigene Kreation verkaufen darf.

Workshop

Kontrollfragen

1. Welche drei Aspekte eines Steuerelements sind für die Container-Anwendung sichtbar?
2. Warum müssen Sie eine Eigenschaftsseite für Ihr Steuerelement entwerfen?
3. Welche vier Arten von Eigenschaften kann ein Steuerelement besitzen?
4. Was passiert mit den Parametern, die an die Methoden eines Steuerelements übergeben werden?
5. Mit welchem Hilfsprogramm können Sie Ihre Steuerelemente testen?

Übungen

1. Nehmen Sie in Ihr Steuerelement eine Methode auf, um der Container-Anwendung zu ermöglichen, das Generieren einer neuen Squiggle-Zeichnung auszulösen.
2. Nehmen Sie in Ihr Steuerelement eine Methode auf, um eine Squiggle-Zeichnung zu speichern. Verwenden Sie die Flags `CFile::modeWrite` und `CArchive::store`, wenn Sie die Objekte `CFile` und `CArchive` erstellen.

Woche 3**Tag 20****Internet und Netzwerke**

Immer mehr Anwendungen besitzen die Fähigkeit, mit anderen Anwendungen über Netzwerke, einschließlich des Internet, zu kommunizieren. Das ist nicht zuletzt auf die explosionsartige Zunahme der Popularität des Internet zurückzuführen. Microsoft hat mit Windows NT und Windows 95 die Netzwerkfähigkeiten in seine Betriebssysteme integriert und damit zu einer allgemeinen Verbreitung in allen Arten von Anwendungen beigetragen.

Einige Anwendungen führen einfache Netzwerkaufgaben aus. Beispielsweise prüfen sie eine Website, ob irgendwelche Updates für das Programm vorhanden sind, und bieten dann dem Benutzer die Option, das Programm mit diesen Updates auf den neuesten Stand zu bringen. Verschiedene Textverarbeitungen formatieren Dokumente als Webseiten und geben damit dem Benutzer die Möglichkeit, die Seiten auf den Webserver zu laden. Es gibt auch Computerspiele, bei denen Benutzer auf der ganzen Welt gegeneinander spielen können und nicht nur für sich selbst am Computer werkeln müssen.

Anwendungen können eine Vielzahl von Netzwerkfunktionen aufweisen, und sie sind alle um die Winsock-Schnittstelle herum aufgebaut. Wenn Sie sich mit der Winsock-Schnittstelle auskennen und wissen, wie man Programme mit der Winsock-Schnittstelle und den MFC-Winsock-Klassen erstellt, eröffnet sich Ihnen dieses wunderbare Reich der Anwendungsprogrammierung, so daß Sie Ihre Programmiermöglichkeiten beträchtlich erweitern können.

Heute lernen Sie, ...

- wie Anwendungen mit der Winsock-Schnittstelle arbeiten, um die Netzwerkverbindungen zwischen zwei oder mehr Computern zu realisieren,
- worin der Unterschied zwischen einer Client- und einer Server-Anwendung liegt und welche Rolle jede beim Einrichten einer Kommunikationsverbindung spielt,
- wie die MFC-Winsock-Klassen das Schreiben von Internet-Anwendungen vereinfachen,
- wie man eine eigene Winsock-Klasse von den MFC-Winsock-Klassen ableitet und damit eine ereignisgesteuerte Netzwerkanwendung erstellt.

Wie funktionieren Netzwerkübertragungen?

Die meisten Anwendungen, die über ein Netzwerk kommunizieren, ob es sich nun um das Internet oder ein kleines lokales Netzwerk handelt, arbeiten nach den gleichen Prinzipien und bauen auf der gleichen Funktionalität auf, um die Datenübertragung zu realisieren. Eine Anwendung wartet auf dem

einen Computer darauf, daß eine andere Anwendung eine Verbindung öffnet. Die erste Anwendung »horcht« auf diese Verbindungsanforderung, genau wie man auf das Klingeln des Telefons hört, wenn man einen Anruf erwartet.

In der Zwischenzeit versucht eine andere Anwendung, höchstwahrscheinlich (aber nicht notwendigerweise) auf einem anderen Computer, sich mit der ersten Anwendung zu verbinden. Dieser Versuch, eine Verbindung zu öffnen, entspricht dem Anrufen eines anderen Telefonteilnehmers. Man wählt die Nummer und hofft, daß der andere das Telefonklingeln am anderen Ende der Leitung hört. Als Anrufer müssen Sie die Rufnummer des Angerufenen kennen. Wenn Sie die Nummer nicht wissen, können Sie sie im Telefonbuch anhand des Namens ermitteln. Analog dazu muß die Anwendung, die eine Verbindung zur ersten Anwendung herzustellen versucht, den Netzwerkstandort - oder die Adresse - der ersten Anwendung kennen.

Nachdem eine Verbindung zwischen den beiden Anwendungen hergestellt ist, lassen sich Nachrichten in beiden Richtungen zwischen den Anwendungen austauschen - auch hier wieder die Analogie zu einem Telefongespräch. Diese Verbindung stellt einen bidirektionalen Kommunikationskanal dar, bei dem beide Seiten Informationen senden können (siehe Abbildung 20.1).

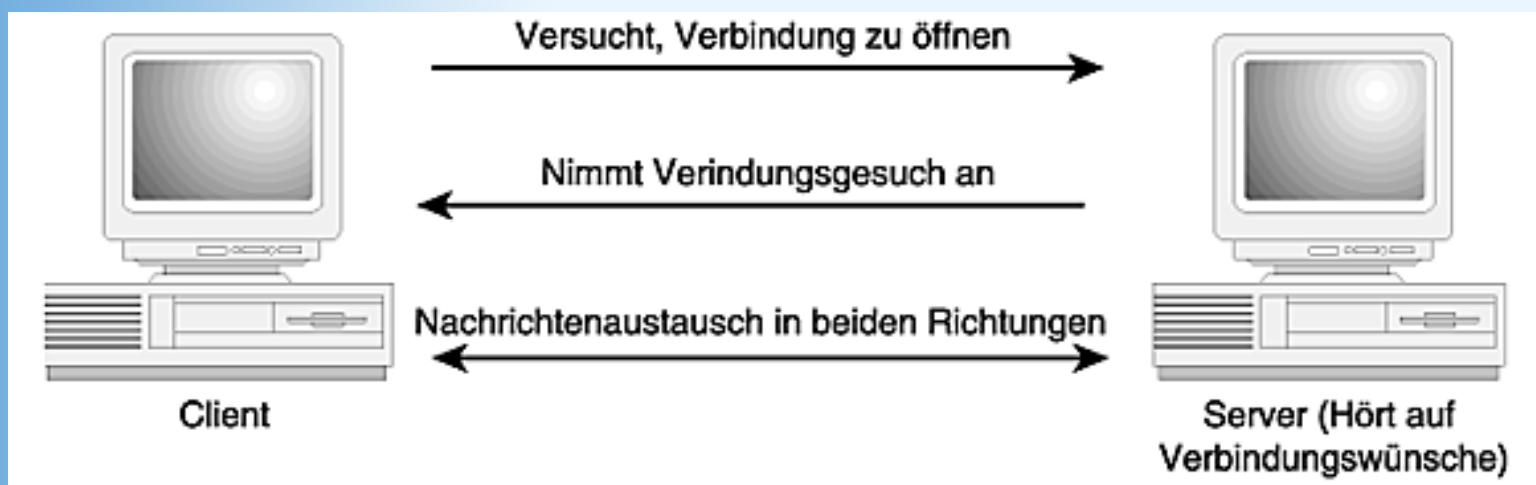


Abbildung 20.1:
Der grundlegende Socket-Verbindungsprozeß

Schließlich beendet eine Seite (oder beide Seiten) die Konversation. Die Verbindung wird geschlossen, so wie Sie den Telefonhörer auflegen, wenn Sie das Gespräch beenden. Nachdem die Verbindung von einer der beiden Seiten geschlossen wurde, erkennt die andere Seite diesen Zustand und schließt ihrerseits die Verbindung, so wie Sie feststellen, daß der andere Telefonteilnehmer aufgelegt hat oder wenn Sie aus anderen Gründen unterbrochen wurden. Grundsätzlich laufen Netzwerkverbindungen zwischen zwei oder mehr Anwendungen nach diesem stark vereinfacht dargestellten Schema ab.



Diese grundlegende Beschreibung bezieht sich auf Netzwerkverbindungen, die nach dem TCP/IP-Protokoll arbeiten, das hauptsächlich als Netzwerkprotokoll im Internet zum Einsatz kommt. Viele andere Netzwerkprotokolle weichen etwas von dieser Beschreibung ab. Andere Protokolle wie etwa das UDP-Protokoll verhalten sich eher wie Rundfunkübertragungen, wo es keine Verbindung zwischen den beiden Anwendungen gibt. Die eine Anwendung sendet Nachrichten, und die andere ist dafür verantwortlich, den Empfang aller dieser Nachrichten sicherzustellen. Diesen Protokollen können wir uns im Rahmen der heutigen Lektion allerdings nicht widmen. Wenn Sie mehr über Netzwerkprotokolle und deren Arbeitsweise erfahren möchten, sollten Sie sich mit der einschlägigen Literatur beschäftigen. Gerade zum Thema Kommunikation gibt es zahlreiche Bücher, die sich auch mit den verschiedenartigen Internet-Anwendungen befassen und die Kommunikation über die eingerichteten Verbindungen beschreiben.

Sockets, Anschlüsse und Adressen

Das grundlegende Objekt, über das eine Anwendung den größten Teil der Netzwerkkommunikation realisiert, bezeichnet man als Socket. Sockets wurden zuerst unter Unix an der Berkeley-Universität entwickelt. Ziel war es, daß Anwendungen den Hauptteil der Netzwerkkommunikation zwischen Anwendungen in der gleichen Weise abwickeln können wie das Lesen und Schreiben von Dateien. Seit dieser Zeit haben sich Sockets natürlich etwas verändert, wobei aber die grundsätzliche Arbeitsweise gleich geblieben ist.

Während der Tage von Windows 3.x, bevor die Netzwerke in das Betriebssystem Windows integriert wurden, konnte man Netzwerkprotokolle für die Datenübertragung von zahlreichen Anbietern erwerben. Jede dieser Firmen ging etwas anders an die Netzwerkübertragung zwischen Anwendungen heran. Im Ergebnis mußte jede Anwendung, die Netzwerkverbindungen realisieren wollte, eine Liste von unterschiedlichen Netzwerkprogrammen beachten, mit denen die Anwendung arbeiten konnte. Viele Anwendungsentwickler waren nicht gerade glücklich mit dieser Situation. Das führte letztendlich dazu, daß die Netzwerkfirmen - einschließlich Microsoft - zusammenkamen und das Winsock-API (Windows Sockets) entwickelten. Damit steht allen Anwendungsentwicklern eine einheitliche Programmierschnittstelle (API - Anwendungsprogrammierschnittstelle) zur Verfügung, die unabhängig von der verwendeten Netzwerksoftware ist.

Wenn man eine Datei lesen oder schreiben möchte, verwendet man ein Dateiobjekt, das auf die Datei zeigt. Obwohl dies in den meisten Visual-C++-Anwendungen, die Sie bisher erstellt haben, vor Ihnen verborgen geblieben ist, mußten Sie für das gestern erzeugte ActiveX-Steuerelement diese Schritte zum Anlegen des Dateiobjekts zum Lesen und Schreiben durcharbeiten. Ein Socket ist damit vergleichbar. Es handelt sich um ein Objekt, das man zum Lesen und Schreiben von Nachrichten verwendet, die zwischen den Anwendungen reisen.

Um eine Socket-Verbindung zu einer anderen Anwendung herzustellen, sind andere Informationen erforderlich als beim Öffnen einer Datei. Wenn man eine Datei öffnen möchte, muß man den Dateinamen und den Standort der Datei kennen. Bei einer Socket-Verbindung muß man wissen, auf welchem Computer die andere Anwendung läuft und über welchen Anschluß die Anwendung hört. Ein Anschluß entspricht etwa einem Telefonanschluß, während die Computeradresse mit der Telefonnummer vergleichbar ist. In gleicher Weise werden Anschlüsse benutzt, um Netzwerkübertragungen weiterzuleiten (siehe Abbildung 20.2). Wie für eine Telefonnummer gibt es auch Mittel, um die Anschlußnummer zu ermitteln, wenn man sie nicht bereits kennt. Das erfordert aber, daß der Computer mit den Angaben konfiguriert ist, über welchen Anschluß die verbindende

Anwendung hört. Wenn man die falsche Computeradresse - oder Anschlußnummer - spezifiziert, erhält man gegebenenfalls eine Verbindung zu einer anderen Anwendung. Das entspricht dem vom Telefon bekannten »falsch verbunden«. Vielleicht erhalten Sie aber auch überhaupt keine Antwort, wenn es am anderen Ende der Leitung keine Anwendung gibt, die hört.

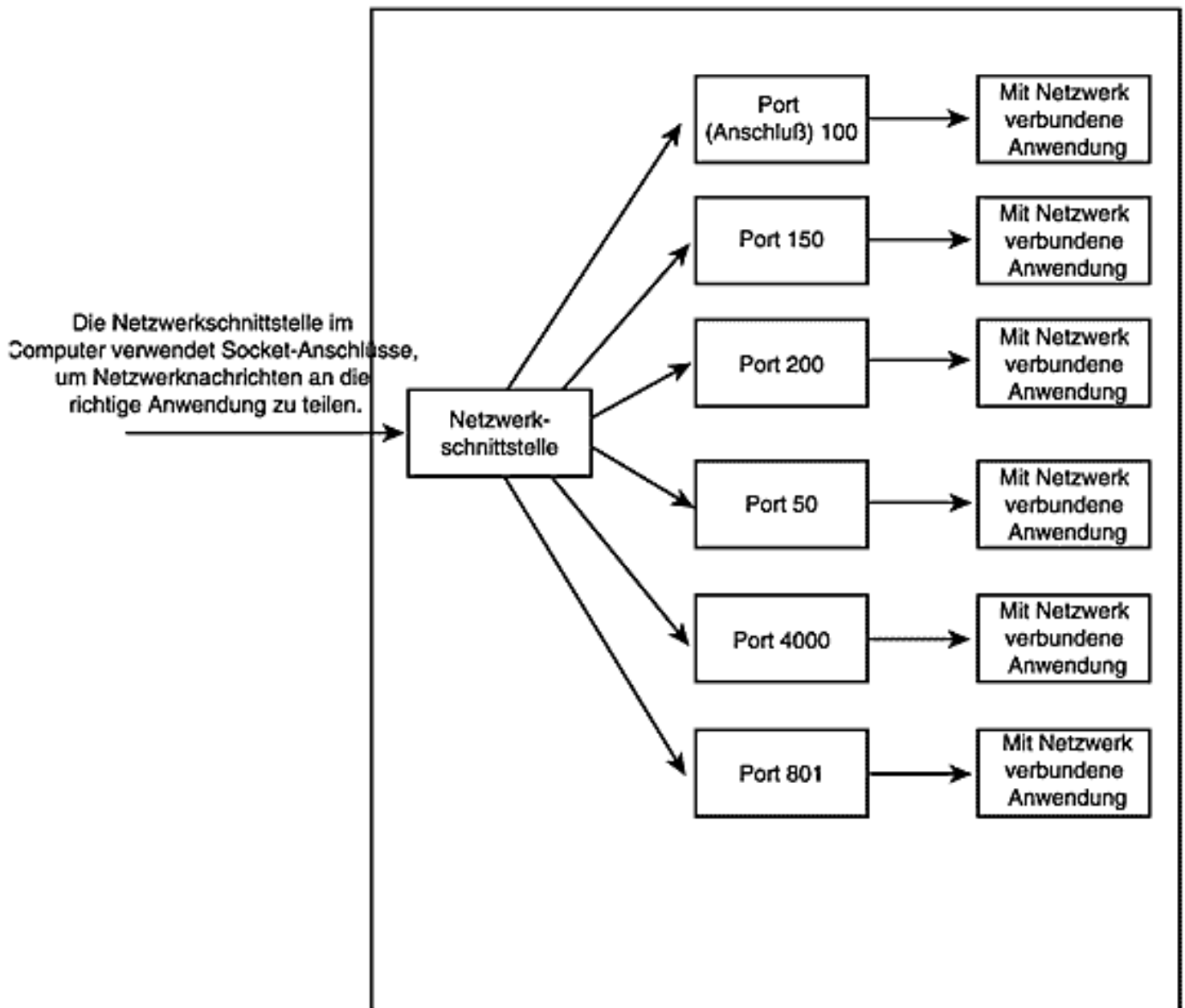


Abbildung 20.2:

Anschlüsse dienen dazu, die Netzwerkverbindungen zur richtigen Anwendung weiterzuleiten.



Nur eine Anwendung kann auf einem einzelnen Computer an einem bestimmten Anschluß hören. Obwohl auch mehrere Anwendungen auf ein und denselben Computer gleichzeitig auf Verbindungsgesuche warten können, müssen diese Anwendungen dazu verschiedene Anschlüsse verwenden.

Einen Socket erstellen

Wenn Sie Anwendungen mit Visual C++ erstellen, können Sie auf die MFC-Winsock- Klassen zurückgreifen, um die Fähigkeiten zur Netzwerkübertragung ziemlich einfach zu realisieren. Die Basisklasse `CAsyncSocket` stellt vollständige, ereignisgesteuerte Verbindungen bereit. Sie können Ihre eigene abgeleitete Socket-Klasse erstellen, die jedes dieser Ereignisse abfangen und darauf antworten kann.



Diese Behandlung der Socket-Kommunikation geht davon aus, daß Sie im Anwendungs-Assistenten die Optionen für das Hinzufügen von Windows Sockets einschalten. Damit wird die unterstützende Funktionalität in die Anwendung aufgenommen, auf die wir hier nicht eingehen.

Um einen Socket zu erstellen, den Sie in Ihrer Anwendung einsetzen können, müssen Sie zunächst eine Variable von `CAsyncSocket` (oder Ihrer davon abgeleiteten Klasse) als Klassenelement für eine der Hauptanwendungsklassen deklarieren:

```
class CMyDlg : public CDialog
{
.
.
.
private:
CAsyncSocket m_sMySocket;
};
```

Bevor Sie das Socket-Objekt verwenden können, müssen Sie dessen `Create`-Methode aufrufen. Diese Methode erstellt den eigentlichen Socket und bereitet ihn zum Einsatz vor. Wie Sie die Methode `Create` aufrufen, hängt von der Verwendungsart des Sockets ab. Wollen Sie - als Client - einen Ruf an eine andere Anwendung absetzen, um eine Verbindung zu dieser Anwendung herzustellen, dann brauchen Sie der `Create`- Methode keinerlei Parameter übergeben:

```
if (m_sMySocket.Create())
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

Wenn allerdings der Socket auf den Verbindungswunsch einer anderen Anwendung hören soll, um sich mit dieser Anwendung zu verbinden, und dabei - als Server - auf den Ruf wartet, dann müssen Sie zumindest die Nummer des Anschlusses übergeben, auf dem der Socket hören soll:

```
if (m_sMySocket.Create(4000))
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

In den Aufruf der `Create`-Methode können Sie weitere Parameter einbinden, wie etwa den Typ des zu erzeugenden Sockets, die Ereignisse, auf die der Socket reagieren soll, und die Adresse, die der

Socket abhören soll (falls der Computer über mehrere Netzwerkkarten verfügt). Alle diese Optionen erfordern ein etwas tiefergehendes Verständnis der Sockets, als wir heute behandeln können.

Eine Verbindung herstellen

Nachdem Sie einen Socket erstellt haben, können Sie damit eine Verbindung öffnen. Das Öffnen einer einzelnen Verbindung verläuft in drei Schritten. Zwei davon finden auf dem Server - der Anwendung, die auf die Verbindung prüft - statt, und der dritte Schritt läuft auf dem Client ab, der den Anruf ausführt.

Auf der Client-Seite ist einfach die Methode `Connect` aufzurufen. Der Client muß zwei Parameter an die Methode übergeben: den Computernamen oder die Netzwerkadresse und den Anschluß der Anwendung, mit der die Verbindung herzustellen ist. Die Methode `Connect` läßt sich auf zwei Arten verwenden. Die erste sieht folgendermaßen aus:

```
if (m_sMySocket.Connect("thatcomputer.com", 4000))
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

Die zweite Form lautet:

```
if (m_sMySocket.Connect("178.1.25.82", 4000))
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

Nachdem die Verbindung hergestellt ist, wird ein Ereignis ausgelöst, um Ihre Anwendung wissen zu lassen, daß sie verbunden ist oder daß es Probleme gegeben hat und sich die Verbindung nicht einrichten läßt. (Auf diese Ereignisse geht der Abschnitt »Socket-Ereignisse« weiter hinten in dieser Lektion näher ein.)

Auf der Server-Seite oder dem Hörer der Verbindung muß die Anwendung zuerst den Socket anweisen, auf hereinkommende Verbindungen zu prüfen. Dazu ist die Methode `Listen` aufzurufen. Die Methode `Listen` übernimmt nur ein einziges Argument, das Sie nicht bereitstellen müssen. Dieser Parameter legt die Anzahl der anhängigen Verbindungen fest, die in eine Warteschlange eingereiht werden können und auf den Abschluß der Verbindung warten. Per Vorgabe ist dieser Wert 5, der das Maximum darstellt. Die Methode `Listen` läßt sich folgendermaßen aufrufen:

```
if (m_sMySocket.Listen())
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

Immer dann, wenn eine andere Anwendung versucht, sich mit der hörenden Anwendung in Verbindung zu setzen, wird ein Ereignis ausgelöst, um der Anwendung mitzuteilen, daß eine Verbindungsanforderung vorliegt. Die hörende Anwendung muß die Verbindungsanforderung durch

Aufruf der Methode `Accept` entgegennehmen. Diese Methode erfordert eine zweite `CAsyncSocket`-Variable, die mit der anderen Anwendung verbunden ist. Nachdem ein Socket in den Hörermodus gebracht wurde, bleibt er in diesem Modus. Immer wenn ein Verbindungsgesuch empfangen wird, erzeugt der hörende Socket einen weiteren Socket, der mit der anderen Anwendung verbunden wird. Für diesen zweiten Socket sollte man nicht die Methode `Create` aufrufen, da die Methode `Accept` den Socket erstellt. Die Methode `Accept` läßt sich folgendermaßen aufrufen:

```
if (m_sMySocket.Accept(m_sMySecondSocket))
{
// Fortfahren
}
else
// Hier Fehlerbehandlung durchführen
```

Zu diesem Zeitpunkt ist die verbindende Anwendung mit dem zweiten Socket der hörenden Anwendung verbunden.

Nachrichten senden und empfangen

Das Senden und Empfangen von Nachrichten über eine Socket-Verbindung ist wenig spektakulär. Da man Sockets einsetzen kann, um alle Arten von Daten zu senden, und sich die Sockets nicht darum kümmern, welche Daten das sind, erwarten die Funktionen zum Senden und Empfangen von Daten einen Zeiger auf einen generischen Puffer. Zum Senden von Daten sollte dieser Puffer die zu sendenden Daten enthalten. Beim Empfangen von Daten nimmt dieser Puffer die empfangenen Daten auf. Solange Sie Strings und Text senden und empfangen, können Sie ziemlich einfache Umwandlungen der Pufferinhalte über `CStrings` vornehmen.

Um eine Nachricht über eine Socket-Verbindung zu versenden, rufen Sie die Methode `Send` auf. Diese Methode erfordert zwei Parameter und einen dritten optionalen Parameter, mit denen man steuern kann, wie die Nachricht zu versenden ist. Der erste Parameter ist ein Zeiger auf den Puffer, der die zu sendenden Daten enthält. Wenn Ihre Nachricht in einer `CString`-Variablen steht, können Sie die `CString`-Variable mit Hilfe des Operators `LPCTSTR` als Puffer übergeben. Der zweite Parameter gibt die Länge des Puffers an. Die Methode liefert die Anzahl der Daten zurück, die an die andere Anwendung gesendet wurden. Wenn ein Fehler auftritt, liefert die Funktion `Send` den Wert `SOCKET_ERROR` zurück. Die Methode `Send` rufen Sie folgendermaßen auf:

```
CString strMyMessage;
int iLen;
int iAmtSent;
.
.
.
iLen = strMyMessage.GetLength();
iAmtSent = m_sMySocket.Send(LPCTSTR(strMyMessage), iLen);
if (iAmtSent == SOCKET_ERROR)
{
// Hier Fehlerbehandlung durchführen
}
else
{
// Alles OK
```

}

Wenn von einer anderen Anwendung empfangene Daten verfügbar sind, wird in der empfangenden Anwendung ein Ereignis ausgelöst. Damit wird der Anwendung mitgeteilt, daß sie die Nachricht empfangen und verarbeiten kann. Um die Nachricht zu erhalten, ist die Methode `Receive` aufzurufen. Diese Methode übernimmt die gleichen Parameter wie die Methode `Send`, mit einem kleinen Unterschied. Der erste Parameter ist ein Zeiger auf einen Puffer, in den die Nachricht kopiert werden kann. Der zweite Parameter gibt die Größe des Puffers an. Aus diesem Wert weiß der Socket, wie viele Daten zu kopieren sind (falls mehr empfangen wurden, als in den Puffer passen). Wie bei der Methode `Send` liefert die Methode `Receive` die Anzahl der in den Puffer kopierten Daten zurück. Wenn ein Fehler auftritt, gibt die Methode ebenfalls den Wert `SOCKET_ERROR` zurück. Wenn die Nachricht, die Ihre Anwendung empfängt, eine Textnachricht ist, können Sie sie direkt in eine `CString`-Variable kopieren. Damit läßt sich die Methode `Receive` wie folgt aufrufen:

```
char *pBuf = new char[1025];
int iBufSize = 1024;
int iRcvd;
CString strRcvd;

iRcvd = m_sMySocket.Receive(pBuf, iBufSize);
if (iRcvd == SOCKET_ERROR)
{
    // Hier Fehlerbehandlung durchführen
}
else
{
    pBuf[iRcvd] = NULL;
    strRcvd = pBuf;
    // Nachricht weiterverarbeiten
}
```



Beim Empfang einer Textnachricht empfiehlt es sich immer, eine `NULL` unmittelbar nach dem letzten empfangenen Zeichen in den Puffer zu schreiben, wie es das obige Beispiel zeigt. Im Puffer können noch willkürliche Zeichen stehen, die Ihre Anwendung vielleicht als Teil der Nachricht interpretiert, wenn Sie den String nicht mit `NULL` abtrennen.

Die Verbindung schließen

Nachdem die Anwendung den Datenaustausch mit der anderen Anwendung beendet hat, kann sie die Verbindung durch Aufruf der Methode `Close` schließen. Die Methode `Close` übernimmt keinerlei Parameter und läßt sich folgendermaßen aufrufen:

```
m_sMySocket.Close();
```



Die Methode `Close` gehört zu den wenigen Methoden von `CAsyncSocket`, die keinen Statuscode zurückgibt. Bei allen bisher untersuchten Member- Funktionen können Sie anhand des Rückgabewerts prüfen, ob ein Fehler aufgetreten ist.

Socket-Ereignisse

Der Hauptgrund, daß Sie Ihre eigene von `CAsyncSocket` abgeleitete Klasse erstellen, liegt darin, die Ereignisse abzufangen, die zum Beispiel beim Empfang einer Nachricht oder beim Beenden der Verbindung auftreten. Die Klasse `CAsyncSocket` verfügt über eine Reihe von Funktionen, die für die jeweiligen Ereignisse aufgerufen werden. Die Funktionen verwenden alle die gleiche Definition - nur der Funktionsname unterscheidet sich - und sind dafür vorgesehen, in den abgeleiteten Klassen überschrieben zu werden. Die Funktionen sind als geschützte (`protected`) Elemente der Klasse `CAsyncSocket` deklariert. Auch in Ihren abgeleiteten Klassen sollten Sie diese Funktionen als geschützt deklarieren. Die Funktionen haben alle einen einzelnen Integer-Parameter, der einen Fehlercode darstellt und den Sie auswerten sollten, um gegebenenfalls auf Fehler zu reagieren. Tabelle 20.1 listet die Ereignisfunktionen zusammen mit den auslösenden Ereignissen auf.

Tabelle 20.1: Überschreibbare Benachrichtigungsfunktionen der Klasse `CAsyncSocket`

Funktion	Ereignisbeschreibung
<code>OnAccept</code>	Diese Funktion wird auf einem hörenden Socket aufgerufen, um zu signalisieren, daß eine Verbindungsanforderung von einer anderen Anwendung auf die Annahme wartet.
<code>OnClose</code>	Diese Funktion wird auf einem Socket aufgerufen, um zu signalisieren, daß die Anwendung am anderen Ende der Verbindung ihren Socket geschlossen hat oder die Verbindung unterbrochen wurde. Daraufhin sollte man den Socket schließen, der diese Nachricht erhalten hat.
<code>OnConnect</code>	Diese Funktion wird auf einem Socket aufgerufen, um zu signalisieren, daß die Verbindung mit einer anderen Anwendung eingerichtet wurde und daß die Anwendung jetzt Nachrichten über den Socket senden und empfangen kann.
<code>OnReceive</code>	Diese Funktion wird aufgerufen, um zu signalisieren, daß Daten über die Socket-Verbindung empfangen wurden und die Daten über die Funktion <code>Receive</code> zum Abruf bereitliegen.
<code>OnSend</code>	Diese Funktion wird aufgerufen, um zu signalisieren, daß der Socket für das Senden von Daten bereit und verfügbar ist. Die Funktion wird unmittelbar im Anschluß an die erfolgreiche Einrichtung der Verbindung aufgerufen. Ansonsten ruft man die Funktion gewöhnlich auf, wenn die Anwendung der Funktion <code>Send</code> mehr Daten übergeben hat, als in ein einziges Paket passen. In diesem Fall ist das ein Signal, daß alle Daten gesendet wurden, und die Anwendung den nächsten mit Daten gefüllten Puffer übertragen kann.

Fehler erkennen

Wenn eine Member-Funktion von `CAsyncSocket` einen Fehler zurückgibt, entweder `FALSE` bei den meisten Funktionen oder `SOCKET_ERROR` bei den Funktionen `Send` und `Receive`, können Sie mit der Methode `GetLastError` den Fehlercode ermitteln. Die Funktion liefert nur Fehlercodes zurück, und Sie müssen sich selbst um eine Übersetzung in Klartext kümmern. Alle Winsock-Fehlercodes sind mit Konstanten definiert, so daß Sie die Konstanten in Ihrem Code verwenden können, um die anzuzeigende Fehlermeldung (falls vorhanden) für den Benutzer zu bestimmen. Die Funktion `GetLastError` läßt sich wie folgt verwenden:

```
int iErrCode;
```



```

iErrCode = m_sMySocket.GetLastError();
switch (iErrCode)
{
case WSA_NOTINITIALISED:
.
.
.
}

```

Eine Netzerkanwendung erstellen

Mit der heutigen Beispielanwendung erstellen Sie eine einfache dialogbasierte Anwendung, die entweder als Client oder als Server in einer Winsock-Verbindung agieren kann. Damit können Sie für jedes Ende der Verbindung zwei Kopien derselben Anwendung starten, entweder auf demselben Computer, oder Sie kopieren die Anwendung auf einen anderen Computer, so daß Sie die beiden Kopien auf getrennten Computern ausführen und die Nachrichtenübertragung über das Netzwerk verfolgen können. Nachdem die Anwendung eine Verbindung mit einer anderen Anwendung eingerichtet hat, können Sie Textnachrichten eingeben und sie an die andere Anwendung verschicken. Wenn die Nachricht gesendet wurde, kommt sie in eine Liste der gesendeten Nachrichten. Jede empfangene Nachricht wird in eine andere Liste aller empfangenen Nachrichten kopiert. Damit können Sie sich die vollständige Liste aller gesendeten und empfangenen Nachrichten ansehen. Außerdem können Sie daraus ersehen, was eine der Anwendungskopien gesendet und was die andere empfangen hat. (Die beiden Listen sollten gleich sein.)

Das Anwendungsgerüst erstellen

Um die heutige Beispielanwendung einfach zu halten, erstellen Sie eine dialogbasierte Anwendung. Alles, was in der heutigen Anwendung zu tun ist, läßt sich genauso einfach auch in einer SDI- oder MDI-Anwendung realisieren. Durch den Einsatz einer dialogbasierten Anwendung halten wir alles, was sonst von der grundlegenden Socket- Funktionalität ablenkt (beispielsweise Fragen, ob die Socket-Variable zur Dokument- oder zur Ansichts-klasse gehört, welche Funktionen der Anwendung in welche der beiden Klassen gehören usw.) von der Beispielanwendung fern.

Erstellen Sie zunächst mit dem MFC-Anwendungs-Assistenten ein Projekt, etwa mit dem Namen **Sock**. Im ersten Dialogfeld des Anwendungs-Assistent wählen Sie die Option **Dialogfeldbasierend**. Im zweiten Schritt des Assistenten schalten Sie das Kontrollkästchen für die Unterstützung von Windows-Sockets ein (siehe Abbildung 20.3). Bei den übrigen Dialogfeldern des Anwendungs-Assistenten können Sie die Standardeinstellungen übernehmen.

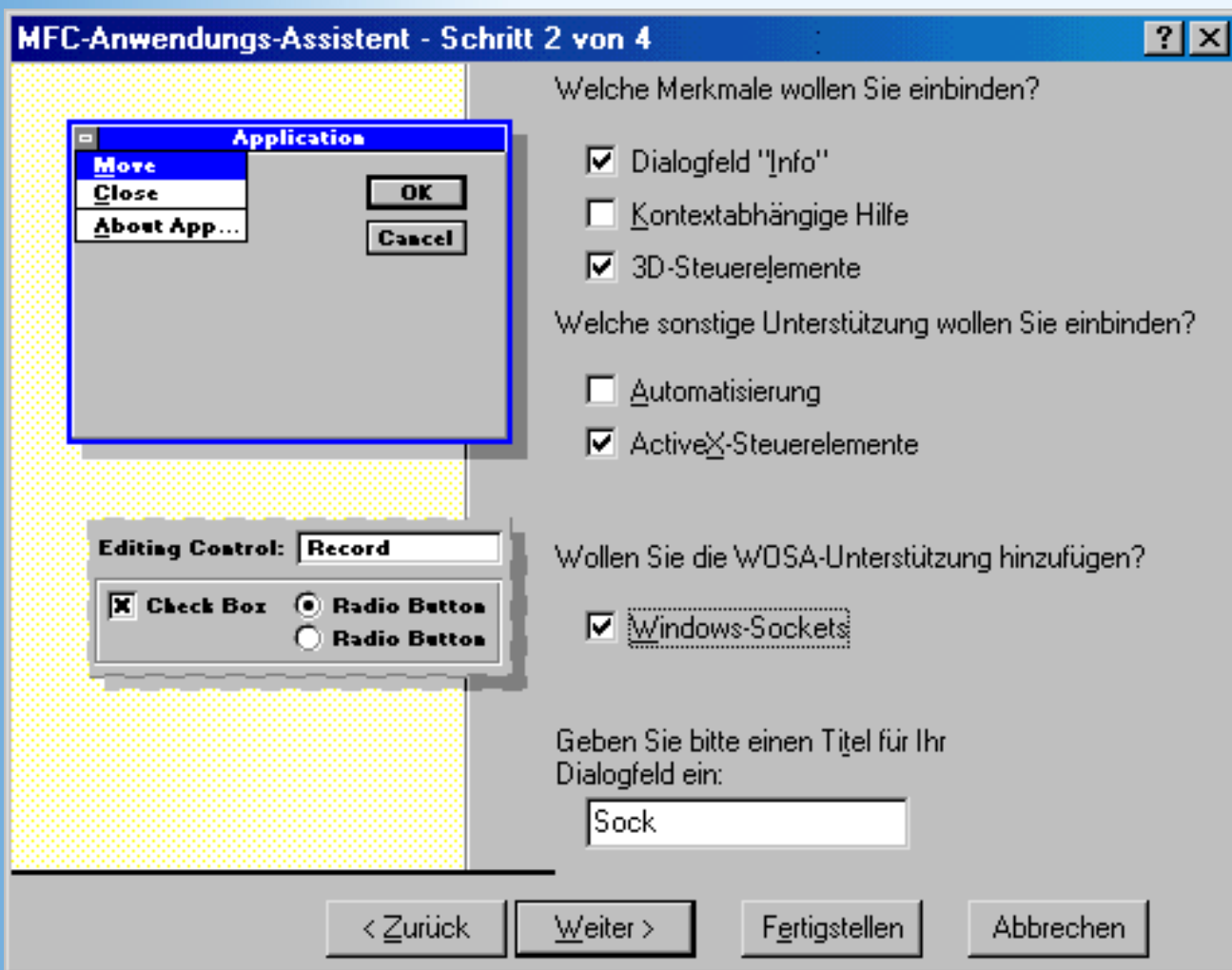


Abbildung 20.3:
Unterstützung für Sockets einbinden

Fenstergestaltung und Funktionalität beim Start

Nachdem Sie über das Anwendungsgerüst verfügen, können Sie das Hauptdialogfeld für Ihre Anwendung entwerfen. In diesem Dialogfeld brauchen Sie eine Gruppe von Optionsfeldern, um festzulegen, ob die Anwendung als Client oder als Server läuft. Außerdem ist eine Reihe von Eingabefeldern für den Computernamen und den Anschluß, den der Server abhören soll, erforderlich. Als nächstes brauchen Sie eine Schaltfläche, um das Hören der Anwendung auf dem Socket zu starten oder die Verbindung zum Server zu öffnen, sowie eine Schaltfläche, um die Verbindung zu schließen. Weiterhin sind ein Eingabefeld für die zu verschickenden Nachricht an die andere Anwendung und eine Schaltfläche zum Senden der Nachricht erforderlich. Schließlich brauchen wir eine Reihe von Listefeldern, in die sich die gesendeten und empfangenen Nachrichten eintragen lassen. Diese Steuerelemente platzieren Sie im Dialogfeld, wie es Abbildung 20.4 zeigt. Die Eigenschaften der Steuerelemente legen Sie gemäß Tabelle 20.2 fest.

Tabelle 20.2: Eigenschaftseinstellungen der Steuerelemente

Objekt	Eigenschaft	Einstellung
Gruppenfeld	ID	IDC_STATICTYPE
	Titel	Socket-Typ

Optionsfeld	ID	IDC_RCLIENT
	Titel	&Client
	Gruppe	eingeschaltet
Optionsfeld	ID	IDC_RSERVER
	Titel	&Server
Text	ID	IDC_STATICNAME
	Titel	Server-&Name:
Eingabefeld	ID	IDC_ESERVERNAME
Text	ID	IDC_STATICPORT
	Titel	Server-&Anschluß:
Eingabefeld	ID	IDC_ESERVPORT
Schaltfläche	ID	IDC_BCONNECT
	Titel	&Verbinden
Schaltfläche	ID	IDC_BCLOSE
	Titel	Schlie&ßen
	Deaktiviert	eingeschaltet
Text	ID	IDC_STATICMSG
	Titel	Nachrich&t:
	Deaktiviert	eingeschaltet
Eingabefeld	ID	IDC_EMSG
	Deaktiviert	eingeschaltet
Schaltfläche	ID	IDC_BSEND
	Titel	Sen&den
	Deaktiviert	eingeschaltet
Text	ID	IDC_STATIC
	Titel	&Gesendet:
Listenfeld	ID	IDC_LSENT
	Tabstopp	ausgeschaltet
	Sortieren	ausgeschaltet
	Auswahl	Keine

Text	ID	IDC_STATIC
	Titel	Empfangen:
Listenfeld	ID	IDC_LRECVD
	Tabstopp	ausgeschaltet
	Sortieren	ausgeschaltet
	Auswahl	Keine

Abbildung 20.4:
Das Layout des Hauptdialogfelds

Nachdem Sie das Dialogfeld fertig entworfen haben, öffnen Sie den Klassen-Assistenten, um die Variablen gemäß Tabelle 20.3 mit den Steuerelementen im Dialogfeld zu verbinden.

Tabelle 20.3: Variablen der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_BCONNECT	m_ctlConnect	Control	CButton
IDC_EMMSG	m_strMessage	Wert	CString
IDC_ESERVNAME	m_strName	Wert	CString
IDC_ESERVPORT	m_iPort	Wert	int
IDC_LRECVD	m_ctlRecvd	Control	CListBox

IDC_LSENT	m_ctlSent	Control	CListBox
IDC_RCLIENT	m_iType	Wert	int

Damit man die Schaltfläche **Verbinden** zweifach nutzen kann, um auch den Server in den Hörermodus zu versetzen, fügen Sie für das Klickereignis der beiden Optionsfelder eine Funktion hinzu, die den Text auf der Schaltfläche in Abhängigkeit von der momentan ausgewählten Option ändert. Um diese Funktionalität in die Anwendung einzubauen, fügen Sie für die Nachricht BN_CLICKED der Steuerelement-ID IDC_RCLIENT eine Funktion namens `OnRType` hinzu. Die gleiche Funktion fügen Sie für die Nachricht BN_CLICKED der Steuerelement-ID IDC_RSERVER hinzu. In die Funktion übernehmen Sie den Code aus Listing 20.1.

Listing 20.1: Die Funktion `OnRType` der Klasse `CSockDlg`

```

1: void CSockDlg::OnRType()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4: // Steuerelemente mit Variablen synchronisieren
5: UpdateData(TRUE);
6: // Welcher Modus ist eingestellt?
7: if (m_iType == 0) // Entsprechenden Text auf der Schaltfläche
festlegen
8: m_ctlConnect.SetWindowText("&Verbinden");
9: else
10: m_ctlConnect.SetWindowText("&Hören");
11: }
```

Wenn Sie die Anwendung jetzt kompilieren und ausführen, sollte sich der Text auf der Schaltfläche ändern und die jeweilige Rolle der Anwendung wiedergeben, je nachdem, welches der beiden Optionsfelder ausgewählt ist (siehe Abbildung 20.5).

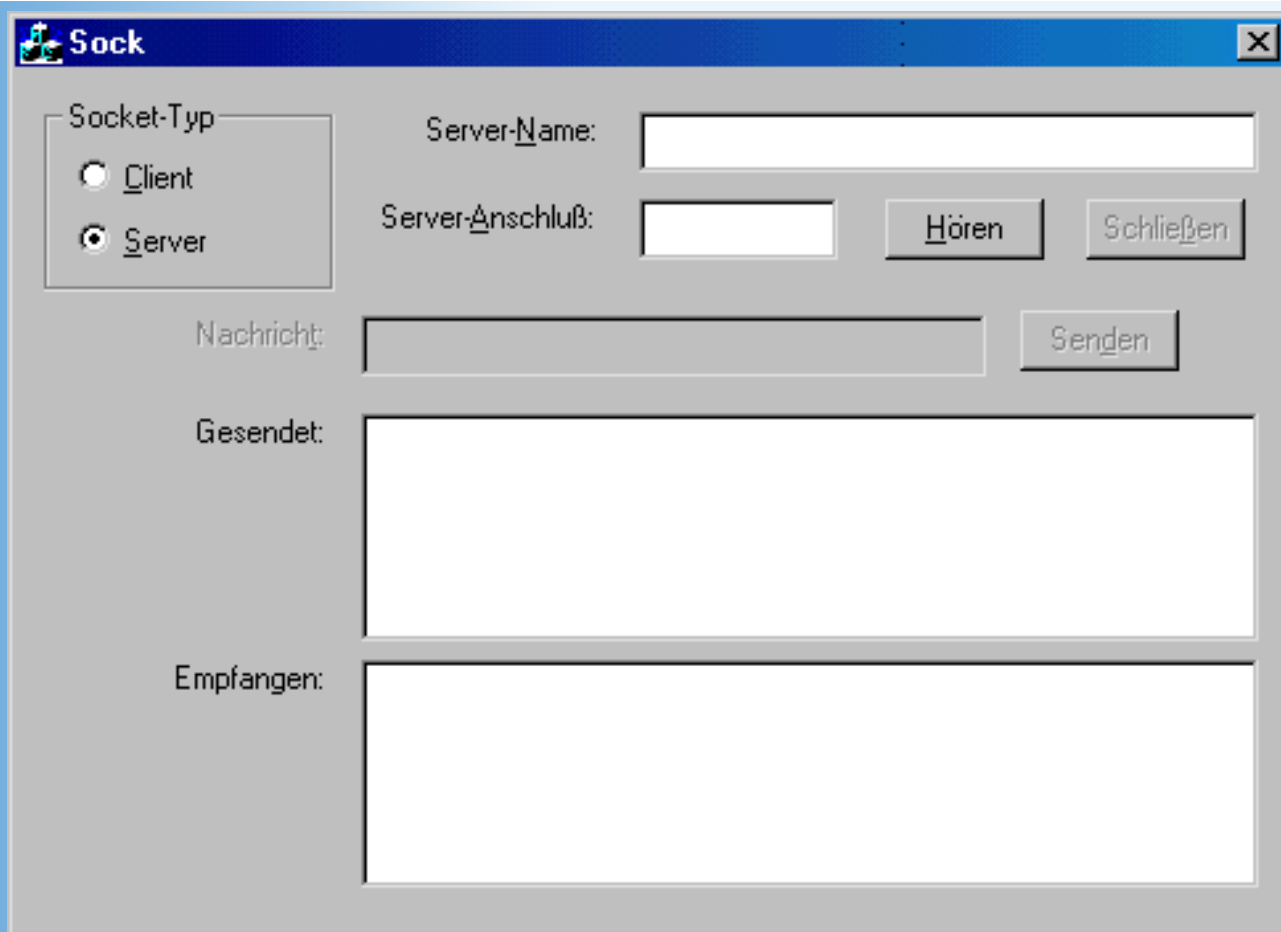


Abbildung 20.5:
Der Schaltflächentext ändert sich je nach gewählter Option

Von der Klasse `CAsyncSocket` erben

Damit Sie die Socket-Ereignisse abfangen und darauf reagieren können, erstellen Sie Ihre eigene Klasse, die von `CAsyncSocket` abgeleitet ist. Diese Klasse benötigt sowohl ihre eigenen Versionen der Ereignisfunktionen als auch die Mittel, das Ereignis an das Dialogfeld weiterzuleiten, zu dem das Objekt gehört. Um jedes dieser Ereignisse auf der Ebene der Dialogfeldklasse übergeben zu können, fügen Sie einen Zeiger auf die übergeordnete Dialogfeldklasse als Elementvariable Ihrer Socket-Klasse hinzu. Über diesen Zeiger rufen Sie Ereignisfunktionen, die Elementfunktionen des Dialogfelds sind, für die Socket-Ereignisse auf, nachdem Sie - selbstverständlich - geprüft haben, daß keine Fehler aufgetreten sind.

Um diese Klasse in Ihrer Anwendung zu erzeugen, wählen Sie den Befehl **Einfügen / Neue Klasse**. Im Dialogfeld **Neue Klasse** übernehmen Sie den Vorgabewert für den Klassentyp **MFC-Klasse**. Geben Sie einen Namen wie etwa `CMySocket` für die Klasse ein, und wählen Sie `CAsyncSocket` aus der Liste der verfügbaren Basisklassen. Mehr können Sie im Dialogfeld **Neue Klasse** nicht festlegen. Klicken Sie also auf OK, um diese neue Klasse in Ihre Anwendung einzubinden.

Nachdem Sie die Socket-Klasse erzeugt haben, nehmen Sie in die Klasse eine Member-Variable auf, die als Zeiger auf das übergeordnete Dialogfeld dient. Legen Sie den Variablentyp als `CDialog*`, den Variablennamen mit `m_pwnd` und den Zugriff als *Privat* fest. Weiterhin müssen Sie der Klasse eine Methode hinzufügen, um den Zeiger zu setzen. Nehmen Sie also in die neue Socket-Klasse eine Member-Funktion auf. Legen Sie den Funktionstyp als `void`, die Deklaration als `SetParent(CDialog* pwnd)` und den Zugriff als *Public* fest. In dieser Funktion setzen Sie den als Parameter übergebenen Zeiger auf den Member-Variablenzeiger, wie es Listing 20.2 zeigt.

Listing 20.2: Die Funktion SetParent der Klasse CMySocket

```

1: void CMySocket::SetParent(CDialog *pWnd)
2: {
3: // Elementzeiger setzen
4: m_pWnd = pWnd;
5: }

```

Das einzige, was Sie noch in Ihre Socket-Klasse aufnehmen müssen, sind die Ereignisfunktionen, über die Sie ähnlich benannte Funktionen in der Dialogfeldklasse aufrufen. Um eine Funktion für das Ereignis `OnAccept` aufzunehmen, fügen Sie Ihrer Socket-Klasse eine Member-Funktion hinzu. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration als `OnAccept(int nErrorCode)` und den Zugriff als *Protected* fest. Schalten Sie außerdem das Kontrollkästchen *Virtual* ein. In die Funktion schreiben Sie den Code aus Listing 20.3.

Listing 20.3: Die Funktion OnAccept der Klasse CMySocket

```

1: void CMySocket::OnAccept(int nErrorCode)
2: {
3: // Sind Fehler aufgetreten?
4: if (nErrorCode == 0)
5: // Nein, OnAccept-Funktion des Dialogfelds aufrufen
6: ((CsockDlg*)m_pWnd)->OnAccept();
7: }

```

Analoge Funktionen fügen Sie der Socket-Klasse für die Funktionen `OnConnect`, `OnClose`, `OnReceive` und `OnSend` hinzu, die gleichnamige Funktionen in der Dialogfeldklasse aufrufen. (Die Funktionen der Dialogfeldklasse realisieren wir später.) Nachdem Sie diese Funktionen erstellt haben, müssen Sie die Header-Datei für das Anwendungsdialogfeld in Ihre Socket-Klasse einbinden, wie es Zeile 7 von Listing 20.4 zeigt.

Listing 20.4: Die einzubindenden Header-Dateien für die Klasse CMySocket

```

1: // MySocket.cpp: Implementierungsdatei
2: //
3:
4: #include "stdafx.h"
5: #include "Sock.h"
6: #include "MySocket.h"
7: #include "SockDlg.h"

```

Nachdem Sie die Ereignisfunktionen in Ihre Socket-Klasse eingebunden haben, nehmen Sie eine Variable Ihrer Socket-Klasse in die Dialogfeldklasse auf. Für die Server- Funktionalität sind zwei Variablen in der Dialogfeldklasse erforderlich - eine zum Hören auf Verbindungsgesuche und die andere zum Herstellen einer Verbindung zur anderen Anwendung. Da Sie zwei Socket-Objekte brauchen, nehmen Sie zwei Member- Variablen in die Dialogfeldklasse (`CsockDlg`) auf. Legen Sie den Typ der beiden Variablen mit Ihrer Socket-Klasse (`CMySocket`) und den Zugriff als *Privat* fest. Die Variable, die für das Hören auf Verbindungswünsche verwendet wird, nennen Sie `m_sListenSocket`, die andere Variable, die für die Übertragung der Nachrichten in beide Richtungen verantwortlich ist, `m_sConnectSocket`.

Nachdem Sie die Socket-Variablen hinzugefügt haben, bauen Sie noch den Initialisierungscode für alle Variablen ein. Als Vorgabe setzen Sie den Anwendungstyp auf *Client*, den Servernamen auf

Loopback und den Anschluß auf 4000. Zusammen mit diesen Variablen setzen Sie die Zeiger des übergeordneten Dialogfelds in Ihren zwei Socket-Objekten, so daß sie auf die Dialogfeldklasse zeigen. Dazu nehmen Sie den Code aus Listing 20.5 in die Funktion `OnInitDialog` der Dialogfeldklasse auf.



Der Computername `loopback` ist ein spezieller Name, der im Netzwerkprotokoll TCP/IP den Computer kennzeichnet, auf dem Sie arbeiten. Es handelt sich um einen internen Computernamen, der in die Netzwerkadresse `127.0.0.1` aufgelöst wird. Diesen Computernamen und diese Adresse verwenden gewöhnlich Anwendungen, die eine Verbindung zu anderen Anwendungen, die auf demselben Computer laufen, herstellen müssen.

Listing 20.5: Die Funktion `OnInitDialog` der Klasse `CSockDlg`

```
1: BOOL CSockDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:     // Hinzufügen des Menübefehls "Info..." zum Systemmenü.
6:
7:     .
8:     .
9:     .
10:
11: 26: SetIcon(m_hIcon, FALSE); // Kleines Symbol verwenden
12: 27:
13: 28: // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen
14: 29: // Steuerelementvariablen initialisieren
15: 30: m_iType = 0;
16: 31: m_strName = "loopback";
17: 32: m_iPort = 4000;
18: 33: // Steuerelemente aktualisieren
19: 34: UpdateData(FALSE);
20: 35: // Socket-Zeiger auf Dialogfeld setzen
21: 36: m_sConnectSocket.SetParent(this);
22: 37: m_sListenSocket.SetParent(this);
23: 38:
24: 39: return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement
25:    soll den ÂFokus erhalten
26: 40: }
```

Die Anwendung verbinden

Wenn der Benutzer auf die Schaltfläche **Verbinden** klickt, deaktivieren Sie alle Steuerelemente im oberen Teil des Dialogfelds. Zu diesem Zeitpunkt soll der Benutzer nicht annehmen, daß er die Einstellungen des Computers, der gerade eine Verbindung aufbaut, oder die Art und Weise, wie die Anwendung hört, ändern kann. Sie rufen die Funktion `Create` auf der jeweiligen Socket-Variablen auf, je nachdem, ob die Anwendung als Client oder als Server läuft. Schließlich rufen Sie entweder die Funktion `Connect` oder die Funktion `Listen` auf, um die Anwendungsseite der Verbindung zu initialisieren. Um diese Funktionalität in Ihre Anwendung aufzunehmen, öffnen Sie den

Klassen-Assistenten und fügen eine Funktion für das Ereignis BN_CLICKED der Schaltfläche **Verbinden** (IDC_BCONNECT) hinzu. In die Funktion übernehmen Sie den Code aus Listing 20.6.

Listing 20.6: Die Funktion OnBconnect der Klasse CSockDlg

```

1: void CSockDlg::OnBconnect()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Âbenachrichtigung hier einfügen
4: // Variablen mit Steuerelementen synchronisieren
5: UpdateData(TRUE);
6: // Steuerelemente für Verbindung und Typ deaktivieren
7: GetDlgItem(IDC_BCONNECT)->EnableWindow(FALSE);
8: GetDlgItem(IDC_ESERVNAME)->EnableWindow(FALSE);
9: GetDlgItem(IDC_ESERVPORT)->EnableWindow(FALSE);
10: GetDlgItem(IDC_STATICNAME)->EnableWindow(FALSE);
11: GetDlgItem(IDC_STATICPORT)->EnableWindow(FALSE);
12: GetDlgItem(IDC_RCLIENT)->EnableWindow(FALSE);
13: GetDlgItem(IDC_RSERVER)->EnableWindow(FALSE);
14: GetDlgItem(IDC_STATICTYPE)->EnableWindow(FALSE);
15: // Läuft Anwendung als Client oder Server?
16: if (m_iType == 0)
17: {
18: // Client, einen Standard-Socket erzeugen
19: m_sConnectSocket.Create();
20: // Verbindung zum Server öffnen
21: m_sConnectSocket.Connect(m_strName, m_iPort);
22: }
23: else
24: {
25: // Server, einen Socket für den angegebenen Anschluß erzeugen
26: m_sListenSocket.Create(m_iPort);
27: // Auf Verbindungsgesuche hören
28: m_sListenSocket.Listen();
29: }
30: }

```

Um die Verbindung zu vervollständigen, fügen Sie in die Dialogfeldklasse die Socket-Ereignisfunktionen für die Ereignisfunktionen **OnAccept** und **OnConnect** hinzu. Diese Funktionen ruft Ihre Socket-Klasse auf. Es sind keine Parameter erforderlich, und die Funktionen liefern auch keine Rückgabewerte. In der Funktion **OnAccept**, die für den hörenden Socket aufgerufen wird, wenn eine andere Anwendung eine Verbindung herstellen will, rufen Sie die Funktion **Accept** des Socket-Objekts auf und übergeben dabei die Socket-Variable für die Verbindung. Nachdem Sie die Verbindung akzeptiert haben, aktivieren Sie die Aufforderung und das Eingabefeld, damit der Benutzer Nachrichten an die andere Anwendung eingeben und abschicken kann.

Fügen Sie zu diesem Zweck der Dialogfeldklasse (CSockDlg) hinzueine Member-Funktion. Legen Sie den Funktionstyp als **void**, die Deklaration mit **OnAccept** und den Zugriff als *Privat* fest. In die Funktion übernehmen Sie den Code aus Listing 20.7.

Listing 20.7: Die Funktion OnAccept der Klasse CSockDlg

```

1: void CSockDlg::OnAccept()
2: {
3: // Verbindungsanforderung annehmen
4: m_sListenSocket.Accept(m_sConnectSocket);
5: // Steuerelemente für Text und Nachrichten aktivieren
6: GetDlgItem(IDC_EMMSG)->EnableWindow(TRUE);
7: GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
8: GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
9: }

```

Nachdem die Verbindung vollständig eingerichtet ist, bleibt auf der Client-Seite nichts weiter zu tun, als die Steuerelemente für die Eingabe und das Senden der Nachrichten zu aktivieren. Weiterhin aktivieren Sie die Schaltfläche **Schließen**, damit sich die Verbindung von der Client-Seite (nicht jedoch von der Server-Seite) schließen lässt. Dazu fügen Sie der Dialogfeldklasse (CSockDlg) eine weitere Member-Funktion hinzu. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration als `OnConnect` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code aus Listing 20.8.

Listing 20.8: Die Funktion OnConnect der Klasse CSockDlg

```

1: void CSockDlg::OnConnect()
2: {
3: // Steuerelemente für Text und Nachrichten aktivieren
4: GetDlgItem(IDC_EMMSG)->EnableWindow(TRUE);
5: GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
6: GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
7: GetDlgItem(IDC_BCLOSE)->EnableWindow(TRUE);
8: }

```

Falls sich die Anwendung jetzt schon kompilieren und ausführen ließe, könnten Sie zwei Kopien der Anwendung starten, eine davon in den Hörermodus setzen und zu dieser mit der anderen Anwendung eine Verbindung herstellen. Leider können Sie wahrscheinlich noch nicht einmal die Anwendung kompilieren, da die Socket-Klasse nach verschiedenen Funktionen in Ihrer Dialogfeldklasse sucht, die Sie noch nicht hinzugefügt haben. Nehmen Sie die noch fehlenden drei Member-Funktionen in die Dialogfeldklasse (CSockDlg) auf. Legen Sie alle drei Funktionen als `void` mit Zugriffsstatus *Public* fest. Die Deklaration der ersten Funktion geben Sie mit `OnSend`, die der zweiten Funktion mit `OnReceive` und die der dritten mit `OnClose` an. Nun sollte sich die Anwendung kompilieren lassen.

Nachdem Sie die Anwendung erfolgreich kompiliert haben, starten Sie zwei Kopien und platzieren Sie nebeneinander auf dem Bildschirm. Die eine Anwendung legen Sie als Server fest und klicken auf die Schaltfläche **Hören**, um sie in den Hörermodus zu versetzen. Die andere Anwendung belassen Sie als Client und klicken auf die Schaltfläche **Verbinden**. Die Verbindungssteuerelemente sollten nun deaktiviert und die Steuerelemente zum Senden der Nachrichten aktiviert sein, nachdem die Verbindung hergestellt ist, wie es Abbildung 20.6 zeigt.

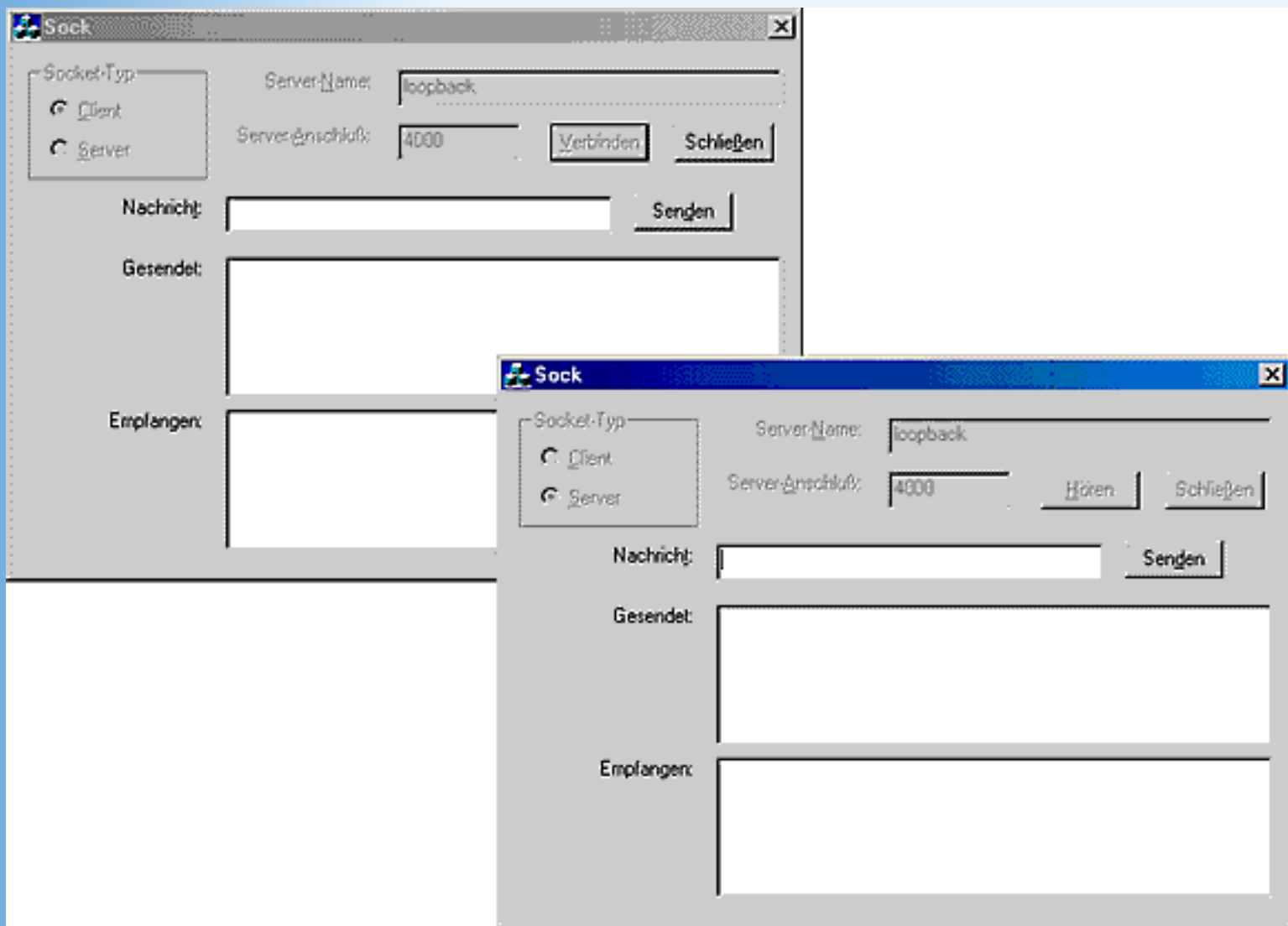


Abbildung 20.6:
Die beiden Anwendungen verbinden



Stellen Sie sicher, daß die Server-Anwendung bereits hört, bevor Sie versuchen, die Verbindung mit der Client-Anwendung herzustellen. Wenn Sie mit dem Client eine Verbindung aufbauen wollen und der Server noch nicht hört, wird die Verbindung zurückgewiesen. Ihre Anwendung erkennt aber nicht, daß die Verbindung abgelehnt wurde, da Sie bisher keinerlei Fehlerbehandlung für dieses Ereignis vorgesehen haben.



Um die Anwendungen ausführen und verbinden zu können, muß TCP/IP auf Ihrem Computer laufen. Diese Bedingung ist wahrscheinlich schon erfüllt, wenn Ihr Computer mit einer Netzwerkkarte ausgestattet ist. Haben Sie keine Netzwerkkarte und bauen Verbindungen zum Internet über ein Modem auf, müssen Sie sich wahrscheinlich mit dem Internet verbinden, wenn Sie die Anwendungen ausführen und testen. Falls Sie die Verbindung zum Internet per Modem abwickeln, startet Ihr Computer gewöhnlich TCP/IP, nachdem die Internet-Verbindung hergestellt ist. Wenn Sie weder über eine Netzwerkkarte verfügen, noch eine Möglichkeit haben, sich mit dem Internet oder einem anderen Netzwerk zu verbinden, das Ihnen die Ausführung von Netzanwendungen gestattet, können Sie die heutigen Beispielanwendungen nicht auf Ihrem Computer ausführen und testen.

Senden und Empfangen

Wenn Sie die beiden laufenden Anwendungen verbinden können, müssen Sie nun noch die Funktionalität hinzufügen, um Nachrichten zu senden und zu empfangen. Nachdem die Verbindung zwischen den Anwendungen eingerichtet ist, kann der Benutzer Textnachrichten in das Eingabefeld in der Mitte des Dialogfelds eingeben und auf die Schaltfläche **Senden** klicken, um die Nachricht an die andere Anwendung abzuschicken. Nachdem die Nachricht gesendet wurde, erscheint sie im Listefeld der gesendeten Nachrichten. Die Anwendung hat also zu prüfen, ob beim Klicken auf die Schaltfläche **Senden** eine zu sendende Nachricht vorhanden ist. In diesem Fall muß die Anwendung die Länge der Nachricht ermitteln, die Nachricht senden und sie dann in das Listefeld der gesendeten Nachrichten eintragen. Um diese Funktionalität in Ihrer Anwendung zu realisieren, fügen Sie mit dem Klassen-Assistenten eine Funktion für das Klickereignis der Schaltfläche **Senden** (IDC_BSEND) hinzu. In die Funktion schreiben Sie den Code gemäß Listing 20.9.

Listing 20.9: Die Funktion OnBsend der Klasse CSockDlg

```

1: void CSockDlg::OnBsend()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4: int iLen;
5: int iSent;
6:
7: // Steuerelemente mit Variablen synchronisieren
8: UpdateData(TRUE);
9: // Zu sendende Nachricht vorhanden?
10: if (m_strMessage != "")
11: {
12: // Länge der Nachricht ermitteln
13: iLen = m_strMessage.GetLength();
14: // Nachricht senden
15: iSent = m_sConnectSocket.Send(LPCTSTR(m_strMessage), iLen);
16: // Konnte die Nachricht gesendet werden?
17: if (iSent == SOCKET_ERROR)
18: {
19: }
20: else
21: {
22: // Nachricht in Listefeld hinzufügen.
23: m_ctlSent.AddString(m_strMessage);
24: // Variablen mit Steuerelementen synchronisieren
25: UpdateData(FALSE);
26: }
27: }
28: }
```

Wenn die Ereignisfunktion **OnReceive** ausgelöst wird, weil eine Nachricht angekommen ist, rufen Sie die Nachricht aus dem Socket über die Funktion **Receive** ab. Dann konvertieren Sie die Nachricht in einen **CString** und fügen sie dem Listefeld der empfangenen Nachrichten hinzu. Um diese Funktionalität zu realisieren, übernehmen Sie den Code aus Listing 20.10 in die Funktion

OnReceive.

Listing 20.10: Die Funktion OnReceive der Klasse CSockDlg

```

1: void CSockDlg::OnReceive()
2: {
3: char *pBuf = new char[1025];
4: int iBufSize = 1024;
5: int iRcvd;
6: CString strRcvd;
7:
8: // Nachricht empfangen
9: iRcvd = m_sConnectSocket.Receive(pBuf, iBufSize);
10: // Wurde etwas empfangen?
11: if (iRcvd == SOCKET_ERROR)
12: {
13: }
14: else
15: {
16: // Ende der Nachricht abschneiden
17: pBuf[iRcvd] = NULL;
18: // Nachricht in einen CString kopieren
19: strRcvd = pBuf;
20: // Nachricht in Listenfeld der empfangenen Nachrichten eintragen
21: m_ctlRcvd.AddString(strRcvd);
22: // Variablen mit den Steuerelementen synchronisieren
23: UpdateData(FALSE);
24: }
25: }

```

Jetzt sollten Sie die Anwendung kompilieren und zwei Kopien ausführen können, wobei Sie die beiden Anwendungen - wie bereits weiter oben geschehen - miteinander verbinden. Nachdem Sie die Verbindung eingerichtet haben, können Sie eine Nachricht in die eine Anwendung eingeben und sie an die andere Anwendung schicken, wie es Abbildung 20.7 verdeutlicht.

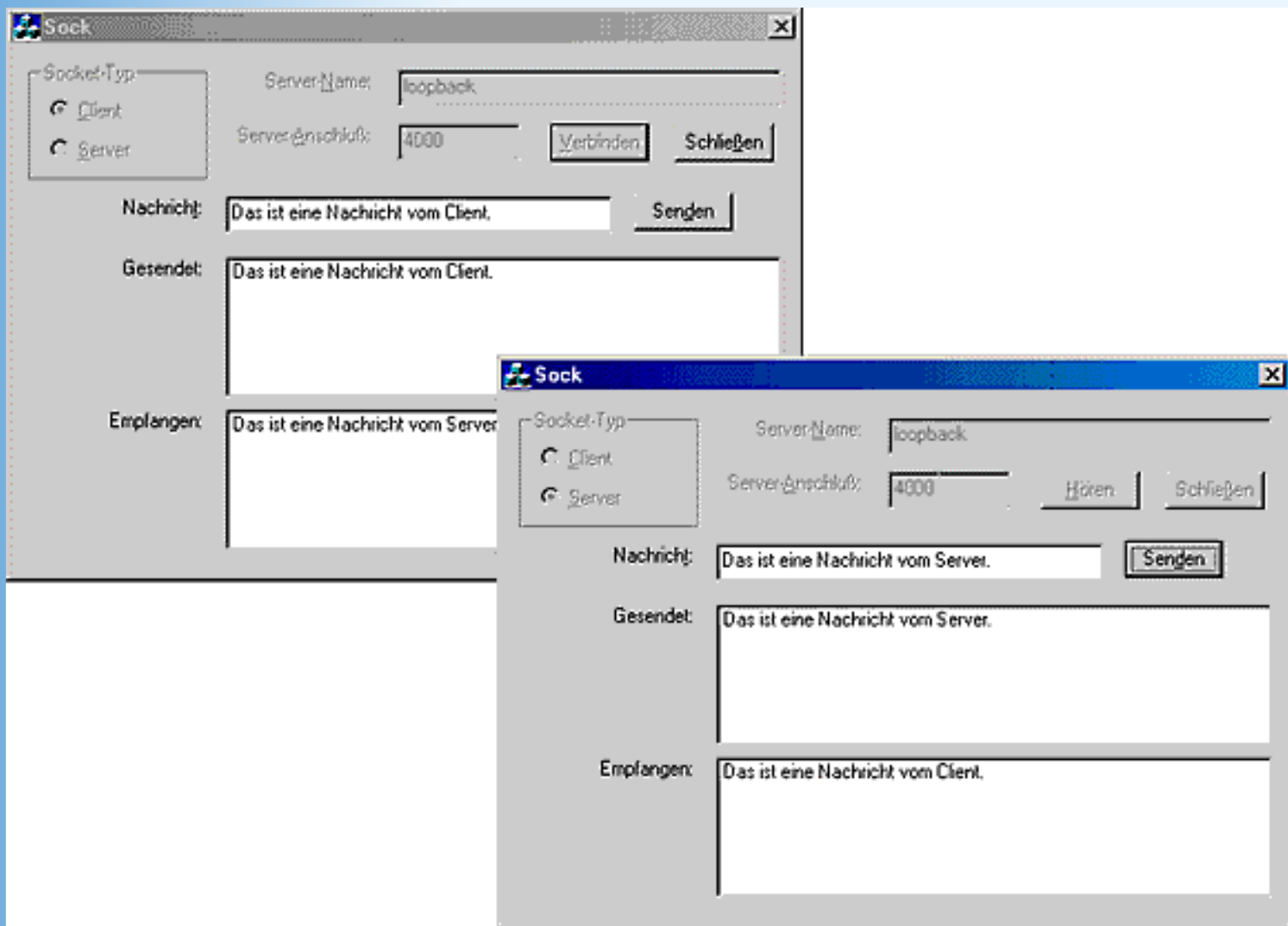


Abbildung 20.7:
Nachrichten zwischen den Anwendungen senden

Die Verbindung beenden

Um die Verbindung zwischen den beiden Anwendungen zu schließen, kann der Benutzer der Client-Anwendung auf die Schaltfläche **Schließen** klicken, um die Verbindung zu beenden. Die Server-Anwendung empfängt daraufhin das Socket-Ereignis `OnClose`. Das gleiche muß in beiden Fällen passieren. Der verbundene Socket ist zu schließen, und die Steuerelemente für das Senden von Nachrichten sind zu deaktivieren. Auf der Client-Seite können die Verbindungssteuerlemente aktiviert bleiben, da der Client beispielsweise die Informationen ändern und eine Verbindung zu einer anderen Server-Anwendung öffnen kann. In der Zwischenzeit horcht die Server-Anwendung weiter den Anschluß ab, für den sie zu diesem Zweck konfiguriert wurde. Um die beschriebene Funktionalität in Ihrer Anwendung umzusetzen, bearbeiten Sie die Funktion `OnClose` und übernehmen den Code aus Listing 20.11.

Listing 20.11: Die Funktion `OnClose` der Klasse `CSockDlg`

```
1: void CSockDlg::OnClose()
2: {
3: // Den verbundenen Socket schließen
4: m_sConnectSocket.Close();
5: // Steuerelemente zum Senden von Nachrichten deaktivieren
```

```

6: GetDlgItem(IDC_EMMSG)->EnableWindow(FALSE);
7: GetDlgItem(IDC_BSEND)->EnableWindow(FALSE);
8: GetDlgItem(IDC_STATICMSG)->EnableWindow(FALSE);
9: GetDlgItem(IDC_BCLOSE)->EnableWindow(FALSE);
10: // Läuft Anwendung im Client-Modus?
11: if (m_iType == 0)
12: {
13: // Ja, Steuerelemente zum Konfigurieren der Verbindung aktivieren
14: GetDlgItem(IDC_BCONNECT)->EnableWindow(TRUE);
15: GetDlgItem(IDC_ESERVNAME)->EnableWindow(TRUE);
16: GetDlgItem(IDC_ESERVPORT)->EnableWindow(TRUE);
17: GetDlgItem(IDC_STATICNAME)->EnableWindow(TRUE);
18: GetDlgItem(IDC_STATICPORT)->EnableWindow(TRUE);
19: GetDlgItem(IDC_RCLIENT)->EnableWindow(TRUE);
20: GetDlgItem(IDC_RSERVER)->EnableWindow(TRUE);
21: GetDlgItem(IDC_STATICTYPE)->EnableWindow(TRUE);
22: }
23: }

```

Schließlich rufen Sie für die Schaltfläche **Schließen** die Funktion `OnClose` auf. Fügen Sie dazu mit dem Klassen-Assistenten eine Funktion für das Klickereignis der Schaltfläche **Schließen** (`IDC_BCLOSE`) hinzu. In die Funktion schreiben Sie den Code aus Listing 20.12.

Listing 20.12: Die Funktion `OnBclose` der Klasse `CSockDlg`

```

1: void CSockDlg::OnBclose()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  ÂBenachrichtigung hier einfügen
4: // OnClose-Funktion aufrufen
5: OnClose();
6: }

```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie die Client-Anwendung mit dem Server verbinden, Nachrichten in beiden Richtungen austauschen und die Verbindung des Clients trennen, indem Sie auf die Schaltfläche **Schließen** klicken. In beiden Anwendungen werden die Steuerelemente zum Senden von Nachrichten deaktiviert, wie es Abbildung 20.8 zeigt. Eine erneute Verbindung des Clients mit dem Server ist möglich, indem Sie wieder auf die Schaltfläche **Verbinden** klicken und dann weitere Nachrichten zwischen den beiden austauschen, so als ob sie noch gar nicht verbunden gewesen wären. Wenn Sie eine dritte Kopie der Anwendung starten, deren Anschlußnummer ändern, sie als Server bekanntmachen und in den Hörermodus setzen, können Sie Ihren Client abwechselnd mit beiden Servern verbinden. Dazu schließen Sie die eine Verbindung, ändern die Anschlußnummer und stellen dann die Verbindung zum anderen Server her.

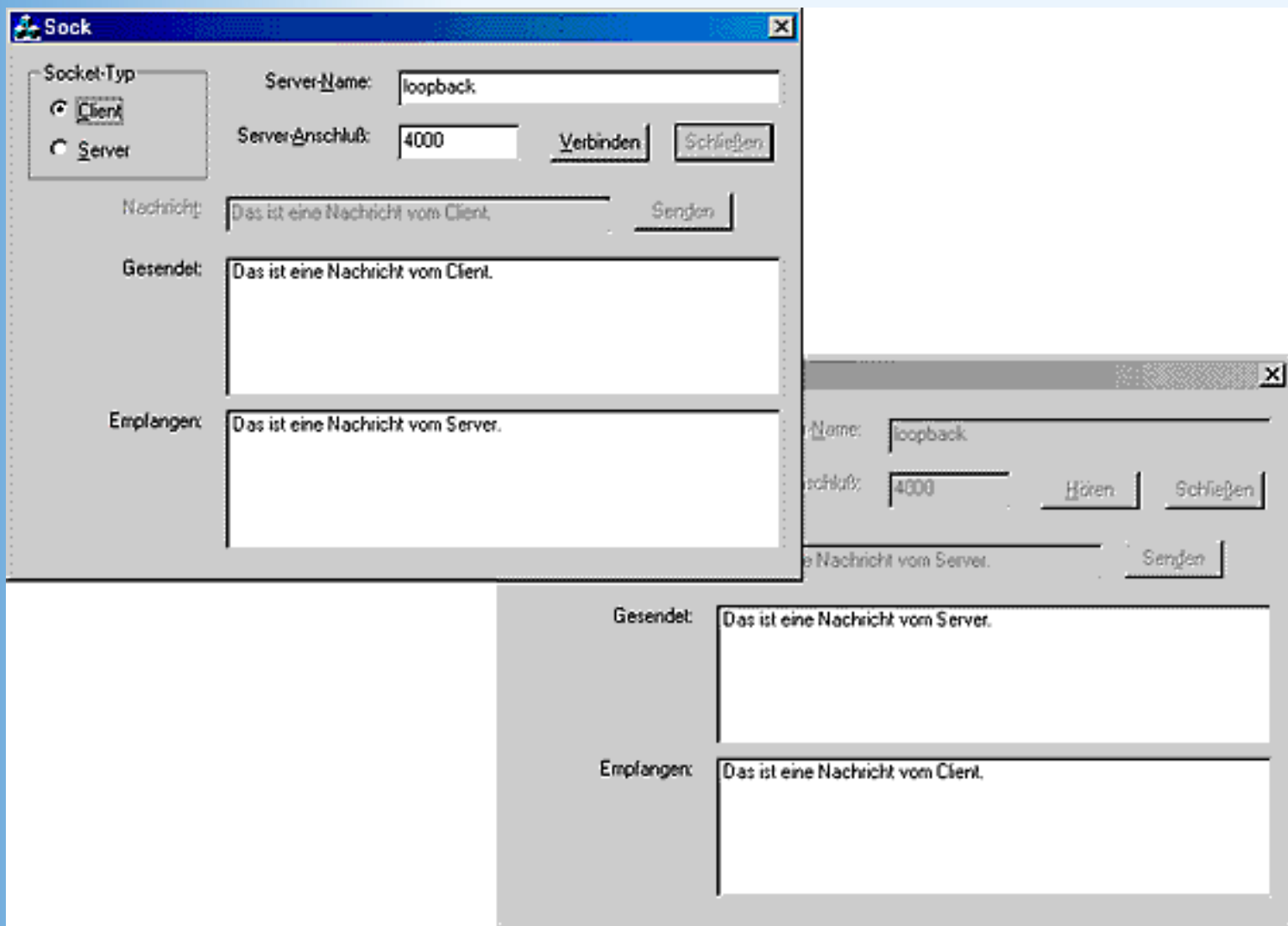


Abbildung 20.8:
Die Verbindung zwischen den Anwendungen schließen

Zusammenfassung

Heute haben Sie gelernt, wie man Anwendungen mit MFC-Winsock-Klassen erstellt, um mit anderen Anwendungen über ein Netzwerk oder das Internet kommunizieren zu können. Die Klasse `CAsyncSocket` wurde näher beleuchtet, und Sie haben gesehen, wie man davon eigene Klassen ableitet, um in einer Anwendung die ereignisgesteuerte Kommunikation zu realisieren. Es wurde gezeigt, wie man eine Server-Anwendung erstellt, die auf Verbindungen mit anderen Anwendungen prüft und die Verbindung herstellt. Weiterhin haben Sie gelernt, wie man eine Client-Anwendung erstellt, die eine Verbindung mit einem Server herstellen kann. Die Lektion ist ebenfalls darauf eingegangen, wie sich Nachrichten über eine Socket-Verbindung zwischen zwei Anwendungen senden und empfangen lassen. Schließlich haben Sie erfahren, wie man die Verbindung schließt und wie man erkennt, daß die Verbindung geschlossen wurde.

Fragen und Antworten

Frage:
Wie arbeiten Internet-Anwendungen?

Antwort:

Die meisten Internet-Anwendungen bauen auf der gleichen Funktionalität auf, die Sie mit der heutigen Beispielanwendung realisiert haben. Der Hauptunterschied besteht darin, daß die Anwendungen mit Nachrichtenskripten arbeiten, die in beiden Richtungen übertragen werden. Die Nachrichten bestehen aus einem Befehl und den Daten, die für diesen Befehl erforderlich sind. Der Server liest den Befehl, verarbeitet die Daten entsprechend und schickt einen Statuscode zurück, aus dem der Client den Erfolg oder das Scheitern des Befehls ablesen kann. Wenn Sie mehr über diese Abläufe in Internet-Anwendungen erfahren möchten, sollte Sie sich mit der einschlägigen Literatur befassen.

Frage:
Wie behandelt eine Server-Anwendung eine größere Anzahl gleichzeitiger Client-Verbindungen?

Antwort:

Bei einem Vollserver sind die Verbindungs-Sockets nicht als Klassenvariablen deklariert. Der Server verwendet statt dessen eine Art dynamischer Zuweisung von Sockets in einem Array oder einer verknüpften Liste, um Sockets für die Clients zu erzeugen, sobald Verbindungsanforderungen hereinkommen. Eine andere von Servern oftmals verwendete Lösung besteht darin, einen eigenen Thread für jede Verbindungsanforderung einzurichten. Damit kann die Anwendung eine einzige Socket-Verbindung pro Thread behandeln, was es wesentlich erleichtert, die Sockets zu verfolgen. In allen Fällen arbeiten Server-Anwendungen normalerweise nicht nur mit einer einzigen Variablen für die Verbindungs-Sockets.

Workshop

Kontrollfragen

1. Welche zwei Dinge muß eine Client-Anwendung kennen, damit sie sich mit einer Server-Anwendung verbinden kann?
2. Welche `CAsyncSocket`-Funktion kommt zum Einsatz, um eine Server-Anwendung in die Lage zu versetzen, Verbindungsgesuche von Client-Anwendungen zu erkennen?
3. Welche `CAsyncSocket`-Elementfunktion wird aufgerufen, um zu signalisieren, daß Daten über eine Socket-Verbindung angekommen sind?
4. Welche Funktion wird aufgerufen, um zu signalisieren, daß eine Verbindung eingerichtet wurde?
5. Welche Funktion verwenden Sie, um eine Nachricht über eine Socket-Verbindung zur Anwendung am anderen Ende der Verbindung zu senden?

Übung

Die Server-Anwendung, die Sie heute geschrieben haben, kann nur eine einzelne Verbindung zu einem Zeitpunkt behandeln. Wenn eine zweite Anwendung versucht, eine Verbindung zum Server zu öffnen, während der Server mit einer anderen Anwendung in Verbindung steht, stürzt die Server-Anwendung ab. Der Server versucht, die zweite Verbindung in den Socket, der bereits mit der ersten Client-Anwendung verbunden ist, zu übernehmen. Fügen Sie Ihrer Anwendung ein drittes Socket-Objekt hinzu, das die zusätzlichen Client-Verbindungen zurückweist, bis der erste Client die Verbindung schließt.

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Woche 3

Tag 21

Funktionen für Webbrowser

Als Microsoft vor wenigen Jahren die Entscheidung getroffen hat, alle seine Anwendungen Internet-fähig zu machen, ging es nicht nur darum, daß man Word das Lesen und Schreiben von HTML-Seiten beibrachte. Das Internet sollte integraler Teil jeder Anwendung werden, auf die eine oder andere Art. Aus dem Blickwinkel der Entwicklungswerkzeuge betrachtet, handelt es sich natürlich nicht um eine wirklich praktische Integration, wenn man einem Editor die Doppelfunktion eines E-Mail-Clients einhaucht. Erleichtert man es allerdings den Benutzern von Entwicklungswerkzeugen, Internet-fähige Anwendungen zu erstellen, ist das schon eher ein praktisches Feature. Und das ist genau das, was Microsoft umgesetzt hat.

Eine der Fähigkeiten, die Microsoft in seinen Werkzeugen zur Anwendungsentwicklung verfügbar gemacht hat, ist der Einsatz des Internet Explorer als integraler Teil einer beliebigen Anwendung. Das bedeutet, daß man den Internet Explorer und alle damit verbundenen Komponenten in eigene Anwendungen einbinden kann. Die Möglichkeiten gehen weit über die Bereitstellung der Webbrowser-Fähigkeiten hinaus. Ihre Anwendungen können auch Java-Applets aufnehmen und damit zusammenarbeiten. Man kann den Benutzern nicht nur eine, sondern zwei Makrosprachen bieten - VBScript und JScript (Microsofts Version von JavaScript).

Heute lernen Sie, wie ...

- das ActiveX-Objektmodell des Internet Explorer die Integration all dieser Komponenten in Ihre Anwendungen gestattet,
- die Ansichtsklasse `CHtmlView` den größten Teil der Funktionalität des Internet Explorer in einer vorgefertigten Klasse verkapselt,
- man einen einfachen Webbrowser mit Hilfe der Klasse `CHtmlView` und dem Internet Explorer erstellt.

Das ActiveX-Modell des Internet Explorer

Als sich bei Microsoft der Gedanke durchsetzte, ActiveX mit dem Webbrowser Internet Explorer zu integrieren, wurde klar, daß man den Internet Explorer technisch überholen mußte, um den Einsatz von ActiveX-Steuerelementen zu unterstützen. Die Entwickler sahen sich an, was man tun mußte und was im Rahmen des Möglichen lag, und entschieden sich dafür, den Internet Explorer zu mehr als zu einem Webbrowser zu machen.

Als erstes hat Microsoft den eigentlichen Webbrowser von den ActiveX-Objekten, die die gesamte

Arbeit verrichten, getrennt. Herausgekommen ist die Anwendung Internet Explorer, die etwas mehr als einen Dokument-Container für ActiveX darstellt, und das Internet-Explorer-HTML-Viewer-Steuerelement, das als ActiveX-Dokument-Server innerhalb der Anwendung läuft. Das bedeutet, daß die Anwendung Internet Explorer nicht nur Webseiten aufnehmen kann, sondern auch Word-Dokumente, Excel- Tabellenblätter, PowerPoint-Präsentationen und jedes beliebige ActiveX-Dokument, für das ein ActiveX-Dokument-Server auf demselben Computer installiert ist, wie es Abbildung 21.1 verdeutlicht.

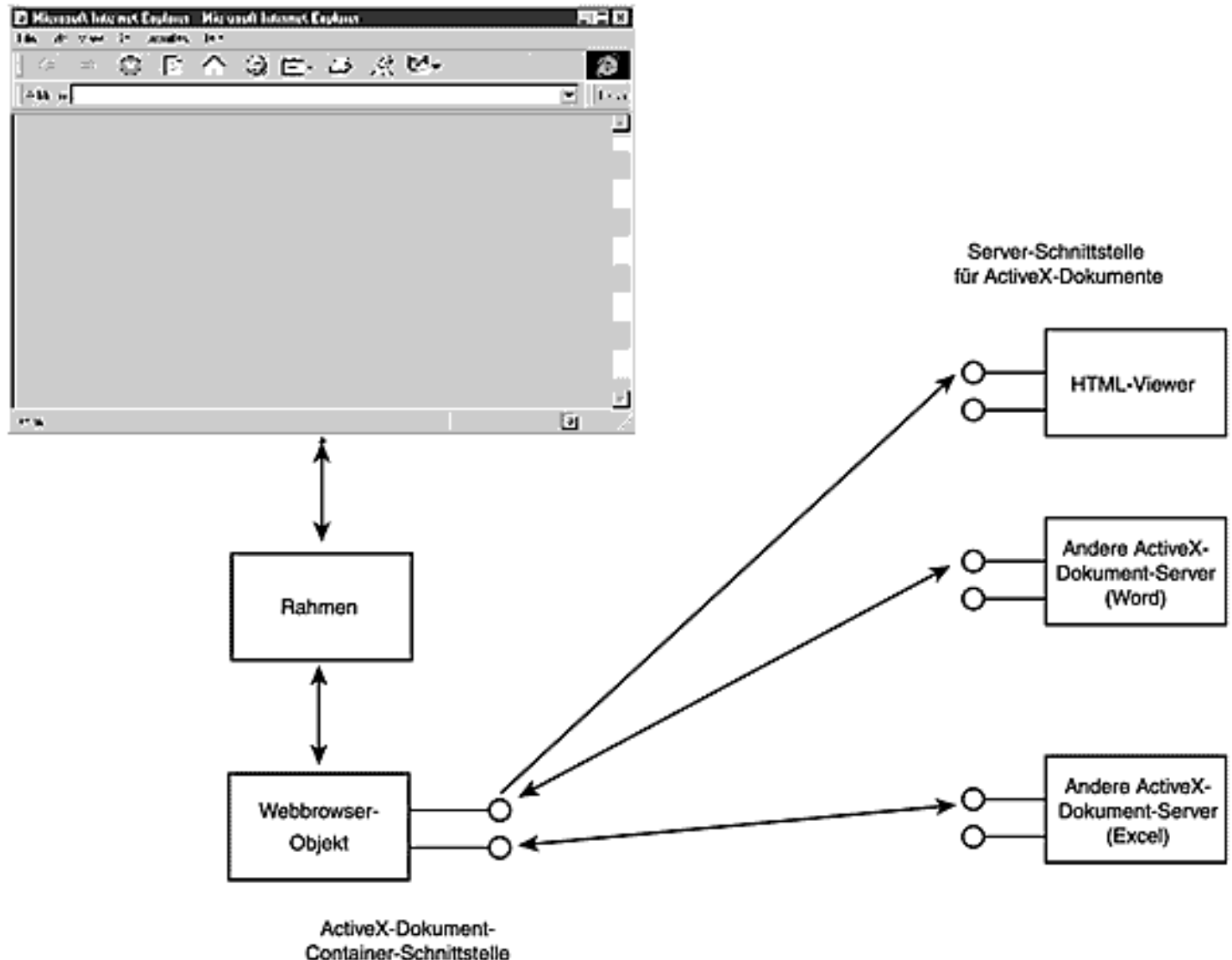


Abbildung 21.1:
Das ActiveX-Dokumentmodell des Internet Explorer

Innerhalb der HTML-Viewer-Komponente hat Microsoft die Fähigkeit eingebaut, andere Steuerelemente aufzunehmen, einschließlich Skripting-Engines und ActiveX- Steuerelemente, wie in Abbildung 21.2 dargestellt. Damit hat Microsoft eine Flexibilität erreicht, um dem Internet Explorer weitere Skriptsprachen hinzuzufügen, wenn diese gefordert und erstellt werden. Darüber hinaus kann der Internet Explorer beliebige ActiveX-Steuerelemente aufnehmen, die noch in den Schubladen der Entwickler liegen.

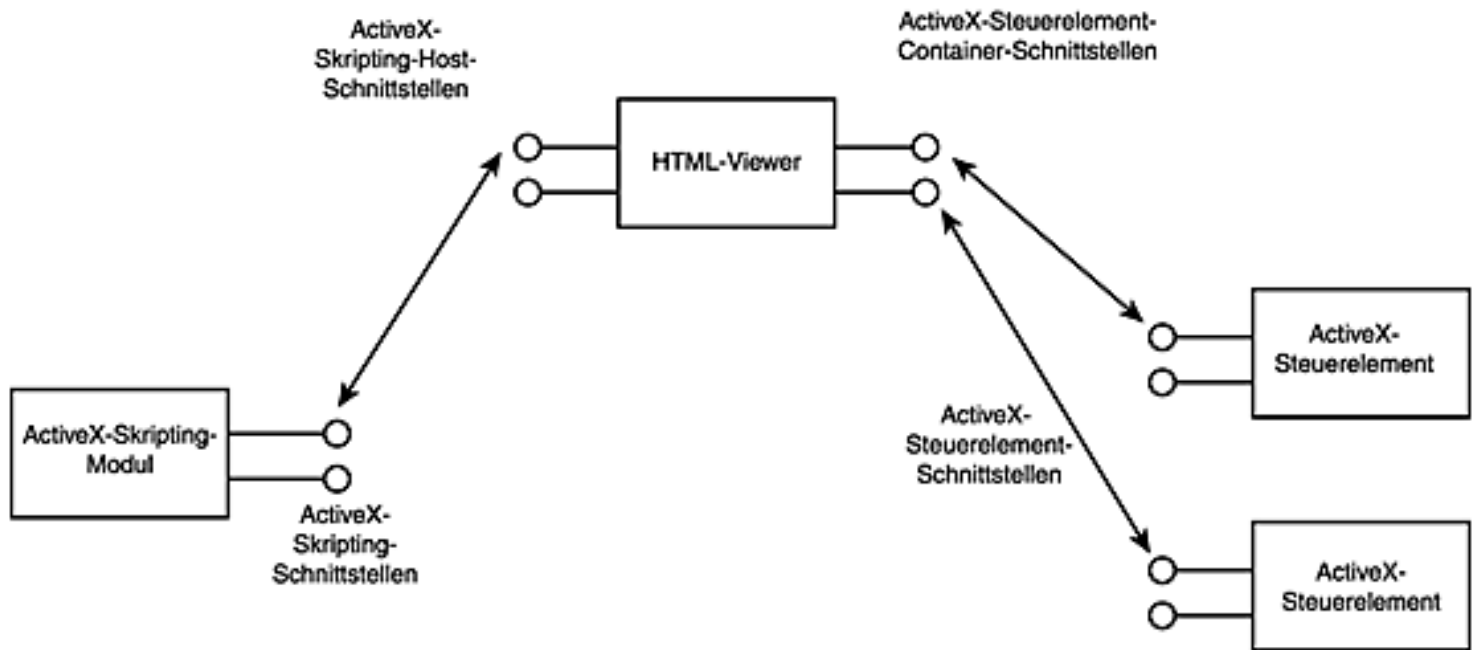


Abbildung 21.2:
Das ActiveX-Objektmodell HTML Viewer des Internet Explorer

Durch dieses Konzept des Internet Explorer hat sich Microsoft nicht nur selbst eine Menge Flexibilität für zukünftige Erweiterungen der vom Internet Explorer unterstützten Funktionalität verschafft, sondern auch die gesamten Tätigkeiten des Internet Explorer jedem Entwickler verfügbar gemacht, der diese nutzen und in seine Anwendungen integrieren möchte.

Die Klasse CHtmlView

Um den Internet Explorer HTML-Viewer leicht in Visual-C++-Anwendungen einbinden zu können, hat ihn Microsoft in die Klasse CHtmlView verpackt. Diese Klasse lässt sich als Basisklasse für die Ansichtsklassen in SDI- oder MDI-Anwendungen einsetzen. Man kann in einfacher Weise Anwendungen erstellen, die die Fähigkeiten eines Webbrowsers mitbringen.

Im Web navigieren

In der Klasse CHtmlView sind verschiedene Funktionen verfügbar, die sich auf die Navigation im Web beziehen. Dazu gehören Funktionen, die dem Browser die Startseite des Benutzers zurückgeben oder den Benutzer auf eine Suchseite des Internet bringen. Weiterhin gibt es Funktionen, über die der Benutzer zur vorhergehenden oder nächsten Seite gelangt oder sogar zu einer entfernten Webseite wechseln kann. Alle diese Funktionen sind Elemente der Klasse CHtmlView und demzufolge Member-Funktionen der Ansichtsklasse Ihrer Anwendung (wenn Sie die Klasse CHtmlView als Basisklasse für Ihre Ansichtsklasse verwenden).

Die Navigationsfunktionen der Klasse CHtmlView faßt Tabelle 21.1 zusammen.

Tabelle 21.1: Navigationsfunktionen der Klasse CHtmlView

Funktionsdefinition	Beschreibung
GoBack()	Bringt den Benutzer auf die vorhergehende Webseite.

GoForward()	Bringt den Benutzer zur nächsten Webseite. (Setzt voraus, daß der Benutzer vorher mindestens eine Webseite zurückgegangen ist.)
GoHome()	Bringt den Benutzer zur Startseite für Internet Explorer.
GoSearch()	Bringt den Benutzer zu einer Suchseite im Internet.
Navigate(LPCTSTR URL)	Bringt den Benutzer auf die Webseite, die in der URL-Variablen angegeben ist.

Die ersten vier Funktionen übernehmen keine Argumente und führen genau die gleiche Funktion aus wie ihre Äquivalente auf der Symbolleiste des Internet Explorer. Von den Argumenten an die letzte Funktion ist nur der URL der anzuzeigenden Webseite erforderlich.

Den Browser steuern

Zusammen mit den Funktionen zur Navigation im Web verwenden Sie auch bestimmte Funktionen, mit denen Sie den Browser steuern. Zwei davon sind `Refresh()`, die das HTML-Viewer-Steuerelement veranlaßt, die aktuelle Webseite neu zu laden, und `Stop()`, die einen laufenden Download anhält. Wie bei den meisten Navigationsfunktionen übernehmen diese Funktionen keine Argumente und arbeiten genau wie ihre äquivalenten Schaltflächen auf der Symbolleiste des Internet Explorer.

Den Browser-Status ermitteln

Eine weitere Kategorie von Funktionen, die in der Klasse `CHtmlView` verfügbar sind, haben informellen Charakter. Man kann diese Funktionen verwenden, um Informationen über den aktuellen Status des Browsers zu ermitteln. Wenn Sie zum Beispiel die aktuelle Webseite im Browser wissen möchten, rufen Sie die Funktion `GetLocationURL()` auf, die einen `CString` mit dem URL zurückgibt. Wollen Sie bestimmen, ob der Browser gerade mit einem Download beschäftigt ist, rufen Sie die Funktion `GetBusy()` auf. Diese Funktion gibt einen booleschen Wert zurück und zeigt damit an, ob der Browser beschäftigt ist.

In der Klasse `CHtmlView` sind viele weitere Funktionen verfügbar. Einige davon arbeiten nur im Internet Explorer selbst und nicht im Browser-Steuerelement.

Eine Webbrowser-Anwendung erstellen

Als Beispiel, wie man die Webbrowser-Komponente des Internet Explorer in eigene Anwendungen einbindet, erstellen Sie heute eine einfache Webbrowser-Anwendung. Das Ganze baut auf einer SDI-Anwendung auf, die die Klasse `CHtmlView` als Basisklasse der Ansichtsklasse verwendet. Für die Navigation fügen Sie ein Menü mit den Befehlen **Zurück** und **Vorwärts** hinzu. Weiterhin erhält die Anwendung ein Dialogfeld, um dem Benutzer die Eingabe eines URL zu ermöglichen. Damit navigieren Sie den Browser zur angegebenen Webseite.

Das Anwendungsgerüst erstellen

Für eine Webbrowser-Anwendung können Sie auf ein normales SDI- oder MDI-Anwendungsgerüst zurückgreifen. Als einzige Voraussetzung muß der Internet Explorer auf dem Computer installiert sein, auf dem Ihre Anwendung läuft. Bei Ihrem Entwicklungscomputer ist das kein Problem, weil die Installation von Visual C++ ohnehin darauf dringt, daß Sie die neueste Version des Internet Explorer installieren. Bei jedem anderen Computer, auf dem Ihre Anwendung laufen soll, müssen Sie sich allerdings zuerst davon überzeugen, ob der Internet Explorer installiert ist. Andernfalls müssen Sie die Installation selbst vornehmen.

Um das Gerüst der heutigen Beispielanwendung zu erstellen, legen Sie mit dem Anwendungs-Assistenten ein neues Projekt an. Nennen Sie das Projekt zum Beispiel **WebBrowse**, und klicken Sie auf OK, um den Assistenten zu starten.

Mit dem Anwendungs-Assistenten können Sie genauso leicht eine MDI-Anwendung als Webbrowser erstellen wie eine SDI-Anwendung. Für die heutige Beispielanwendung wählen Sie die Option **Einzelnes Dokument** (SDI). In den übrigen Dialogfeldern des Assistenten übernehmen Sie die Standardeinstellungen. Für das Beispiel wählen Sie allerdings im vierten Dialogfeld des Assistenten die Option **Internet Explorer-Inforeisten** als Erscheinung der Menüs.

Schließlich legen Sie im sechsten Dialogfeld des Anwendungs-Assistenten die Klasse **CHtmlView** als Basisklasse für Ihre Ansichtsklasse fest. Damit erstellt der Assistent Ihre Anwendung mit dem Webbrowser-Steuerelement des Internet Explorer als Hauptansicht der Anwendung.

Wenn Sie dieses Gerüst der Anwendung kompilieren und starten sowie eine Verbindung zum Internet herstellen, haben Sie bereits einen funktionsfähigen Webbrowser vor sich, wie es Abbildung 21.3 zeigt. Allerdings läßt sich noch nicht festlegen, wohin Sie der Browser bringen soll, es sei denn, Sie klicken auf die Links in den angezeigten Webseiten.

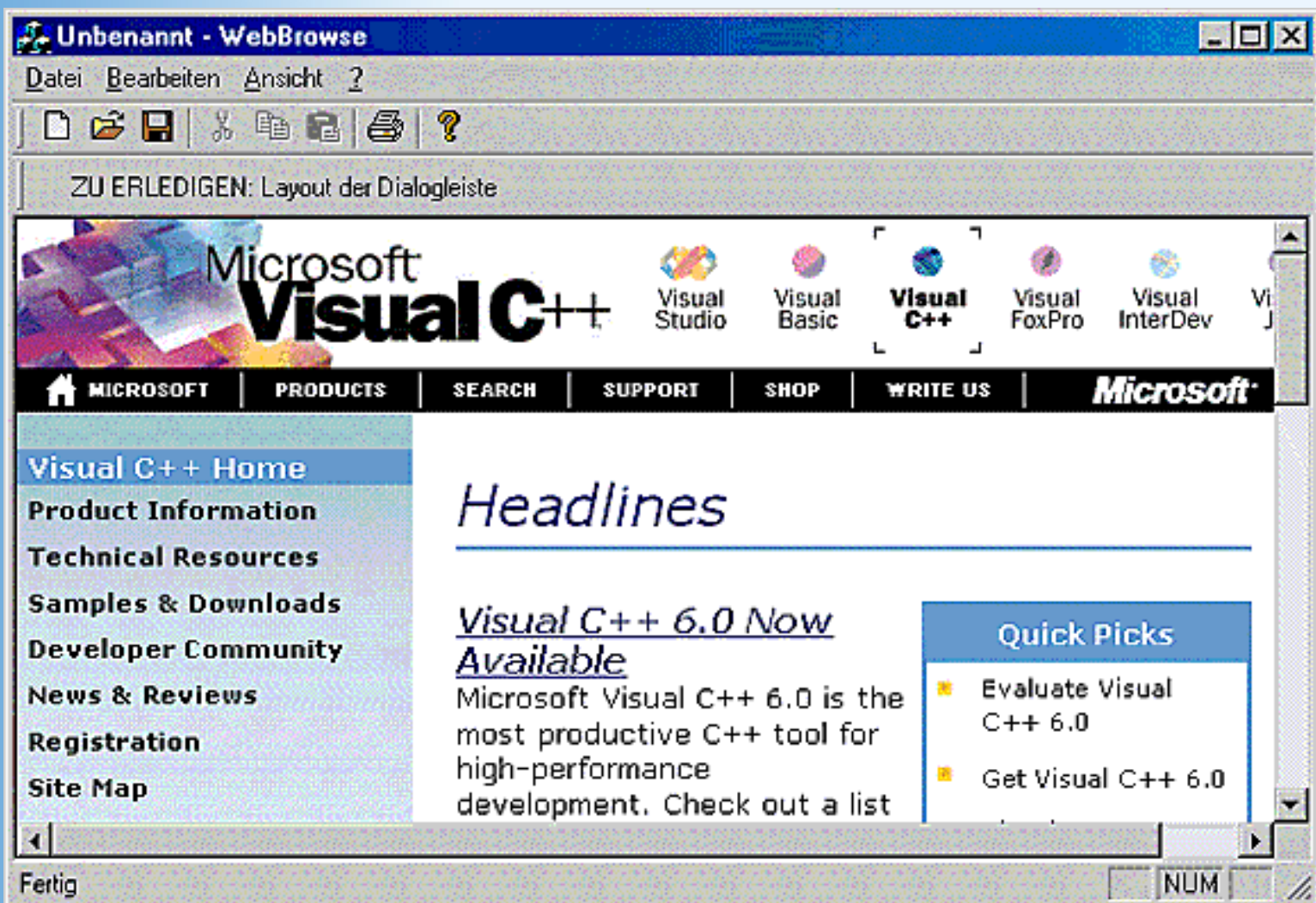


Abbildung 21.3:
Die anfängliche Webbrowser-Anwendung

Funktionen zum Navigieren hinzufügen

Mit einem funktionsfähigen Browser allein ist es noch nicht getan. Man muß auch steuern können, wohin der Browser navigieren soll. Es ist ein Eingabefeld erforderlich, in das der Benutzer einen URL eingeben kann. Wenn Sie sich die Symbolleiste der laufenden Anwendung ansehen, finden Sie einen Platz, wo sich dieses Steuerelement unterbringen läßt.

Einen URL spezifizieren

Wenn Sie die Anwendung ausführen, haben Sie wahrscheinlich schon den Text in der zweiten Symbolleiste bemerkt, daß an dieser Stelle das Layout der Dialogleiste unterzubringen ist. Die Dialogleiste unterscheidet sich von dem, was Sie bisher kennengelernt haben. Es handelt sich um eine Symbolleiste, auf der normale Steuerelemente wie bei einem Dialogfeld enthalten sind. Die Leiste entwerfen Sie sogar mit dem Dialog-Editor. Wenn Sie auf der Registerkarte **Ressourcen** nach der Dialogleiste suchen, finden Sie sie nicht im Ordner **Toolbar**, sondern im Ordner **Dialog**.

Öffnen Sie den Ordner **Dialog**, und doppelklicken Sie auf das Dialogfeld **IDR_MAINFRAME**, um es im Dialog-Editor zu öffnen. Es zeigt sich, daß dieses Dialogfeld die zweite Symbolleiste in Ihrer Anwendung verkörpert. Auf dieser Symbolleiste lassen sich Eingabefelder, Schaltflächen, Kombinationsfelder und Kontrollkästchen platzieren - praktisch jedes beliebige Steuerelement, das Sie auch in einem normalen Dialogfeld verwenden können.

Modifizieren Sie das bereits auf der Dialogleiste vorhandene Textfeld, und fügen Sie ein Eingabefeld hinzu, wie es Abbildung 21.4 zeigt. Legen Sie für dieses Beispiel die ID des Eingabefelds mit IDC_EADDRESS fest.

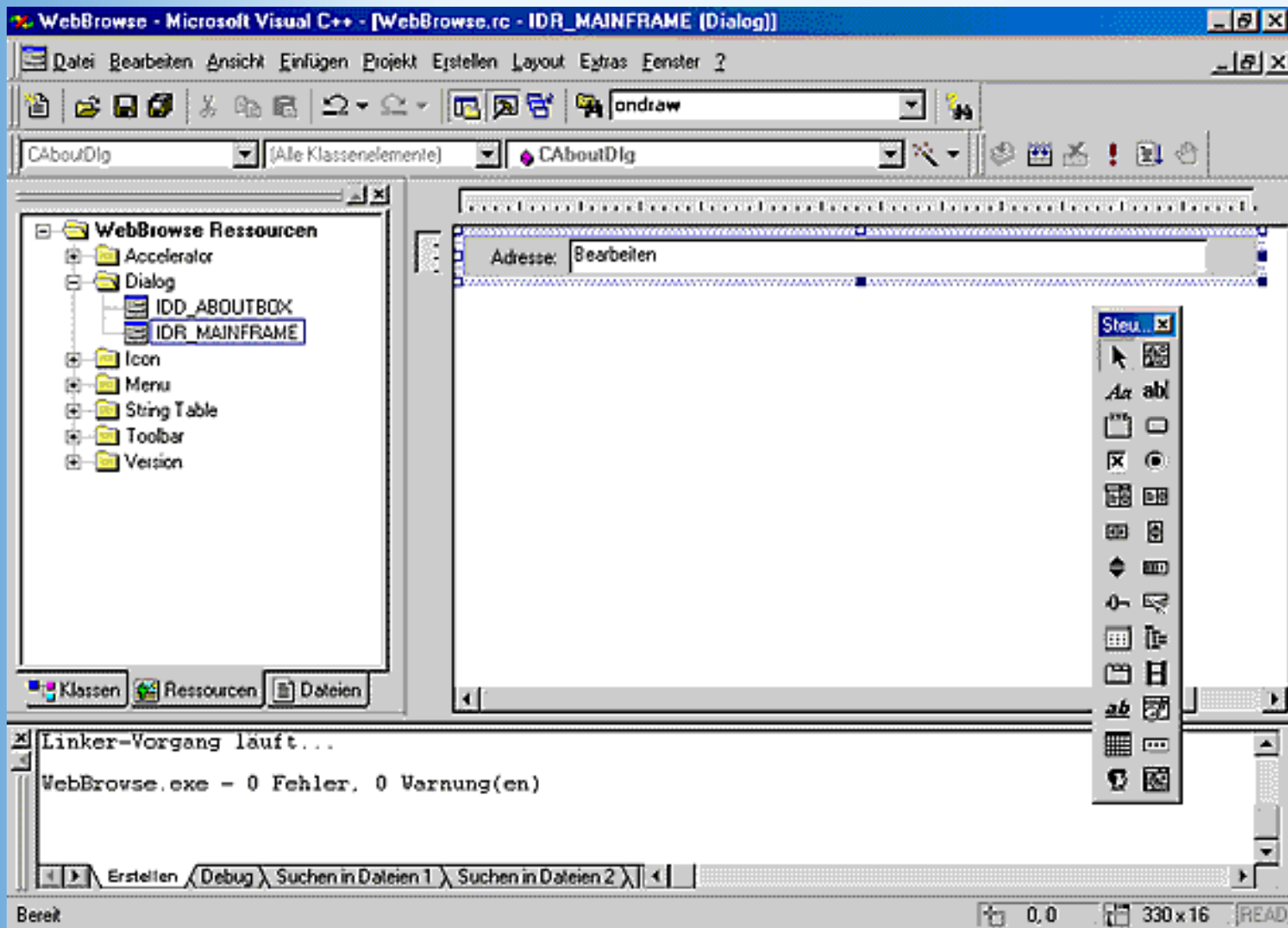


Abbildung 21.4:
Das Layout der Dialogleiste

Bevor Sie den Klassen-Assistenten öffnen, um Variablen und Behandlungsroutinen für die Dialogleiste hinzuzufügen, sollten Sie wissen, daß die Dialogleiste ihre Ereignisse automatisch an die Hauptrahmenklasse Ihrer Anwendung schickt. Sobald Sie den Klassen-Assistenten öffnen, nimmt dieser an, daß Sie die Dialogleiste mit einer Klasse verbinden möchten, und fordert Sie auf, eine neue Klasse zu erstellen. Diese Verbindung ist jedoch nicht erforderlich, da Sie alle Ereignisse über den Rahmen abbilden und von dort aus an die Ansichts- oder Dokumentklassen weiterleiten können.

Für die heutige Beispielanwendung brauchen Sie nicht einmal den Klassen-Assistenten zu bemühen, um irgendwelche Behandlungsroutinen für die Dialogleiste aufzunehmen. Sie müssen eine Aktion auslösen, wenn der Benutzer den URL vollständig in das Eingabefeld eingegeben hat. Von den im Klassen-Assistenten verfügbaren Ereignissen kommt am ehesten EN_CHANGED in Frage, das jeden vom Benutzer eingegebenen Buchstaben meldet. Wir brauchen aber ein Ereignis, das ausgelöst wird, wenn der Benutzer die (E)-Taste drückt. Zum Glück wird die Befehls-ID IDOK an die Rahmenklasse gesendet, wenn der Benutzer das Eingabefeld in der Dialogleiste ausfüllt und die (E)-Taste drückt. Man kann in diesem Fall eine Behandlungsroutine in die Nachrichtenzuordnungstabelle einfügen, um eine Funktion für den Befehl IDOK aufzurufen.

In der Behandlungsroutine des Befehls müssen Sie den Fenstertext aus dem Eingabefeld in der Dialogleiste ermitteln. Diesen String können Sie an die Funktion `Navigate` in der Ansichtsklasse übergeben und damit den Browser veranlassen, zu der vom Benutzer angegebenen Seite zu gehen.

Fügen Sie zu diesem Zweck eine neue Member-Funktion in die Klasse `CMainFrame` ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration als `OnNewAddress` und den Zugriff als `Public` fest. In die Funktion schreiben Sie den Code aus Listing 21.1.

Listing 21.1: Die Funktion `OnNewAddress` der Klasse `CMainFrame`

```
1: void CMainFrame::OnNewAddress()
2: {
3:     CString sAddress;
4:
5:     // Den neuen URL holen
6:     m_wndDlgBar.GetDlgItem(IDC_EADDRESS)->GetWindowText(sAddress);
7:     // Zum neuen URL navigieren
8:     ((CWebBrowseView*)GetActiveView())->Navigate(sAddress);
9: }
```

Die Funktion holt in Zeile 6 mit der Funktion `GetWindowText` den Text im Eingabefeld und schreibt ihn in die Variable `m_sAddress`. Die Dialogleiste ist in der Klasse `CMainFrame` als Variable `m_wndDlgBar` deklariert, so daß sich die Funktion `GetDlgItem` auf der Variablen der Dialogleiste verwenden läßt, um einen Zeiger auf das Eingabefeld zu ermitteln.

In Zeile 8 findet die Typumwandlung des aus der Funktion `GetActiveView` zurückgegebenen Zeigers in einen Zeiger auf die Klasse `CWebBrowseView` statt. Damit kann man die Funktion `Navigate` auf der Ansichtsklasse aufrufen und den URL übergeben, den der Benutzer im Eingabefeld spezifiziert hat.

Nachdem Sie nun den vom Benutzer eingegebenen URL übernehmen und die Browser-Komponente anweisen können, zu dieser Webseite zu gehen, wie lösen Sie dann diese Funktion aus? Dazu müssen Sie manuell einen Eintrag in die Nachrichtenzuordnungstabelle einfügen, da der Klassen-Assistent dazu nicht in der Lage ist. Fügen Sie in der Nachrichtenzuordnungstabelle nach der schließenden Markierung von `AFX_MSG_MAP` (dem vom Klassen-Assistenten verwalteten Bereich) das Makro `ON_COMMAND` ein, um damit den Befehl `IDOK` und Ihre neue Funktion als aufzurufenden Handler zu spezifizieren, wie es Listing 21.2 zeigt. Sie können diesen Eintrag auch vor dem vom Klassen-Assistenten verwalteten Abschnitt einfügen, solange der Eintrag nicht innerhalb dieses Bereichs steht.

Listing 21.2: Die Nachrichtenzuordnungstabelle von `CMainFrame`

```
1: BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
2: //{{AFX_MSG_MAP(CMainFrame)
3: // HINWEIS - Hier werden Mapping-Makros vom Klassen-Assistenten
   eingefügt und entfernt.
4: // Innerhalb dieser generierten Quelltextabschnitte NICHTS
   ÄVERÄNDERN!
5: ON_WM_CREATE()
6: //}}AFX_MSG_MAP
7: ON_COMMAND(IDOK, OnNewAddress)
8: END_MESSAGE_MAP()
```

Wenn Sie die Anwendung kompilieren und ausführen, können Sie einen URL in das Eingabefeld auf der Symbolleiste eintragen und die (€)-Taste drücken, und Ihre Anwendung sollte zur angegebenen Webseite gehen, wie es Abbildung 21.5 zeigt.

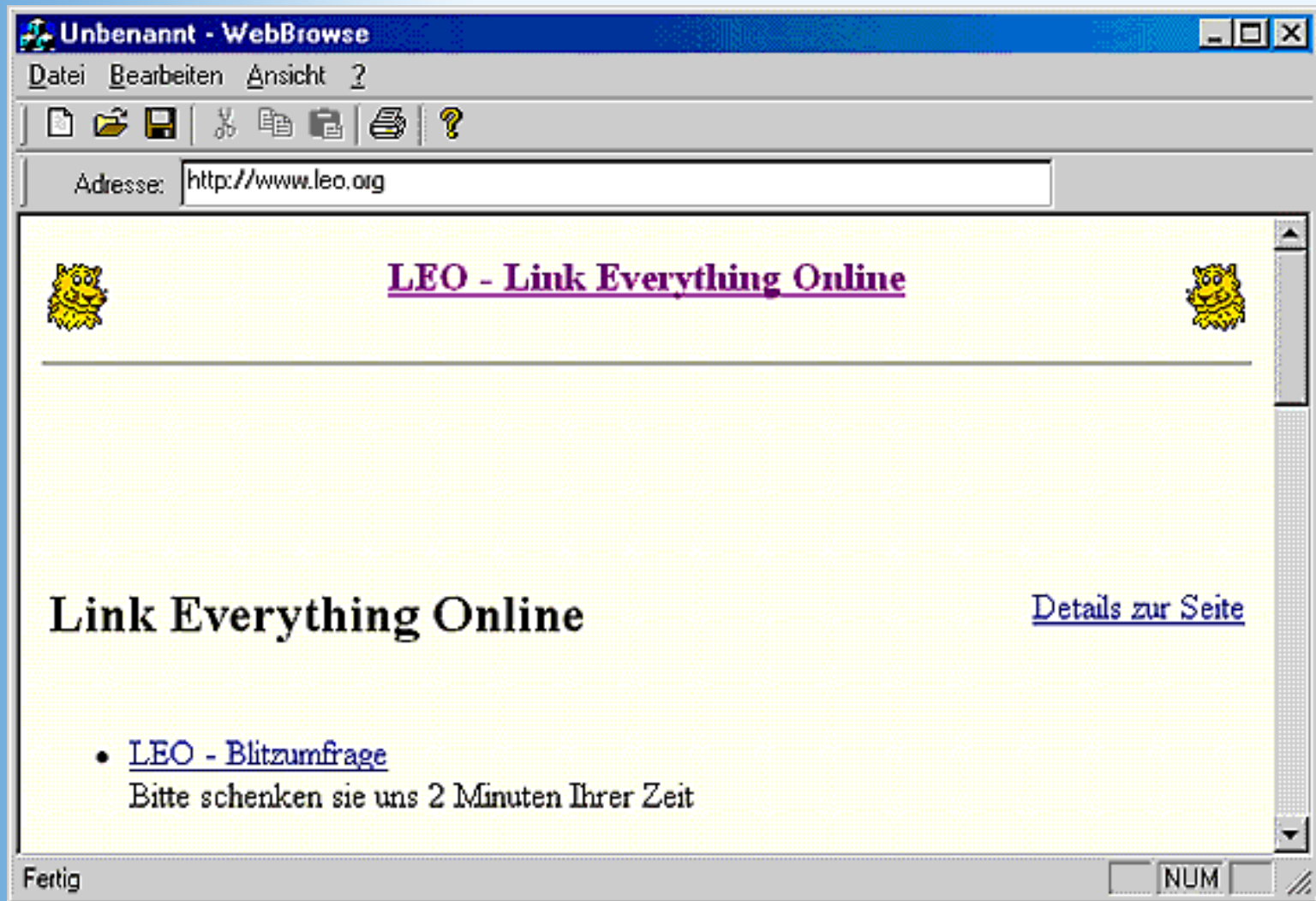


Abbildung 21.5:
Zu einem bestimmten URL gehen

Den aktuellen URL anzeigen

Wenn Sie durch das Web surfen, folgen Sie oftmals Verweisen auf Webseiten, die zu anderen Websites führen. In diesem Fall wissen Sie nicht, auf welche Website Sie zugreifen, wenn Ihr Browser nicht den URL in das Adreßfeld stellt und damit Ihren aktuellen Standort angibt sowie die Möglichkeit bietet, den URL zu kopieren oder zu modifizieren, um eine andere Seite auf derselben Site zu suchen.

Das Abfragen des aktuellen URL vom Browser ist einfach: Man ruft die Funktion `GetLocationURL` auf und übergibt das Ergebnis an die Dialogleiste. Das Problem besteht darin, wann man den URL holt. Es stellt sich heraus, daß sich manche Ereignisfunktionen in der Klasse `CHtmlView` nicht in Ihrer Klasse überschreiben lassen. Diese Funktionen werden bei verschiedenen Ereignissen ausgelöst, die das Browser-Steuerelement initiiert. Es gibt Ereignisfunktionen für das Starten der Navigation, den Beginn eines Download, die Überwachung des Download-Fortschritts und - am wichtigsten für unsere Erfordernisse - die Anzeige, daß der Download abgeschlossen ist. Keine dieser Behandlungsfunktionen können Sie in Ihre Ansichtsklasse über den Klassen-Assistenten aufnehmen. Das müssen Sie in eigener Regie bewerkstelligen.

Um die Behandlungsroutine für einen abgeschlossenen Download in Ihre Anwendung einzubauen,

fügen Sie eine neue Member-Funktion in die Ansichtsklasse der Anwendung ein. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration mit `OnDocumentComplete(LPCSTR lpszUrl)` und den Zugriff als *Public* fest. In die Funktion schreiben Sie den Code aus Listing 21.3.

Listing 21.3: Die Funktion `OnDocumentComplete` der Klasse `CWebBrowseView`

```
1: void CWebBrowseView::OnDocumentComplete(LPCTSTR lpszUrl)
2: {
3: // Den neuen URL an die Adreßleiste übergeben
4: ((CMainFrame*)GetParentFrame())->SetAddress(lpszUrl);
5: }
```

In dieser Funktion fällt auf, daß man die Funktion `GetLocationURL` überhaupt nicht aufrufen muß. Der heruntergeladene URL wird als Argument an die Funktion übergeben. Damit können Sie den URL an den Rahmen weiterreichen, wo Sie eine andere Funktion hinzufügen, um das Eingabefeld auf der Dialogleiste mit dem URL zu füllen.

Nehmen Sie zu diesem Zweck eine neue Member-Funktion in die Hauptrahmenklasse `CMainFrame` auf. Legen Sie den Funktionstyp als `void`, die Funktionsdeklaration als `SetAddress(LPCSTR lpszURL)` und den Zugriff als *Public* fest. In die Funktion übernehmen Sie den Code aus Listing 21.4.

Listing 21.4: Die Funktion `SetAddress` der Klasse `CMainFrame`

```
1: void CMainFrame::SetAddress(LPCTSTR lpszURL)
2: {
3: // Den neuen URL ins Eingabefeld der Adresse setzen
4: m_wndDlgBar.GetDlgItem(IDC_EADDRESS)->SetWindowText(lpszURL);
5: }
```

Hier haben Sie den entgegengesetzten Weg eingeschlagen, mit dem Sie den Text aus dem Eingabefeld geholt haben. Die Funktion `SetAddress` ändert den Text im Eingabefeld mit der Funktion `SetWindowText` auf den übergebenen URL. Wenn Sie die Anwendung ausführen, sollte die URL-Adresse in der Dialogleiste die momentan angezeigte Webseite widerspiegeln.

Zurück und Vorwärts

Nachdem Sie einen URL in die Dialogleiste eingeben und Ihre Anwendung veranlassen können, zu dieser Website zu gehen, und auch die Adresse aller betrachteten Websites zu sehen sind, wäre es schön, wenn man wieder an die Stelle zurückkehren könnte, woher man gekommen ist. Dazu ruft man einfach die Funktionen `GoBack` und `GoForward` in der Ansichtsklasse der Anwendung auf. Die Funktionen lassen sich über Menübefehle aktivieren, was Ihnen außerdem ermöglicht, dieselben Aufrufe über Schaltflächen der Symbolleiste zu realisieren.

Um diese Funktionalität zu implementieren, öffnen Sie das Hauptmenü im Menü-Editor. Aus der Menüleiste können Sie das Menü **Bearbeiten** mit allen zugehörigen Menübefehlen löschen, da sie für die heutige Beispielanwendung keine Rolle spielen. Ziehen Sie den leeren Menübefehl auf der Menüleiste links neben das Menü **Hilfe**. Öffnen Sie das Eigenschaftsdialogfeld für diesen Menübefehl, und tragen Sie als Titel `&Wechseln` zu ein. In diesem Menü sind alle Navigationsfunktionen zusammengefaßt.

Für die Navigation in Vorwärts- und Rückwärtsrichtung fügen Sie zwei Menübefehle hinzu, einen für die Funktion `GoForward` und einen für die Funktion `GoBack`. Legen Sie die Eigenschaften für die

beiden Menübefehle gemäß Tabelle 21.2 fest.

Tabelle 21.2: Eigenschaftseinstellungen des Menüs Wechseln zu

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_GO_BACK
	Titel	&Zurück\tAlt+Nach-Links
	Statuszeilentext	Wechselt zum vorherigen Objekt\nZurück
Menübefehl	ID	IDM_GO_NEXT
	Titel	&Vorwärts\tAlt+Nach-Rechts
	Statuszeilentext	Wechselt zum nächsten Objekt\nVorwärts

Nachdem Sie die Menübefehle hinzugefügt haben, können Sie mit dem Klassen-Assistenten die Funktionen in die Ansichtsklasse für beide Menüereignisse hinzufügen. Für die Menü-ID `IDM_GO_BACK` fügen Sie eine Behandlungsfunktion für die Nachricht `COMMAND` hinzu und schreiben den Code aus Listing 21.5 in die Funktion.

Listing 21.5: Die Funktion `OnGoBack` der Klasse `CWebBrowseView`

```
1: void CWebBrowseView::OnGoBack()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Zur vorherigen Seite gehen
6: GoBack();
7: }
```

Öffnen Sie den Klassen-Assistenten erneut, und fügen Sie eine Behandlungsfunktion für die Objekt-ID `IDM_GO_NEXT` für die Nachricht `COMMAND` hinzu. In diese Funktion nehmen Sie den Code aus Listing 21.6 auf.

Listing 21.6: Die Funktion `OnGoNext` der Klasse `CWebBrowseView`

```
1: void CWebBrowseView::OnGoNext()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Zur nächsten Seite gehen
6: GoForward();
7: }
```

Nun können Sie Ihre Anwendung ausführen und - egal, wohin Sie gerade gesurft sind - über die Menüs zu den vorherigen Webseiten zurückgehen und dann die gleiche Route wieder vorwärts einschlagen. Allerdings ist es schwerer, mit Menüs zu arbeiten, so daß man für die beiden Menübefehle Zugriffstasten vorsehen sollte.

Wenn Sie die Accelerator-Tabelle im Ressourcenbaum öffnen, finden Sie bereits ein ganzes Bündel von Zugriffstasten, die mit Menü-IDs verbunden sind. Jede dieser Zugriffstasten besteht aus einer ID und einer Tastenkombination. Wenn Sie an einer beliebigen Stelle in der Accelerator-Tabelle mit der

rechten Maustaste klicken, finden Sie im Kontextmenü den Befehl **Neue Zugriffstaste**. Wenn Sie diesen Befehl wählen, erscheint ein Dialogfeld, in das Sie die Angaben für die Zugriffstaste eintragen können. Als erstes ist die Menü-ID anzugeben, mit der die Zugriffstaste verbunden wird. (Wie bei Symbolleistschaltflächen sind Zugriffstasten mit Menübefehlen gekoppelt.) Darunter geben Sie die Taste an, die als Zugriffstaste wirksam ist, oder Sie wählen eine Taste aus der Dropdown-Liste aus.

Im rechten Teil des Dialogfelds können Sie die Zusatztaste festlegen. Dabei handelt es sich um Tasten, die in Verbindung mit der bereits festgelegten Zugriffstaste zu drücken sind, um die Funktion auszulösen. Nachdem Sie alle erforderlichen Informationen für die Zugriffstaste eingegeben haben, schließen Sie das Dialogfeld, um die spezifizierten Informationen in die Tabelle aufzunehmen.



Es empfiehlt sich, die Tasten (Strg) oder (Alt) als eine der Zusatztasten für alle Zugriffstasten zu verwenden, die mit Standardtasten arbeiten. Wenn Sie nicht eine dieser beiden Tasten als Teil der Zugriffstaste spezifizieren, kann die Anwendung nicht unterscheiden, ob der Benutzer Informationen eingeben oder die mit einer Zugriffstaste verbundene Funktion auslösen möchte.

Um Zugriffstasten für die Menüs **Zurück** und **Vorwärts** in Ihre Anwendung aufzunehmen, wählen Sie den Befehl **Neue Zugriffstaste** aus dem Kontextmenü und legen die ID mit **IDM_GO_BACK**, die Taste mit **VK_LEFT** und die Zusatztaste mit **Alt** fest. Für die zweite Zugriffstaste geben Sie **IDM_GO_NEXT** als ID, **VK_RIGHT** als Taste und **Alt** als Zusatztaste an. Wenn Sie die Anwendung ausführen, können Sie mit der Tastenkombination (Alt)+(æ) zur vorherigen Seite und mit der Tastenkombination (Alt)+(Æ) zur nächsten Seite gehen.



Das Feld **Taste** akzeptiert ein einzelnes Zeichen, einen ASCII-Wert oder einen virtuellen Tastenbezeichner. Da sich die Pfeiltasten nicht mit einem einzelnen Zeichen eingeben lassen, bleiben noch die beiden anderen Möglichkeiten. Wenn Sie nicht gerade alle ASCII-Werte für die Tasten im Kopf haben, sollten Sie die Taste aus der Dropdown-Liste auswählen. Der virtuelle Tastenbezeichner für die Pfeiltaste nach links lautet **VK_LEFT**, für die Pfeiltaste nach rechts **VK_RIGHT**.

Damit sich die Anwendung wie die meisten verfügbaren Webbrowser verhält, fügen Sie für die beiden Menübefehle noch Symbolleistschaltflächen hinzu, die mit einem Pfeil nach links bzw. rechts versehen sind.

Den Browser steuern

Beim Browsen gelangt man oft auf eine Webseite, bei der man das Herunterladen gar nicht abwarten will. Die Übertragung läßt sich auch mittendrin anhalten. Es kann sein, daß Sie den falschen URL eingegeben haben oder Ihnen der Download zu lange dauert. Es spielt keine Rolle, warum Sie den Download anhalten wollen. Es genügt, daß Sie es wollen. Aus diesem Grund verfügt die Klasse **CHTMLView** über die Funktion **Stop**. Diese Funktion bricht den momentan laufenden Download ab. Um diese Funktionalität in die Beispielanwendung aufzunehmen, fügen Sie dem Menü **Ansicht** mit dem Menü-Editor einen neuen Menübefehl hinzu. Legen Sie die Eigenschaften des Menübefehls gemäß Tabelle 21.3 fest.

Tabelle 21.3: Eigenschaften des Menübefehls Abbrechen

Objekt	Eigenschaft	Einstellung

Menübefehl	ID	IDM_VIEW_STOP
	Titel	A&bbrechen
	Statuszeilentext	Bricht die aktuelle Übertragung ab\nAbbrechen

Mit dem Klassen-Assistenten nehmen Sie eine Behandlungsroutine in die Ansichtsklasse für diese Menü-ID für die Nachricht **COMMAND** auf. In die Funktion schreiben Sie den Code aus Listing 21.7.

Listing 21.7: Die Funktion OnViewStop der Klasse CWebBrowseView

```

1: void CWebBrowseView::OnViewStop()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Bricht die aktuelle Übertragung ab
6: Stop();
7: }
```

Wenn Sie die Anwendung ausführen, können Sie mit diesem Menübefehl jegliches Herunterladen einer Webseite stoppen, auf die Sie nicht warten wollen. Es ist allerdings bequemer, wenn man das über eine Schaltfläche auf der Symbolleiste vornehmen könnte.

Die meisten Browser verfügen außerdem über die Möglichkeit, die aktuelle Seite neu zu laden. Diese Funktion ist vor allem für Webseiten praktisch, die dynamische Elemente enthalten, die sich bei jedem Herunterladen der Seite ändern. Sie ist auch hilfreich für Webseiten, die Ihr Browser im Cache abgelegt hat, so daß er die neueste Version der Seite vielleicht gar nicht abruft. Man muß den Browser dazu zwingen können, die Seite erneut zu laden und nicht einfach die Cache-Version anzuzeigen (insbesondere bei einer Webseite, die Sie gerade entwerfen). Diese Fähigkeit realisiert die Browser-Funktion **Refresh**. Ein Aufruf dieser Funktion bedeutet, daß die aktuelle Seite erneut geladen wird.

In die Beispielanwendung können Sie diese Funktionalität mit einem weiteren Menübefehl im Menü **Ansicht** realisieren. Legen Sie die Eigenschaften für den neuen Menübefehl gemäß Tabelle 21.4 fest. Zwischen den ursprünglich vorhandenen und den neuen Menübefehlen können Sie auch eine Trennlinie einfügen, so daß Ihr Menü wie in Abbildung 21.6 aussieht.

Tabelle 21.4: Eigenschaften des Menübefehls Aktualisieren

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_VIEW_REFRESH
	Titel	&Aktualisieren
	Statuszeilentext	Aktualisiert den Inhalt der aktuellen Seite\nAktualisieren

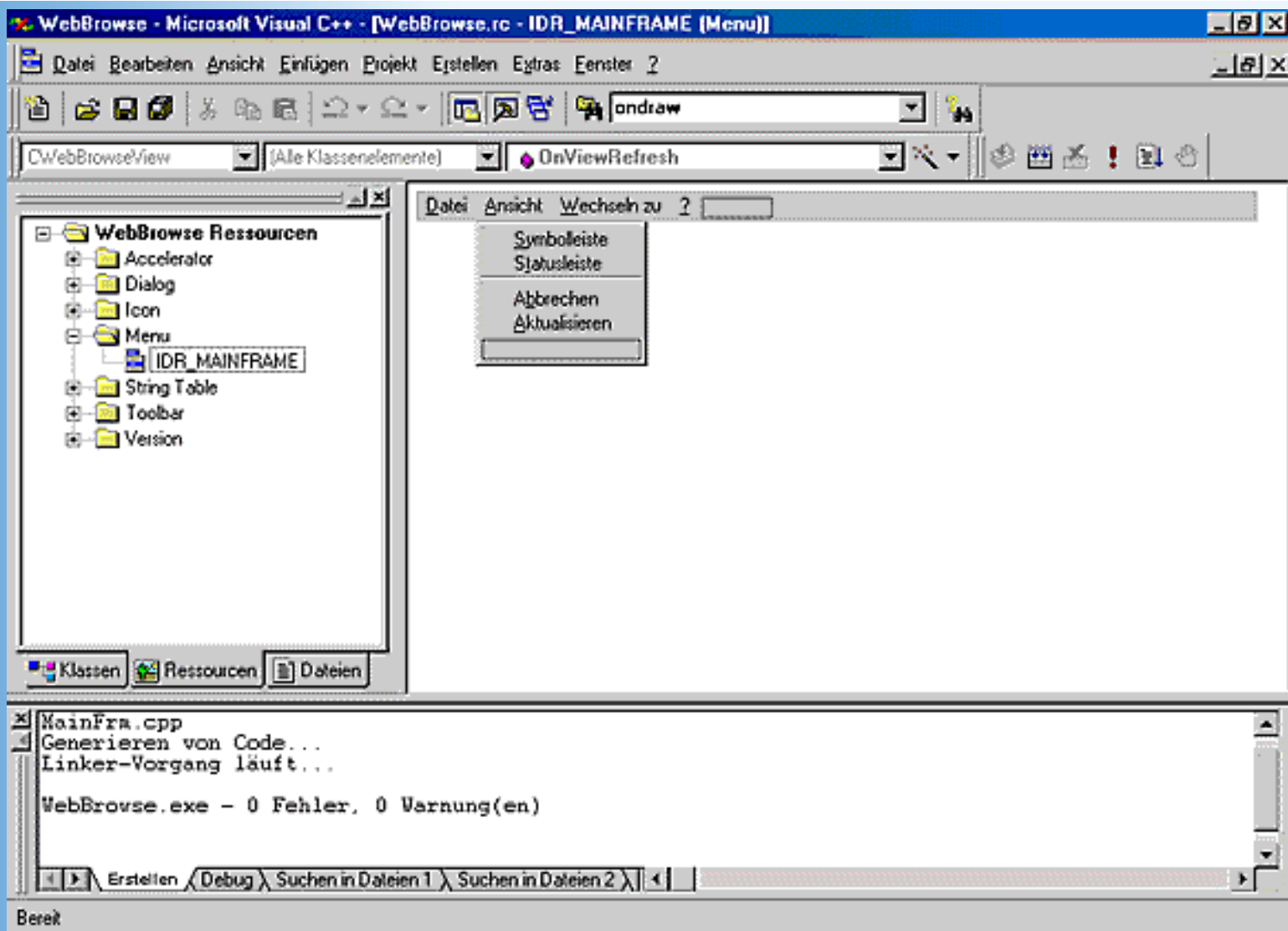


Abbildung 21.6:
Das modifizierte Menü Ansicht

Nachdem Sie den Menübefehl hinzugefügt haben, nehmen Sie mit dem Klassen-Assistenten eine Behandlungsroutine in die Ansichtsklasse für die Nachricht **COMMAND** für diesen Menübefehl auf. In die Funktion schreiben Sie den Code aus Listing 21.8.

Listing 21.8: Die Funktion OnViewRefresh der Klasse CWebBrowseView

```
1: void CWebBrowseView::OnViewRefresh()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Lädt die aktuelle Seite erneut
6: Refresh();
7: }
```

Diese Funktionalität können Sie nun testen, indem Sie eine Website suchen, die jedesmal eine andere Seite zurückliefert, wenn Sie den Browser aktualisieren, wie zum Beispiel **Random Monty Python Skit Server** (siehe Abbildung 21.7). Wie bei den übrigen Menüfunktionen, die Sie in die Beispielanwendung aufgenommen haben, sollten Sie auch für diese eine Schaltfläche auf der Symbolleiste vorsehen.



Abbildung 21.7:
Die Funktion Refresh leitet einen neuen Download ein.

Zusammenfassung

Heute haben Sie gelernt, wie Microsoft den Webbrowser Internet Explorer als eine Reihe von ActiveX-Komponenten konzipiert hat, die sich auch in anderen Anwendungen nutzen lassen. Es wurde gezeigt, wie Microsoft den Browser in der Klasse `CHtmlView` abgekapselt hat. Diese Klasse kann man in SDI- und MDI-Anwendungen nutzen, um die Funktionalität eines Browsers in nahezu jeder Anwendung umsetzen zu können. Sie haben gesehen, wie man die Dialogleiste verwendet, um Steuerelemente auf einer Symbolleiste unterzubringen, und wie sich die Ereignisse für diese Steuerelemente in der Rahmenklasse der Anwendung behandeln lassen. Schließlich haben Sie gelernt, wie man Menüs in die Anwendung aufnimmt, um die verschiedenen Funktionen des Webbrowsers aufzurufen, damit das Surferlebnis komplett wird.

Fragen und Antworten

Frage:

Warum gehört die Seitenansicht nicht zu den Standardmenüs, wenn ich die Klasse `CHtmlView` als Basisklasse für meine Ansichtsklasse wähle?

Antwort:

Die Druckfunktionen für die Klasse `CHtmlView` übernimmt der Browser und nicht die Ansichtsklasse.

Die Seitenansicht ist nicht vorhanden, da sie der Browser nicht unterstützt.

Frage:

Wie kann die den HTML-Quellcode vom Browser beziehen, damit ich ihn ansehen oder bearbeiten kann?

Antwort:

Die Klasse CHtmlView verfügt über die Member-Funktion GetHtmlDocument, die einen Zeiger auf die Schnittstelle IDispatch des Dokumentobjekts im Browser zurückliefert. Verwenden Sie die Methode Invoke der Schnittstelle IDispatch, um die Funktionen im Dokumentobjekt aufzurufen und den HTML-Quellcode zu erhalten. Weitere Hinweise dazu finden Sie auf der Microsoft Developer Network CD, die zum Lieferumfang von Visual C++ gehört.

Workshop

Kontrollfragen

1. Was verkapselt die Klasse CHtmlView für den Einsatz in Visual-C++-Anwendungen?
2. Wie kann man den URL für die aktuelle Webseite aus der Klasse CHtmlView erhalten?
3. Welcher Befehl wird für die Rahmenklasse ausgelöst, wenn der Benutzer die (C)- Taste im Eingabefeld der Dialogleiste drückt?
4. Welche Funktionen können Sie aufrufen, um den Browser zur vorherigen und zur nächsten Webseite zu navigieren?
5. Wie kann man einen laufenden Download anhalten?

Übungen

1. Fügen Sie die Funktion GoSearch in das Menü und die Symbolleiste ein.
2. Nehmen Sie die Funktion GoHome in das Menü und die Symbolleiste auf.
3. Deaktivieren Sie die Schaltfläche **Abbrechen** und den zugehörigen Menübefehl, wenn die Anwendung keine Webseite herunterlädt.

Woche 3 im Rückblick

Sie haben es geschafft! Nunmehr verfügen Sie über die Fertigkeiten, sich den meisten Programmieraufgaben mit Visual C++ unter Windows anzunehmen, ausgenommen diejenigen, die spezielle Kenntnisse erfordern. Obwohl es noch viel zu lernen gibt, haben Sie den größten Teil der Themen in einer sehr kurzen Zeit abgehandelt. Von hier aus werden Sie sich wahrscheinlich einem oder zwei spezielleren Bereichen der Windows-Programmierung - zu denen selbst ganze Bücher erschienen sind - widmen, da Sie momentan über ein umfangreiches Allgemeinwissen verfügen.

Für den Fall, daß Sie noch nicht alle Themen beherrschen, empfiehlt es sich, daß Sie noch einmal an Ihren eigenen Anwendungen arbeiten und das Gelernte hier ausprobieren. Das hilft Ihnen, die Bereiche herauszufinden, die Sie vielleicht wiederholen sollten, bevor Sie sich komplizierteren Themen zuwenden. Sehen wir uns zur Sicherheit noch einmal kurz an, was Sie in der letzten Woche gelernt haben.

Die Woche hat mit der neuesten Datenbanktechnologie von Microsoft, den ActiveX Data Objects (ADO), begonnen. Sie haben erfahren, wie ADO auf der Technologie OLE DB aufsetzt und dabei den Zugriff auf Datenbanken beträchtlich vereinfacht. Es wurde gezeigt, wie leicht sich eine Datenbankanwendung mit ADO als ActiveX-Steuererelement erstellen läßt und wie man sie mit bestimmten ADO-kompatiblen, datengebundenen ActiveX-Steuererelementen verbindet. Weiterhin haben Sie gelernt, wie man die DLL importiert und in den Code eintaucht. Dadurch können Sie vollständig Einfluß darauf nehmen, wie die Anwendung mit der Datenbank zusammenarbeitet. Es wurde gezeigt, wie man die einzelnen Felder eines Datensatzes im Recordset liest und schreibt, indem man sie aus und in den Datentyp `Variant` konvertiert. Darüber hinaus haben Sie Ihre eigene benutzerdefinierte Datensatzklasse entworfen und sie mit dem Recordset-Objekt gebunden, was Ihnen einen wesentlich bequemeren Weg zum Zugriff auf die einzelnen Feldwerte in der Datenbank eröffnet. Wenn Sie Probleme mit diesem Thema haben, sollten Sie Tag 15 wiederholen.

Sie haben drei verschiedene Arten der gemeinsamen Nutzung von funktionellen Modulen kennengelernt, die Sie mit anderen Programmierern entwickeln, ohne daß Sie den eigentlichen Code freigeben müssen. Als erstes haben Sie Ihre Module in Bibliotheksdateien untergebracht, die sich zu Anwendungen anderer Visual- C++-Entwickler linken lassen. Diese Lösung hat den Nachteil, daß bei Änderungen am Bibliotheksmodul alle Anwendungen, die auf das Modul zurückgreifen, neu zu erstellen sind. Sie haben gelernt, wie man diese Module erstellt, ohne daß spezielle Änderungen beim Schreiben des Codes erforderlich sind. Wenn Sie nicht mehr genau wissen, wie das alles zu realisieren ist, gehen Sie noch einmal zu Tag 16 zurück.

Die zweite Möglichkeit, um Funktionalität mit anderen Programmierern gemeinsam zu nutzen, ist das Erstellen von DLLs. Dabei haben Sie zwei verschiedene Arten von DLLs erstellt: Der eine Typ läßt sich nur von anderen Visual-C++-Anwendungen nutzen, der andere von beliebigen Anwendungen, unabhängig davon, mit welcher Programmiersprache sie erstellt wurden. Sie haben eine DLL erstellt, die andere Visual-C++-Programmierer nutzen können, ohne daß Sie gänzlich anders vorgehen

mußten, um die Module zu entwerfen oder zu kodieren. Sie haben aber auch gelernt, daß Ihr Modul grundsätzlich anders aufzubauen ist, damit der Einsatz und die Interaktion in DLLs, die man mit allen Programmiersprachen einsetzen kann, gewährleistet sind. Es wurde gezeigt, wie man gewöhnliche Funktionsaufrufe als Schnittstelle für andere Anwendungen bereitstellt, wobei alle erforderlichen Informationen als Parameter an die Funktionen zu übergeben sind. Ein weiteres Thema waren die Definitionsdateien, die die Namen aller zu exportierenden Funktionen enthalten. Sehen Sie sich Tag 17 noch einmal an, falls Ihnen noch nicht alle Verfahren in diesem Themenbereich geläufig sind.

Der dritte und letzte Weg, um die Funktionalität Ihrer Module gemeinsam zu nutzen, ohne den eigentlichen Code freizulegen, war die Verpackung als ActiveX- Steuerelement. Sie haben mit dem Steuerelement-Assistenten und dem Klassen- Assistenten die Eigenschaften, Methoden und Ereignisse hinzugefügt, die für das Steuerelement erforderlich waren. Außerdem haben Sie gelernt, wie man die Eigenschaften des Steuerelements liest und schreibt. Die Eigenschaften haben Sie nach zwei unterschiedlichen Verfahren implementiert, und Sie haben gelernt, wie man den jeweils passenden Typ für die Eigenschaften des Steuerelements wählt. Es wurde gezeigt, wie man Ereignisse in der Containeranwendung vom Steuerelement aus auslöst, indem man das Ereignis im Code initiiert. Das Steuerelement haben Sie dann mit dem Testcontainer für ActiveX-Steuerelemente ausprobiert und dabei alle Methoden aufgerufen und die vorhandenen Ereignisse des Steuerelements abgefangen. Sie haben gesehen, wie man die Ereignisse überwacht, die das Steuerelement in der Containeranwendung auslöst, um zu gewährleisten, daß die Ereignisse zum richtigen Zeitpunkt ausgelöst werden. Wiederholen Sie Tag 19, falls Ihnen zu diesem Themengebiet noch etwas unklar ist.

Die gleichzeitige Ausführung von Aufgaben bildete einen weiteren Schwerpunkt der dritten Woche. Mehr und mehr moderne Anwendungen bieten eine derartige Funktionalität. Dabei haben Sie nicht nur gelernt, wie man mehrere Aufgaben gleichzeitig ausführt, sondern es wurden auch zwei unterschiedliche Lösungsmöglichkeiten dargestellt. Als erstes wurde die Funktion `OnIdle` behandelt und gezeigt, wie man sich in diese Funktion einklinkt, um die eigene Funktionalität auszulösen, wenn die Anwendung im Leerlauf arbeitet. Dabei wurde auch auf die Mängel dieses Verfahrens eingegangen, die sich zeigen, wenn man eine zweite Task in die Anwendung aufnimmt. Diese kann verhindern, daß der Benutzer auf die Anwendung reagieren kann. Die Hintergrundverarbeitung mußte in kleinere Teile zerlegt werden, die sich schnell ausführen lassen. Dazu war eine Möglichkeit herauszuarbeiten, wie man verfolgen kann, in welcher Phase die Task gerade arbeitet und wo sie die Arbeit aufnehmen muß, wenn die Anwendung wieder in den Leerlaufzustand gelangt.

Als zweites Verfahren, um einer Anwendung die Ausführung einer zweiten oder dritten Task zu ermöglichen, wurden separate Threads vorgestellt, die vollkommen unabhängig vom Thread der Benutzeroberfläche laufen. In diesem Zusammenhang haben Sie Callback-Funktionen kennengelernt, die die höchste Ausführungsebene für die Threads steuern, und wie man den Thread bei Bedarf starten und anhalten kann. Sie haben auch gesehen, wie diese unabhängigen Threads vollkommen losgelöst von der übrigen Anwendung arbeiten und wie sie weiterlaufen, selbst wenn die übrige Anwendung beschäftigt ist. Wenn Sie sich mit diesem Thema nochmals befassen möchten, gehen Sie zurück zu Tag 18.

Ein Bereich, der zunehmend Bedeutung gewinnt, ist das Erstellen von Internet-Anwendungen mit den Klassen der Winsock-Schnittstelle. Sie haben eine Anwendung erstellt, die eine Verbindung mit einer anderen Anwendung über ein Netzwerk eingeht und Nachrichten in beiden Richtungen austauscht. Wie beim Telefon muß die zweite Anwendung auf ein Verbindungsgesuch der ersten Anwendung hören. Es wurde gezeigt, wie einfach es ist, Nachrichten zu senden und benachrichtigt zu werden, wenn die Nachricht angekommen ist, nachdem die Verbindung zwischen beiden Anwendungen

hergestellt wurde. Sollten sich in diesem Zusammenhang bei Ihnen Fragen ergeben, wiederholen Sie den Stoff von Tag 20.

Schließlich haben Sie den Webbrowser Internet Explorer von Microsoft in Ihre Anwendung ohne großen Aufwand integriert. Sie haben gelernt, wie man den Browser steuert, indem man den URL angibt, der zu laden und anzuzeigen ist. Es wurde auch dargestellt, wie man Informationen für den Benutzer sichtbar macht, was der Browser gerade ausführt und wann er beschäftigt ist. Falls Sie Ihre Kenntnisse zu diesem Thema auffrischen möchten, gehen Sie noch einmal zurück zu Tag 21.

Das war's. Sie haben es geschafft. Sie haben bei den Grundlagen begonnen und sich bis zu komplizierteren Themen - insbesondere in der letzten Woche - vorgearbeitet. Jetzt ist es an der Zeit, das Buch beiseite zu legen und aktiv in die Programmierung einzusteigen. Erstellen Sie auf der Basis des Gelernten eigene Anwendungen. Viel Glück. Wenn Sie Hilfe oder einen Rat brauchen, sollten Sie sich in den Newsgroups von Microsoft im Internet umsehen. Hier gibt es viele Gleichgesinnte, die sowohl kenntnisreich als auch hilfreich sind.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Tag A

C++ im Überblick

Dieser Anhang soll Ihnen einen schnellen Überblick über die Grundlagen der Programmiersprache C++ verschaffen. Nachdem Sie sich mit diesem Anhang beschäftigt haben, sollten Sie ein umfassendes Verständnis für die verschiedenen Aspekte von C++ und seiner Syntax besitzen.

Die erste Anwendung erstellen

Das erste Beispiel ist ein einfaches Programm, das die Meldung "Hello World" auf dem Bildschirm ausgibt. Dazu erstellen Sie einen Arbeitsbereich und die für das Programm erforderliche C++-Datei. Mit Visual C++ läßt sich ein C++-Programm schnell und einfach schreiben. Führen Sie die folgenden Schritte aus:

1. Starten Sie Visual C++ über **Start / Programme / Microsoft Visual Studio 6 / Microsoft Visual C++ 6.0**.
2. Wählen Sie **Datei / Neu** im Menü von Visual C++.

Gehen Sie auf die Registerkarte **Projekte** (siehe Abbildung A.1)

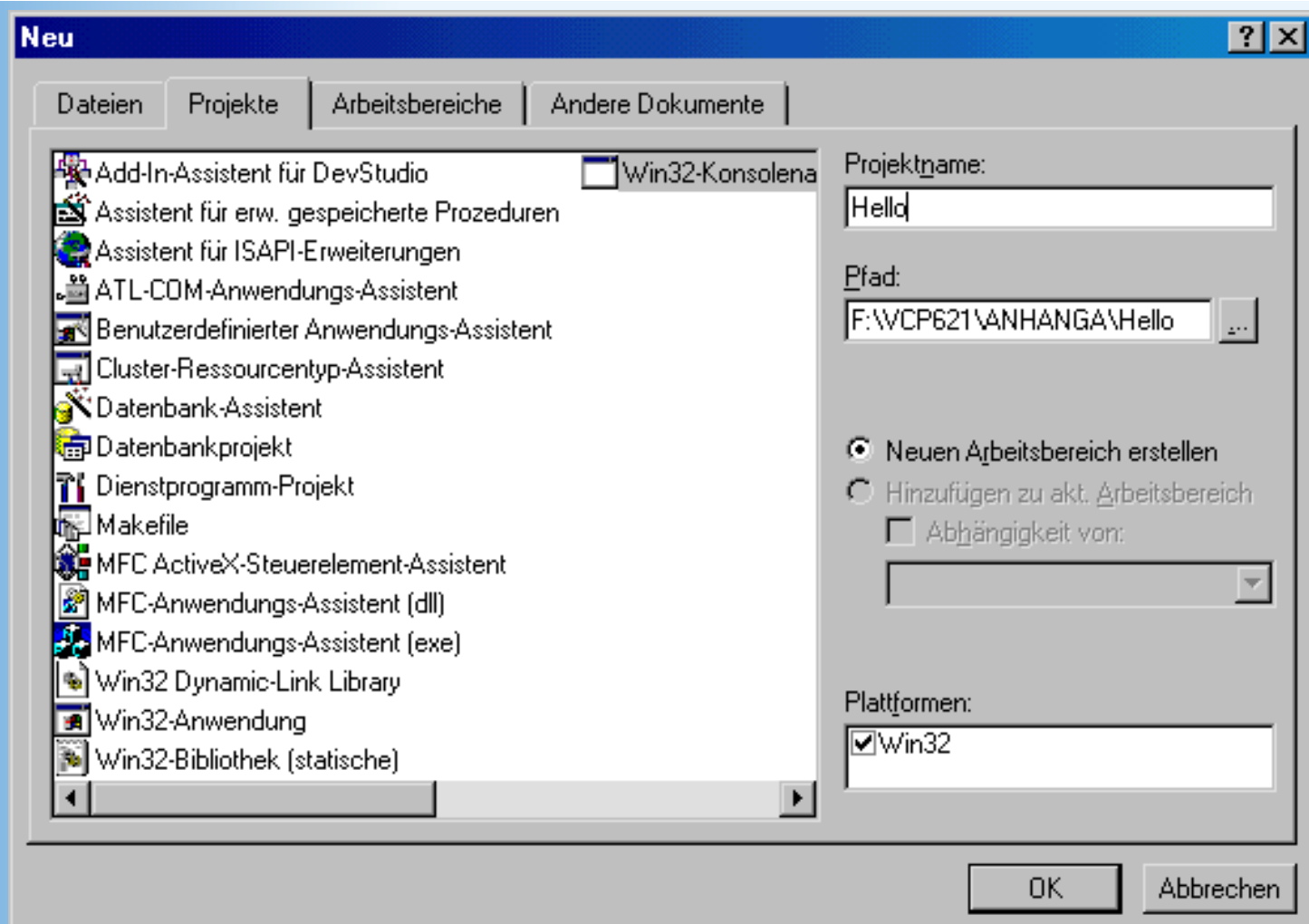


Abbildung A.1:
Den Arbeitsbereich für das Projekt Hello einrichten

3. Markieren Sie **Win32-Konsolenanwendung** im linken Fensterbereich.
4. Tippen Sie **He l l o** in das Eingabefeld **Projektname** ein.
5. Klicken Sie auf **OK**. Übernehmen Sie im ersten und einzigen Schritt des Assistenten die Option **Leeres Projekt**. Klicken Sie auf **Fertigstellen** und im nächsten Dialogfeld auf **OK**.

Visual C++ erstellt den Arbeitsbereich Ihrer Anwendung. Zu diesem Zweck legt Visual C++ ein Verzeichnis **He l l o** an, so daß Sie alle Dateien, die sich auf ein bestimmtes Projekt beziehen, in einem bestimmten Bereich unterbringen können. Als nächstes fügen Sie die erforderlichen Dateien für das Projekt hinzu:

1. Wählen Sie wieder **Datei / Neu**.
2. Gehen Sie auf die Registerkarte **Dateien**, falls diese nicht bereits ausgewählt ist.
3. Markieren Sie den Eintrag **C++-Quellcodedatei**.
4. Schalten Sie das Kontrollkästchen **Dem Projekt hinzufügen** im rechten Fensterbereich ein.
5. In das Eingabefeld **Dateiname** geben Sie **He l l o w o r l d** ein (siehe Abbildung A.2).

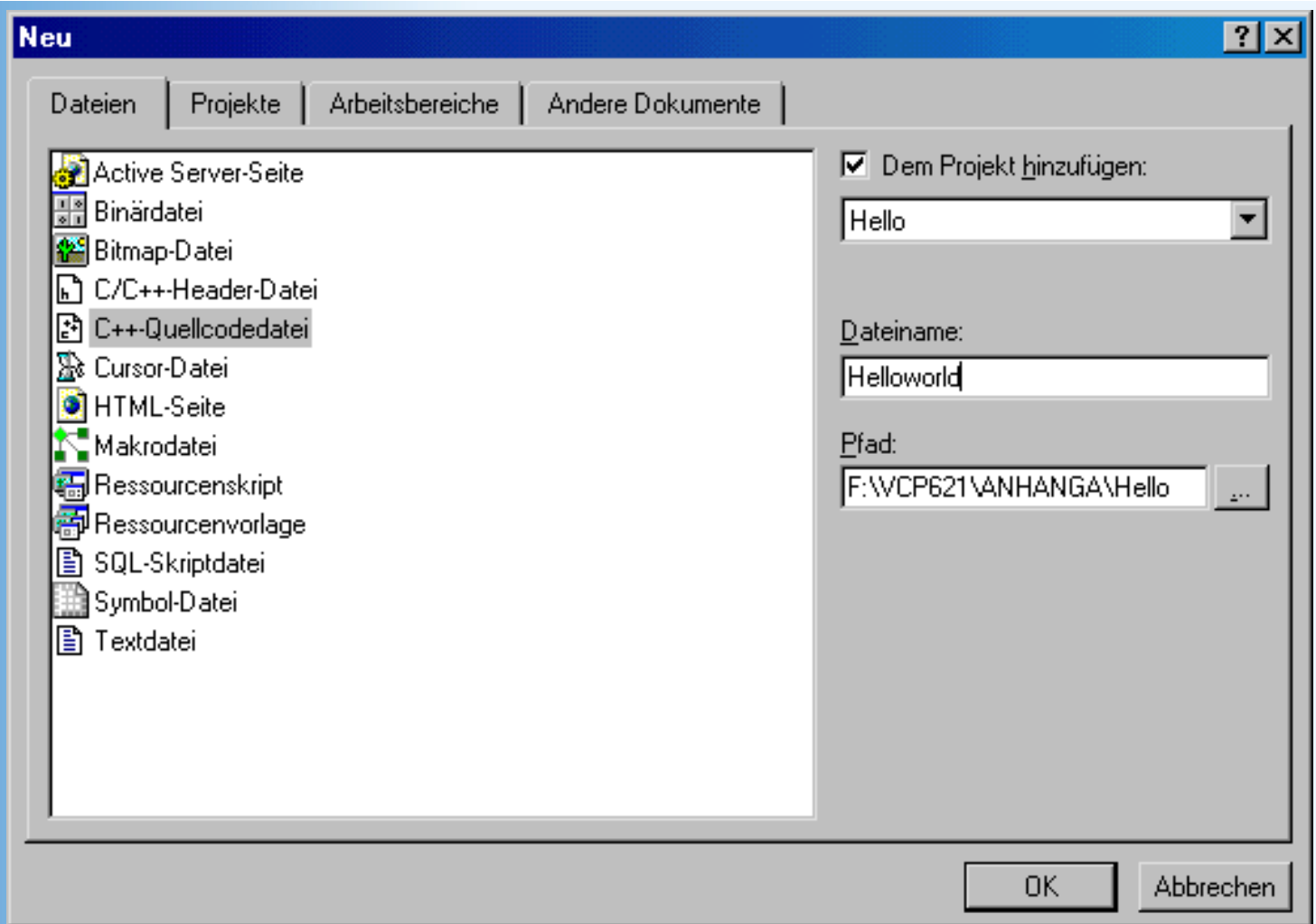


Abbildung A.2:
Das Projekt Helloworld einrichten

6. Klicken Sie auf **OK**.

In die Datei `Helloworld.cpp` schreiben Sie den C++-Quellcode. Alle C++-Quelldateien weisen die Erweiterung `.cpp` auf. Später kommen wir noch auf andere Dateitypen zu sprechen.

Alle Lehrbeispiele in diesem Abschnitt erstellen Sie in ähnlicher Weise, sie unterscheiden sich einzig in den Namen der Arbeitsbereiche und der Dateien.

Helloworld.cpp

Das Programm `Helloworld` zeigt den Text `HELLO WORLD` auf dem Bildschirm an. Listing A.1 enthält den Code. Tippen Sie den Code exakt wie angegeben in das Fenster `Helloworld.cpp` ein. Die Zeilennummern lassen Sie aber bitte weg, sie sind in den Listings nur angegeben, um im Text einfacher auf bestimmte Codeabschnitte verweisen zu können. C++ beachtet die Groß-/Kleinschreibung. Demzufolge ist `main` nicht dasselbe wie `MAIN` oder `Main`.

Listing A.1: Die Quelldatei `Helloworld.cpp`

```

1: // Arbeitsbereich: Hello
2: // Programmname: Helloworld.cpp
3:
4: # include <iostream.h>
5:
6: int main()
7:

```

```

8: {
9: cout<< "HELLO WORLD \n";
10: return 0;
11: }

```

Das Programm führen Sie folgendermaßen aus:

1. Wählen Sie **Datei / Speichern**, um Ihre Arbeit zu sichern.
2. Wählen Sie **Erstellen / Aktive Konfiguration festlegen** (siehe Abbildung A.3).

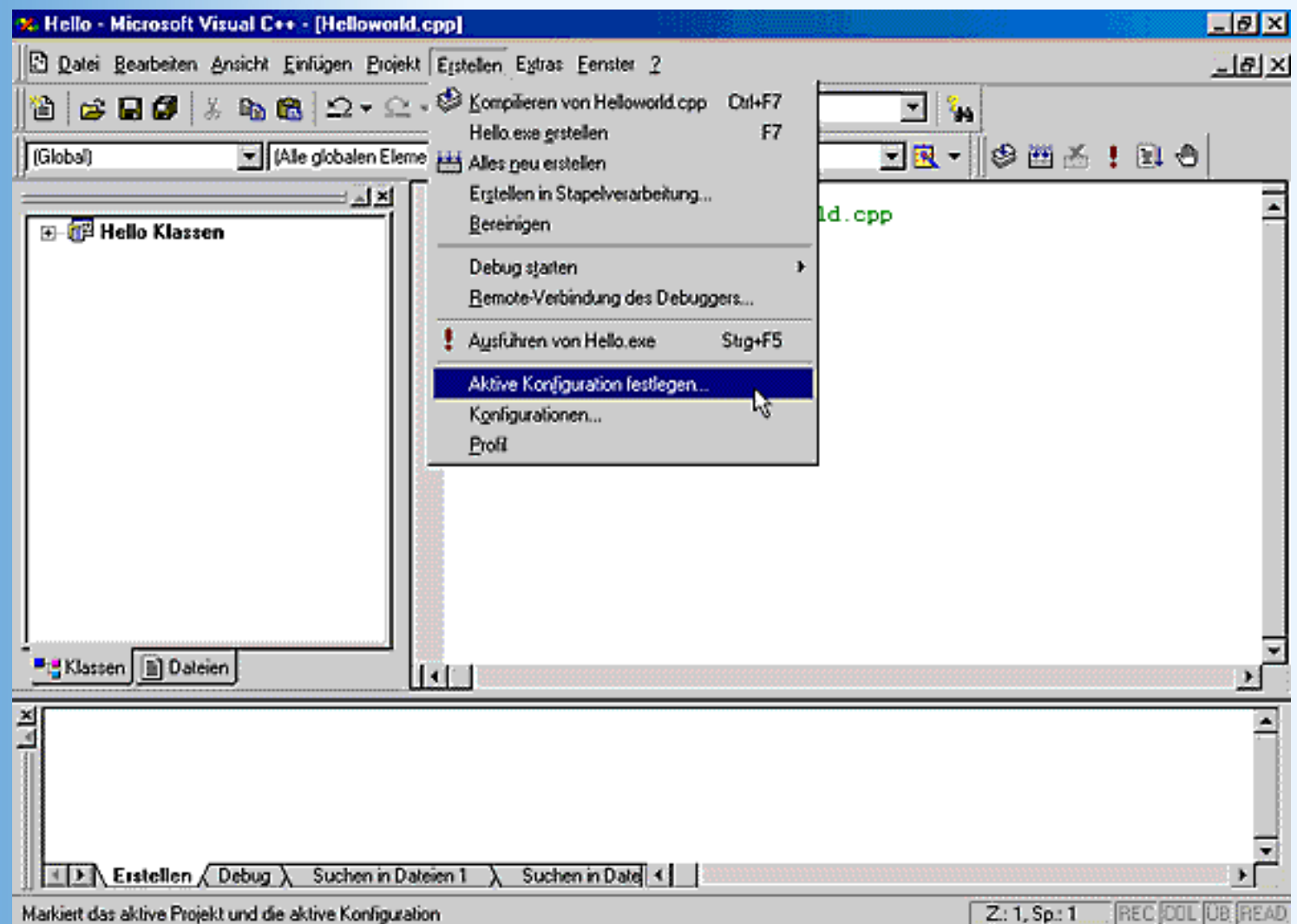


Abbildung A.3:
Die aktive Konfiguration festlegen

3. Markieren Sie **Hello - Win32 Debug**, und klicken Sie auf **OK** (siehe Abbildung A.4).

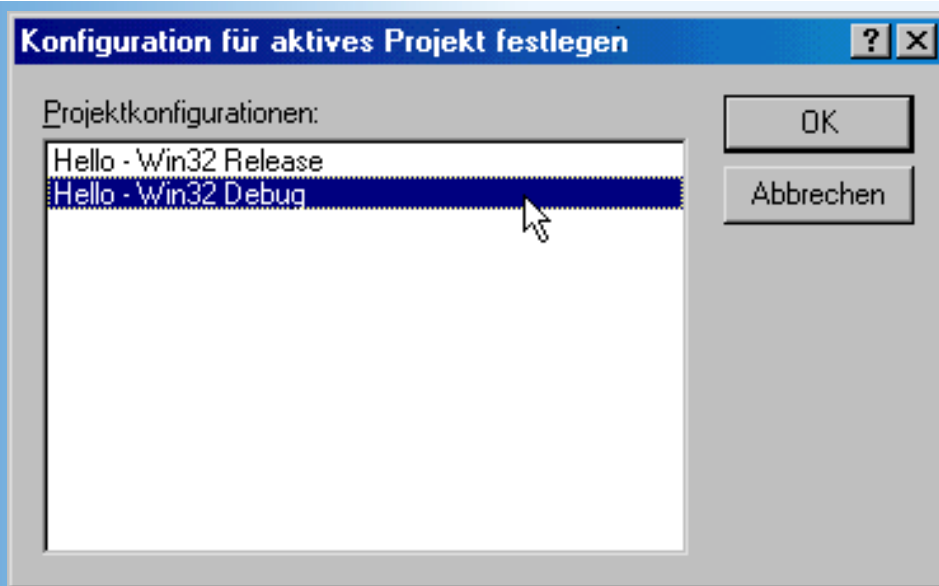


Abbildung A.4:
Win32 Debug auswählen

4. Wählen Sie **Erstellen / Hello.exe erstellen**.

Visual C++ kompiliert und linkt das Programm, um eine ausführbare Datei zu erzeugen. Das Ausgabefenster informiert Sie über den Erfolg oder das Scheitern der Kompilierung. Eine erfolgreiche Kompilierung sollte die Ausgabe

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

liefern. Falls irgendwelche Fehler auftreten, prüfen Sie bitte nach, ob Sie alle Programmzeilen genau wie im Listing (ohne Zeilennummern) eingegeben haben.

Um das Programm zu starten, wählen Sie **Erstellen / Ausführen von Hello.exe**.

Das Programm öffnet eine MS-DOS-Eingabeaufforderung und zeigt den Text HELLO WORLD an (siehe Abbildung A.5).

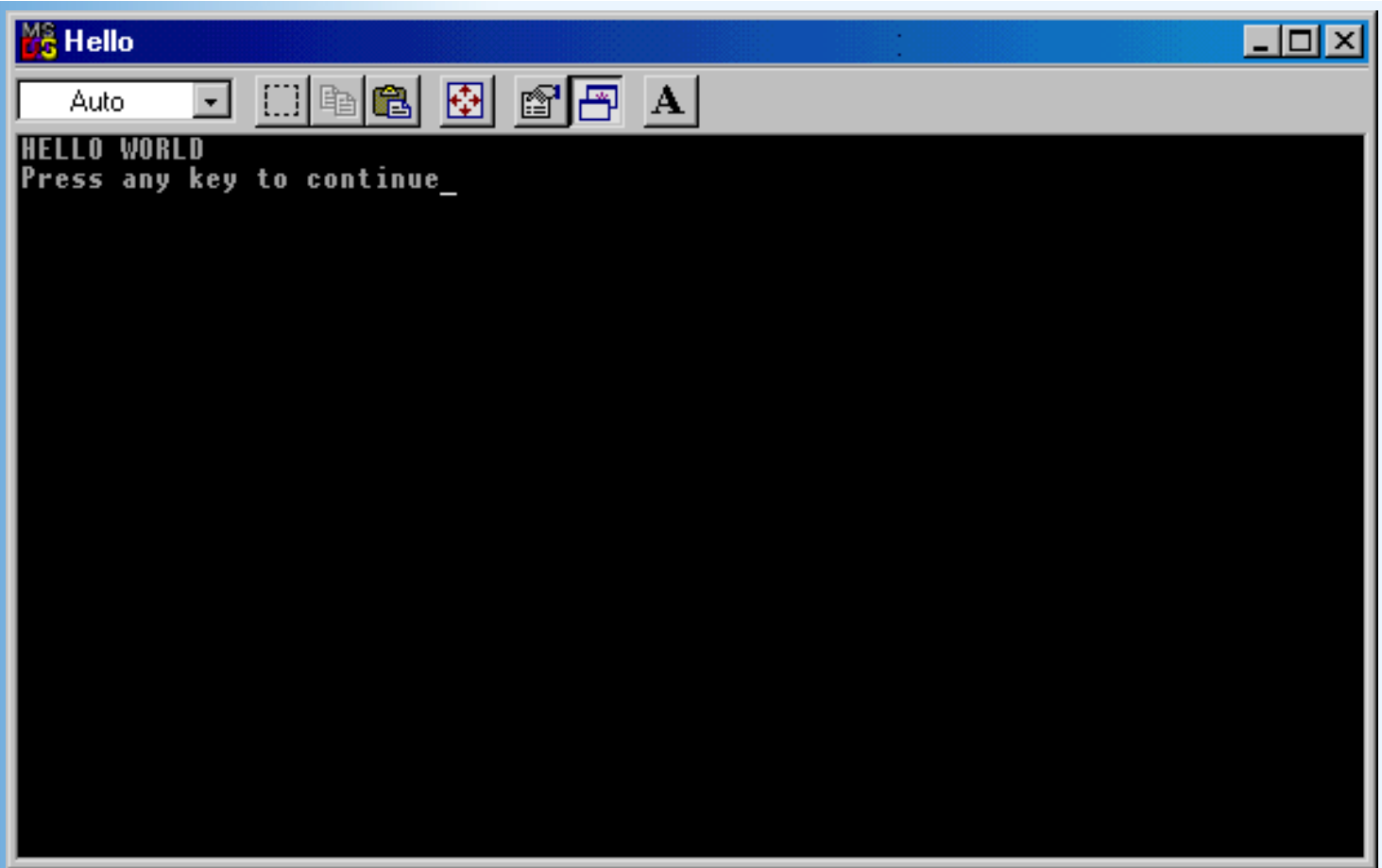


Abbildung A.5:
Die Anzeige HELLO WORLD

Komponenten von Helloworld.cpp

Die beiden ersten Zeilen des Programms sind Kommentare:

```
// Arbeitsbereich: Hello
// Programmname: Helloworld.cpp
```

Der Befehl mit doppelten Schrägstrichen (//) weist den Compiler an, alle nachfolgenden Zeichen auf der Zeile zu ignorieren. Es gehört zum guten Programmierstil, die eigene Arbeit zu kommentieren, da sich dadurch das Programm leichter lesen lässt, insbesondere für jemanden, der es selbst nicht geschrieben hat. Kommentare zählen sich vor allem aus, wenn Sie an einem komplexen Programm über Monate hinweg arbeiten. Bei allen Änderungen helfen Ihnen Kommentare, Ihre vor längerer Zeit gesponnenen Gedankengänge auf Anhieb nachvollziehen zu können.

Zeile 4 beginnt mit dem Nummernzeichen (#):

```
# include <iostream.h>
```

Das ist eine Direktive an den Präprozessor, die angegebene Datei (`iostream.h`) einzubinden (`include`). Die spitzen Klammern (`< >`) weisen den Präprozessor an, nach der Datei in den Standardverzeichnissen zu suchen. Die Datei `iostream.h` enthält Definitionen für die Ausgabe- (`<<`) und Eingabe- (`>>`) Operatoren. Die Direktive ist erforderlich, um die Anweisung `cout` zu verarbeiten, die in Zeile 9 des Programms definiert ist. Die Datei `iostream.h` ist ein vorkompilierter Header, der zum Lieferumfang des Compilers gehört. Mit dem Programm `HelloWorld` können Sie experimentieren, indem Sie die Include-Zeile auskommentieren. Fügen Sie dazu die doppelten Schrägstriche (`//`) vor dem Nummernzeichen (#) ein. Wenn Sie das Programm jetzt kompilieren und ausführen, erhalten Sie einen Fehler:

```
Helloworld.cpp
```

```
f:\vcp621\anhanga\hello\helloworld.cpp(9) : error C2065: 'cout' :
nichtdeklarerter Bezeichner
f:\vcp621\anhanga\hello\helloworld.cpp(9) : error C2297: '<<' :
Ungültig, da der rechte Operand vom Typ 'char [14]' ist
Fehler beim Ausführen von cl.exe.
```

Hello.exe - 2 Fehler, 0 Warnung(en)

Ohne die Datei `iostream.h` erkennt das Programm weder den Befehl `cout` noch den Einfügeoperator (`<<`).

In der nächsten Codezeile (Zeile 6) beginnt die eigentliche Programmausführung. Hier befindet sich der Eintrittspunkt Ihres Codes:

```
int main()
```

Diese Zeile weist den Compiler an, eine Funktion namens `main` zu verarbeiten. Jedes C++-Programm stellt eine Sammlung von Funktionen dar. Auf Funktionen gehen wir weiter hinten in diesem Anhang näher ein. Momentan sollten Sie wissen, daß man eine Funktion mit einem bestimmten Namen als Eintrittspunkt für einen Codeblock definiert. Die leeren Klammern zeigen an, daß die Funktion keinerlei Parameter übernimmt. Die Übergabe von Parametern in Funktionen behandelt der Abschnitt »Funktionen und Variablen« weiter hinten in diesem Anhang.

Jedes C++-Programm muß die Funktion `main()` enthalten. Sie stellt den Eintrittspunkt für die Programmausführung dar. Wenn eine Funktion einen Rückgabewert liefert, muß vor dem Funktionsnamen der Typ des zurückgegebenen Wertes stehen. In diesem Fall gibt die Funktion `main()` einen Wert vom Typ `int` zurück.

Der von einer Funktion definierte Codeblock ist in geschweifte Klammern (`{ }`) einzuschließen:

```
{
cout<< "HELLO WORLD \n";
return 0;
}
```

Der gesamte Code innerhalb dieser geschweiften Klammern gehört zur benannten Funktion - in diesem Fall `main()`.

Die nächste Zeile führt das `cout`-Objekt aus. Daran schließt sich der Ausgabeoperator (`<<`) an, der die anzuzeigenden Informationen übergibt. Der anzuzeigende Text ist in Anführungszeichen zu setzen. Auf den eigentlichen Text folgt das Zeichen `\n`, das den Wechsel auf eine neue Zeile bewirkt. Der Ausgabe- oder Einfügeoperator (`<<`) bedeutet, daß alle nachfolgenden Zeichen in `cout` einzufügen sind.



Zeile 9 endet mit einem Semikolon. Alle Anweisungen in C++ sind mit einem Semikolon abzuschließen.

Zeile 10 des Codebeispiels enthält eine `return`-Anweisung. Programmierer geben damit in der Regel bestimmte Werte oder Fehlercodes zurück. Sehen Sie sich noch einmal Zeile 7 an, in der die Funktion `main()` steht. Dort ist auch der Rückgabotyp als ganze Zahl - oder Integer-Zahl - mit `int` definiert. Führen Sie den Code erneut aus, diesmal aber ohne die `return`-Anweisung in Zeile 10. Dazu ist Zeile 7 wie folgt zu modifizieren:

```
void main()
```

Es gehört zum guten Programmierstil, daß man bei komplexen Programmen Rückgabecodes vorsieht.

Damit lassen sich Fehler im Programm besser erkennen und verfolgen.

Funktionen und Variablen

Das Programm `HelloWorld` enthält nur eine Funktion, `main()`. Ein funktionales C++- Programm besteht normalerweise aus wesentlich mehr Funktionen. Um eine Funktion zu verwenden, muß man sie zuerst deklarieren. Eine Funktionsdeklaration bezeichnet man auch als Prototyp. Ein Prototyp liefert eine knappe Repräsentation der gesamten Funktion. Wenn man den Prototyp einer Funktion angibt, schreibt man eigentlich eine Anweisung. Wie bereits weiter oben erwähnt, müssen alle Anweisungen in C++ mit Semikolons enden. Ein Funktionsprototyp besteht aus einem Rückgabetyt, dem Namen der Funktion und einer Parameterliste. Der Rückgabetyt in der Funktion `main()` lautet `int`, der Name `main` und die Parameterliste ist `()` oder leer.

Eine Funktion muß einen Prototyp und eine Definition haben. Der Prototyp und die Definition einer Funktion müssen bezüglich Rückgabetyt, Name und Parameterliste übereinstimmen. Der einzige Unterschied besteht darin, daß der Prototyp eine Anweisung ist und demzufolge mit einem Semikolon endet. Listing A.2 verdeutlicht diesen Punkt mit einem einfachen Programm, das die Fläche eines Dreiecks berechnet.

Listing A.2: Das Programm `Area.cpp`

```
1: // Arbeitsbereich: Dreieck
2: // Programmname: Area.cpp
3: // Die Dreiecksfläche berechnet sich zu Grundseite mal Höhe durch 2
4: // Dreiecksfläche = (Grundseite * Höhe)/2
5:
6: #include <iostream.h> // Vorkompilierter Header
7:
8: double base,height,area; // Variablen deklarieren
9: double Area(double,double); // Prototyp/Deklaration der Funktion
10:
11: int main()
12: {
13:     cout << "Hoehe des Dreiecks eingeben: "; // Eine Zahl eingeben
14:     cin >> height; // Eingabe in Variable speichern
15:     cout << "Grundseite des Dreiecks eingeben: "; // EinZahl eingeben
16:     cin >> base; // Eingabe in Variable speichern
17:
18:     area = Area(base,height); // Ergebnis der Funktion Area in der
19:     // Variablen area speichern
20:     cout << "Die Dreiecksfläche betraegt: " << area << endl ; // Fläche
    ausgeben
21:
22:     return 0;
23: }
24:
25: double Area (double base, double height) // Funktionsdefinition
26: {
27:     area = (0.5*base*height);
28:     return area;
29: }
```

Das Programm deklariert in Zeile 8 drei Variablen, `base`, `height` und `area`. Variablen speichern Werte, auf die das Programm zurückgreift. Der Typ der Variablen spezifiziert die in der Variablen zu speichernden Werte. Tabelle 22.1 zeigt die von C++ unterstützten Typen.

Tabelle A.1: Datentypen von Variablen

Datentyp	Wertebereich
unsigned short int	0 bis 65,535
short int	-32,768 bis 32,767
unsigned long int	0 bis 4,294,967,925
long int	-2,147,483,648 bis 2,147,483,647
int	-2,147,483,648 bis 2,147,483,647 (32 Bit)
unsigned int	0 bis 4,294,967,925 (32 Bit)
char	256 Zeichenwerte
float	1.2e-38 bis 3.4e38
double	2.2e-308 bis 1.8e308

Um eine Variable zu definieren, geben Sie zuerst den Typ an und daran anschließend den Namen. Der Variablen lassen sich mit dem Zuweisungsoperator (=) Werte zuweisen, wie es die beiden folgenden Beispiele zeigen:

```
double base = 5;
unsigned long int base = 5;
```

In C++ kann man auch eine eigene Typdefinition festlegen. Dazu dient das Schlüsselwort `typedef`, auf das der vorhandene Typ und der Name folgen:

```
typedef unsigned long int ULONG;
ULONG base = 5;
```

Wenn man eigene Typen festlegt, kann man sich den Aufwand sparen, die gesamte Deklaration zu schreiben.

Die nächste Codezeile (Zeile 9) definiert den Prototyp der Funktion:

```
double Area (double, double);
```

Diese Funktion hat den Typ `double`, den Namen `Area` und eine Parameterliste mit zwei Variablen vom Typ `double`. Bei der Definition des Prototyps ist es zwar nicht erforderlich, die eigentlichen Parameter zu definieren, dennoch gehört es zum guten Programmierstil, die Parameternamen aufzuführen. Dieses Programm übernimmt zwei Eingaben vom Benutzer, nämlich `base` (die Länge der Grundseite) und `height` (die Höhe) des Dreiecks, und berechnet dessen Fläche (`area`). Die Angaben `base`, `height` und `area` sind Variablen. Das Beispiel `HelloWorld.cpp` hat den Ausgabeoperator (<<) verwendet. Im jetzigen Beispiel kommt der Eingabeoperator (>>) vor. Das Programm fordert den Benutzer in Zeile 13 auf, einen Wert für die Höhe des Dreiecks einzugeben. Wenn der Benutzer einen Wert für `height` eingibt, werden die Daten vom Bildschirm geholt und in die Variable `height` geschrieben. Dieser Ablauf wiederholt sich für die Grundseite des Dreiecks in den Zeilen 15 und 16. Nach der Übernahme der Benutzereingaben übergibt die Funktion `main()` die Programmausführung an die Funktion `Area(base, height)` zusammen mit den Parameterwerten für `base` und `height`. Wenn `main()` die Ausführung an die Funktion `Area(base, height)` übergibt, erwartet sie einen Wert vom Typ `double` als Rückgabe

der Funktion. Die Berechnung der Dreiecksfläche findet in Zeile 27 statt:

```
area = (0.5*base*height);
```



Es ist zu unterscheiden zwischen **Area** als dem Namen der Funktion und **area** als Variablenname. Da C++ die Groß-/Kleinschreibung beachtet, handelt es sich hier um zwei vollkommen verschiedene Namen.

Die Anweisung verwendet die Standardoperatoren für Zuweisung (=) und Multiplikation (*). Der Zuweisungsoperator weist das Ergebnis aus $(0.5 * \text{base} * \text{height})$ an die Variable **area** zu. Der Multiplikationsoperator (*) berechnet das Ergebnis aus den Werten $(0.5 * \text{base} * \text{height})$. Für den Zuweisungsoperator gilt eine Auswertungsreihenfolge von rechts nach links. Demzufolge wird die Multiplikation vor der Zuweisung der Werte an **area** ausgeführt. Zu den fünf grundlegenden mathematischen Operatoren gehören Addition (+), Subtraktion (-), Multiplikation (*), Division (/) und Modulo-Division (%).

In Zeile 28 gibt die Funktion **Area** den Wert der Variablen **area** an die Funktion **main()** zurück. Das Programm setzt nun die Ausführung in der Funktion **main()** ab Zeile 18 nach dem Funktionsaufruf von **Area** fort. Der Rest des Programms zeigt das Ergebnis von **area** auf dem Bildschirm an.

Die if-Anweisung, Operatoren und Polymorphismus

In größeren und komplexeren Programmen ist es häufig erforderlich, den Programmablauf in Abhängigkeit von einem bestimmten Wert - beispielsweise von einer Benutzereingabe - fortzusetzen. Das lässt sich mit einer **if**-Anweisung (if - wenn) erreichen. Das nächste Beispiel demonstriert den Einsatz einer **if**-Anweisung. Das Format der Anweisung sieht folgendermaßen aus:

```
if (dieser_Ausdruck)
    tue dies;
```

Ein weiteres Format der **if**-Anweisung lautet:

```
if (dieser_Ausdruck)
    tue dies;
else
    tue das;
```

Da man in **if**-Anweisungen häufig relationale Operatoren verwendet, sehen wir uns zunächst diese Operatoren im Überblick an. Mit relationalen Operatoren bestimmt man, ob zwei Ausdrücke oder Zahlen gleich sind. Sind sie nicht gleich, liefert die Anweisung entweder **0** oder **false**. Tabelle 22.2 listet die sechs relationalen Operatoren von C++ auf.

Tabelle A.2: Relationale Operatoren

Operator	Name
==	Vergleich
!=	Nicht gleich
>	Größer als
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich

C++ verfügt außerdem über logische Operatoren. Der Vorteil der logischen Operatoren liegt darin, daß sie zwei einzelne Ausdrücke vergleichen und daraus schließen können, ob sie wahr (true) oder falsch (false) sind. Tabelle 22.3 führt die drei logischen Operatoren auf.

Tabelle A.3:
Logische
Operatoren

Symbol	Operator
&&	AND
	OR
!	NOT

Ein wichtiges und leistungsfähiges Merkmal von C++ ist das Überladen von Funktionen, der sogenannte **Polymorphismus**. Dabei handelt es sich um die Möglichkeit, mehrere Funktionen mit dem gleichen Namen zu definieren, die sich aber in ihren Parameterlisten unterscheiden. Das nächste Beispiel erweitert den vorherigen Code zur Dreiecksberechnung. Zunächst soll der Benutzer entscheiden, ob er die Fläche eines Dreiecks oder eines Kreises berechnen will. In Abhängigkeit von seiner Antwort - 1 für das Dreieck und 2 für den Kreis - ruft das Programm die erforderlichen Eingaben zur Berechnung der Fläche ab. In Listing A.3 wird die Funktion `Area` überladen. Unter demselben Funktionsnamen verbirgt sich sowohl die Berechnung des Dreiecks als auch des Kreises. Die Funktionen unterscheiden sich lediglich in ihren Parameterlisten.

Listing A.3: Das Programm Overload.cpp

```

1: // Arbeitsbereich: Overload
2: // Programmname: Overload.cpp
3:
4: # include <iostream.h>
5:
6: double base,height,radius; // Globale Variablen
7: double Area_of_triangle,Area_of_circle; // Globale Variablen
8: int choice; // Globale Variable
9:
10: double Area (double,double); // Funktionsprototyp
11: double Area (double); // Funktionsprototyp
12:
13: const double pi = 3.14; // Konstante Variable
14:
15: void main() // Hauptfunktion
16:
17: {
18: cout << "Fuer Dreiecksflaeche 1 eingeben\n";
19: cout << "Fuer Kreisflaeche 2 eingeben\n";
20: cin >> choice;
21:
22: if (choice == 1)
23:
24: {

```



```

25: cout << "Grundseite des Dreiecks eingeben: ";
26: cin >> base;
27: cout << "Hoehe des Dreiecks eingeben: ";
28: cin >> height;
29:
30: Area_of_triangle = Area(base,height);
31:
32: cout << "Die Dreiecksflaeche betraegt: "<<Area_of_triangle<<endl;
33: }
34:
35: if (choice == 2)
36:
37: {
38: cout << "Radius des Kreises eingeben: ";
39: cin >> radius;
40: Area_of_circle = Area(radius);
41: cout << "Die Kreisflaeche betraegt: "<<Area_of_circle<<endl;
42: }
43:
44: if (choice != 1 && choice != 2)
45:
46: {
47: cout << "Sorry! Sie muessen entweder 1 oder 2 eingeben.\n";
48: }
49: }
50:
51: double Area (double base, double height)
52: {
53: return (0.5*base*height)
54: }
55:
56: double Area(double radius)
57: {
58: return (pi*radius*radius);
59: }

```

Globale und lokale Variablen

In den bisher gezeigten Beispielen wurden die Variablen am Anfang des Programms deklariert, noch vor der Definition der Funktion `main()`. Diese Vorgehensweise entspricht eher einem C- als einem C++-Programm. Die Variablen sind in diesem Fall global und allen Funktionen zugänglich. Allerdings kann man auch lokale Variablen deklarieren. Der Gültigkeitsbereich lokaler Variablen beschränkt sich auf die Funktion, in der sie deklariert sind. Lokale Variablen können den gleichen Namen wie lokale Variablen haben, ändern aber nicht den Wert der globalen Variablen. Lokale Variablen beziehen sich nur auf die Funktion, in der sie definiert sind. Dieser Unterschied kann verwirrend sein und zu rätselhaften Ergebnissen führen.

Das Programm in Listing A.4 zeigt klar den Unterschied zwischen globalen und lokalen Variablen. Die Flächenberechnung für einen Kreis führen Sie sowohl mit globalen als auch lokalen Variablen aus.

Listing A.4: Das Programm Global.cpp

```

1: // Arbeitsbereich: Variable

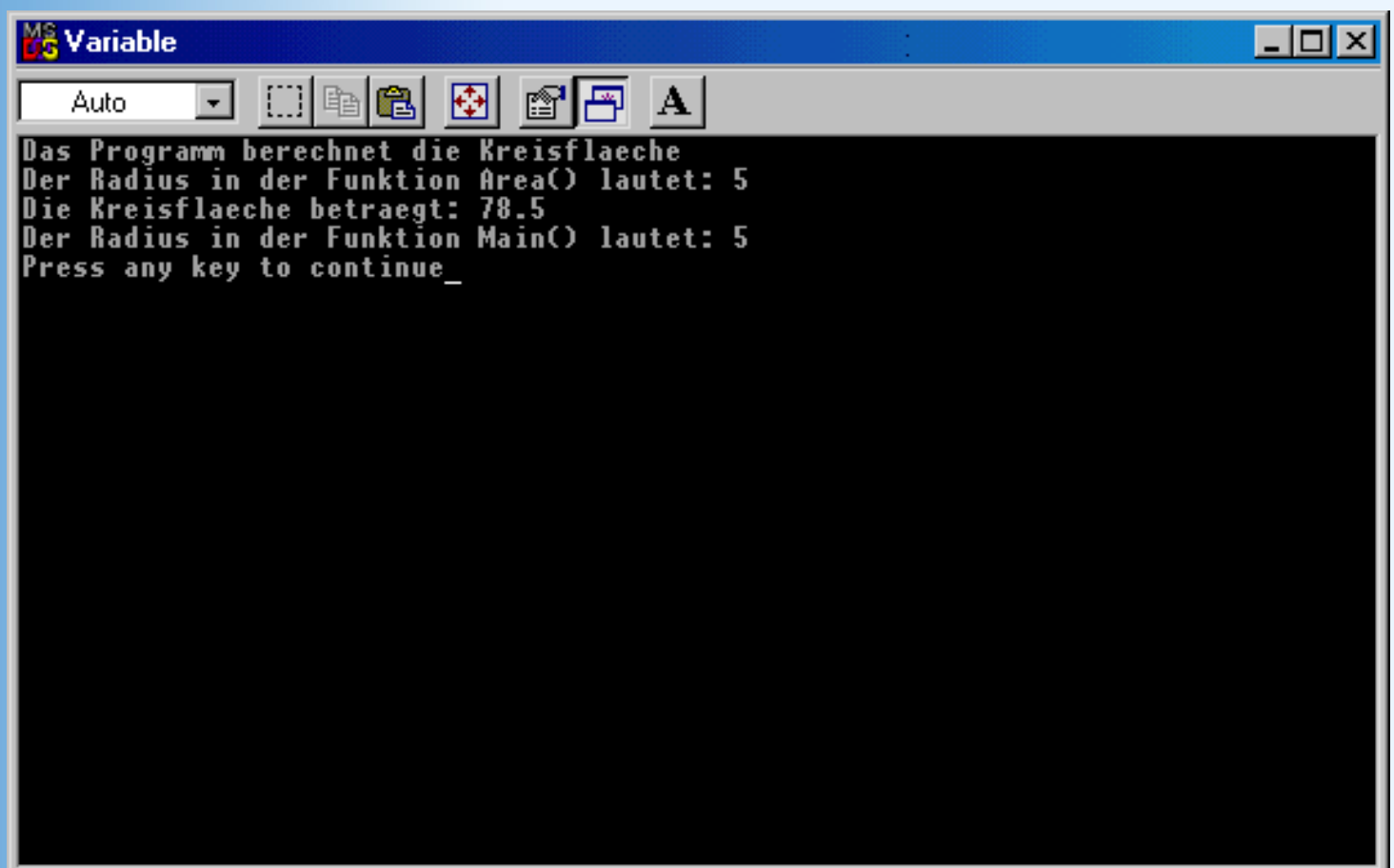
```

```

2: // Programmname: Global.cpp
3:
4: #include <iostream.h>
5:
6: double area;
7: double Area (double);
8: const double pi = 3.14;
9: double radius = 5;
10:
11: int main()
12: {
13: {
14: cout<<"Das Programm berechnet die Kreisflaeche\n";
15: area = Area (radius);
16: cout << "Die Kreisflaeche betraegt: "<<area<<endl;
17: cout << "Der Radius in der Funktion Main() lautet: "<<radius<<endl;
18: return 0;
19: }
20:
21: double Area (double radius)
22: {
23: area = (pi*radius*radius);
24: cout<<"Der Radius in der Funktion Area() lautet: "<<radius<<endl;
25: return area;
26: }

```

Die Variable `radius` ist sowohl in der Funktion `main()` als auch in der Funktion `Area()` zugänglich und in beiden Funktionen identisch. Das Ergebnis des ausgeführten Programms zeigt Abbildung A.6.



```

MS Variable
Auto
Das Programm berechnet die Kreisflaeche
Der Radius in der Funktion Area() lautet: 5
Die Kreisflaeche betraegt: 78.5
Der Radius in der Funktion Main() lautet: 5
Press any key to continue_

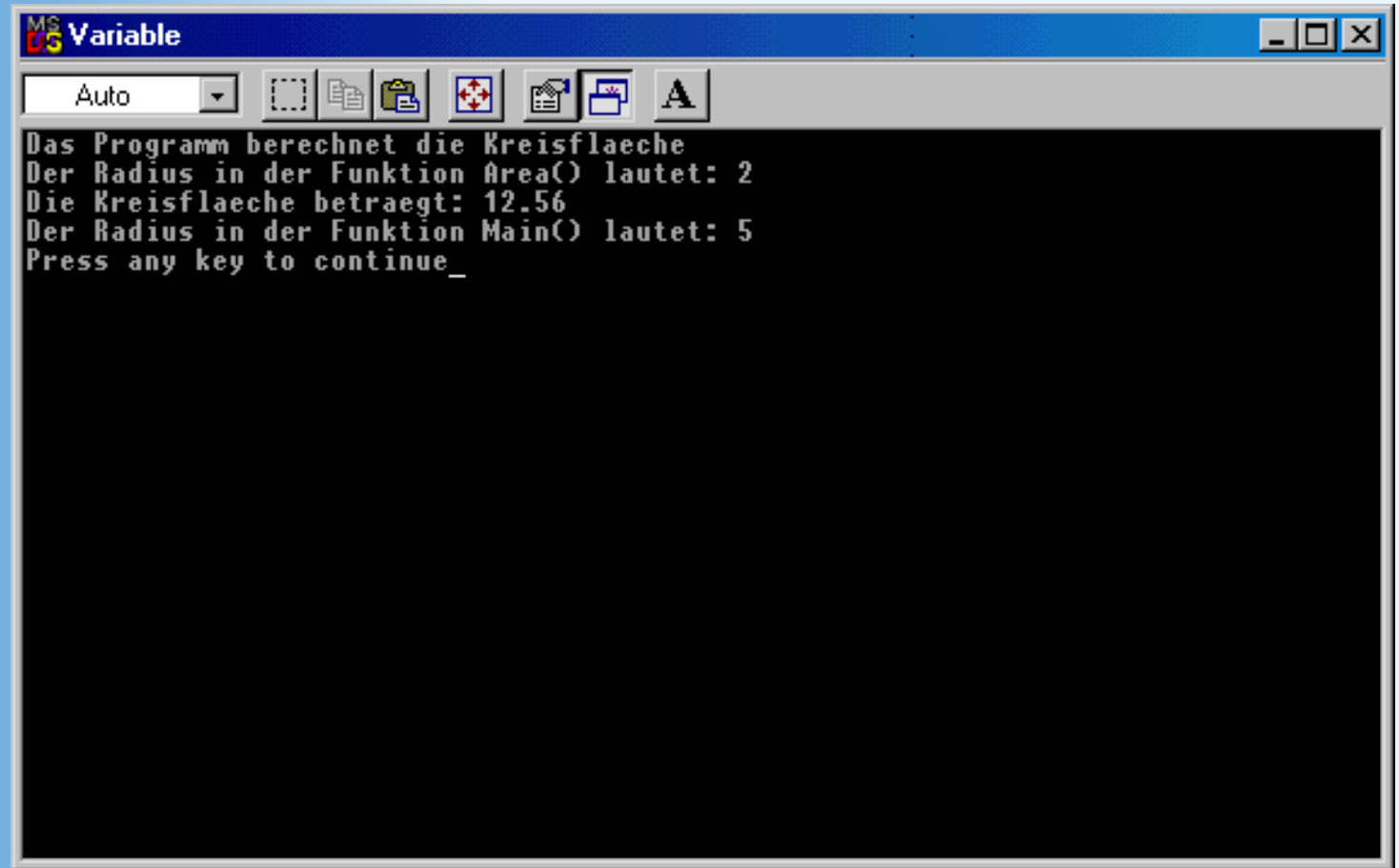
```

Abbildung A.6:**Das Programm Global.cpp - mit einer globalen Variablen**

Bei der Ausführung zeigt das Programm den Wert der Variablen `radius` in verschiedenen Funktionen an. Jetzt ändern Sie die globale Variable in eine lokale Variable. Fügen Sie in die Funktion `Area` eine zusätzliche Zeile ein, um eine lokale Variable zu definieren:

```
radius = 2;
```

Kompilieren und starten Sie das Programm. Die Ergebnisse sind in Abbildung A.7 dargestellt.

**Abbildung A.7:****Das modifizierte Programm Global.cpp - mit globalen und lokalen Variablen**

Der Wert der Variablen `radius` bleibt in der Funktion `main()` unverändert, während er sich lokal in der Funktion `Area()` ändert. Die Kreisfläche wird mit dem Wert der lokalen Variablen berechnet, während gleichzeitig der Wert der globalen Variablen nicht geändert wird und vor der Funktion `Area()` verborgen bleibt.



Es empfiehlt sich immer, globale und lokale Variablen im Quelltext zu unterscheiden. Zum Beispiel können Sie vor den Variablennamen das Präfix `g` für global und `l` für lokal schreiben.

Zeiger

Zeiger gehören zu den wichtigsten Einrichtungen von C++. Allerdings stiften sie bei Einsteigern in die C++-Programmierung oftmals Verwirrung. Zeiger ermöglichen den direkten Zugriff auf die Originaldaten, was die Effizienz eines Programms erhöht. Hauptsächlich verwendet man Zeiger mit zwei Operatoren, dem Indirektionsoperator (`*`) und dem Adreßoperator (`&`). Es ist üblich, vor den Namen einer Zeigervariablen

ein `p` (für pointer - Zeiger) zu setzen, um ihn von anderen Variablennamen zu unterscheiden. Prinzipiell stellt ein Zeiger eine Variable dar, er enthält aber nicht den Wert der Variablen selbst, sondern eine Adresse auf eine Speicherstelle. Um einen Zeiger zu deklarieren, schreibt man ein Sternchen (*) vor den Zeigernamen. Auf die Adresse der Variablen greift man zu, indem man den Operator & vor den Variablennamen setzt.

Zum Verständnis der Zeiger muß man wissen, wie Variablen gespeichert werden. In Tabelle 22.1 weiter vorn in diesem Anhang haben Sie verschiedene Datentypen von Variablen kennengelernt. Tabelle 22.4 zeigt die von den Variablentypen belegte Größe im Arbeitsspeicher.

Tabelle A.4: Größe der Variablentypen

Variablentyp	Größe in Byte
unsigned short int	2 Byte
short int	2 Byte
unsigned long int	4 Byte
long int	4 Byte
int	4 Byte (32 Bit)
insigned int	2 Byte (32 Bit)
char	1 Byte
float	4 Byte
double	8 Byte

Im Programm `Address.cpp` in Listing A.5 belegen die Variablen `base` und `radius` 8 bzw. 4 Byte. Nehmen wir an, daß im Speicher Ihres Computers ein bestimmter Platz für diese Variablen zur Verfügung steht. Diese Speicherzellen sind fortlaufend von 1 bis 12 nummeriert. Jeder Schritt ist 1 Byte groß. Wenn Sie die Variable `base` vom Typ `double` deklarieren, belegt sie 8 Byte. Nehmen wir an, daß sich diese 8 Byte an den Speicherstellen von 1 bis 8 befinden. Weiterhin ist eine Variable `radius` vom Typ `int` deklariert, die 4 Byte einnimmt und die Plätze 9 bis 12 belegen soll. Die Position dieser Variablen bezeichnet man als Adresse. Demzufolge hat die Variable `base` eine Anfangsadresse von 1 und eine Endadresse von 8. Analog dazu hat die Variable `radius` die Anfangsadresse 9 und die Endadresse 12. Wenn Sie den Adreßoperator (&) auf eine Variable anwenden, liefert er die Adresse der Variablen zurück. Die Variable `base` belegt einen Adreßbereich von 1 bis 8, der Adreßoperator liefert aber nur die Adresse 1 zurück. Intern weiß das System bereits, daß zu dieser Adresse 8 aufeinanderfolgende Speicherstellen gehören, da Sie den Typ der Variablen als `double` definiert haben.



Die in Tabelle A.4 angegebenen Größen liegen nicht genau fest und hängen vom jeweiligen Compiler und der verwendeten Hardware ab. Die Größe der Variablen für Ihren Compiler und Ihre Hardware können Sie mit der Funktion `sizeof()` bestimmen, wie es in Listing A.5 in den Zeilen 13 und 16 implementiert ist.

Das Programm in Listing A.5 zeigt, wie man auf die Speicheradresse von Variablen zugreift.

Listing A.5: Das Programm `Address.cpp`

```
1: // Arbeitsbereich: Zeiger
2: // Programmname: Address.cpp
```

```

3:
4: #include <iostream.h>
5:
6: double base = 5.0;
7: int radius = 2;
8:
9: void main()
10: {
11: cout<<"Der Wert WERT von base lautet: "<<base<<endl;
12: cout<<"Die ADRESSE von base lautet: "<<&base<<endl;
13: cout<<"Die GROESSE von double base ist: "<<sizeof(double)<< "
Byte\n";
14: cout<<"Der WERT von radius lautet: "<<radius<<endl;
15: cout<<"Die ADRESSE von radius lautet: "<<&radius<<endl;
16: cout<<"Die GROESSE von integer radius ist: "<<sizeof(int)<< "
Byte\n";
17: }

```

Die Zeilen 12 und 15 greifen mit dem Adreßoperator (&) direkt auf die Adresse der Variablen zu. Abbildung A.8 zeigt die Adressen der Variablen `base` und `radius`. Die konkreten Adressen der Variablen hängen von Ihrem System und den laufenden Programmen ab, so daß sie nicht mit der Abbildung übereinstimmen brauchen.

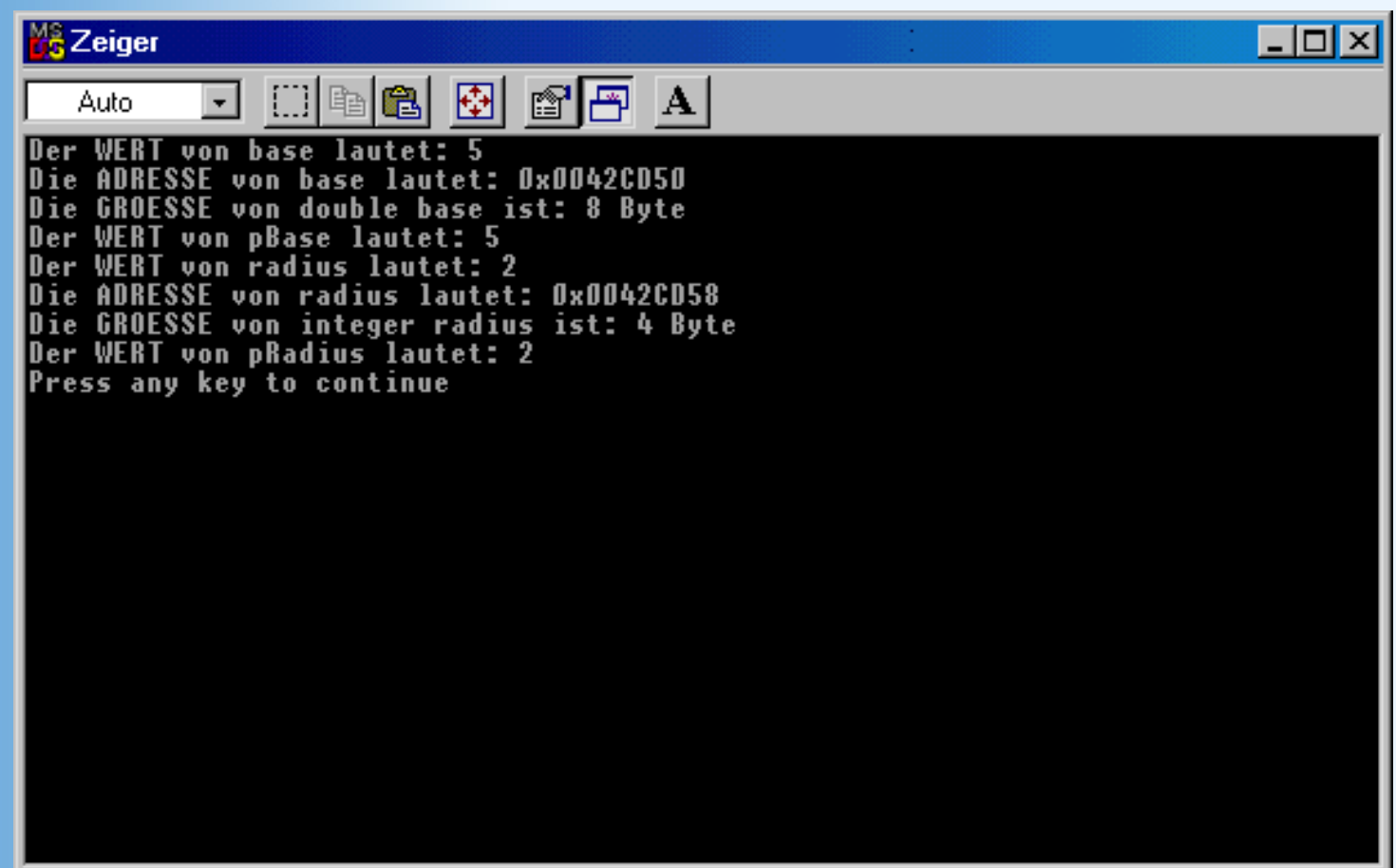


Abbildung A.8:
Der Adreßoperator im praktischen Einsatz

Der Indirektionsoperator (*) liefert den Zugriff auf den Wert, der an der Adresse der Variablen gespeichert ist. Wenn ein Zeiger für einen bestimmten Variablentyp (beispielsweise `int`) deklariert ist, sollte man den Zeiger nicht für einen anderen Typ einsetzen, solange man keine Typumwandlung auf den anderen Typ

vornimmt. Erinnern wir uns: Ein Zeiger ist eine Variable, und wie jede andere Variable muß man sie deklarieren und initialisieren. Ein nicht initialisierter Zeiger kann gefährlich sein. Listing A.5 wurde modifiziert, um auf die Werte der Variablen `radius` und `base` zuzugreifen. Das modifizierte Programm ist in Listing A.6 wiedergegeben.

Listing A.6: Das modifizierte Programm `Address.cpp`

```

1: // Arbeitsbereich: Zeiger
2: // Programmname: Address.cpp
3:
4: #include <iostream.h>
5:
6: double base =5.0;
7: int radius =2;
8:
9: double *pBase =0; // Zeigervariable initialisieren
10: int *pRadius =0; // Zeigervariable initialisieren
11:
12: void main()
13: {
14: pBase = &base; // Adresse von base zuweisen
15: pRadius = &radius; // Adresse von radius zuweisen
16: cout<<"Der WERT von base lautet: "<<base<<endl; // Wert von base
ausgeben
17: cout<<"Die ADRESSE von base lautet: "<<&base<<endl; // Adresse von
base ausgeben
18: cout<<"Die GROESSE von double base ist: "<<sizeof(double)<< " Byte\n
19: cout<<"Der WERT von pBase lautet: "<<*pBase<<endl; // Umgeleiteten
Wert von base ausgeben
20:
21: cout<<"Der WERT von radius lautet: "<<radius<<endl; // Wert von
radius ausgeben
22: cout<<"Die ADRESSE von radius lautet: "<<&radius<<endl; // Adresse
von radius ausgeben
23: cout<<"Die GROESSE von integer radius ist: "<<sizeof(int)<<"
Byte\n";
24: cout<<"Der WERT von pRadius lautet: "<<*pRadius<<endl; //
Umgeleiteten Wert von radius ausgeben
25:
26: }
```

Referenzen

Ein weiteres wichtiges Merkmal von C++ sind Referenzen, die man oft in Verbindung mit Funktionsparametern einsetzt. Eine Referenz ist einfach ein Synonym für eine Variable. Bis jetzt haben Sie Parameter an Funktionen als Wert übergeben. Jetzt geht es darum, wie man Parameter als Referenz übergibt. Um eine Referenzvariable zu erzeugen, legt man deren Typ fest und setzt vor den Namen den Referenzoperator (&). Für eine Variable `float radius` erzeugen Sie eine Referenz mit

```
void Funktionsname (float &rfradius);
```

Die Referenzvariable kann einen beliebigen Namen erhalten. Im folgenden Beispiel steht vor dem Namen der Referenzvariablen das Präfix `rf`. Der Vorteil einer Referenz liegt darin, daß man sie wie jede andere

Variable als Parameter übergeben kann. Im Gegensatz zu normalen Parametern wirken sich aber die in einer Funktion am Wert der Referenz vorgenommenen Änderungen auf die Originalvariable aus. Das Beispiel in Listing A.7 zeigt, wie die Referenz den Wert der Variablen in der Funktion `main()` ändert.

Listing A.7: Das Programm Refer.cpp

```

1: // Arbeitsbereich: Referenz
2: // Programmname: Refer.cpp
3:
4: #include <iostream.h>
5:
6: void squareit (float &num);
7: int main()
8:
9: {
10: float num=5.0;
11:
12: cout<<"In Main: vor dem Quadrieren: "<<num*num<<"\n";
13:
14: squareit (num);
15: cout<<"In Main: nach dem Quadrieren: "<<num*num<<"\n";
16: return 0;
17:
18: }
19:
20: void squareit (float &rnum)
21: {
22:
23: cout<<"In Squareit: vor dem Quadrieren: "<<rnum*rnum<<"\n";
24:
25: rnum = rnum+5;
26:
27: cout<<"In Squareit: nach dem Quadrieren: "<<rnum*rnum<<"\n";
28:
29: }

```

In Zeile 6 steht der Prototyp der Funktion `squareit`, die den übergebenen Wert quadriert. Der Parameter ist eine Referenz. Zeile 10 setzt den Wert der Variablen `num` auf 5. Das Quadrat der Zahl zeigt der Code in Zeile 15 auf dem Bildschirm an. In Zeile 14 wird die Funktion `squareit` aufgerufen.



Sie übergeben hier die Variable `num` und nicht deren Adresse.

Nur wenn die Programmausführung von Zeile 14 zu Zeile 20 springt, sind die Variablen als Referenzen gekennzeichnet. Zeile 27 quadriert die Referenzen und zeigt das Ergebnis an. Die Referenzen sollten mit den Variablen übereinstimmen, da es sich lediglich um Aliase für die Variablen handelt. Zeile 25 addiert 5 zur Referenz, die wiederum die Variable `num` verändert. Der höhere Wert wird quadriert und auf dem Bildschirm angezeigt. Die Ausführung kehrt in die Funktion `main()` zurück. Zeile 15 zeigt zur Kontrolle an, daß sich die Variable geändert hat. Die Ausgabe des Programms ist in Abbildung A.9 zu sehen.

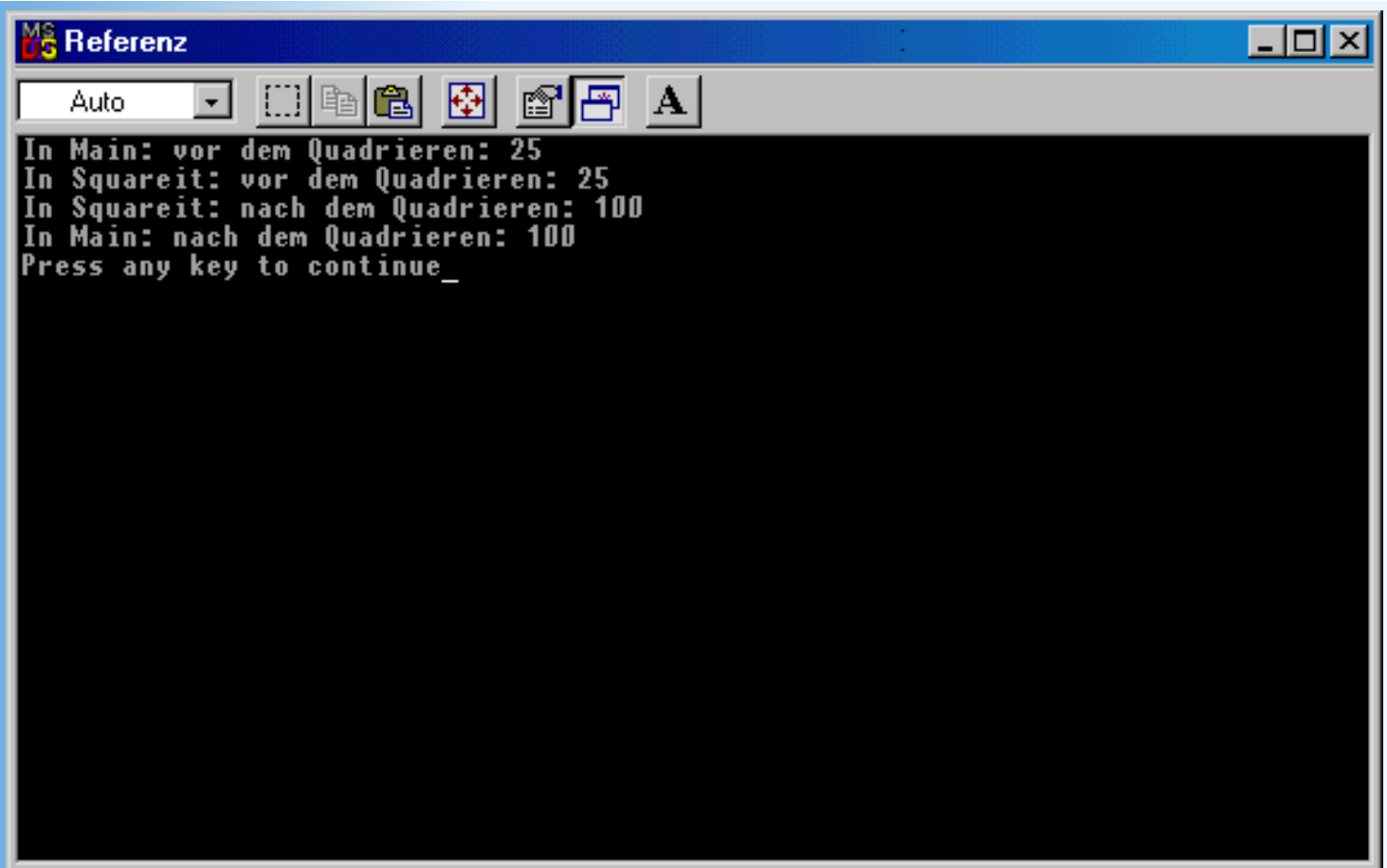


Abbildung A.9:
Parameter als Referenz übergeben

Klassen

In den vorherigen Abschnitten haben Sie Datentypen (`int`, `float` usw.) verwendet, die fester Bestandteil von C++ sind. In großen komplexen Programmen ist es einfacher, eigene Typen zu definieren, die man aus den vordefinierten Typen zusammensetzen kann. Hauptsächlich zu diesem Zweck wurden in C++ Klassen aufgenommen - um dem Programmierer zu ermöglichen, benutzerdefinierte Datentypen und Methoden zu definieren. Das Konzept der Klassen in C++ entwickelte sich aufgrund bestimmter Beschränkungen im Konzept der Strukturen von C. Ein tiefergehendes Verständnis der Klassen erfordert es, daß wir zuerst die Strukturen von C untersuchen.

Eine Struktur stellt in C/C++ ein Mittel dar, eigene benutzerdefinierte Daten darzustellen. Bei der Definition von Variablen definieren Sie zuerst deren Datentypen und schließen die Variablennamen an:

```
int radius;
```

Zur Definition eigener Datentypen verwendet man das Schlüsselwort `struct`. Die Syntax für die Deklaration einer Struktur lautet:

```
struct [StrukturName]
{
    Datenelemente
}
```

Die *Datenelemente* einer Struktur sind Variablen und Funktionen. Wenn von Funktionen im Zusammenhang mit Klassen die Rede ist, spricht man besser von Methoden. Von jetzt an verwenden wir den Begriff Funktion für Programmcode, der nicht Teil einer Struktur oder Klasse ist. Ein Bezug auf Methoden weist darauf hin, daß die Funktion mit einer Klassenstruktur gekoppelt ist. Um zu verstehen, wie

man Strukturen einsetzt, sehen wir uns das Beispiel in Listing A.8 an:

Listing A.8: Das Programm Struct.cpp

```

1: // Arbeitsbereich: Klasse1
2: // Programmname: Struct.cpp
3: #include <iostream.h>
4:
5: struct farm_house
6: {
7: int pig_values;
8: };
9:
10: int main()
11: {
12: farm_house pig1, pig2, pig3;
13:
14: pig1.pig_values = 12;
15: pig2.pig_values = 13;
16: pig3.pig_values = 14;
17:
18: cout << "Der Wert von pig1 lautet: " << pig1.pig_values<< "\n";
19: cout << "Der Wert von pig2 lautet: " << pig2.pig_values << "\n";
20: cout << "Der Wert von pig3 lautet: " << pig3.pig_values << "\n";
21:
22: return 0;
23: }

```

In Zeile 5 folgt dem Schlüsselwort `struct` der Name der Struktur. Die eigentliche Definition der Struktur ist in die geschweiften Klammern eingeschlossen. Die hier gezeigte Struktur definiert ein Datenelement vom Typ `int` mit dem Namen `pig_values`. Wie bereits erwähnt, definieren Sie mit einer Struktur grundsätzlich einen benutzerdefinierten Datentyp. Alle Datentypen enden mit einem Semikolon, so daß die Struktur ebenfalls mit einem Semikolon abzuschließen ist. Zeile 12 definiert drei Instanzen des gleichen Typs von `farm_house`, von denen jede eine einzelne `int`-Variable enthält.



Wenn Sie sich streng an die Syntax von C halten, sind die Instanzen in Zeile 12 mit Hilfe des Schlüsselworts `struct` zu definieren:

```
struct farm_house pig1, pig2, pig3;
```

In C++ ist das nicht mehr erforderlich.

Die Zeilen 14 bis 16 weisen Werte an die Elementvariablen jeder Struktur zu. Der auch als Punktoperator bezeichnete Strukturelementoperator (`.`) dient dem Zugriff auf die Elementvariablen der Struktur. Die Zeilen 18 bis 20 geben die zugewiesenen Werte auf dem Bildschirm aus. Abbildung A.10 zeigt die Ausgabe dieses Programms.

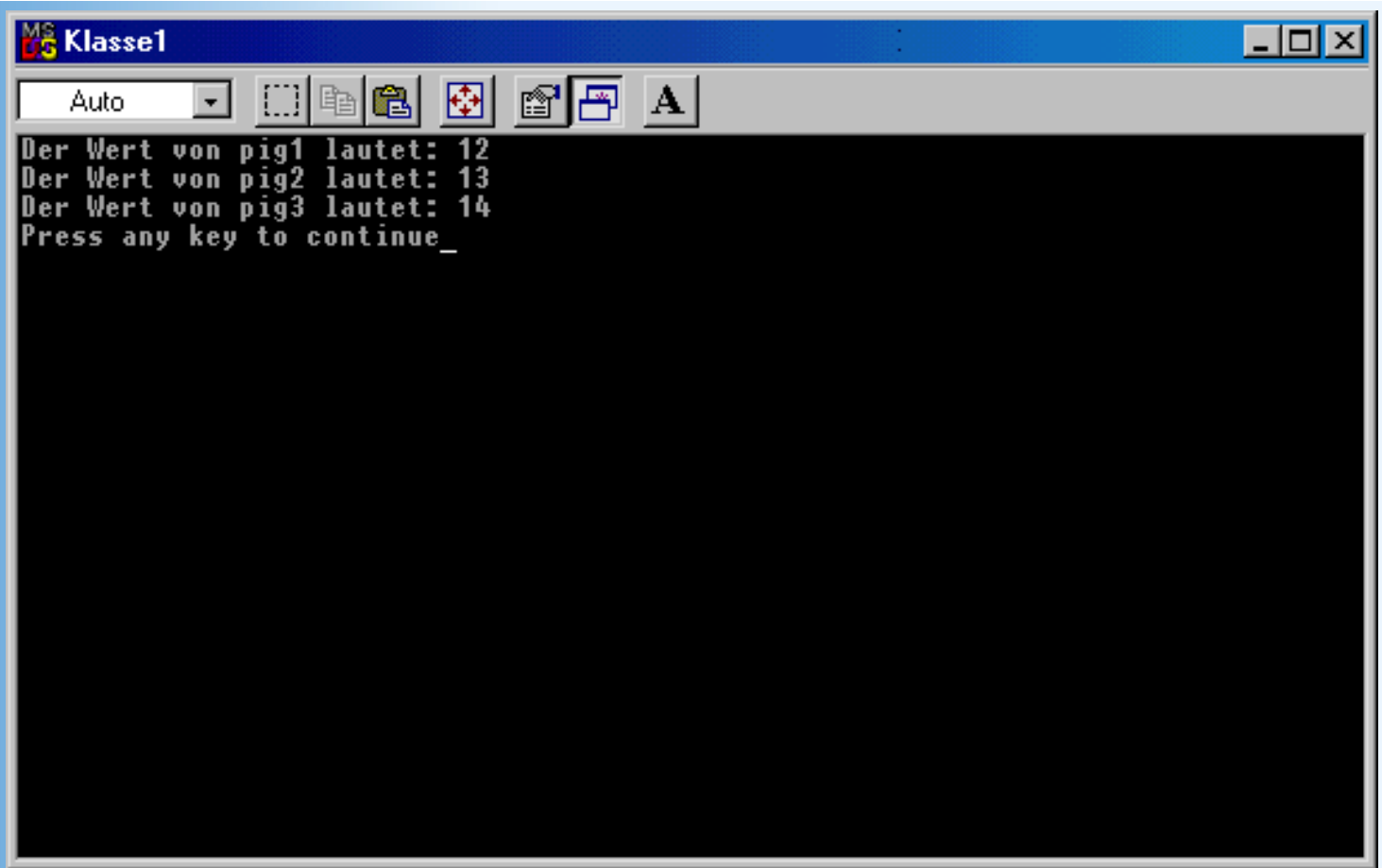


Abbildung A.10:
Ausgabe des Strukturprogramms

Das wichtigste Konzept der objektorientierten Programmierung ist die Kapselung, die sich auf eine oder mehrere Klassen erstrecken kann. Kapselung fördert Schutzmaßnahmen und das Verbergen von Daten. Das Programm `struct.cpp` wies keine Kapselung oder Klassen auf. Was bedeuten nun Kapselung und Klassen in der objektorientierten Programmierung?

Beschreiben wir zunächst die Syntax und die Komponenten einer Klasse:

```
class Klassenname
{
public:
Klassenname_Konstruktor;
~Klassenname_Destruktor;
Klassenmethode_Prototypen();
Elementvariablen_der_Klasse;
private:
Klassenmethode_Prototypen();
Elementvariablen_der_Klasse;
};
```

Die fettgedruckten Wörter sind Schlüsselwörter. Eine Klasse deklariert man mit dem Schlüsselwort `class`. Daran schließt sich der Name der Klasse an. Die Daten und Methoden einer Klasse sind in geschweifte Klammern (`{ }`) eingeschlossen. Die Methoden einer Klasse stellen Prototypen von Funktionen dar und bestimmen das Verhalten der Objekte einer Klasse. Die Elementvariablen - oder Member-Variablen - sind die Variablen der Klasse. Weiterhin verfügen Klassen über Konstruktoren und Destruktoren. Die Methoden und Variablen lassen sich entweder als `public` (öffentlich) oder `private` (privat) klassifizieren.

Das vorherige Beispiel des Programms `Struct.cpp` in Listing A.8 erstellen Sie nun neu und nutzen dazu die Methodik der Klasse und der Kapselung. Die Ausgabe des Programms in Listing A.9 ist identisch mit dem vorherigen Beispiel `Struct.cpp`.

Listing A.9: Das Programm `Clasfarm.cpp`

```

1: // Arbeitsbereich: Klasse2
2: // Programmname: Clasfarm.cpp
3: #include <iostream.h>
4:
5: class farm_house
6: {
7:     int pig_values;
8: public:
9:     void set(int input);
10:    int get(void);
11: };
12:
13: void farm_house::set(int input)
14: {
15:     pig_values = input;
16: }
17:
18: int farm_house::get(void)
19: {
20:     return pig_values;
21: }
22:
23: int main()
24: {
25:     farm_house pig1, pig2, pig3;
26:
27:
28:     pig1.set(12);
29:     pig2.set(13);
30:     pig3.set(14);
31:
32:     cout << "Der Wert von pig1 lautet: " << pig1.get() << "\n";
33:     cout << "Der Wert von pig2 lautet: " << pig2.get() << "\n";
34:     cout << "Der Wert von pig3 lautet: " << pig3.get() << "\n";
35:
36:     return 0;
37:
38: }
```

Vergleichen Sie die Strukturdeklaration des Programms `Struct.cpp` in Listing A.8 (Zeilen 5 bis 7) mit der Klassendeklaration des Programms `Clasfarm.cpp` in Listing A.9 (Zeilen 5 bis 11). Der Unterschied liegt in den privaten und öffentlichen Teilen der jeweiligen Deklarationen. In der `struct`-Deklaration ist alle öffentlich, während die Klassendeklaration mit einem privaten Abschnitt beginnt. Alle Daten und Methoden am Beginn einer Klasse sind privat. Das bedeutet, daß die Elementvariable

`int pig_values;`

privat ist und den Methoden außerhalb der Klasse verborgen bleibt. Die Variable `pig_values` ist

demnach nicht in der Funktion `main()` zugänglich. Mit anderen Worten ist die Elementvariable versteckt. Den Methoden ihrer Klasse ist sie zugänglich, und zwar

```
void set (int input);
int get(void);
```

Diese Methoden sind als öffentlich definiert. Aus diesem Grund lassen sich die Methoden durch beliebige Objekte der Klasse ansprechen. Zeile 25 definiert `pig1`, `pig2` und `pig3` als Instanzen oder Objekte der Klasse. Sicherlich fragen Sie sich jetzt, warum `pig1` ein Objekt ist.

Zeile 5 definiert eine Klasse `farm_house`. Wie bereits erwähnt, deklarieren Sie mit einer Klasse eigentlich nur einen neuen Typ. Wenn Sie eine Variable deklarieren, geben Sie zuerst ihren Typ und dann den Variablennamen an, wie es folgendes Beispiel zeigt:

```
long EineVariable, EineAndere, NochEine;
```

Bei einem Objekt einer Klasse definieren Sie analog dazu den Typ, der in diesem Fall `farm_house` lautet, und den Objektnamen, hier `pig1`:

```
farm_house pig1, pig2, pig3;
```

In Zeile 28 setzen Sie den Wert von `pig1` auf 12. Dabei kommt der Punktoperator (`.`) zum Einsatz. Das Objekt `pig1` hat Zugriff auf die Methode `set()`. Die Methode `set()` ist eine Methode der Klasse `farm_house`, so daß sie Zugriff auf deren private Daten hat. Die Implementierung der Methode `set()` zeigt Zeile 13. Damit das Programm weiß, daß die Methode `set()` innerhalb des Gültigkeitsbereichs der Klasse `farm_house` liegt, verwendet man den Zugriffsoperator (`::`), der verschiedentlich auch als Bereichs- Auflösungsoperator firmiert. In Zeile 15 wird die Variable `input` auf die Variable `pig_values` gesetzt.

In der Klasse `farm_house` sind zwei öffentliche Methoden deklariert. Die andere Methode, `get()`, ist in Zeile 18 implementiert. Die Methode `get()` übernimmt keine Parameter, gibt aber `pig_values` zurück, da sie ebenfalls innerhalb des Gültigkeitsbereichs der Klasse `farm_house` liegt.

In den Zeilen 32 bis 34 wird die Methode `get()` erneut aufgerufen, und zwar durch die Objekte `pig1`, `pig2` und `pig3`, um die Werte von `pig_values` auf dem Bildschirm auszugeben.

Wenn Sie die beiden Programme `Struct.cpp` und `Clasfarm.cpp` vergleichen, fällt auf, daß ein Programm nur 23 Zeilen umfaßt, während das andere 38 Zeilen lang ist. Der Code ist durch die Implementierung von Klassen umfangreicher geworden! Das stimmt zwar. Allerdings treten die Vorteile aus der Verwendung von Klassen erst in komplexeren und größeren Programmen zutage. Da sie außerdem sensible Daten vor dem Benutzer verbergen können, sind Klassen sicherer und weniger fehleranfällig. Der Compiler kann bereits Fehler erkennen, bevor sie sich unangenehm im laufenden Programm bemerkbar machen.

Konstruktoren und Destruktoren

Weiter oben haben Sie bereits die Syntaxdefinition einer Klasse kennengelernt. In diesem Zusammenhang war auch von Konstruktoren und Destruktoren die Rede. Im Beispiel `Clasfarm.cpp` finden sich jedoch weder Konstruktoren noch Destruktoren. Wenn ein Konstruktor oder Destruktor nicht explizit definiert ist, erzeugt ihn der Compiler automatisch.

Die Konstruktorfunktion

Ein **Konstruktor** ist eine Funktion, die eine Klasse initialisiert. Der Aufruf des Konstruktors erfolgt, wenn man die Instanz einer Klasse erzeugt. Für einen Konstruktor gelten folgende Regeln:

- Der Konstruktor muß den gleichen Namen wie seine Klasse haben:


```
class farm_house
{
public:
farm_house(); // Konstruktor
.....
.....
}
```

- Ein Konstruktor läßt sich nicht mit einem Rückgabewert definieren.
- Ein Konstruktor ohne Argumente ist ein Standardkonstruktor.
- Der Konstruktor muß mit dem Schlüsselwort `public` deklariert sein.

Die Destruktorfunktion

Eine Destruktorfunktion stellt das Gegenteil zur Konstruktorfunktion dar. Der Aufruf des Destruktors erfolgt automatisch, wenn man den Block, indem das Objekt initialisiert wurde, verläßt. Ein Destruktor gibt das Objekt und demzufolge auch den zugewiesenen Speicher frei. Für einen Destruktor gelten die folgenden Regeln:

- Der Destruktor muß den gleichen Namen wie seine Klasse haben.
- Dem Destruktor ist das Zeichen `~` (Tilde) voranzustellen.
- Ein Destruktor hat weder Argumente noch einen Rückgabewert.
- Der Destruktor ist mit dem Schlüsselwort `public` zu deklarieren.

```
class farm_house
{
public:
farm_house (); // Konstruktor
~farm_house(); // Destruktor
.....
}
```

Friend-Funktionen und Friend-Klassen

Als privat deklarierte Methoden und Elemente sind nur dem Teil des Programms zugänglich, der zur Klasse gehört. Allerdings kann man eine Funktion außerhalb der Klasse oder eine andere Klasse als Friend-Klasse oder -Funktion definieren. Die Friend-Definition läßt sich auf die gesamte Klasse oder nur auf einzelne Funktionen anwenden. Bei der Deklaration von Friend-Funktionen sind folgende wichtige Regeln zu beachten:

- Friends sollte man möglichst sparsam einsetzen, da sie die Vorteile aufheben, die sich aus dem Verbergen von Daten ergeben.
- Deklariert man `x` als Friend von `y`, bedeutet das nicht unbedingt, daß `y` Zugriff auf die Methoden und Elemente von `x` hat.

Deklarationen und Definitionen von Klassen

Klassen verfügen über eigene private und öffentliche Elementvariablen und Methoden. Wie Sie im vorherigen Beispiel `Clasfarm.cpp` gesehen haben, ist das Programm umfangreicher als ein Programm ohne Klassen. Es gibt keine strengen Regeln, es haben sich aber bestimmte Standardtechniken durchgesetzt, die fast alle Programmierer einhalten. Zuerst einmal stellt man alle Klassendeklarationen in die Header-Dateien (mit der Erweiterung `.h` oder `.hpp`). Alle Klassendefinitionen bringt man in der Quellcodedatei (mit der Erweiterung `.cpp`) unter. Am Anfang der Quellcodedatei steht eine

Include-Anweisung für die Header-Datei. Beispielsweise läßt sich das Programm `Clasfarm` in die Dateien `Clasfarm.h` und `Clasfarm.cpp` aufspalten. Die Datei `Clasfarm.h` sieht dann wie in Listing A.10 aus.

Listing A.10: Die Header-Datei `Clasfarm.h`

```
1: // Arbeitsbereich: Klasse2
2: // Programmname: Clasfarm.h
3: #include <iostream.h>
4:
5: class farm_house
6: {
7:     int pig_values;
8: public:
9:     void set(int input);
10:    int get(void);
11: };
```

Die Quelldatei `Clasfarm.cpp` ist in Listing A.11 wiedergegeben.

Listing A.11: Die Quelldatei `Clasfarm.cpp`

```
1: #include "clasfarm.h"
2: void farm_house::set(int input)
3: {
4:     pig_values = input;
5: }
6:
7: int farm_house::get(void)
8: {
9:     return pig_values;
10: }
11:
12: int main()
13: {
14:     farm_house pig1, pig2, pig3;
15:
16:
17:     pig1.set(12);
18:     pig2.set(13);
19:     pig3.set(14);
20:
21:     cout << "Der Wert von pig1 lautet: " << pig1.get() << "\n";
22:     cout << "Der Wert von pig2 lautet: " << pig2.get() << "\n";
23:     cout << "Der Wert von pig3 lautet: " << pig3.get() << "\n";
24:
25:     return 0;
26:
27: };
```

Klassen innerhalb einer Klasse

Es ist durchaus zulässig, innerhalb einer Klasse eine andere Klassendeklaration vorzusehen. Man spricht in diesem Fall von verschachtelten Klassen. Das folgende Beispiel deklariert zwei Klassen, `Parzelle` und

`Steuer_bescheid`. Das Objekt `steuern` der Klasse `Steuer_bescheid` ist innerhalb der Klasse `Parzelle` definiert. Die Methode `main()` hat keine Objekte der Klasse `Steuer_bescheid`, so daß die Methoden oder Elemente der Klasse `Steuer_bescheid` nicht direkt aus der Funktion `main()` heraus zugänglich sind. Untersuchen wir das Programm in Listing A.12 etwas näher.

Listing A.12: Das Programm `Class3.cpp`

```

1: // Arbeitsbereich Name: Klasse3
2: // Programmname: Klasse3.cpp
3: #include <iostream.h>
4:
5: class Steuer_bescheid
6: {
7: int Grund_steuer;
8: int Verm_steuer;
9: public:
10: void set(int in_stadt, int in_verm)
11: {Grund_steuer = in_stadt; Verm_steuer = in_verm; }
12: int get_Verm_steuer(void) {return Verm_steuer;}
13: int get_Grund_steuer(void) {return Grund_steuer;}
14: };
15:
16:
17: class Parzelle {
18: int laenge;
19: int breite;
20: Steuer_bescheid steuern;
21: public:
22: void set(int l, int b, int s, int p) {
23: laenge = l;
24: breite = b;
25: steuern.set(s, p); }
26: int get_area(void) {return laenge * breite;}
27: int get_data(void) {return steuern.get_Verm_steuer() ;}
28: int get_data2(void) {return steuern.get_Grund_steuer() ;}
29: };
30:
31:
32: int main()
33: {
34: Parzelle klein, mittel, gross;
35:
36: klein.set(5, 5, 5, 25);
37: mittel.set(10, 10, 10, 50);
38: gross.set(20, 20, 15, 75);
39:
40:
41: cout << "Bei einer kleinen Parzelle von "<< klein.get_area ()<<
"\n";
42: cout << "betragen die Grundsteuern DM "<< klein.get_data2 () <<
"\n";
43: cout << "und die Vermoegenssteuern DM " << klein.get_data ()<< "\n";

```

```

44:
45: cout << "Bei einer mittleren Parzelle von "<< mittel.get_area ()<<
"\n";
46: cout << "betragen die Grundsteuern DM "<< mittel.get_data2 () <<
"\n";
47: cout << "und die Vermoegenssteuern DM " << mittel.get_data ()<<
"\n";
48:
49: cout << "Bei einer grossen Parzelle von "<< gross.get_area ()<<
"\n";
50: cout << "betragen die Grundsteuern DM "<< gross.get_data2 () <<
"\n";
51: cout << "und die Vermoegenssteuern DM " << gross.get_data ()<< "\n";
52: return 0;
53: }

```

Dieses Programm gibt die Fläche eines Rechtecks und auch die hypothetischen Steuern für einen rechteckigen Bereich aus. Die Ausgabe ist in Abbildung A.11 zu sehen.

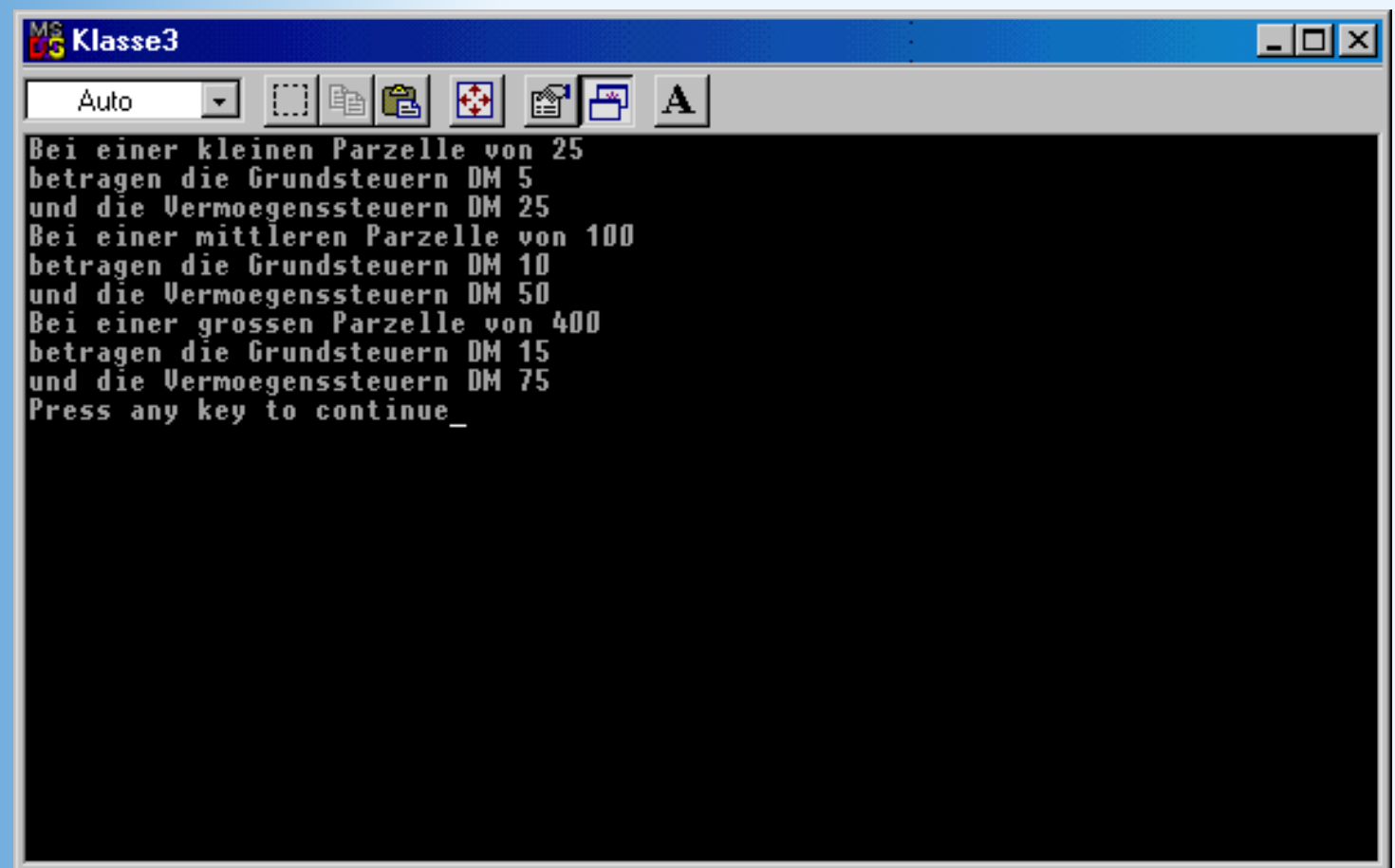


Abbildung A.11:
Ausgabe des Programms Class3.cpp

Die Zeilen 5 bis 14 definieren die Klasse `Steuer_bescheid`. Die Klasse enthält die zwei privaten Datenelementen `int Grund_steuer` und `int Verm_steuer` sowie drei öffentliche Methoden. Insbesondere sollten Sie sich die Deklaration und Definition dieser Methoden ansehen. In den früheren Beispielen haben Sie nur die Methoden in der Klasse deklariert. Der Zugriff auf die Funktionsdefinitionen erfolgte mit dem Zugriffsoperator (`::`). Im vorliegenden Beispiel deklarieren Sie die Methode und schreiben auch gleich die Definition. Man bezeichnet diese Technik als Inline-Implementierung der Funktion. Diese Technik bietet sich an, wenn die Definition einer Funktion kurz und knapp ist. Darüber

hinaus läßt sich damit die Effizienz des Programms (Ausführungsgeschwindigkeit) verbessern, weil das Programm keine Sprünge in die und aus der Funktionsdefinition unternehmen muß.

Die Datenelemente `Grund_steuer` und `Verm_steuer` sind privat, so daß man darauf nur über deren Elementmethoden - nämlich `set()`, `get_Verm_steuer()` und `get_Grund_steuer()` - zugreifen kann.

Die Zeilen 17 bis 29 deklarieren die Klasse `Parzelle` mit ihren Datenelementen und Methoden. In Zeile 20 wird die Klasse `Steuer_bescheid` in die Klasse eingebettet. In dieser Zeile wird auch das Objekt `steuern` deklariert, das unter dem Schutz der Klasse `Parzelle` steht. Die einzigen Methoden, über die man auf dieses Objekt zugreifen kann, sind diejenigen, die zur Klasse `Parzelle` gehören. Die Klasse `Parzelle` verfügt über vier öffentliche Methoden, die in Zeile 22 und den Zeilen 26 bis 28 deklariert und definiert sind. Zeile 25 der Methode `set()` läßt eine weitere `set()`-Methode definieren. Es handelt sich hier nicht um eine rekursive Methode, sondern eher um ein weiteres Beispiel für das Überladen von Funktionen. Die `set()`-Methoden in Zeile 10 und Zeile 22 unterscheiden sich in der Anzahl der Parameter. Die Methode `set()` in Zeile 25 kann auf das Objekt `steuern` zugreifen, da es unter der Klasse `Steuer_bescheid` in Zeile 20 definiert ist.

Die Funktion `main()` beginnt in Zeile 32 und hat den Rückgabotyp `int`. In Zeile 34 werden die Objekte der Klasse `Parzelle` deklariert. Die Zeilen 36 bis 38 setzen die Werte der Objekte mittels der Methode `set()`. Hierbei ist besonders zu beachten, daß die Klasse `Steuer_bescheid` keine Objekte in der Methode `main()` hat, so daß Sie auf keine Datenelemente oder Methoden dieser Klasse von `main()` aus zugreifen können.

Zeile 41 gibt die Fläche der `Parzelle` durch Aufruf der Methode `get_area()` auf einem Objekt der Klasse `Parzelle` aus. Die Zeile 42 enthält die Ausgabe der Grundsteuer durch Aufruf der Methode `get_data2()` auf einem Objekt der Klasse `Parzelle`. Dieses Vorgehen ist erforderlich, weil die `Grund_steuer` ein Datenelement der Klasse `Steuer_bescheid` ist, auf die man in der Methode `main()` nicht direkt zugreifen kann. Man verwendet hier die Methode `get_data2()`, die eine Methode der Klasse `Parzelle` ist und Zugriff auf das Objekt `steuern` hat, auf das man wiederum über `get_Grund_steuer` zugreifen kann.

Vererbung

Einer der Vorteile bei der Programmierung in C++ oder einer anderen objektorientierten Sprache besteht darin, daß man eine globale Herangehensweise in eine lokale überführen kann. Nehmen wir an, Sie entwickeln ein Programm, das alle Metalle und deren Eigenschaften einschließt. Wenn Sie mit der Klassenlösung des vorherigen Abschnitts arbeiten, haben Sie wahrscheinlich eine Klasse namens `Metalle`. Die Datenelemente der Klasse `Metalle` könnten `Dichte` und `Volumen` sein. Weiterhin könnten Sie eine Klasse `Gold` und eine Klasse `Aluminium` erstellen. Die Datenelemente, die `Gold` und `Aluminium` beschreiben, brauchen alle Eigenschaften der `Metalle` und daneben ihre eigenen Datenelementen wie `Farbe` und `Glanz`. Wenn Sie eine Hierarchie von Klassen entwickeln, so daß etwa die Klassen für `Gold` und `Aluminium` nur ihre individuellen Datenelemente enthalten, aber die allgemeinen Eigenschaften von der übergeordneten Klasse `Metalle` übernehmen - dann arbeiten Sie mit Vererbung.

Vererbung bezeichnet man auch als Ableitung. Die neue Klasse erbt die Funktionalität einer vorhandenen Klasse. Die vorhandene Klasse heißt Basisklasse und die neue Klasse abgeleitete Klasse. Eine ähnliche Vererbung läßt sich für Tiere, Säugetiere und Hunde ableiten.

Mit dem Doppelpunktopperator (`:`) kann man eine neue Klasse von einer Basisklasse ableiten:

```
class Mensch : public Saeugetier
```


In diesem Beispiel ist die neue Klasse **Mensch** von der Basisklasse **Saeugetier** abgeleitet. Die abgeleitete Klasse **Mensch** erhält die gesamte Funktionalität der Basisklasse **Saeugetier**. Darüber hinaus kann die Klasse **Mensch** weitere Funktionen enthalten, wie etwa die Fähigkeit, Auto zu fahren oder seinen Lebensunterhalt zu verdienen. Das Beispiel in Listing A.13 zeigt, wie man die Objekte vom Typ **Mensch** erzeugt und auf dessen Daten und Funktionen zugreift.

Listing A.13: Das Programm Vererbg1.cpp

```

1: // Arbeitsbereich Name: Vererbung
2: // Programmname : Vererbg1.cpp
3:
4: #include <iostream.h>
5: enum GESCHLECHT { MAENNLICH, WEIBLICH };
6:
7: class Saeugetier
8: {
9: public:
10: // Konstruktoren
11: Saeugetier():seinAlter(35), seinGewicht(90){}
12: ~Saeugetier(){}
13:
14: int GetAlter()const { return seinAlter; }
15: void SetAlter(int alter) { seinAlter = alter; }
16: int GetGewicht() const { return seinGewicht; }
17: void SetGewicht(int gewicht) { seinGewicht = gewicht; }
18:
19:
20: protected:
21: int seinAlter;
22: int seinGewicht;
23: };
24:
25: class Mensch : public Saeugetier
26: {
27: public:
28:
29: // Konstruktoren
30: Mensch():seinGeschlecht(MAENNLICH){}
31: ~Mensch(){}
32:
33: GESCHLECHT GetGeschlecht() const { return seinGeschlecht; }
34: void SetGeschlecht(GESCHLECHT geschlecht) { seinGeschlecht =
geschlecht; }
35:
36: void Fahren() { cout << "Zur Arbeit fahren...\n"; }
37: void Arbeiten() { cout << "Arbeiten...\n"; }
38:
39: private:
40: GESCHLECHT seinGeschlecht;
41: };
42:
43: void main()

```



```

44: {
45: Mensch Otto_normal;
46: Otto_normal.Fahren();
47: Otto_normal.Arbeiten();
48: cout << "Otto_normal ist " << Otto_normal.GetAlter() << " Jahre
alt\n";
49: cout << "und wiegt " << Otto_normal.GetGewicht() << " Kg \n";
50: }

```

Die Ausgabe des Programms von Listing A.13 zeigt Abbildung A.12.

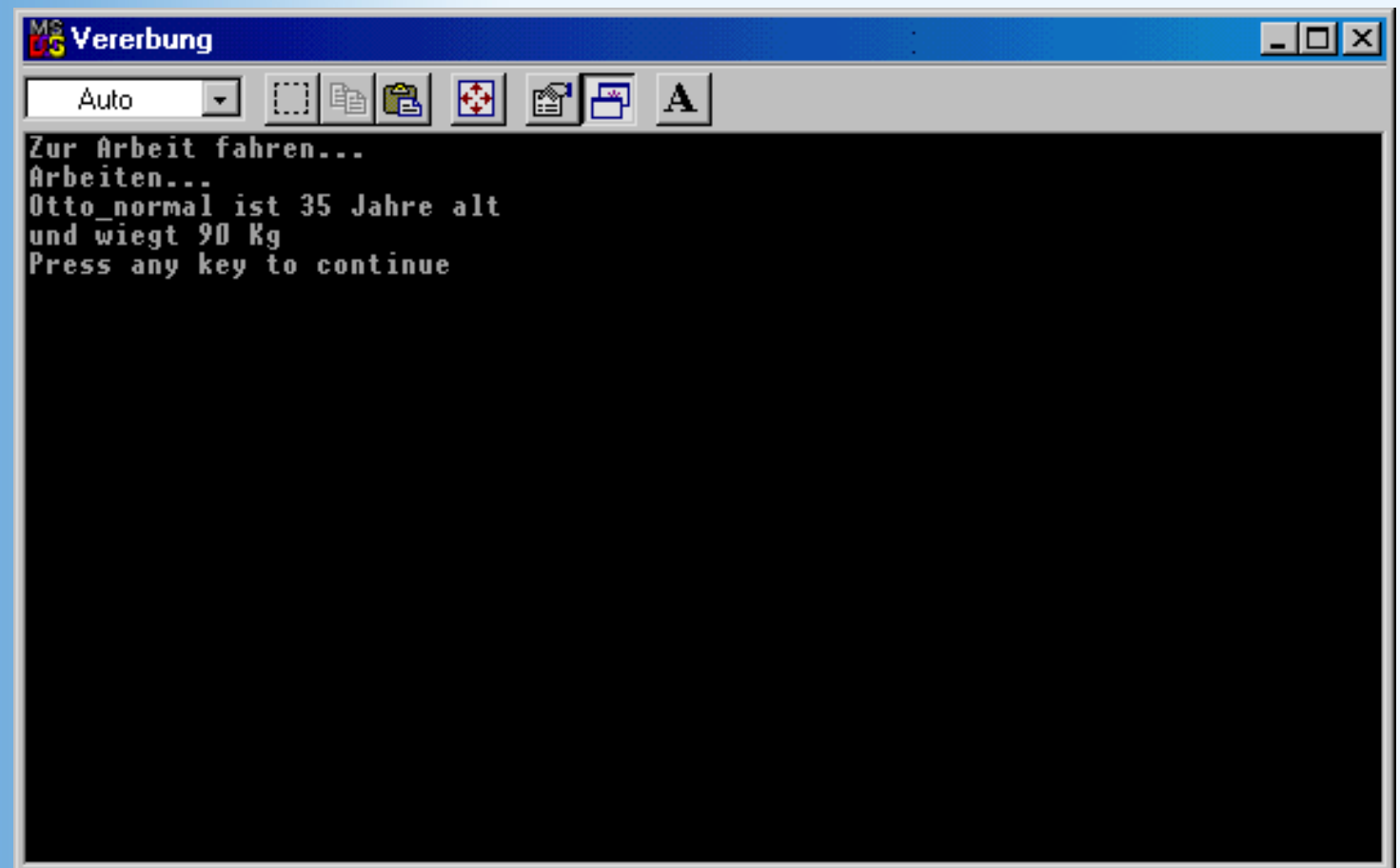


Abbildung A.12:
Ausgabe des Vererbungsprogramms

Zeile 5 enthält eine Definition mit dem Schlüsselwort `enum`. Aufzählungen (enumerations) definieren einen neuen Datentyp mit einer Liste von Bezeichnern. Den Bezeichnern sind feste Werte in aufsteigender Reihenfolge zugeordnet. In diesem Beispiel hat die Variable `MAENNLICH` den Wert 0 und die Variable `WEIBLICH` den Wert 1. Es handelt sich dabei um die automatisch erzeugten Standardwerte. Man kann die Werte auch selbst neu festlegen:

```
enum Alphabet { A, B, C=5, D=1 }
```

In diesem Beispiel hat A den Wert 0, B den Wert 1, C ist 5 und D gleich 1. Wenn Sie keine speziellen Werte angeben, gilt die folgende Zuordnung:

```

A ist 0
B ist 1
C ist 2
D ist 3

```

In Zeile 20 finden Sie ein weiteres neues Schlüsselwort in der Definition der Klasse `Saeugetier`: `protected` (geschützt). Auf die Schlüsselwörter `public` und `private` sind wir bereits weiter oben

eingegangen, als Klassen besprochen wurden, deren Datenelemente alle unter dem Schlüsselwort `private` definiert waren. Sobald ein Datenelement als `private` definiert ist, kann die abgeleitete Klasse nicht mehr darauf zugreifen. Damit den abgeleiteten Klassen die Datenelemente und Methoden der Basisklasse zugänglich sind, muß man sie als `protected` definieren. Das Schlüsselwort `protected` erlaubt den Zugriff nur durch die abgeleiteten Klassen. Alternativ kann man diese Methoden und Elemente als `public` definieren, wodurch man allen Klassen freien Zugriff gewährt. Diese Lösung empfiehlt sich allerdings nicht, da man von der Kapselung abkommt.

Zeile 7 deklariert die Basisklasse `Saeugetier`. Der Konstruktor steht in Zeile 11, der Destruktor in Zeile 12. Der Aufruf des Konstruktors einer Klasse erfolgt immer dann, wenn ein Objekt der Klasse erzeugt wird. Die Konstruktorklasse führt eine zusätzliche Funktion aus, um die Datenelemente `seinAlter(35)` und `seinGewicht(90)` zu initialisieren. Das hätte man auch im Rumpf des Konstruktors wie im folgenden Beispiel erreichen können:

```
Saeugetier()
{
    seinAlter = 35;
    seinGewicht = 180;
};
```

Diese Art der Initialisierung der Datenelemente in der Konstruktordeklaration (wie in Zeile 11 von Listing A.13) ist infolge der internen Initialisierung von Klassen in C++ weitaus effizienter. Greifen Sie möglichst immer auf dieses Verfahren zurück, da es die Effizienz des Codes verbessert.

Wenn man ein Objekt einer abgeleiteten Klasse erzeugt, wird zuerst der Konstruktor der Basisklasse und danach der Konstruktor der abgeleiteten Klasse aufgerufen. Wenn im Beispiel `Otto_normal` erstmals erzeugt wird, erfolgt der Aufruf des Konstruktors der Basisklasse `Saeugetier`. Das Objekt `Otto_normal` wird erst dann erstellt, wenn sowohl der Konstruktor der Basisklasse als auch der Konstruktor der abgeleiteten Klasse aufgerufen wurden. Bei Destrukturen ist die Reihenfolge umgekehrt. Nachdem das Objekt `Otto_normal` nicht mehr existiert, wird der Destruktor der abgeleiteten Klasse vor dem Destruktor der Basisklasse aufgerufen. Zeile 25 definiert den Namen der abgeleiteten Klasse und gibt den Bezug zur betreffenden Basisklasse an.

Im Hinblick auf den Zugriff und die Ausgabe der Daten verdienen die Zeilen 48 und 49 eine besondere Beachtung. In diesen Zeilen greift das `Mensch`-Objekt `Otto_normal` direkt auf die Informationen aus der Basisklasse `Saeugetier` zu. Wie Sie noch aus dem Beispiel `Klasse3.cpp` wissen, mußte man indirekt auf die Klasse `Steuer_bescheid` zugreifen, um Daten aus einer verschachtelten Klasse auszugeben.

Die Vererbung stellt ein bedeutsames Werkzeug in der objektorientierten Programmierung dar. Und wenn man sie effektiv einsetzt, gewährleistet sie die Wiederverwendbarkeit von Code. Das Programm `Vererbg1.cpp` hat Ihnen einen ersten Gesamteindruck der Vererbung und ihrer Eigenschaften gebracht. Wenn man »richtige« Programme schreibt, strukturiert man sie effizienter. Das nächste Programm geht das Ganze logischer und formeller an.

Nehmen wir an, daß Sie ein Programm für den Automarkt schreiben. Hier haben wir es mit PKWs, LKWs, Minivans und Sportwagen zu tun. Die übergeordnete oder Basisklasse ist `Autos`, die anderen Klassen sind abgeleitet. Definieren wir zunächst die Klasse `Autos`. Listing A.14 zeigt den Code.

Listing A.14: Die Header-Datei `Auto.h`

```
1: // Arbeitsbereich: Vererbg2
2: // Programmname: Auto.h
3:
4: #ifndef AUTO_H
```

```

5: #define AUTO_H
6:
7: class autos
8: {
9: protected:
10: int verbrauch;
11: float tankfuellung;
12: public:
13: void initialize(int in_verbr, float in_tank);
14: int get_verbr(void);
15: float get_tank(void);
16: float reichweite(void);
17: };
18:
19: #endif

```

Die Zeilen 4 und 5 enthalten Direktiven an den Präprozessor. Auf die Direktive in Zeile 4 gehen wir gegen Ende dieses Abschnitts näher ein. Zeile 7 definiert die Klasse `Autos`. Diese Klasse umfaßt zwei Datenelemente und vier Methoden. In der Header-Datei steht nur die Deklaration der Klasse. Die Definition der zugehörigen Methoden befindet sich in der Datei `Auto.cpp`, die in Listing A.15 wiedergegeben ist.

Listing A.15: Die Quellcodedatei Auto.cpp

```

1: // Arbeitsbereich: Vererbg2
2: // Programmname : Auto.cpp
3: #include "Auto.h"
4:
5:
6: void autos::initialize(int in_verbr, float in_tank)
7: {
8: verbrauch = in_verbr;
9: tankfuellung = in_tank;
10: }
11:
12: // Durchschnittsverbrauch bestimmen
13: int autos::get_verbr()
14: {
15: return verbrauch;
16: }
17:
18: // Tankkapazität ermitteln
19: float autos::get_tank()
20: {
21: return tankfuellung;
22: }
23:
24: // Mögliche Reichweite zurückgeben
25: float autos::reichweite()
26: {
27: return verbrauch * tankfuellung;
28: }

```

Die Methode `get_verbr` liefert den Wert für den Kraftstoffverbrauch für ein bestimmtes Fahrzeug. Über die Methode `get_tank` erhält man das Fassungsvermögen des Tanks zurück. Als nächstes definieren Sie die erste abgeleitete Klasse, eine `Wagen`- Klasse gemäß Listing A.16.

Listing A.16: Die Header-Datei `Wagen.h`

```
1: // Arbeitsbereich: Vererb2
2: // Programmname: Wagen.h
3:
4: #ifndef WAGEN_H
5: #define WAGEN_H
6:
7: #include "Auto.h"
8:
9: class Wagen : public Autos
10: {
11:     int anzahl_tueren;
12: public:
13:     void initialize(int in_verbr, float in_tank, int tuer = 4);
14:     int tueren(void);
15: };
16:
17: #endif
```

Die Klasse `Wagen` ist von der Klasse `Autos` abgeleitet. In dieser Eigenschaft hat sie Zugriff auf alle Methoden der Basisklasse `Autos`. Zusätzlich verfügt die Klasse `Wagen` über ein Datenelement für die Anzahl der Türen. Die Methoden der Klasse `Wagen` sind in der Datei `Wagen.cpp` gemäß Listing A.17 definiert.

Listing A.17: Die Quellcodedatei `Wagen.cpp`

```
1: // Arbeitsbereich: Vererb2
2: // Programmname: Wagen.cpp
3:
4: #include "Wagen.h"
5:
6: void Wagen::initialize(int in_verbr, float in_tank, int tuer)
7: {
8:     anzahl_tueren = tuer;
9:     verbrauch = in_verbr;
10:    tankfuellung = in_tank;
11: }
12:
13:
14: int Wagen::tueren(void)
15: {
16:    return anzahl_tueren;
17: }
```

Die Methode `initialization` ist in den Zeilen 6 bis 11 definiert. Hierbei ist zu beachten, daß auch die Basisklasse `Autos` (`Auto.h`) über eine Methode `initialization` verfügt. Die Methode `initialization` in der Klasse `Wagen` überschreibt die Methode `initialization` der Basisklasse. Zu guter Letzt folgt die Definition von `main()`. Die Methode `main()` ist in der Datei `Fuhrpark.cpp` (Listing A.18) definiert.

Listing A.18: Die Quellcodedatei Fuhrpark.cpp

```
1: // Arbeitsbereich: Vererb2
2: // Programmname: Fuhrpark.cpp
3:
4: #include <iostream.h>
5: #include "Auto.h"
6: #include "Wagen.h"
7:
8: int main()
9: {
10:
11: Wagen sedan;
12:
13: sedan.initialize(24, 20.0, 4);
14: cout << "Der Sedan hat eine Reichweite von " << sedan.reichweite()
<<
15: " Kilometern.\n";
16: cout << "Der Sedan hat " << sedan.tueren() << " Tueren.\n";
17:
18: return 0;
19: }
```

In `main()` ist nur ein einziges Objekt definiert. Zeile 11 deklariert ein Objekt der Klasse `Wagen`. Die Initialisierung steht in Zeile 13. Die Methode `initialization` übergibt den Durchschnittsverbrauch des Autos (`verbrauch`) und das Fassungsvermögen des Tanks. Mit diesen Informationen erfolgt der Zugriff auf die Methode `reichweite` in der Basisklasse, die in `Auto.cpp` definiert ist. Die abgeleitete Klasse hat Zugriff auf die Methoden der Basisklasse. Darüber hinaus übergibt die abgeleitete Klasse Informationen an ihr eigenes Datenelement zur Anzahl der Fahrzeugtüren. Das Ergebnis bei Ausführung des Programms zeigt Abbildung A.13.

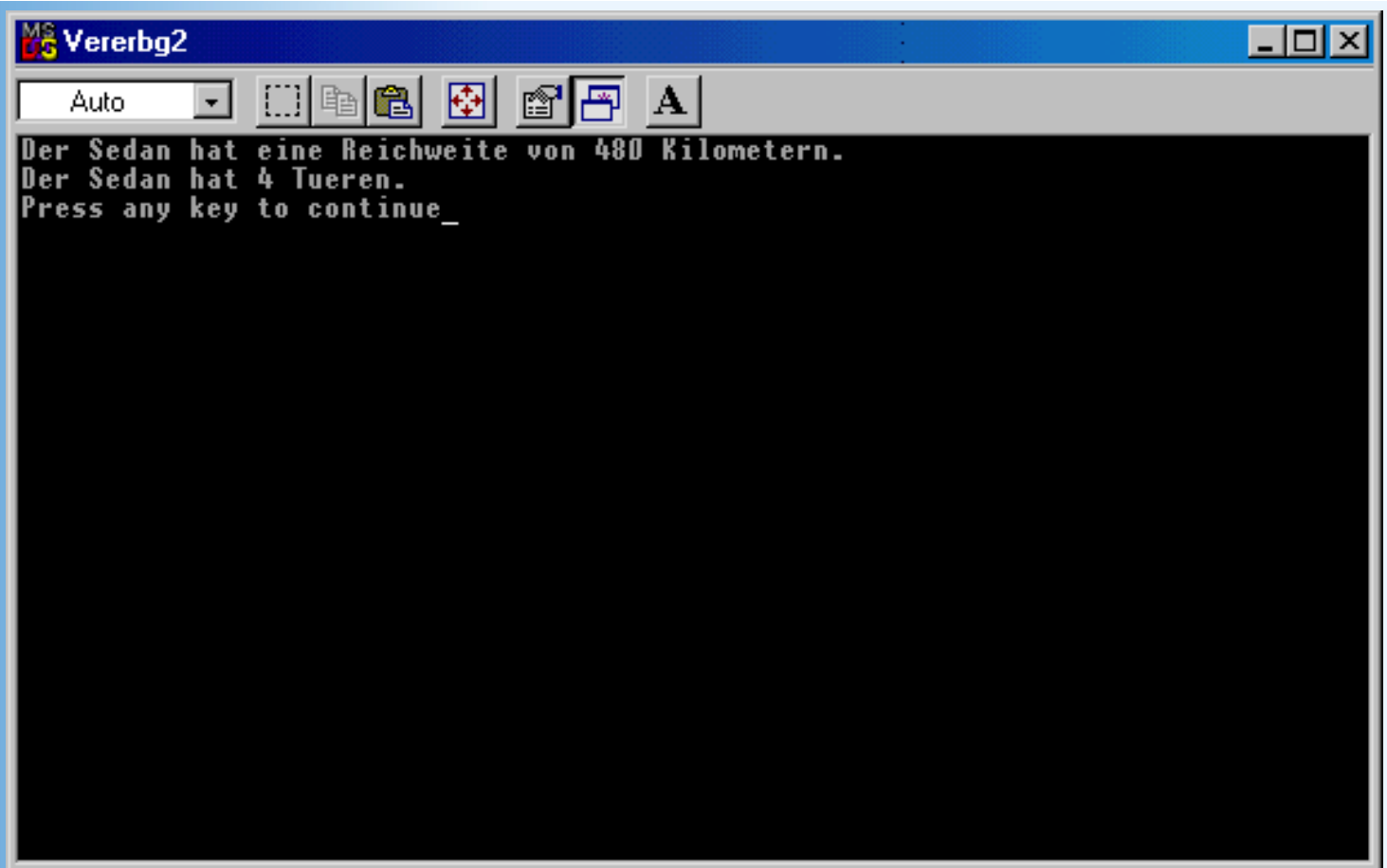


Abbildung A.13:
Klassenergebnisse von Vehicle

Sie können nun weitere Klassen für anderen Fahrzeugtypen hinzufügen. Erstellen Sie eigene Klassen für einen LKW und einen Minivan, und leiten Sie sie von der Basisklasse genauso ab wie die Klasse `Wagen`.

Wenn Sie eine andere Klasse für LKWs hinzufügen, achten Sie darauf, die Präprozessordirektiven der Zeilen 4 und 19 von Listing A.14 einzubinden. Diese Zeilen seien hier noch einmal genannt:

```
4 #ifndef AUTO_H
5 #define AUTO_H
.
.
.
19 #endif
```

Da die Klasse `LKW` von der übergeordneten Klasse `Autos` abgeleitet ist, muß sie die Datei `Auto.h` in `LKW.h` einbinden. Der Header der Klasse `Wagen`, `Wagen.h`, bindet bereits `Auto.h` aus dem gleichen Grund ein. Wenn Sie nun eine Methode erzeugen, die sowohl die Klasse `LKW` als auch die Klasse `Wagen` verwendet, binden Sie `Auto.h` möglicherweise zweimal ein, was zu einem Compilerfehler führt. Um das zu verhindern, fügen Sie die Zeilen 4 und 5 von Listing A.14 hinzu. Durch den Befehl in Zeile 4 prüft der Compiler, ob die Klasse `Autos` (anhand des Bezeichners `AUTO_H`) bereits definiert ist. Wenn das nicht der Fall ist, springt das Programm zu Zeile 5, definiert den Bezeichner und dann die Klasse. Ist die Klasse bereits definiert, springt das Programm zu Zeile 19 und endet hier.

Zusammenfassung

Gratulation! Sie haben nun fast alle Merkmale und Eigenschaften von C++ kennengelernt. Damit besitzen Sie eine solide Grundlage, um die Vorteile von Visual C++ und der objektorientierten Programmierung umfassend nutzen zu können.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).

Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Tag B

Antworten

Antworten für Tag 1

Antworten zu den Kontrollfragen

1. Markieren Sie im Dialog-Editor die zu ändernde Schaltfläche. Klicken Sie mit der rechten Maustaste, und wählen Sie **Eigenschaften** aus dem Kontextmenü. Ändern Sie den Wert im Feld **Titel**.
2. Mit dem Anwendungs-Assistenten erstellen Sie das Gerüst Ihrer Anwendung basierend auf dem Typ der Anwendung und der geplanten Funktionalität. Das Gerüst unterstützt dann die in den Dialogfeldern des Assistenten spezifizierte Funktionalität.
3. Mit dem Klassen-Assistenten erzeugen Sie eine Funktion und verbinden sie mit einem Objekt, das eine bestimmte Windows-Nachricht behandelt. Der Klassen-Assistent erzeugt die Funktion und öffnet die zugehörige Quelldatei bereits an der richtigen Stelle, so daß Sie sofort mit der Eingabe des eigenen Codes beginnen können.

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**.
2. Erweitern Sie den Zweig **Dialog**, und doppelklicken Sie auf das Dialogfeld **IDD_ABOUTBOX**. Dieses Dialogfeld wird daraufhin im Editor des Visual Studios zur Bearbeitung angezeigt.
3. Klicken Sie in der Werkzeugleiste auf das Symbol für Schaltfläche.
4. Klicken und ziehen Sie die Maus an die Stelle im Dialogfeld, wo Sie die Schaltfläche plazieren möchten.
5. Öffnen Sie den Eigenschaftsdialog für die neue Schaltfläche. Ändern Sie die ID und den Titel, um den Text auf der Schaltfläche festzulegen. Schließen Sie das Eigenschaftsdialogfeld.
6. Öffnen Sie den Klassen-Assistenten, und fügen Sie eine neue Funktion für die Klicknachricht (**BN_CLICKED**) für Ihre neue Schaltfläche hinzu.
7. Klicken Sie im Klassen-Assistenten auf die Schaltfläche **Code bearbeiten**. Der Klassen-Assistent öffnet daraufhin die Quelldatei und zeigt die neue Funktion zur Bearbeitung an.
8. Fügen Sie die Funktion **MessageBox** hinzu, um die Nachricht für den Benutzer auszugeben.
9. Kompilieren und starten Sie die Anwendung, um die neue Schaltfläche zu testen.

Antworten für Tag 2

Antworten zu den Kontrollfragen

1. Indem Sie die Tabulator-Reihenfolge der Steuerelemente in Ihrem Anwendungsfenster festlegen, können Sie bestimmen, auf welchem Weg der Benutzer durch das Anwendungsfenster navigiert. Wenn der Benutzer mit der Tastatur arbeitet, kann er mit der <Tab>-Taste von einem Steuerelement zum nächsten wechseln und über die Zugriffstasten ein bestimmtes Steuerelement direkt anspringen. Die Tabulator-Reihenfolge bietet dem Benutzer ein einheitliches und vorhersagbares Verhalten, wenn er im Anwendungsfenster navigiert.
2. Wenn man eine Zugriffstaste für ein Textfeld vorsieht und dann sicherstellt, daß sich das Textfeld unmittelbar vor dem zugeordneten Eingabefeld befindet, kann der Benutzer die Zugriffstaste des Textfelds wählen, um direkt zum Eingabefeld zu gelangen.
3. Die eindeutigen Objekt-IDs für die beiden statischen Textfelder sind erforderlich, da diese beiden Steuerelemente in Abhängigkeit vom Zustand der Kontrollkästchen, die bestimmte Gruppen von Steuerelementen aktivieren oder deaktivieren, manipuliert werden sollen.
4. Wenn der Benutzer den Wert des Steuerelements auf dem Bildschirm ändert, ist die Funktion `UpdateData` mit dem Argument `TRUE` aufzurufen, um die Werte aus den Steuerelementen im Fenster in die Variablen, die mit diesen Steuerelementen verbunden sind, zu übertragen. Ruft man `UpdateData` nicht auf, spiegeln die Variablen nicht die aktuellen Änderungen wider, die der Benutzer in den Steuerelementen auf dem Bildschirm vorgenommen hat.

Lösungen zu den Übungen

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Klicknachricht der Schaltfläche **Standardnachricht** (IDC_DFLTMSG) hinzu. In die Funktion schreiben Sie den Code gemäß Listing B.1.

Listing B.1: Tag2Dlg.cpp - Dieser Code plazierte eine Standardnachricht im Eingabefeld.

```

1: void CTag2Dlg::OnDfltmsg()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Nachricht auf Standardnachricht setzen
10: m_strMessage = "Nachricht hier eingeben";
11:
12: // Bildschirm aktualisieren
13: UpdateData(FALSE);
14:
15: //////////////////////////////////
16: // EIGENER CODE, ENDE
17: //////////////////////////////////
18: }
```

2. Fügen Sie Funktionen für die Kontrollkästchen **Programmaktion aktivieren** und **Programmaktion zeigen** hinzu. In diese Funktionen übernehmen Sie den Code aus Listing B.2.

Listing B.2: Tag2Dlg.cpp - Die zur Ausführung von Programmen relevanten Steuerelemente werden mit diesem Code aktiviert/deaktiviert und angezeigt/ausgeblendet.

```

1: void CTag2Dlg::OnCkenblpgm()
```

```

2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Aktuelle Werte vom Bildschirm holen
10: UpdateData(TRUE);
11:
12: // Kontrollkästchen 'Programmaktion aktivieren' eingeschaltet?
13: if (m_bEnablePgm == TRUE)
14: {
15: // Ja, dann alle Steuerelemente aktivieren, die
16: // für den Start eines Programms relevant sind.
17: GetDlgItem(IDC_PROGTORUN)->EnableWindow(TRUE);
18: GetDlgItem(IDC_RUNPGM)->EnableWindow(TRUE);
19: GetDlgItem(IDC_STATICPGM)->EnableWindow(TRUE);
20: }
21: else
22: {
23: // Nein, dann alle Steuerelemente deaktivieren, die
24: // für den Start eines Programms relevant sind.
25: GetDlgItem(IDC_PROGTORUN)->EnableWindow(FALSE);
26: GetDlgItem(IDC_RUNPGM)->EnableWindow(FALSE);
27: GetDlgItem(IDC_STATICPGM)->EnableWindow(FALSE);
28: }
29:
30: //////////////////////////////////
31: // EIGENER CODE, ENDE
32: //////////////////////////////////
33: }
34:
35: void CTag2Dlg::OnCkshwpgm()
36: {
37: // TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
38:
39: //////////////////////////////////
40: // EIGENER CODE, ANFANG
41: //////////////////////////////////
42:
43: // Aktuelle Werte vom Bildschirm holen
44: UpdateData(TRUE);
45:
46: // Kontrollkästchen 'Programmaktion zeigen' eingeschaltet?
47: if (m_bShowPgm == TRUE)
48: {
49: // Ja, dann alle Steuerelemente anzeigen, die
50: // für den Start eines Programms relevant sind.
51: GetDlgItem(IDC_PROGTORUN)->ShowWindow(TRUE);
52: GetDlgItem(IDC_RUNPGM)->ShowWindow(TRUE);
53: GetDlgItem(IDC_STATICPGM)->ShowWindow(TRUE);
54: }

```

```

55: else
56: {
57: // Nein, dann alle Steuerelemente ausblenden, die
58: // für den Start eines Programms relevant sind.
59: GetDlgItem(IDC_PROGTORUN)->ShowWindow(FALSE);
60: GetDlgItem(IDC_RUNPGM)->ShowWindow(FALSE);
61: GetDlgItem(IDC_STATICPGM)->ShowWindow(FALSE);
62: }
63:
64: //////////////////////////////////////
65: // EIGENER CODE, ENDE
66: //////////////////////////////////////
67: }

```

3. Modifizieren Sie die Funktion OnRunpgm gemäß Listing B.3.

Listing B.3: Tag2Dlg.cpp - Dieser Code startet das Programm, dessen Name der Benutzer in das Kombinationsfeld Programm starten eingibt.

```

1: void CTag2Dlg::OnRunpgm()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
4: // Benachrichtigung hier einfügen
5:
6: //////////////////////////////////////
7: // EIGENER CODE, ANFANG
8: //////////////////////////////////////
9:
10: // Aktuelle Werte vom Bildschirm holen
11: UpdateData(TRUE);
12:
13: // Lokale Variable zur Aufnahme des Programmnamens deklarieren
14: CString strPgmName;
15:
16: // Programmname in die lokale Variable kopieren
17: strPgmName = m_strProgToRun;
18:
19: // Programmname in Großbuchstaben umwandeln
20: strPgmName.MakeUpper();
21:
22: // Programm Paint gewählt?
23: if (strPgmName == "PAINT")
24: // Ja, Paint starten
25: WinExec("pbrush.exe", SW_SHOW);
26:
27: // Editor (Notepad) gewählt?
28: if (strPgmName == "EDITOR")
29: // Ja, Editor starten
30: WinExec("notepad.exe", SW_SHOW);
31:
32: // Solitär gewählt?
33: if (strPgmName == "SOLITÄR")
34: // Ja, Solitär starten
35: WinExec("sol.exe", SW_SHOW);
36:
37: // Programm zu eingegebenem Namen in das Kombinationsfeld starten

```

```

37: if ((strPgmName != "PAINT") && (strPgmName != "EDITOR") &&
38: (strPgmName != "SOLITÄR"))
39: // Ja, eingegebenen Programmnamen aufrufen
40: WinExec(strPgmName, SW_SHOW);
41:
42: //////////////////////////////////
43: // EIGENER CODE, ENDE
44: //////////////////////////////////
45: }

```

Antworten für Tag 3

Antworten zu den Kontrollfragen

1. WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDOWNBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP, WM_RBUTTONDOWNBLCLK, WM_MOUSEMOVE und WM_MOUSEWHEEL.

2. Man kann die Flags, die an die Funktion OnMouseMove übergeben werden, mit dem Flag MK_LBUTTON wie folgt maskieren:

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

3. Geben Sie in der Behandlungsroutine OnSetCursor den Wert TRUE zurück. Damit verhindern Sie, daß die Funktion OnSetCursor der Basisklasse aufgerufen wird.

Lösungen zu den Übungen

1. Fügen Sie eine Funktion für die Nachricht WM_RBUTTONDOWN hinzu, und schreiben Sie in die Funktion den Code aus Listing B.4.

Listing B.4: MausDlg.cpp - Die Funktion OnRButtonDown

```

1: void CMAusDlg::OnRButtonDown(UINT nFlags, CPoint point)
2: {
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
  und/oder Standard aufrufen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Aktuellen Punkt als Startpunkt setzen
10: m_iPrevX = point.x;
11: m_iPrevY = point.y;
12:
13: //////////////////////////////////
14: // EIGENER CODE, ENDE
15: //////////////////////////////////
16:
17: CDialog::OnRButtonDown(nFlags, point);
18: }

```

Erweitern Sie die Funktion OnMouseMove gemäß Listing B.5.

Listing B.5: MausDlg.cpp - Die modifizierte Funktion OnMouseMove

```

1: void CMAusDlg::OnMouseMove(UINT nFlags, CPoint point)
2: {

```



```
3: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
und/oder Standard aufrufen
4:
5: //////////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////////
8:
9: // Linke Maustaste gedrückt?
10: if ((nFlags & MK_LBUTTON) == MK_LBUTTON)
11: {
12: // Gerätekontext holen
13: CClientDC dc(this);
14:
15: // Neuen Stift erzeugen
16: CPen lpen(PS_SOLID, 16, RGB(255, 0, 0));
17:
18: // Den neuen Stift verwenden
19: dc.SelectObject(&lpen);
20:
21: // Linie vom letzten zum aktuellen Punkt zeichnen
22: dc.MoveTo(m_iPrevX, m_iPrevY);
23: dc.LineTo(point.x, point.y);
24:
25: // Aktuellen Punkt als letzten Punkt speichern
26: m_iPrevX = point.x;
27: m_iPrevY = point.y;
28: }
29:
30: // Rechte Maustaste gedrückt?
31: if ((nFlags & MK_RBUTTON) == MK_RBUTTON)
32: {
33: // Gerätekontext holen
34: CClientDC rdc(this);
35:
36: // Neuen Stift erzeugen
37: CPen rpen(PS_SOLID, 16, RGB(0, 0, 255));
38:
39: // Den neuen Stift verwenden
40: rdc.SelectObject(&rpen);
41:
42: // Linie vom letzten zum aktuellen Punkt zeichnen
43: rdc.MoveTo(m_iPrevX, m_iPrevY);
44: rdc.LineTo(point.x, point.y);
45:
46: // Aktuellen Punkt als letzten Punkt speichern
47: m_iPrevX = point.x;
48: m_iPrevY = point.y;
49: }
50:
51: //////////////////////////////////////
52: // EIGENER CODE, ENDE
53: //////////////////////////////////////
54:
55: CDialog::OnMouseMove(nFlags, point);
56: }
```

2. Die modifizierte Funktion OnKeyDown kann etwa folgendermaßen aussehen:

```
void CMAusDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
// TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen und/
oder Standard aufrufen

////////////////////
// EIGENER CODE, ANFANG
////////////////////

char lsChar; // Zeichen der gedrückten Taste
HCURSOR lhCursor; // Handle zum anzuzeigenden Cursor

// Code der gedrückten Taste in Zeichen umwandeln
lsChar = char(nChar);

// Ist Zeichen ein "A"
if (lsChar == 'A')
{
// Pfeilcursor laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "B"
if (lsChar == 'B')
{
// Balkencursor laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_IBEAM);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "C"
if (lsChar == 'C')
{
// Sanduhrzeiger laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_WAIT);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "D"
if (lsChar == 'D')
{
// Fadenkreuz-Cursor laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_CROSS);
```

```
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "E"
if (lsChar == 'E')
{
// Cursor mit Pfeil nach oben laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_UPARROW);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "F"
if (lsChar == 'F')
{
// Cursor zur Größenveränderung laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZEALL);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "G"
if (lsChar == 'G')
{
// Cursor zur Größenveränderung, von Nordwest nach Südost, laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENWSE);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "H"
if (lsChar == 'H')
{
// Cursor zur Größenveränderung, von Nordost nach Südwest, laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENESW);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "I"
if (lsChar == 'I')
{
// Cursor zur Größenveränderung, von West nach Ost, laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZEWE);
```

```
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "J"
if (lsChar == 'J')
{
// Cursor zur Größenveränderung, von Nord nach Süd, laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_SIZENS);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

if (lsChar == 'K')
{
// "Kein" Cursor laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_NO);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

if (lsChar == 'L')
{
// Cursor beim Starten der Anwendung (Pfeil und Sanduhr) laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_APPSTARTING);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

if (lsChar == 'M')
{
// Hilfe-Cursor (Pfeil und Fragezeichen) laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_HELP);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
SetCursor(lhCursor);
}

// Ist Zeichen ein "X"
if (lsChar == 'X')
{
// Pfeilcursor laden
lhCursor = AfxGetApp()->LoadStandardCursor(IDC_ARROW);
// Cursorflag setzen
m_bCursor = TRUE;
// Bildschirmcursor setzen
```

```

SetCursor(lhCursor);
// Anwendung beenden
OnOK();
}

////////////////////
// EIGENER CODE, ENDE
////////////////////

CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
}

```

Antworten für Tag 4

Antworten zu den Kontrollfragen

1. Durch die Definition der beiden IDs sind diese als Konstanten durch die gesamte Anwendung hindurch verfügbar.

2. Fügen Sie sie als `#define`-Konstanten in die Header-Datei der Klasse (`TimerDlg.h`) wie folgt ein:

```

////////////////////////////////////
CTimerDlg Dialogfeld

```

```

#define ID_CLOCK_TIMER 1
#define ID_COUNT_TIMER 2

```

```

class CTimerDlg : public CDialog
{
.
.
.
}

```

3. Anhand der Timer-ID läßt sich bestimmen, welcher Timer das Ereignis ausgelöst hat.

4. Genau ein Timer-Ereignis.

Lösung zur Übung

Um das Intervall zu ändern, mit dem ein Timer läuft, müssen Sie zuerst den Timer stoppen und dann erneut starten, wie es aus Listing B.6 hervorgeht.

Listing B.6: Die überarbeiteten Funktionen `OnStarttime` und `OnStoptimer`

```

1: void CTimerDlg::OnStarttime()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
  Benachrichtigung hier einfügen
4:
5: //////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////
8:
9: // Variablen aktualisieren
10: UpdateData(TRUE);
11:
12: // Zähler aktualisieren

```

```
13: m_iCount = 0;
14: // Zähler für Anzeige formatieren
15: m_sCount.Format("%d", m_iCount);
16:
17: // Dialogfeld aktualisieren
18: UpdateData(FALSE);
19: // Timer starten
20: SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
21:
22: // Uhren-Timer anhalten
23: KillTimer(ID_CLOCK_TIMER);
24: // Uhren-Timer mit dem Intervall des Zählers neu starten
25: SetTimer(ID_CLOCK_TIMER, m_iInterval, NULL);
26:
27: // Schaltfläche Timer anhalten aktivieren
28: m_cStopTime.EnableWindow(TRUE);
29: // Schaltfläche Timer starten deaktivieren
30: m_cStartTime.EnableWindow(FALSE);
31:
32: //////////////////////////////////////
33: // EIGENER CODE, ENDE
34: //////////////////////////////////////
35: }
36:
37: void CTimerDlg::OnStoptimer()
38: {
39: // TODO: Code für die Behandlungsroutine der Steuerelement-
Benachrichtigung hier einfügen
40:
41: //////////////////////////////////////
42: // EIGENER CODE, ANFANG
43: //////////////////////////////////////
44:
45: // Timer anhalten
46: KillTimer(ID_COUNT_TIMER);
47:
48: // Uhren-Timer anhalten
49: KillTimer(ID_CLOCK_TIMER);
50: // Uhren-Timer mit Intervall 1 Sekunde neu starten
51: SetTimer(ID_CLOCK_TIMER, 1000, NULL);
52:
53: // Schaltfläche Timer anhalten deaktivieren
54: m_cStopTime.EnableWindow(FALSE);
55: // Schaltfläche Timer starten aktivieren
56: m_cStartTime.EnableWindow(TRUE);
57:
58: //////////////////////////////////////
59: // EIGENER CODE, ENDE
60: //////////////////////////////////////
61: }
```


Antworten für Tag 5

Antworten zu den Kontrollfragen

1. IDRETRY und IDCANCEL.
2. Als MFC-Klassen sind die folgenden allgemeinen Windows-Dialogfelder definiert:
 - Dateiauswahl
 - Schriftauswahl
 - Farbauswahl
 - Seite zum Drucken einrichten
 - Drucken
 - Suchen und Ersetzen
3. Ein modales Dialogfeld stoppt alle Programmabläufe in der Anwendung, bis der Benutzer auf das Dialogfeld antwortet. Ein nichtmodales Dialogfeld gestattet dem Benutzer, mit der übrigen Anwendung weiterzuarbeiten, während das Dialogfeld geöffnet bleibt und Eingaben entgegennehmen kann.
4. In der Variablendeklaration der Klasseninstanz übergeben Sie FALSE statt TRUE. Damit sieht die Variablendeklaration folgendermaßen aus:
`CFileDialog m_ldFile(FALSE);`
5. Im benutzerdefinierten Dialogfeld ist lediglich die Funktion `UpdateData` aufzurufen, bevor das Dialogfeld geschlossen wird. Da Sie die Schaltflächen **OK** und **Abbrechen** nicht aus dem Dialogfeld gelöscht haben, führt die Schaltfläche **OK** automatisch diese Funktion aus.

Lösungen zu den Übungen

1. Modifizieren Sie die Funktion `OnFileopen` wie folgt:

```
void CDialogDlg::OnFileopen()
{
// TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen

////////////////////////////////////
// EIGENER CODE, ANFANG
////////////////////////////////////

CFileDialog m_ldFile(TRUE);

// Dialogfeld Öffnen zeigen und Ergebnis auffangen
if (m_ldFile.DoModal() == IDOK)
{
// Gewählten Dateinamen ermitteln
m_sResults = m_ldFile.GetPathName();
// Dialogfeld aktualisieren
UpdateData(FALSE);
}

////////////////////////////////////
// EIGENER CODE, ENDE
////////////////////////////////////
}
```

Die Funktion `GetPathName` gibt den Pfad und Dateinamen zurück, so daß bei Änderung des Funktionsaufrufs von `GetFileName` in `GetPathName` der Pfad zusammen mit dem Dateinamen in der Anzeige erscheint.

2. Führen Sie die folgenden Schritte aus:

1. Fügen Sie über den Klassen-Assistenten eine Member-Variable in die Klasse `CMsgDlg` ein. Legen Sie den Variablentyp als `int`, den Namen mit `m_iYesNo` und den Zugriff als `Public` fest.
2. Gehen Sie auf die Registerkarte **Ressourcen**, und öffnen Sie das benutzerdefinierte Dialogfeld zur Bearbeitung. Ziehen Sie eine Schaltfläche auf das Dialogfeld, tragen Sie `IDC_YESNO` als ID und `&Ja oder Nein` als Titel ein.

3. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die neue Schaltfläche hinzu, und schreiben Sie in diese Funktion den folgenden Code:

```
void CMsgDlg::OnYesno()
{
// TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen

////////////////////////////////////
// EIGENER CODE, ANFANG
////////////////////////////////////

// Benutzer fragen
m_iYesNo = MessageBox("Ja oder Nein wählen",
"Ja oder Nein",
MB_YESNO);

////////////////////////////////////
// EIGENER CODE, ENDE
////////////////////////////////////
}
```

4. Nehmen Sie in das Hauptdialogfeld eine Schaltfläche mit der ID `IDC_YESNO` und dem Titel `Ja oder &Nein` auf.

5. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die neue Schaltfläche hinzu, und schreiben Sie den folgenden Code in diese Funktion:

```
void CDialogeDlg::OnYesno()
{
// TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen

////////////////////////////////////
// EIGENER CODE, ANFANG
////////////////////////////////////

// Was hat Benutzer geantwortet?
switch (m_dMsgDlg.m_iYesNo)
{
case IDYES: // Mit Ja geantwortet?
m_sResults = "Ja!";
break;
case IDNO: // Mit Nein geantwortet?
m_sResults = "Nein!";
break;
}
```

```

}

// Dialogfeld aktualisieren
UpdateData(FALSE);

////////////////////
// EIGENER CODE, ENDE
////////////////////
}

```

Antworten für Tag 6

Antworten zu den Kontrollfragen

1. COMMAND
2. Öffnen Sie im Dialog-Editor den Eigenschaftsdialog für das Fenster, und wählen Sie das Menü aus der Dropdown-Liste der Menüs aus.
3. Die Dialogklasse für das Fenster, in dem das Menü erscheint.
4. Durch das Ereignis WM_CONTEXTMENU.

Lösungen zu den Übungen

1. Führen Sie die folgenden Schritte aus:

1. Nehmen Sie eine Schaltfläche in das Dialogfeld auf. Tragen Sie für diese Schaltfläche die ID IDC_HELLO und den Titel &He llo ein.
2. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Schaltfläche hinzu. Verwenden Sie OnHe llo als Funktionsname.

2. Führen Sie die folgenden Schritte aus:

1. Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht WM_CONTEXTMENU in Ihr Dialogfeld ein.

2. Schreiben Sie den folgenden Code in die Funktion:

```

void CMenuDlg:: OnContextMenu(CWnd* pWnd, CPoint point)
{
// TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen

////////////////////
// EIGENER CODE, ANFANG
////////////////////

// Lokale Variablen deklarieren
CMenu *m_lMenu; // Zeiger auf Menü
CPoint m_pPoint; // Kopie der Mausposition

// Mausposition in lokale Variable kopieren
m_pPoint = point;
// Position in Bildschirmkoordinaten konvertieren
ClientToScreen(&m_pPoint);
// Zeiger auf Fenstermenü holen
m_lMenu = GetMenu();

```

```
// Zeiger auf zweites Untermenü holen
m_lMenu = m_lMenu->GetSubMenu(1);
// Popup-Menü anzeigen
m_lMenu->TrackPopupMenu(TPM_CENTERALIGN + TPM_LEFTBUTTON,
m_pPoint.x, m_pPoint.y, this, NULL);

////////////////////
// EIGENER CODE, ENDE
////////////////////
}
```

Antworten für Tag 7

Antworten zu den Kontrollfragen

1. Übergeben Sie für das Argument `bUnderline` an die Funktion `CreateFont` den Wert 1.
2. Übergeben Sie für das Argument `nEscapement` an die Funktion `CreateFont` den Wert 1800.
3. Diese Funktion wird für jede im System verfügbare Schrift je einmal aufgerufen, außer wenn die Callback-Funktion den Wert 0 zurückgibt und die Auflistung der Schriften stoppt.

Lösungen zu den Übungen

1. Nehmen Sie das Kontrollkästchen in das Dialogfeld auf. Legen Sie für die Eigenschaft `ID` den Wert `IDC_CBUSETEXT` und als Titel `Text &` verwenden fest.

Weisen Sie dem Steuerelement mit dem Klassen-Assistenten eine Variable zu. Legen Sie den Variablentyp als `BOOL` und den Namen mit `m_bUseText` fest.

Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `BN_CLICKED` des Kontrollkästchens hinzu. In die Funktion schreiben Sie den folgenden Code:

```
void CTag7Dlg::OnCbusetext()
{
// TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen

////////////////////
// EIGENER CODE, ANFANG
////////////////////

// Variablen mit den Steuerelementen des Dialogfelds aktualisieren
UpdateData(TRUE);
// Schriftname für Schriftprobe verwenden?
if (!m_bUseText)
// Schriftname verwenden
m_strDisplayText = m_strFontName;
else
// Eingegebenen Text verwenden
m_strDisplayText = m_strSampText;

// Dialogfeld mit Variablen aktualisieren
UpdateData(FALSE);

////////////////////
```

```
// EIGENER CODE, ENDE
////////////////////
}
```

Um das Kontrollkästchen zu initialisieren, modifizieren Sie die Funktion `OnInitDialog` wie folgt:

```
BOOL CTag7Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    .
    .
    .
    // ZU ERLEDIGEN: Hier zusätzliche Initialisierung einfügen

    //////////////////
    // EIGENER CODE, ANFANG
    //////////////////

    // Listenfeld für Schriften füllen
    FillFontList();

    // Einzugebenden Text initialisieren
    m_strSampText = "Testen";
    // Text in Feld für Schriftbeispiel kopieren
    m_strDisplayText = m_strSampText;
    // Kontrollkästchen initialisieren
    m_bUseText = TRUE;
    // Dialogfeld aktualisieren
    UpdateData(FALSE);

    //////////////////
    // EIGENER CODE, ENDE
    //////////////////

    return TRUE; // Geben Sie TRUE zurück, außer ein Steuerelement soll den Fokus
erhalten
}
```

Modifizieren Sie die Funktion `OnSelchangeLFonts` wie folgt:

```
void CTag7Dlg::OnSelchangeLFonts()
{
    // TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen

    //////////////////
    // EIGENER CODE, ANFANG
    //////////////////

    // Variablen mit Steuerelementen des Dialogfelds aktualisieren
    UpdateData(TRUE);
    // Schriftname für Schriftprobe verwenden?
    if (!m_bUseText)
    {
        // Schriftname in Feld für Schriftbeispiel kopieren
        m_strDisplayText = m_strFontName;
        // Dialogfeld mit Variablen aktualisieren
    }
}
```

```
UpdateData(FALSE);
}
// Schrift für Beispiel festlegen
SetMyFont();
```

```
////////////////////
// EIGENER CODE, ENDE
////////////////////
}
```

Schließlich ändern Sie noch die Funktion `OnChangeEsamptext` folgendermaßen ab:

```
void CTag7Dlg::OnChangeEsamptext()
{
// TODO: Wenn dies ein RICHEDIT-Steuerelement ist, sendet das Steuerelement
diese
// Benachrichtigung nicht, bevor Sie nicht die Funktion
CDialog::OnInitDialog()
// überschreiben und CRichEditCtrl().SetEventMask() aufrufen, wobei
// eine ODER-Operation mit dem Attribut ENM_CHANGE und der Maske erfolgt.
```

```
// TODO: Fügen Sie hier Ihren Code für die
Benachrichtigungsbehandlungsroutine des Steuerelements hinzu
```

```
////////////////////
// EIGENER CODE, ANFANG
////////////////////
```

```
// Variablen mit Steuerelementen des Dialogfelds aktualisieren
```

```
UpdateData(TRUE);
// Text für Schriftprobe verwenden?
if (m_bUseText)
{
// Aktuellen Text in Schriftprobe kopieren
m_strDisplayText = m_strSampText;
// Dialogfeld mit Variablen aktualisieren
UpdateData(FALSE);
}
```

```
////////////////////
// EIGENER CODE, ENDE
////////////////////
}
```

2. Nehmen Sie das Kontrollkästchen in das Dialogfeld auf. Legen Sie für die Eigenschaft `ID` den Wert `IDC_CBITALIC` und als Titel `&Kursiv` fest.

Weisen Sie dem Steuerelement mit dem Klassen-Assistenten eine Variable zu. Legen Sie den Variablentyp als `BOOL` und den Namen mit `m_bItalic` fest.

Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `BN_CLICKED` des Kontrollkästchens hinzu. In die Funktion schreiben Sie den folgenden Code:

```
void CTag7Dlg::OnCbitalic()
{
// TODO: Code für die Behandlungsroutine der Steuerelement- Benachrichtigung
hier einfügen
```



```

////////////////////
// EIGENER CODE, ANFANG
////////////////////

```

```

// Variablen mit Steuerelementen des Dialogfelds aktualisieren
UpdateData(TRUE);
// Schrift für Beispiel festlegen
SetMyFont();

```

```

////////////////////
// EIGENER CODE, ENDE
////////////////////
}

```

Ändern Sie die Funktion SetMyFont entsprechend dem nachstehenden Listing ab:

```

void CTag7Dlg::SetMyFont()
{
    CRect rRect; // Rechteck des Anzeigebereichs
    int iHeight; // Höhe des Anzeigebereichs
    int iItalic = 0; // Schrift kursiv ausgeben?

    // Wurde eine Schrift ausgewählt?
    if (m_strFontName != "")
    {
        // Abmessungen des Feldes für Schriftbeispiel ermitteln
        m_ctlDisplayText.GetWindowRect(&rRect);
        // Höhe des Feldes berechnen
        iHeight = rRect.top - rRect.bottom;
        // Sicherstellen, daß Höhe positiv
        if (iHeight < 0)
            iHeight = 0 - iHeight;
        // Schrift kursiv ausgeben?
        if (m_bItalic)
            iItalic = 1;
        // Zu verwendende Schrift erzeugen
        m_fSampFont.CreateFont((iHeight - 5), 0, 0, 0,
            FW_NORMAL, iItalic, 0, 0, DEFAULT_CHARSET,
            OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS,
            DEFAULT_QUALITY, DEFAULT_PITCH |
            FF_DONTCARE, m_strFontName);

        // Schrift für Anzeigefeld der Schriftprobe festlegen
        m_ctlDisplayText.SetFont(&m_fSampFont);
    }
}

```

Antworten für Tag 8

Antworten zu den Kontrollfragen

1. Rot, Grün und Blau.
2. Mit dem Gerätekontext.
3. 8 mal 8 Pixel.

4. Die Nachricht `WM_PAINT`.

5. Rufen Sie die Funktion `Invalidate` auf diesem Fenster auf.

Lösungen zu den Übungen

1. Öffnen Sie das zweite Dialogfeld im Dialog-Editor. Öffnen Sie das Eigenschaftsdialogfeld für das Fenster. Gehen Sie auf die Registerkarte **Formate**. Wählen Sie im Dropdown-Listefeld **Rand** den Eintrag **Größe ändern**. Öffnen Sie den Klassen-Assistenten, und fügen Sie eine Behandlungsroutine für die Nachricht `WM_SIZE` hinzu. In der eben erzeugten Funktion rufen Sie die Funktion `Invalidate` auf, wie es Listing B.7 zeigt.

Listing B.7: Die Funktion `OnSize`

```
1: void CPaintDlg::OnSize(UINT nType, int cx, int cy)
2: {
3: CDialog::OnSize(nType, cx, cy);
4:
5: // TODO: Code für die Behandlungsroutine für Nachrichten hier einfügen
6: // Fenster neu zeichnen
7: Invalidate();
8: }
```

2. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**. Klicken Sie mit der rechten Maustaste auf den obersten Ordner des Ressourcenbaumes. Aus dem Kontextmenü wählen Sie den Befehl **Einfügen**. Im Dialogfeld **Ressource einfügen** markieren Sie in der Liste **Ressourcentyp** den Eintrag **Bitmap**. Zeichnen Sie ein Muster in das gerade erstellte Bitmap. Klicken Sie mit der rechten Maustaste auf die Bitmap-ID im Arbeitsbereich. Öffnen Sie den Eigenschaftsdialog, und ändern Sie die Objekt-ID in `IDB_BITMAPBRUSH`. Öffnen Sie den Quellcode für die Funktion `DrawRegion`. Fügen Sie in diese Funktion die Zeilen 22 bis 24 und 105 bis 112 von Listing B.8 ein. Vergrößern Sie die Anzahl der Schleifen in der `for`-Anweisung in Zeile 39.

Listing B.8: Die Funktion `DrawRegion`

```
1: void CPaintDlg::DrawRegion(CPaintDC *pdc, int iColor, int iTool, int
iShape)
2: {
3: // Stifte deklarieren und erzeugen
.
.
.
19: CBrush lVertBrush(HS_VERTICAL, m_crColors[iColor]);
20: CBrush lNullBrush(RGB(192, 192, 192));
21:
22: CBitmap lBitmap;
23: lBitmap.LoadBitmap(IDB_BITMAPBRUSH);
24: CBrush lBitmapBrush(&lBitmap);
25:
26: // Größe der Zeichenbereiche berechnen
27: CRect lRect;
28: GetClientRect(lRect);
.
.
.
37: int i;
38: // Schleife durch alle Pinsel und Stifte
39: for (i = 0; i < 8; i++)
40: {
41: switch (i)
```

```

42: {
.
.
.
103: pdc->SelectObject(&lVertBrush);
104: break;
105: case 7: // Null - Bitmap
106: // Position für diese Figur bestimmen.
107: lDrawRect.left = lDrawRect.left + liHorz;
108: lDrawRect.right = lDrawRect.left + liWidth;
109: // Passenden Stift und Pinsel auswählen
110: pdc->SelectObject(&lNullPen);
111: pdc->SelectObject(&lBitmapBrush);
112: break;
113: }
114: // Welches Werkzeug wird verwendet?
.
.
.
126: pdc->SelectObject(lOldBrush);
127: pdc->SelectObject(lOldPen);
128:}

```

Antworten für Tag 9

Antworten zu den Kontrollfragen

1. Über die `IDispatch`-Schnittstelle kann der Container die Methode `Invoke` aufrufen. An diese Methode übergibt er die `DISPID` derjenigen Steuerelementmethode, die der Container aufrufen möchte.
2. Die Containeranwendung hat ihre eigene `IDispatch`-Schnittstelle, über die das Steuerelement Ereignisse auslösen kann.
3. Schalten Sie im zweiten Dialogfeld des MFC-Anwendungs-Assistenten das Kontrollkästchen **ActiveX-Steuerelemente** ein.
4. Visual C++ erzeugt C++-Klassen, die die Funktionalität des Steuerelements verkapseln.
5. Ältere Steuerelemente liefern möglicherweise nicht die Informationen, die Visual C++ braucht, um C++-Klassen zu generieren und damit die Funktionalität des Steuerelements zu verkapseln.

Lösung zur Übung

Fügen Sie mit dem Klassen-Assistenten eine Funktion für die Nachricht `DbLClick` des Grid-Steuerelements (`IDC_MSFGGRID`) hinzu. In die Funktion schreiben Sie den folgenden Code:

```

void CActiveXDlg::OnDbLClickMsfgrid()
{
// TODO: Code für die Behandlungsroutine der Steuerelement-Benachrichtigung
hier einfügen

//////////////////////////
// EIGENER CODE, ANFANG
//////////////////////////

// Hat Benutzer auf eine Datenzeile und nicht die

```

```
// Kopfzeile geklickt?
if (m_ctlFGrid.GetMouseRow() != 0)
{
// Wenn ja, dann Spaltenvariable auf 0 setzen
// und beenden
m_iMouseCol = 0;
return;
}
// Angeklickte Spalte sichern
m_iMouseCol = m_ctlFGrid.GetMouseCol();
// Wenn es die erste Spalte war, dann
// ist nichts zu tun.
if (m_iMouseCol == 0)
return;
// Neuzeichnen des Steuerelements abschalten
m_ctlFGrid.SetRedraw(FALSE);
// Gewählte Spaltenposition ändern
m_ctlFGrid.SetColPosition(m_iMouseCol, 0);
// Tabelle neu sortieren
DoSort();
// Neuzeichnen wieder einschalten
m_ctlFGrid.SetRedraw(TRUE);

////////////////////
// EIGENER CODE, ENDE
////////////////////
```

Antworten für Tag 10

Antworten zu den Kontrollfragen

1. Single Document Interface - Einzeldokumentschnittstelle.
2. Die Ansichtsklasse ist für die Anzeige des Dokuments verantwortlich.
3. Um das Dokument neu zu zeichnen, wird die Funktion **OnDraw** in der Ansichtsklasse aufgerufen.
4. Um das aktuelle Dokument zu löschen, schreibt man den entsprechenden Code in die Funktion **DeleteContents** der Dokumentklasse.
5. Die Dokumentklasse verwaltet und manipuliert die Daten. Sie verwaltet die abstrakte Repräsentation des zu bearbeitenden Dokuments.

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Markieren Sie im Arbeitsbereich auf der Registerkarte **Klassen** die Klasse **CLine**. Klicken Sie mit der rechten Maustaste, und wählen Sie aus dem Kontextmenü den Befehl **Member-Variable hinzufügen**.
2. Legen Sie den Variablentyp mit **UINT**, die Deklaration mit **m_nWidth** und den Zugriffsstatus mit **Privat** fest. Klicken Sie auf **OK**, um die Variable hinzuzufügen.
3. Klicken Sie mit der rechten Maustaste im Baum der Registerkarte **Klassen** auf den Konstruktor von **CLine**. Wählen Sie aus dem Kontextmenü den Befehl **Gehe zu Deklaration**.
4. Fügen Sie **UINT nWidth** als viertes Argument in die Konstruktordeklaration ein.

5. Klicken Sie mit der rechten Maustaste im Baum der Registerkarte **Klassen** auf den Konstruktor von **CLine**. Wählen Sie aus dem Kontextmenü den Befehl **Gehe zu Definition**.

6. Fügen Sie im Konstruktor das vierte Argument hinzu, und setzen Sie das Element `m_nWidth` auf das neue Argument, wie es Listing B.9 zeigt.

Listing B.9: Der modifizierte Konstruktor von CLine

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo, COLORREF crColor, UINT nWidth)
2: {
3: // Anfangs- und Endpunkte initialisieren
4: m_ptFrom = ptFrom;
5: m_ptTo = ptTo;
6: m_crColor = crColor;
7: m_nWidth = nWidth;
8: }
```

7. Gehen Sie nach unten zur Funktion `Draw`, und nehmen Sie die Änderungen gemäß Listing B.10 vor.

Listing B.10: Die modifizierte Funktion Draw

```
1: void CLine::Draw(CDC * pDC)
2: {
3: // Stift erzeugen
4: CPen lpen (PS_SOLID, m_nWidth, m_crColor);
5:
6: // Neuen Stift als Zeichenobjekt festlegen
7: CPen* pOldPen = pDC->SelectObject(&lpen);
8: // Linie zeichnen
9: pDC->MoveTo(m_ptFrom);
10: pDC->LineTo(m_ptTo);
11: // Vorherigen Stift zurücksetzen
12: pDC->SelectObject(pOldPen);
13: }
```

8. Gehen Sie nach unten zur Funktion `Serialize`, und modifizieren Sie die Funktion gemäß Listing B.11.

Listing B.11: Die modifizierte Funktion Serialize

```
1: void CLine::Serialize(CArchive &ar)
2: {
3: CObject::Serialize(ar);
4:
5: if (ar.IsStoring())
6: ar << m_ptFrom << m_ptTo << (DWORD) m_crColor << m_nWidth;
7: else
8: ar >> m_ptFrom >> m_ptTo >> (DWORD) m_crColor >> m_nWidth;
9: }
```

9. Markieren Sie im Arbeitsbereich auf der Registerkarte **Klassen** die Klasse **CTag10Doc**. Klicken Sie mit der rechten Maustaste, und wählen Sie aus dem Kontextmenü den Befehl **Member-Variable hinzufügen**.

10. Legen Sie den Variablentyp mit `UINT`, die Deklaration mit `m_nWidth` und den Zugriffsstatus als **Privat** fest. Klicken Sie auf **OK**, um die Variable hinzuzufügen.

11. Öffnen Sie den Quellcode von `CTag10Doc (Tag10.Doc.cpp)`. Gehen Sie nach unten zur Funktion `OnNewDocument`, und bearbeiten Sie die Funktion entsprechend Listing B.12.

Listing B.12: Die modifizierte Funktion OnNewDocument

```
1: BOOL CTag10Doc::OnNewDocument()
```

```

2: {
3: if (!CDocument::OnNewDocument())
4: return FALSE;
5:
6: // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
7: // (SDI-Dokumente verwenden dieses Dokument)
8:
9: ///////////////////////////////////
10: // EIGENER CODE, ANFANG
11: ///////////////////////////////////
12:
13: // Farbe mit Schwarz initialisieren
14: m_nColor = ID_COLOR_BLACK - ID_COLOR_BLACK;
15: // Breite mit Dünn initialisieren
16: m_nWidth = ID_WIDTH_VTHIN - ID_WIDTH_VTHIN;
17:
18: ///////////////////////////////////
19: // EIGENER CODE, ENDE
20: ///////////////////////////////////
21:
22: return TRUE;
23: }

```

12. Gehen Sie nach unten zur Funktion `AddLine`, und nehmen Sie die Änderungen gemäß Listing B.13 vor.

Listing B.13: Die modifizierte Funktion `AddLine`

```

1: CLine * CTag10Doc::AddLine(CPoint ptFrom, CPoint ptTo)
2: {
3: static UINT nWidths[5] = { 1, 8, 16, 24, 32};
4:
5: // Ein neues CLine-Objekt erzeugen
6: CLine *pLine = new CLine(ptFrom, ptTo, m_crColors[m_nColor],
nWidths[m_nWidth]);
7: try
8: {
9: // Die neue Linie in das Objektarray einfügen
10: m_oaLines.Add(pLine);
11: // Dokument als bearbeitet markieren
12: SetModifiedFlag();
13: }
14: // Ist Speicherausnahme aufgetreten?
15: catch (CMemoryException* perr)
16: {
17: // Meldung für Benutzer, schlechte Neuigkeiten
18: // mitteilen
19: AfxMessageBox("Speichermangel", MB_ICONSTOP | MB_OK);
20: // Wurde Linienobjekt erzeugt?
21: if (pLine)
22: {
23: // Objekt löschen
24: delete pLine;
25: pLine = NULL;
26: }
27: // Ausnahmeobjekt löschen
28: perr->Delete();

```



```

29: }
30: return pLine;
31: }

```

13. Fügen Sie eine neue Member-Funktion in die Klasse CTag10Doc ein. Legen Sie den Funktionstyp als **UINT**, die Deklaration als **GetWidth** und den Zugriffsstatus als **Public** fest.

14. Fügen Sie den Code aus Listing B.14 in die Funktion **GetWidth** ein.

Listing B.14: Die Funktion GetWidth

```

1: UINT CTag10Doc::GetWidth()
2: {
3: // Aktuelle Breite zurückgeben
4: return ID_WIDTH_VTHIN + m_nWidth;
5: }

```

15. Gehen Sie im Arbeitsbereich auf die Registerkarte **Ressourcen**. Erweitern Sie den Baum, so daß Sie den Inhalt des Ordners **Menu** sehen. Doppelklicken Sie auf die Menüressource.

16. Ziehen Sie den leeren Menübefehl der obersten Ebene (am rechten Ende der Menüleiste) nach links, und legen Sie ihn vor dem Menü **Ansicht** ab.

17. Öffnen Sie den Eigenschaftsdialog für den leeren Menübefehl. Legen Sie den Titel mit **Breite** fest. Schließen Sie das Eigenschaftsdialogfeld.

18. Fügen Sie die Menübefehle für das Menü **Breite** in der Reihenfolge und mit den Eigenschaften gemäß Tabelle B.1 hinzu.

Tabelle B.1: Einstellungen der Menüeigenschaften

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	ID_WIDTH_VTHIN
	Titel	Sehr d&ünn
Menübefehl	ID	ID_WIDTH_THIN
	Titel	&Dünn
Menübefehl	ID	ID_WIDTH_MEDIUM
	Titel	&Mittel
Menübefehl	ID	ID_WIDTH_THICK
	Titel	D&ick
Menübefehl	ID	ID_WIDTH_VTHICK
	Titel	&Sehr dick

19. Öffnen Sie den Klassen-Assistenten. Wählen Sie CTag10Doc im Kombinationsfeld **Klassenname**.

20. Fügen Sie Funktionen für die Nachrichten **COMMAND** und **UPDATE_COMMAND_UI** für alle Menübefehle des Menüs **Breite** hinzu.

21. Nachdem Sie den letzten Menübefehl hinzugefügt haben, klicken Sie auf die Schaltfläche **Code bearbeiten**.

22. Bearbeiten Sie die Funktionen des Menübefehls **Sehr dünn** entsprechend Listing B.15.

Listing B.15: Die Funktionen des Menübefehls Sehr dünn

```

1: void CTag10Doc::OnWidthVthin()

```

```

2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Aktuelle Breite auf Sehr dünn setzen
10: m_nWidth = ID_WIDTH_VTHIN - ID_WIDTH_VTHIN;
11:
12: //////////////////////////////////
13: // EIGENER CODE, ENDE
14: //////////////////////////////////
15: }
16:
17: void CTag10Doc::OnUpdateWidthVthin(CCmdUI* pCmdUI)
18: {
19: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
Benutzeroberfläche hier einfügen
20:
21: //////////////////////////////////
22: // EIGENER CODE, ANFANG
23: //////////////////////////////////
24:
25: // Prüfen, ob Menübefehl Sehr dünn mit Kontrollhäkchen zu versehen ist
26: pCmdUI->SetCheck(GetWidth() == ID_WIDTH_VTHIN ? 1 : 0);
27:
28: //////////////////////////////////
29: // EIGENER CODE, ENDE
30: //////////////////////////////////
31: }

```

23. Bearbeiten Sie die Funktionen des Menübefehls **Dünn** entsprechend Listing B.16. Gehen Sie bei den restlichen Menüfunktionen in der gleichen Weise vor, wobei Sie die jeweiligen Menü-IDs anstelle von `ID_WIDTH_THIN` einsetzen.

Listing B.16: Die Funktionen des Menüs Dünn

```

1: void CTag10Doc::OnWidthThin()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: //////////////////////////////////
6: // EIGENER CODE, ANFANG
7: //////////////////////////////////
8:
9: // Aktuelle Breite auf Dünn setzen
10: m_nWidth = ID_WIDTH_THIN - ID_WIDTH_VTHIN;
11:
12: //////////////////////////////////
13: // EIGENER CODE, ENDE
14: //////////////////////////////////
15: }
16:
17: void CTag10Doc::OnUpdateWidthThin(CCmdUI* pCmdUI)
18: {

```

```

19: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
Benutzeroberfläche hier einfügen
20:
21: //////////////////////////////////////
22: // EIGENER CODE, ANFANG
23: //////////////////////////////////////
24:
25: // Prüfen, ob Menübefehl Dünn mit Kontrollhäkchen zu versehen ist
26: pCmdUI->SetCheck(GetWidth() == ID_WIDTH_THIN ? 1 : 0);
27:
28: //////////////////////////////////////
29: // MY CODE ENDS HERE
30: //////////////////////////////////////
31: }

```

Antworten für Tag 11

Antworten zu den Kontrollfragen

1. Die fünf Basisklassen in MDI-Anwendungen sind die von `CWinApp` abgeleitete Klasse, die von `CMDIFrameWnd` abgeleitete Klasse, die von `CMDIChildWnd` abgeleitete Klasse, die von `CDocument` abgeleitete Klasse und die von `CView` abgeleitete Klasse.
2. Der Klassen-Assistent versteht den Eintrag `ON_COMMAND_RANGE` in der Nachrichtenzuordnungstabelle nicht und wird ihn demzufolge entweder entfernen oder verfälschen.
3. Die ID der Nachricht.
4. `WM_CONTEXTMENU`.

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Fügen Sie den Behandlungscode für die Breite wie in der gestrigen Übung hinzu.
2. Fügen Sie die Menübefehle für das Menü **Breite** mit den gleichen Einstellungen wie gestern hinzu.
3. Öffnen Sie die Header-Datei `Tag11Doc.h`.
4. Gehen Sie in der Header-Datei nach unten bis zum geschützten Abschnitt, in dem die Nachrichtenzuordnungstabelle `AFX_MSG` deklariert ist (suchen Sie nach `// {{AFX_MSG(CTag11Doc)}}`).
5. Fügen Sie die Funktionsdeklarationen aus Listing B.17 vor der gesuchten Zeile ein. (Der gesuchte String markiert den Beginn der vom Klassen-Assistenten verwalteten Nachrichtenzuordnungstabelle. Alles zwischen dieser Markierung und der Endemarkierung `//}}AFX_MSG` wird vom Klassen-Assistenten möglicherweise entfernt oder verfälscht.)

Listing B.17: Die Deklarationen der Behandlungsroutinen in `Tag11Doc.h`

```

.
.
.
.
1: #ifdef _DEBUG
2: virtual void AssertValid() const;
3: virtual void Dump(CDumpContext& dc) const;
4: #endif
5:

```

```

6: protected:
7:
8: // Generierte Message-Map-Funktionen
9: protected:
10: afx_msg void OnColorCommand(UINT nID);
11: afx_msg void OnWidthCommand(UINT nID);
12: afx_msg void OnUpdateColorUI(CCmdUI* pCmdUI);
13: afx_msg void OnUpdateWidthUI(CCmdUI* pCmdUI);
14://{{AFX_MSG(CTag11Doc)
15: ...
16://}}AFX_MSG
17: DECLARE_MESSAGE_MAP()
18: private:
19: UINT m_nColor;
20: UINT m_nWidth;
21: CObArray m_oaLines;
22: };

```

6. Öffnen Sie die Quellcodedatei Tag11Doc.cpp.

7. Suchen Sie nach der Zeile BEGIN_MESSAGE_MAP, und fügen Sie unmittelbar danach die Zeilen aus Listing B.18 ein. Es ist wichtig, daß dieser Code zwischen der Zeile BEGIN_MESSAGE_MAP und der Zeile //{{AFX_MSG_MAP steht. Wenn Sie diese Befehle zwischen die Zeilen //{{AFX_MSG_MAP und //}}AFX_MSG_MAP schreiben, entfernt oder verstümmelt sie der Klassen-Assistent.

Listing B.18: Die Einträge der Behandlungsroutinen in der Nachrichtenzuordnungstabelle in Tag11Doc.cpp

```

1: //////////////////////////////////////
2: // CTag11Doc
3:
4: IMPLEMENT_DYNCREATE(CTag11Doc, CDocument)
5:
6: BEGIN_MESSAGE_MAP(CTag11Doc, CDocument)
7: ON_COMMAND_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE, OnColorCommand)
8: ON_COMMAND_RANGE(ID_WIDTH_VTHIN, ID_WIDTH_VTHICK, OnWidthCommand)
9: ON_UPDATE_COMMAND_UI_RANGE(ID_COLOR_BLACK, ID_COLOR_WHITE,
OnUpdateColorUI)
10: ON_UPDATE_COMMAND_UI_RANGE(ID_WIDTH_VTHIN, ID_WIDTH_VTHICK,
OnUpdateWidthUI)
11://{{AFX_MSG_MAP(CTag11Doc)
12: ...
13://}}AFX_MSG_MAP
14: END_MESSAGE_MAP()
15:
16: const COLORREF CTag11Doc::m_crColors[8] = {
17: RGB( 0, 0, 0), // Schwarz
18: RGB( 0, 0, 255), // Blau
.
.
.

```

8. Gehen Sie ans Ende der Datei, und fügen Sie die beiden Behandlungsroutinen aus Listing B.19 ein.

Listing B.19: Die Behandlungsroutinen für die Nachrichten des Menüs Breite

```

1: void CTag11Doc::OnWidthCommand(UINT nID)
2: {
3: // Aktuelle Breite setzen

```

```

4: m_nWidth = nID - ID_WIDTH_VTHIN;
5: }
6:
7: void CTag11Doc::OnUpdateWidthUI(CCmdUI* pCmdUI)
8: {
9: // Prüfen, ob Menübefehl mit Kontrollhäkchen zu versehen ist
10: pCmdUI->SetCheck(GetWidth() == pCmdUI->m_nID ? 1 : 0);
11: }

```

9. Öffnen Sie IDR_CONTEXTMENU im Menü-Editor.

10. Fügen Sie in das überlappende Menü **Breite** die Menübefehle genau wie beim Menü IDR_TAG11TYPE und mit den gleichen Eigenschaftseinstellungen hinzu. Die ID können Sie aus der Dropdown-Liste der IDs auswählen, wenn Sie lieber danach suchen als sie einzutippen.

Antworten für Tag 12

Antworten zu den Kontrollfragen

1. Vergeben Sie für die Schaltfläche der Symbolleiste dieselbe Objekt-ID wie für den Menübefehl.
2. Bei beiden muß das Andocken auf den gleichen Seiten in der Funktion `OnCreate` der Rahmenklasse aktiviert sein (mit Hilfe der Funktion `EnableDocking`).
3. Löschen Sie den Eintrag `ID_INDICATOR_NUM` aus der Tabelle für die Statusleistenanzeige am Anfang der Quellcodedatei des Hauptrahmens.
4. In die Symbolleiste ist eine Trennlinie als Platzhalter einzufügen. Der Symbolleisten-Editor hindert Sie in gut gemeinter Absicht daran, Trennlinien einzufügen, weil er hier einen Fehler vermutet.

Lösungen zu den Übungen

1. Fügen Sie einen Eintrag in die Zeichenfolgentabelle mit der ID `ID_INDICATOR_WIDTH` und dem Titel **SEHR DICK** ein.

Nehmen Sie einen weiteren Eintrag in die Tabelle der Statusleistenanzeige am Anfang der Datei `CMainFrame.cpp` auf:

```

static UINT indicators[] =
{
ID_SEPARATOR, // Statusleistenanzeige
ID_INDICATOR_WIDTH,
ID_INDICATOR_COLOR,
ID_INDICATOR_CAPS,
ID_INDICATOR_NUM,
ID_INDICATOR_SCRL,
};

```

Fügen Sie der Klasse `CToolBarDoc` eine neue Member-Funktion hinzu. Legen Sie den Funktionstyp als `afx_msg void`, die Funktionsdeklaration als `OnUpdateIndicatorWidth (CCmdUI *pCmdUI)` und den Zugriffsstatus als **Protected** fest. In die Funktion übernehmen Sie den folgenden Code:

```

void CToolBarDoc::OnUpdateIndicatorWidth(CCmdUI *pCmdUI)
{
CString strWidth;

// Aktuelle Breite?
switch (m_nWidth)

```

```

{
case 0: // Sehr dünn
strWidth = "SEHR DÜNN";
break;
case 1: // Dünn
strWidth = "DÜNN";
break;
case 2: // Mittel
strWidth = "MITTEL";
break;
case 3: // Dick
strWidth = "DICK";
break;
case 4: // Sehr dick
strWidth = "SEHR DICK";
break;
}
// Statusleistenausschnitt aktivieren
pCmdUI->Enable(TRUE);
// Test des Statusleistenausschnitts auf aktuelle
// Breite festlegen
pCmdUI->SetText(strWidth);
}

```

Fügen Sie der Nachrichtenzuordnungstabelle von `CToolBarDoc` einen Eintrag für die Behandlungsroutine der Nachricht `ON_UPDATE_COMMAND_UI` wie folgt hinzu:

```

////////////////////////////////////
// CToolBarDoc

```

```

IMPLEMENT_DYNCREATE(CToolBarDoc, CDocument)

BEGIN_MESSAGE_MAP(CToolBarDoc, CDocument)
ON_UPDATE_COMMAND_UI(ID_INDICATOR_WIDTH, OnUpdateIndicatorWidth)
ON_UPDATE_COMMAND_UI(ID_INDICATOR_COLOR, OnUpdateIndicatorColor)
//{{AFX_MSG_MAP(CToolBarDoc)
ON_COMMAND(ID_COLOR_BLACK, OnColorBlack)
ON_UPDATE_COMMAND_UI(ID_COLOR_BLACK, OnUpdateColorBlack)
...
ON_COMMAND(ID_WIDTH_VTHIN, OnWidthVthin)
ON_UPDATE_COMMAND_UI(ID_WIDTH_VTHIN, OnUpdateWidthVthin)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

```

2. Öffnen Sie die Symbolleiste `IDR_MAINFRAME` im Symbolleisten-Editor. Zeichnen Sie ein Symbol für die leere Schaltfläche am Ende der Symbolleiste. Doppelklicken Sie auf die Schaltfläche, um das zugehörige Eigenschaftsdialogfeld zu öffnen. Legen Sie die ID mit `ID_VIEW_COLORBAR` fest, und geben Sie einen passenden Statuszeilentext ein. Kompilieren Sie die Anwendung erneut, und führen Sie sie aus. Auf der Hauptsymbolleiste sollte nun die Umschaltung der Farbsymbolleiste funktionieren.

Antworten für Tag 13

Antworten zu den Kontrollfragen

1. DECLARE_SERIAL und IMPLEMENT_SERIAL.
2. Rufen Sie dazu die Funktion IsStoring oder IsLoading auf.
3. Den Klassennamen, den Namen der Basisklasse und die Versionsnummer.
4. CFormView.
5. CFile.

Lösung zur Übung

Im Dialog-Editor fügen Sie zwei Optionsfelder und das Textfeld mit der Aufforderung hinzu. Legen Sie die Eigenschaften der Steuerelemente gemäß Tabelle B.2 fest.

Tabelle B.2: Einstellungen der Steuerelementeigenschaften

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Geschlecht:
Optionsfeld	ID	IDC_RMALE
	Titel	&männlich
	Gruppe	eingeschaltet
Optionsfeld	ID	IDC_RFEMALE
	Titel	wie&blich

Verbinden Sie mit den neuen Optionsfeldern eine Variable laut Tabelle B.3.

Tabelle B.3: Variable der Steuerelemente

Objekt	Name	Kategorie	Typ
IDC_RMALE	m_iSex	Wert	int

Inkrementieren Sie die Versionsnummer im Makro IMPLEMENT_SERIAL in der Klasse CPerson. Fügen Sie eine neue Member-Variable in die Klasse CPerson ein. Legen Sie den Typ als `int`, den Namen als `m_iSex` und den Zugriffsstatus als **Privat** fest. Aktualisieren Sie die Konstruktorfunktion von CPerson, indem Sie die Variable `m_iSex` in die Initialisierungen wie in Zeile 8 von Listing B.20 einfügen.

Listing B.20: Der modifizierte Konstruktor von CPerson

```

1: CPerson::CPerson()
2: {
3: // Klassenvariablen initialisieren
4: m_iMaritalStatus = 0;
5: m_iAge = 0;
6: m_bEmployed = FALSE;
7: m_sName = "";
8: m_iSex = 0;
9: }
```

Fügen Sie entsprechend der Zeilen 10 und 16 von Listing B.21 die Inline-Funktionen in die Klassendeklaration von CPerson ein, um den Wert der neuen Variablen zu setzen und abzurufen.

Listing B.21: Die modifizierte Deklaration der Klasse CPerson

```

1: class CPerson : public CObject
2: {
3: DECLARE_SERIAL (CPerson)
4: public:
5: virtual void Serialize(CArchive &ar);
6: // Funktionen zum Setzen der Variablen
7: void SetEmployed(BOOL bEmployed) { m_bEmployed = bEmployed;}
8: void SetMaritalStat(int iStat) { m_iMaritalStatus = iStat;}
9: void SetAge(int iAge) { m_iAge = iAge;}
10: void SetSex(int iSex) { m_iSex = iSex; }
11: void SetName(CString sName) { m_sName = sName;}
12: // Funktionen zum Holen der aktuellen Variableneinstellungen
13: BOOL GetEmployed() { return m_bEmployed;}
14: int GetMaritalStatus() { return m_iMaritalStatus;}
15: int GetAge() {return m_iAge;}
16: int GetSex() {return m_iSex;}
17: CString GetName() {return m_sName;}
18: CPerson();
19: virtual ~CPerson();
20:
21: private:
22: int m_iSex;
23: int m_iMaritalStatus;
24: CString m_sName;
25: int m_iAge;
26: BOOL m_bEmployed;
27: };

```

Aktualisieren Sie die Funktion `Serialize` in der Klasse `CPerson`, um die Variable `m_iSex` entsprechend der Zeilen 9 und 12 von Listing B.22 einzubinden.

Listing B.22: Die modifizierte Funktion `Serialize` der Klasse `CPerson`

```

1: void CPerson::Serialize(CArchive &ar)
2: {
3: // Funktion der Basisklasse aufrufen
4: CObject::Serialize(ar);
5:
6: // Wird geschrieben?
7: if (ar.IsStoring())
8: // Alle Variablen in der richtigen Reihenfolge schreiben
9: ar << m_sName << m_iAge << m_iMaritalStatus << m_bEmployed << m_iSex;
10: else
11: // Alle Variablen in der richtigen Reihenfolge lesen
12: ar >> m_sName >> m_iAge >> m_iMaritalStatus >> m_bEmployed >> m_iSex;
13:
14: }

```

Modifizieren Sie die Funktion `PopulateView` im Ansichtsobjekt, um die Variable `m_iSex` in den Datenaustausch aufzunehmen, wie es Zeile 19 von Listing B.23 zeigt.

Listing B.23: Die modifizierte Funktion `PopulateView` der Klasse `CSerializeView`

```

1: void CSerializeView::PopulateView()
2: {
3: // Einen Zeiger auf das aktuelle Dokument holen
4: CSerializeDoc* pDoc = GetDocument();
5: if (pDoc)
6: {
7: // Aktuelle Datensatzposition in der Menge anzeigen
8: m_sPosition.Format("Datensatz %d von %d", pDoc->GetCurRecordNbr(),
9: pDoc->GetTotalRecords());
10: }
11: // Ist Datensatzobjekt gültig?
12: if (m_pCurPerson)
13: {
14: // Ja, alle Werte des Datensatzes holen
15: m_bEmployed = m_pCurPerson->GetEmployed();
16: m_iAge = m_pCurPerson->GetAge();
17: m_sName = m_pCurPerson->GetName();
18: m_iMaritalStatus = m_pCurPerson->GetMaritalStatus();
19: m_iSex = m_pCurPerson->GetSex();
20: }
21: // Anzeige aktualisieren
22: UpdateData(FALSE);
23: }

```

Fügen Sie eine Behandlungsroutine für das Klickereignis der beiden neuen Optionsfelder hinzu. Nutzen Sie für beide Behandlungsroutinen dieselbe Funktion. Aktualisieren Sie das Feld des Datensatzobjekts mittels der Set-Funktion wie in Listing B.24.

Listing B.24: Die Funktion OnSex der Klasse CSerializeView

```

1: void CSerializeView::OnSex()
2: {
3: // TODO: Code für die Behandlungsroutine der Steuerelement-
4: Benachrichtigung hier einfügen
5: // Daten im Formular mit Variablen synchronisieren
6: UpdateData(TRUE);
7: // Wenn gültiges Person-Objekt vorhanden, Daten an das Objekt übergeben
8: if (m_pCurPerson)
9: m_pCurPerson->SetSex(m_iSex);
10: // Zeiger auf das Dokument holen
11: CSerialDoc * pDoc = GetDocument();
12: if (pDoc)
13: // Veränderungsflag im Dokument setzen
14: pDoc->SetModifiedFlag();
15: }

```

Antworten für Tag 14

Antworten zu den Kontrollfragen

1. Open Database Connectivity, d.h. eine offene, herstellerunabhängige Schnittstelle für den Zugriff auf Datenbanken.
2. Move, MoveNext, MovePrev, MoveFirst, MoveLast und SetAbsolutePosition.

3. CRecordView.

4. AddNew, Update und Requery.

5. Edit

Lösung zur Übung

1. Erstellen Sie ein neues Dialogfeld mit dem in Abbildung B.1 gezeigten Layout. Konfigurieren Sie die Steuerelemente nach Tabelle B.4.

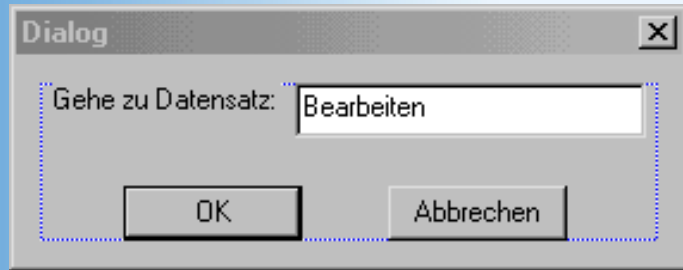


Abbildung B.1:
Das Dialogfeld Gehe zu

Tabelle B.4: Einstellungen der Dialogfeldeigenschaften

Objekt	Eigenschaft	Einstellung
Text	ID	IDC_STATIC
	Titel	Gehe zu Datensatz:
Eingabefeld	ID	IDC_EREENBR

2. Öffnen Sie den Klassen-Assistenten. Erzeugen Sie eine neue Klasse für das neue Dialogfeld. Nennen Sie die neue Klasse CMoveToDlg. Fügen Sie dann eine Variable für das Eingabefeld hinzu. Legen Sie den Variablentyp als long und den Namen als m_lRowNbr fest.

3. Fügen Sie einen weiteren Menübefehl in das Hauptmenü der Anwendung ein. Legen Sie die Menüeigenschaften gemäß Tabelle B.5 fest.

Tabelle B.5: Eigenschaften des neuen Menübefehls

Objekt	Eigenschaft	Einstellung
Menübefehl	ID	IDM_RECORD_MOVE
	Titel	&Gehe zu ...
	Statuszeilentext	Geht zu einem bestimmten Datensatz\nGehe zu

4. Öffnen Sie den Klassen-Assistenten, und fügen Sie eine Behandlungsfunktion für die Nachricht COMMAND des neuen Menübefehls in die Ansichtsklasse ein. In die Funktion übernehmen Sie den Code aus Listing B.25.

Listing B.25: Die Funktion OnRecordMove der Klasse CDbOdbcView

```

1: void CDbOdbcView::OnRecordMove()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4: // Instanz des Dialogfelds 'Gehe zu' erzeugen
5: CMoveToDlg dlgMoveTo;
```

```

6: // Zeilennummer des Ziels ermitteln
7: if (dlgMoveTo.DoModal() == IDOK)
8: {
9: // Zeiger auf den Recordset holen
10: CRecordset* pSet = OnGetRecordset();
11: // Sicherstellen, daß alle Änderungen am aktuellen Datensatz gespeichert
    wurden.
12: if (pSet->CanUpdate() && !pSet->IsDeleted())
13: {
14: pSet->Edit();
15: if (!UpdateData())
16: return;
17:
18: pSet->Update();
19: }
20: // Neue Position einstellen
21: pSet->SetAbsolutePosition(dlgMoveTo.m_lRowNbr);
22: // Formular aktualisieren
23: UpdateData(FALSE);
24: }
25: }

```

5. Binden Sie die Header-Datei für das neue Dialogfeld in den Quellcode der Ansichtsklasse ein, wie es aus Zeile 10 von Listing B.26 hervorgeht.

Listing B.26: Die Include-Anweisungen der Klasse CDbOdbcView

```

1: // DbOdbcView.cpp : Implementierung der Klasse CDbOdbcView
2: //
3:
4: #include "stdafx.h"
5: #include "DbOdbc.h"
6:
7: #include "DbOdbcSet.h"
8: #include "DbOdbcDoc.h"
9: #include "DbOdbcView.h"
10: #include "MoveToDlg.h"

```

6. Fügen Sie eine Symbolleistenschaltfläche für den neuen Menübefehl ein.

Antworten für Tag 15

Antworten zu den Kontrollfragen

1. ActiveX Data Objects.
2. OLE DB.
3. Connection, Command, Parameter, Error, Recordset und Field.
4. `::CoInitialize(NULL);`
5. `pCmd->ActiveConnection = pConn;`
6. Die erste Möglichkeit:
`_RecordsetPtr pRs;`
`pRs = pCmd->Execute();`

Die zweite:

```
_RecordsetPtr pRs;
pRs.CreateInstance(__uuidof(Recordset));
pRs->PutRefSource(pCmd);
```

Lösung zur Übung

Fügen Sie der Dokumentklasse Behandlungsfunktionen für die Nachricht `UPDATE_COMMAND_UI` der Befehle des Navigationsmenüs hinzu. In die Funktionen für die Menübefehle **Erster** und **Vorheriger** schreiben Sie den Code aus Listing B.27, für die Menübefehle **Letzter** und **Nächster** den Code aus Listing B.28.

Listing B.27: Die Funktion `OnUpdateDataFirst` der Klasse `CDbAdoDoc`

```
1: void CDbAdoDoc::OnUpdateDataFirst(CCmdUI* pCmdUI)
2: {
3: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
  Benutzeroberfläche hier einfügen
4: // Existiert der Datensatz?
5: if (m_pRs)
6: {
7: // Am Anfang der Datei (BOF) angelangt?
8: if (m_pRs->BOF)
9: pCmdUI->Enable(FALSE);
10: else
11: pCmdUI->Enable(TRUE);
12: }
13: }
```

Listing B.28: Die Funktion `OnUpdateDataLast` der Klasse `CDbAdoDoc`

```
1: void CDbAdoDoc::OnUpdateDataLast(CCmdUI* pCmdUI)
2: {
3: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
  Benutzeroberfläche hier einfügen
4: // Existiert der Datensatz?
5: if (m_pRs)
6: {
7: // Am Ende der Datei (EOF) angelangt?
8: if (m_pRs->EndOfFile)
9: pCmdUI->Enable(FALSE);
10: else
11: pCmdUI->Enable(TRUE);
12: }
13: }
```

Antworten für Tag 16

Antworten zu den Kontrollfragen

1. Wenn man eine neue Klasse erstellen muß, die von einer existierenden MFC-Klasse abgeleitet ist.
2. Alle Anwendungen sind erneut zu linken.
3. MFC, Allgemein und Formular.
4. Die LIB-Bibliotheksdatei und die Header-Dateien für die Objekte im Modul.

5. Kapselung und Vererbung. Das dritte Prinzip ist der Polymorphismus, der in der heutigen Lektion nicht behandelt wurde.

Lösung zur Übung

1. Erstellen Sie ein neues Projekt. Wählen Sie dazu im Dialogfeld **Neu** auf der Registerkarte **Projekte** den Eintrag **Win32-Bibliothek (statische)**. Tragen Sie einen Projektnamen wie etwa **Line** ein.
2. Aktivieren Sie die Unterstützung für MFC und vorkompilierte Header.
3. Kopieren Sie die Dateien **Line.cpp** und **Line.h** in das Projektverzeichnis. Fügen Sie beide Dateien dem Projekt hinzu. Kompilieren Sie das Bibliotheksmodul.
4. Öffnen Sie das originale Projekt des Bibliotheksmoduls. Löschen Sie die Dateien **Line.cpp** und **Line.h** aus dem Projekt. Fügen Sie am Beginn der Quellcodedatei für das Zeichenobjekt die Header-Datei **Line.h** aus dem Verzeichnis des Projekts **Line** ein, wie es Zeile 9 von Listing B.29 zeigt. Kompilieren Sie das Projekt neu.

Listing B.29: Die Include-Anweisungen und die Farbtabelle der Klasse CModArt

```

1: // ModArt.cpp: Implementierung der Klasse CModArt.
2: //
3: //////////////////////////////////////
4:
5: #include <stdlib.h>
6: #include <time.h>
7:
8: #include "stdafx.h"
9: #include "..\Line\Line.h"
10: #include "ModArt.h"

```

5. Öffnen Sie das Projekt der Testanwendung. Fügen Sie die Bibliotheksdatei **Line** in das Projekt ein. Erstellen Sie das Projekt.

Antworten für Tag 17

Antworten zu den Kontrollfragen

1. Eine erweiterte MFC-DLL.
2. Das Makro **AFX_EXT_CLASS** in der Klassendeklaration.
3. Eine Standard-DLL.
4. Normalerweise nicht. Allerdings müssen Sie die Anwendungen, die auf die DLL zurückgreifen, neu kompilieren, wenn Sie in der exportierten Schnittstelle der DLL Änderungen vorgenommen haben.
5. Die LIB-Datei enthält Funktionsrümpfe der Funktionen in der DLL und den erforderlichen Code, um den Funktionsaufruf zu lokalisieren und an die eigentliche Funktion in der DLL weiterzuleiten.

Lösungen zu den Übungen

1. Erstellen Sie ein neues Projekt. Wählen Sie dazu im Dialogfeld **Neu** auf der Registerkarte **Projekte** den Eintrag **MFC-Anwendungs-Assistent (dll)**. Tragen Sie einen Projektnamen wie beispielsweise **LineDll** ein.

Wählen Sie im ersten Schritt des MFC-Anwendungs-Assistenten die Option **Erweiterungs-MFC-DLL**.

Nachdem Sie das Projektgerüst erstellt haben, kopieren Sie die Quellcode- und Header-Dateien für die Klasse **Line** in das Projektverzeichnis. Nehmen Sie diese Dateien in das Projekt auf.

Fügen Sie in die Deklaration der Klasse `CLine` das Makro `AFX_EXT_CLASS` hinzu.

Kompilieren Sie die DLL. Kopieren Sie die DLL in das Debug-Verzeichnis für die Testanwendung.

Öffnen Sie das Projekt der Standard-DLL. Löschen Sie über die Registerkarte **Dateien** des Arbeitsbereichs die Quellcode- und Header-Dateien der Klasse `Line` aus dem Projekt. Fügen Sie die Linien-DLL-LIB-Datei in das Projekt ein. Ändern Sie in der Quellcodedatei die Zeichenfunktionalität, indem Sie den Header der Linienklasse in der Version aus dem Projektverzeichnis der `CLine`-DLL einbinden, wie es Listing B.30 zeigt.

Listing B.30: Die Include-Anweisungen der Klasse `CModArt`

```
1: // ModArt.cpp: Implementierung der Klasse CModArt.
2: //
3: //////////////////////////////////////
4:
5: #include <stdlib.h>
6: #include <time.h>
7:
8: #include "stdafx.h"
9: #include "..\LineDll\Line.h"
10: #include "ModArt.h"
```

Kompilieren Sie das Projekt. Kopieren Sie die DLL in das Debug-Verzeichnis des Projekts für die Testanwendung.

Starten Sie die Testanwendung.

2. Öffnen Sie das Projekt der Linienklasse-DLL, das Sie in der vorherigen Übung erstellt haben. Ersetzen Sie im Konstruktor der Klasse die Initialisierung der Variablen `m_nWidth` durch einen konstanten Wert, wie es Listing B.31 zeigt.

Listing B.31: Der Konstruktor der Klasse `CLine`

```
1: CLine::CLine(CPoint ptFrom, CPoint ptTo, UINT nWidth, COLORREF crColor)
2: {
3:     m_ptFrom = ptFrom;
4:     m_ptTo = ptTo;
5:     m_nWidth = 1;
6:     m_crColor = crColor;
7: }
```

Kompilieren Sie die DLL. Kopieren Sie die DLL in das Debug-Verzeichnis des Projekts für die Testanwendung. Starten Sie die Testanwendung.

Antworten für Tag 18

Antworten zu den Kontrollfragen

1. Wenn die Anwendung im Leerlauf arbeitet und keine Nachrichten in der Nachrichtenwarteschlange stehen.
2. Die Rückgabe des Wertes `TRUE` bewirkt, daß die Funktion `OnIdle` wiederholt aufgerufen wird, solange die Anwendung im Leerlauf bleibt.
3. Eine `OnIdle`-Task wird nur ausgeführt, wenn die Anwendung im Leerlauf ist und keine Nachrichten in der Nachrichtenwarteschlange stehen. Ein Thread läuft dagegen vollkommen unabhängig von der übrigen Anwendung.
4. Kritische Abschnitte, Mutexe, Semaphoren und Ereignisse.
5. Die verbleibenden Threads und Prozesse, die auf dem Computer laufen, erhalten wesentlich weniger Prozessorzeit.

Lösungen zu den Übungen

1. Bearbeiten Sie die Funktion `OnIdle` gemäß Listing B.32.

Listing B.32: Die modifizierte Funktion `OnIdle` der Klasse `CTaskingApp`

```

1: BOOL CTaskingApp::OnIdle(LONG lCount)
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen
4:
5: // Leerlaufverarbeitung der Basisklasse aufrufen
6: BOOL bRtn = CWinApp::OnIdle(lCount);
7:
8: // Position der ersten Dokumentvorlage holen
9: POSITION pos = GetFirstDocTemplatePosition();
10: // Ist Position der Dokumentvorlage gültig?
11: if (pos)
12: {
13: // Zeiger auf die Dokumentvorlage ermitteln
14: CDocTemplate* pDocTemp = GetNextDocTemplate(pos);
15: // Ist Zeiger gültig?
16: if (pDocTemp)
17: {
18: // Position des ersten Dokuments holen
19: POSITION dPos = pDocTemp->GetFirstDocPosition();
20: // Ist Position des Dokuments gültig?
21: if (dPos)
22: {
23: // Zeiger auf das Dokument ermitteln
24: CTaskingDoc* pDocWnd =
25: (CTaskingDoc*)pDocTemp->GetNextDoc(dPos);
26: // Ist Zeiger gültig?
27: if (pDocWnd)
28: {
29: // Position der Ansicht ermitteln
30: POSITION vPos = pDocWnd->GetFirstViewPosition();
31: // Ist Ansichtsposition gültig?
32: if (vPos)
33: {
34: // Zeiger auf die Ansicht holen
35: CTaskingView* pView = (CTaskingView*)pDocWnd->GetNextView(vPos);
36: // Ist Zeiger gültig?
37: if (pView)
38: {
39: // Fächer für ersten Leerlauf-Thread drehen?
40: if (pView->m_bOnIdle1)
41: {
42: // Ersten Leerlauf-Thread drehen
43: pDocWnd->DoSpin(0);
44: bRtn = TRUE;
45: }
46: // Zweiten Leerlauf-Thread drehen?
47: if (pView->m_bOnIdle2)
48: {
49: // Zweiten Leerlauf-Thread drehen

```

```

50: pDocWnd->DoSpin(2);
51: bRtn = TRUE;
52: }
53: }
54: }
55: }
56: }
57: }
58: }
59: return bRtn;
60: }

```

2. Beim Starten der unabhängigen Threads geben Sie einem Thread die Priorität `THREAD_PRIORITY_NORMAL` und dem anderen die Priorität `THREAD_PRIORITY_LOWEST`.

Bearbeiten Sie die Funktion `SuspendSpinner` entsprechend Listing B.33.

Listing B.33: Die modifizierte Funktion `SuspendSpinner` der Klasse `CTaskingDoc`

```

1: void CTaskingDoc::SuspendSpinner(int nIndex, BOOL bSuspend)
2: {
3: // Thread anhalten?
4: if (!bSuspend)
5: {
6: // Ist Zeiger für den Thread gültig?
7: if (m_pSpinThread[nIndex])
8: {
9: // Handle für den Thread holen
10: HANDLE hThread = m_pSpinThread[nIndex]->m_hThread;
11: // Auf Absterben des Threads warten
12: ::WaitForSingleObject (hThread, INFINITE);
13: }
14: }
15: else // Thread laufen lassen
16: {
17: int iSpnr;
18: int iPriority;
19: // Welchen Fächer verwenden?
20: switch (nIndex)
21: {
22: case 0:
23: iSpnr = 1;
24: iPriority = THREAD_PRIORITY_NORMAL;
25: break;
26: case 1:
27: iSpnr = 3;
28: iPriority = THREAD_PRIORITY_LOWEST;
29: break;
30: }
31: // Thread starten, Zeiger auf Fächer übergeben
32: m_pSpinThread[nIndex] = AfxBeginThread(ThreadFunc,
33: (LPVOID)&m_cSpin[iSpnr], iPriority);
34: }
35: }

```

Antworten für Tag 19

Antworten zu den Kontrollfragen

1. Eigenschaften, Methoden und Ereignisse.
2. Um dem Benutzer die Möglichkeit zu geben, die Eigenschaften des Steuerelements festzulegen.
3. Ambient, erweitert, vordefiniert und benutzerdefiniert.
4. Die Schnittstellenparameter werden zu einer standardisierten, maschinenunabhängigen Struktur gepackt und über Prozeßgrenzen hinweg gesendet. Diesen Vorgang bezeichnet man als Marshaling.
5. Mit dem Testcontainer für ActiveX-Steuerelemente.

Lösungen zu den Übungen

1. Öffnen Sie den Klassen-Assistenten, und gehen Sie auf die Registerkarte **Automatisierung** . Klicken Sie auf die Schaltfläche **Methode hinzufügen**. Geben Sie einen Methodennamen wie etwa `GenNewDrawing` ein, und legen Sie den Rückgabetyt als `void` fest. Klicken Sie auf **OK**, um die Methode hinzuzufügen. In die Methode übernehmen Sie den Code aus Listing B.34.

Listing B.34: Die Funktion `GenNewDrawing` der Klasse `CSquiggleCtrl`

```

1: void CSquiggleCtrl:: GenNewDrawing()
2: {
3: // ZU ERLEDIGEN: Fügen Sie hier den Code für Ihre Dispatch-
  Behandlungsroutine ein
4: // Flag setzen, so daß neue Zeichnung generiert wird
5: m_bGenNewDrawing = TRUE;
6: // Steuerelement ungültig machen, um Funktion OnDraw auszulösen
7: Invalidate();
8: }
```

2. Öffnen Sie den Klassen-Assistenten, und gehen Sie auf die Registerkarte **Automatisierung** . Klicken Sie auf die Schaltfläche **Methode hinzufügen**. Geben Sie einen Methodennamen wie `SaveDrawing` ein, und legen Sie den Rückgabetyt als `BOOL` fest. Nehmen Sie einen einzelnen Parameter, `sFileName`, mit dem Typ `LPCSTR` auf. Klicken Sie auf **OK**, um die Methode hinzuzufügen. In die Methode schreiben Sie den Code aus Listing B.35.

Listing B.35: Die Funktion `SaveDrawing` der Klasse `CSquiggleCtrl`

```

1: BOOL CSquiggleCtrl::SaveDrawing(LPCTSTR sFileName)
2: {
3: // ZU ERLEDIGEN: Fügen Sie hier den Code für Ihre Dispatch-
  Behandlungsroutine ein
4: try
5: {
6: // Ein CFile-Objekt erzeugen
7: CFile lFile(sFileName, CFile::modeWrite);
8: // CArchive-Objekt erzeugen, um die Datei zu speichern
9: CArchive lArchive(&lFile, CArchive::store);
10: // Datei speichern
11: m_maDrawing.Serialize(lArchive);
12: }
13: catch (CFileException err)
14: {
15: return FALSE;
16: }
```

```

17: return TRUE;
18: }

```

Antworten für Tag 20

Antworten zu den Kontrollfragen

1. Die Netzwerkadresse (oder den Namen) des Computers und den Anschluß, den der Server abhört.
2. Listen.
3. OnReceive.
4. OnConnect.
5. Send.

Lösung zur Übung

Führen Sie die folgenden Schritte aus:

1. Nehmen Sie eine Member-Variable in die Dialogfeldklasse (CSockDlg) auf. Legen Sie den Variablentyp als **BOOL**, den Namen als `m_bConnected` und den Zugriffsstatus als **Privat** fest.
2. Initialisieren Sie die Variable in der Funktion `OnInitDialog` als **FALSE**.
3. Setzen Sie die Variable in der Dialogfeldfunktion `OnAccept` auf **TRUE**, nachdem die Verbindung angenommen wurde.
4. Setzen Sie die Variable in der Dialogfeldfunktion `OnClose` auf **FALSE**.
5. Modifizieren Sie die Dialogfeldfunktion `OnAccept` gemäß Listing B.36.

Listing B.36: Die modifizierte Funktion `OnAccept` der Klasse `CSockDlg`

```

1: void CSockDlg::OnAccept()
2: {
3:   if (m_bConnected)
4:   {
5:     // Ablehnungs-Socket erzeugen
6:     CAsyncSocket sRjctSock;
7:     // Zu sendende Meldung erzeugen
8:     CString strMsg = "Zu viele Verbindungen, bitte später noch einmal
versuchen.";
9:     // Mittels Ablehnungs-Socket annehmen
10:    m_sListenSocket.Accept(sRjctSock);
11:    // Ablehnungsmeldung senden
12:    sRjctSock.Send(LPCTSTR(strMsg), strMsg.GetLength());
13:    // Socket schließen
14:    sRjctSock.Close();
15:  }
16:  else
17:  {
18:    // Verbindungsanforderung annehmen
19:    m_sListenSocket.Accept(m_sConnectSocket);\
20:    // Socket als verbunden markieren
21:    m_bConnected = TRUE;
22:    // Steuerelemente für Text und Nachrichten aktivieren

```



```

23: GetDlgItem(IDC_EMMSG)->EnableWindow(TRUE);
24: GetDlgItem(IDC_BSEND)->EnableWindow(TRUE);
25: GetDlgItem(IDC_STATICMSG)->EnableWindow(TRUE);
26: }
27: }

```

Antworten für Tag 21

Antworten zu den Kontrollfragen

1. Den Webbrowser Internet Explorer.
2. Mit der Funktion `GetLocationURL()`.
3. `IDOK`.
4. `GoBack()` und `GoForward()`.
5. Mit der Funktion `Stop()`.

Lösungen zu den Übungen

1. Nehmen Sie einen Menübefehl in das Menü **Wechseln zu** auf. Legen Sie die Eigenschaften des Menübefehls nach Tabelle B.6 fest.

Tabelle B.6: Einstellungen der Menüeigenschaft

Objekt	ID	Einstellung
Menübefehl	ID	IDM_GO_SEARCH
	Titel	Suchen im &Web
	Statuszeilentext	Öffnet die Seite »Suchen im Web“\nSuchen

Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsroutine für die ID `IDM_GO_SEARCH` für die Nachricht `COMMAND` hinzu. In die Funktion schreiben Sie den Code entsprechend Listing B.37.

Listing B.37: Die Funktion `OnGoSearch` der Klasse `CWebBrowseView`

```

1: void CWebBrowseView::OnGoSearch()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Zur Suchseite gehen
6: GoSearch();
7: }

```

Fügen Sie eine Symbolleistenschaltfläche für die Menü-ID `IDM_GO_SEARCH` hinzu.

2. Nehmen Sie einen Menübefehl in das Menü **Wechseln zu** auf. Legen Sie die Eigenschaften des Menübefehls entsprechend Tabelle B.7 fest.

Tabelle B.7: Einstellungen des Menübefehls

Objekt	Eigenschaft	Einstellung

Menübefehl	ID	IDM_GO_START
	Titel	&Startseite
	Statuszeilentext	Öffnet die Startseite\nStartseite

Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsfunktion für die ID `IDM_GO_START` für die Nachricht `COMMAND` hinzu. Übernehmen Sie den Code aus Listing 38 in die Funktion.

Listing B.38: Die Funktion `OnGoStart` der Klasse `CWebBrowseView`

```
1: void CWebBrowseView::OnGoStart()
2: {
3: // TODO: Code für Befehlsbehandlungsroutine hier einfügen
4:
5: // Zur Startseite gehen
6: GoHome();
7: }
```

Fügen Sie eine Symbolleistenschaltfläche für die Menü-ID `IDM_GO_START` hinzu.

3. Fügen Sie der Ansichtsklasse mit dem Klassen-Assistenten eine Behandlungsroutine für die Objekt-ID `IDM_VIEW_STOP` für die Nachricht `UPDATE_COMMAND_UI` hinzu. Übernehmen Sie den Code aus Listing B.39 in diese Funktion.

Listing B.39: Die Funktion `OnUpdateViewStop` der Klasse `CWebBrowseView`

```
1: void CWebBrowseView::OnUpdateViewStop(CCmdUI* pCmdUI)
2: {
3: // TODO: Code für die Befehlsbehandlungsroutine zum Aktualisieren der
  Benutzeroberfläche hier einfügen
4:
5: // Schaltfläche aktivieren, wenn beschäftigt
6: pCmdUI->Enable(GetBusy());
7: }
```

Tag C

Drucken und Druckvorschau

Die Funktionalität des Rahmenprogramms

Die vom Anwendungs-Assistenten erstellten SDI- und MDI-Rahmenprogramme bieten per Vorgabe den Aufhänger für das Drucken und die Druckvorschau (auch als Seitenansicht bezeichnet). Man kann das deaktivieren, wenn man im Schritt 4 des MFC-Anwendungs-Assistenten das Kontrollkästchen **Drucken und Druckvorschau** ausschaltet. Im allgemeinen ist es aber sinnvoll, diese Befehle in jedes Projekt einzubinden. Außerdem erfordern sie nur einen geringen Overhead. Die eigentliche Knochenarbeit beim Drucken erledigt der Gerätekontext und das GDI (Graphics Device Interface - grafische Geräteschnittstelle). Das Programmgerüst stellt für das Drucken der Dokumentseite einen Gerätekontext bereit, den man beinahe wie einen normalen Fenstergerätekontext verwenden kann.

Standardfunktionen zum Drucken

Das Gerüst für eine SDI-Anwendung unterstützt das Drucken von Bildern aus Ansichten, wobei die Informationen aus dem Dokument stammen. Da diese Informationen bereits in den Ansichten der Anwendung angezeigt werden, lassen sie sich gegebenenfalls drucken, indem man die Ansicht modifiziert und Druckfunktionen hinzufügt.

Das Gerüst ruft Ihre Funktion `OnDraw` in der Ansicht auf, um ein Bild anzuzeigen. Es gibt eine korrespondierende Funktion `OnPrint`, die das Gerüst aufruft, damit Ihre Ansicht die zu druckenden Informationen behandeln kann. Häufig benutzt man dazu den gleichen Zeichencode, wie er in der Funktion `OnDraw` vorhanden ist. In diesem Fall brauchen Sie die Funktion `OnPrint` eigentlich gar nicht implementieren. Das Gerüst erledigt das per Voreinstellung in der Basisklasse `CView` und ruft `OnDraw` auf. Der Drucker wird dann genauso behandelt wie der Bildschirm, da die Klasse einen Gerätekontext für die zu verwendenden Zeichenfunktionen als Ersatz für den üblichen Bildschirmgerätekontext bietet. Ihre Funktion `OnDraw` kann bestimmen, ob es sich bei dem an die Funktion übergebenen Gerätekontext um einen Bildschirm- oder Druckergerätekontext handelt. Da aber die Zeichenfunktionen in beiden in der gleichen Weise arbeiten, ist nicht einmal diese Kenntnis erforderlich.

Die vom Standardgerüst bereitgestellten Druckfunktionen können Sie untersuchen, indem Sie eine normale SDI-Anwendung mit dem Anwendungs-Assistenten erstellen. Lassen Sie das Kontrollkästchen **Drucken und Druckvorschau** in Schritt 4 des Anwendungs-Assistenten eingeschaltet (d.h., Sie können bereits im Schritt 1 auf **Fertigstellen** klicken), und nennen Sie das Projekt `PrintIt`.



Druckunterstützung bei einem Standardgerüst

Die Standardunterstützung für das Drucken und die Druckvorschau ist nur in SDI- und MDI-Anwendungen verfügbar. Dialogfeldbasierende Anwendungen müssen ihre eigene Druckunterstützung implementieren.

Zuerst brauchen Sie eine Grafik, die auszudrucken ist. In der Funktion `OnDraw` der Klasse `CPrintItView` (eine normale `CView`-Klasse) können Sie eine Testanzeige erstellen, wie es Listing C.1 zeigt. Das Testbild selbst ist nicht allzu wichtig, erlaubt es aber, die Ausgaben auf dem Drucker und auf dem Bildschirm miteinander zu vergleichen.

Listing C.1: Die Datei `PrintItView.cpp` - ein Druckbeispiel in `OnDraw` erzeugen

```
1: void CPrintItView::OnDraw(CDC* pDC)
2: {
3:     CPrintItDoc* pDoc = GetDocument();
4:     ASSERT_VALID(pDoc);
5:
6:     // ZU ERLEDIGEN: Hier Code zum Zeichnen der ursprünglichen Daten
    hinzufügen
7:
8:     // Metrischen Abbildungsmodus einstellen
9:     pDC->SetMapMode(MM_LOMETRIC);
10:
11:    // Schrift mit Höhe 2,2 cm deklarieren und erzeugen
12:    CFont fnBig;
13:    fnBig.CreateFont(220,0,0,0,FW_HEAVY,FALSE,FALSE,0,
14:        ANSI_CHARSET,OUT_DEFAULT_PRECIS,
15:        CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
16:        FF_SWISS+VARIABLE_PITCH,"Arial");
17:
18:    // Neue Schrift auswählen, originale speichern
19:    CFont* pOldFont = pDC->SelectObject(&fnBig);
20:
21:    // Rechteck im Client-Bereich deklarieren
22:    CRect rcClient;
23:    GetClientRect(&rcClient);
24:
25:    // In logische Einheiten konvertieren
26:    pDC->DPtoLP(&rcClient);
27:
28:    // Variablen zum Zeichnen einrichten
29:    const int nPoints = 50;
30:    int xm = rcClient.Width();
31:    int ym = rcClient.Height();
32:    double dAspW = xm/(double)nPoints;
33:    double dAspH = ym/(double)nPoints;
34:
35:    // Schwarzen Stift auswählen
36:    CPen* pOldPen =
```

```
37: (CPen*)pDC->SelectStockObject(BLACK_PEN);
38:
39: // Linien zeichnen
40: for(int i=0;i<nPoints;i++)
41: {
42:     int xo = (int)(i * dAspW);
43:     int yo = (int)(i * dAspH);
44:
45:     pDC->MoveTo(xo,0);
46:     pDC->LineTo(xm,yo);
47:     pDC->LineTo(xm-xo,ym);
48:     pDC->LineTo(0,ym-yo);
49:     pDC->LineTo(xo,0);
50: }
51:
52: // Alten Stift wiederherstellen
53: pDC->SelectObject(pOldPen);
54:
55: // Vordergrundtext zeichnen
56: pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
57: pDC->SetBkMode(TRANSPARENT);
58:
59: // Grauen Text einstellen
60: pDC->SetTextColor(RGB(64,64,64));
61: pDC->TextOut(xm/2,ym/2,"Probedruck");
62:
63: // Alte Schrift wiederherstellen
64: pDC->SelectObject(pOldFont);
65: }
```

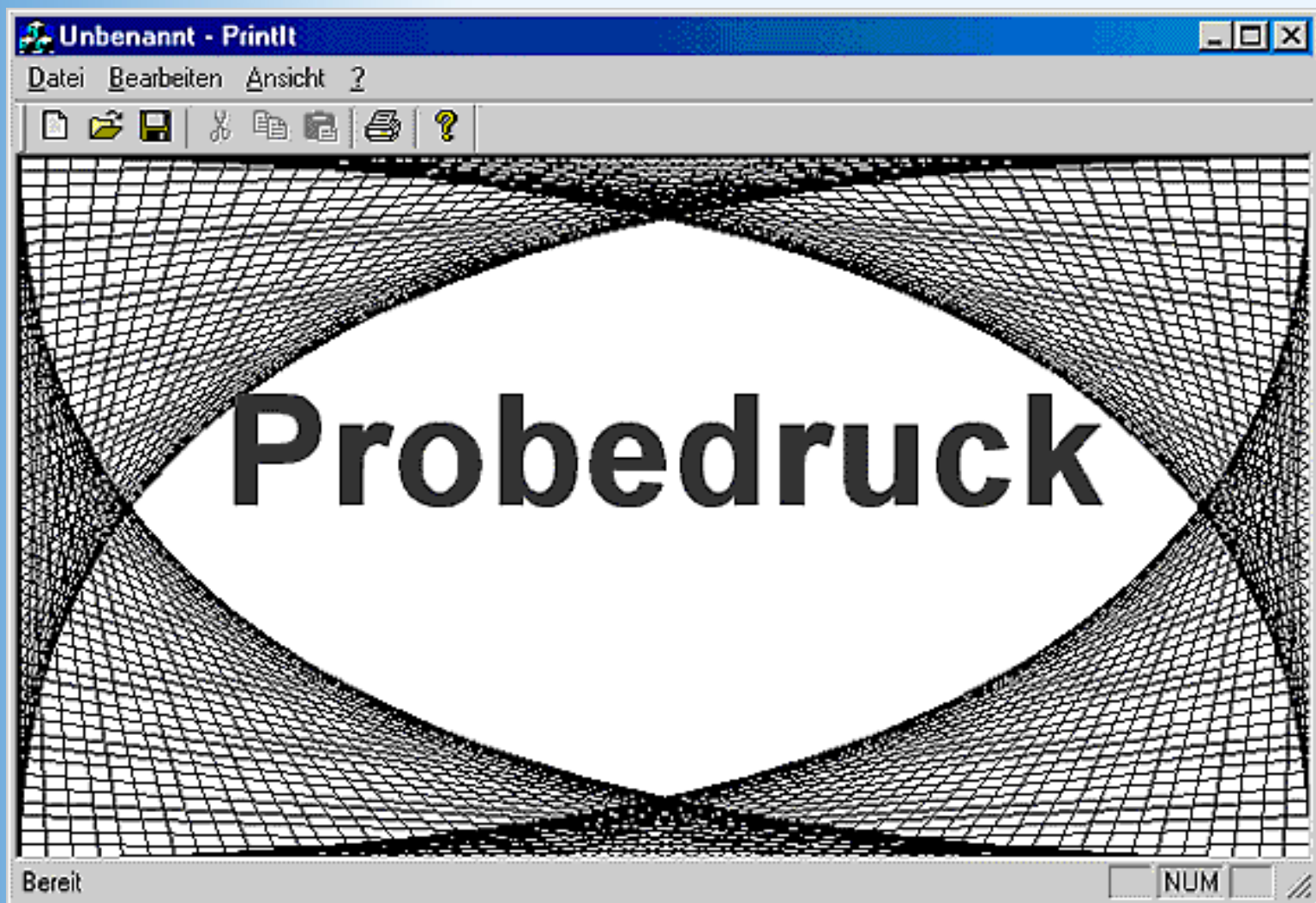



Abbildung C.1:
Testausgabe einer Grafik mit PrintIt in einem Fenster

Obwohl diese OnDraw-Funktion eine Menge Code enthält, ist keine Zeile überflüssig. Der Code zeichnet die Linien innerhalb des Client-Bereichs und schreibt einen Probetext in die Mitte. Beachten Sie, daß Zeile 9 den Abbildungsmodus auf MM_LOMETRIC setzt. Damit werden die logischen Koordinaten in Zehntel Millimeter übersetzt.

Zeile 13 erzeugt eine 2,2 cm hohe Schrift, die Zeile 61 zum Zeichnen des Probetextes verwendet. Die Zeilen 40 bis 50 zeichnen auf der Basis der Koordinaten des Client-Bereichs den Rahmen für das kunstvolle Muster. Die Details können Sie selbst studieren, hier geht es vor allem um den Druckvorgang.

Wenn Sie das Programm erstellen und ausführen, nachdem Sie die Zeilen in die Funktion OnDraw gemäß Listing C.1 eingegeben haben, sollte im Anwendungsfenster eine Grafik wie in Abbildung C.1 zu sehen sein.

Die große Frage lautet nun: Was ist zu tun, um das Bild zu drucken? Überraschenderweise nur wenig - das Standardgerüst versucht bereits, das Bild zu drucken, und übergibt dazu der Funktion OnDraw den Gerätekontext des Druckers anstelle des Fensters.

Wenn Sie das Menü **Datei** der Anwendung PrintIt öffnen und den Befehl **Seitenansicht** wählen, erscheint eine verkleinerte Darstellung des Bildes in der oberen linken Ecke, auch wenn die Schrift für die Linienzeichnung zu groß ist. Das liegt aber nicht am Programmgerüst - dieses tut sein bestes, um das Fenster wiederzugeben. Dem Gerätekontext wurden die falschen Koordinaten übergeben. Das

Problem liegt in Zeile 23 bei der Funktion `GetClientRect`.

Beachten Sie, daß die Funktion `GetClientRect` ein Element der Ansicht und nicht des Gerätekontextes ist. Für das Fenster funktioniert das ausgezeichnet, da der Gerätekontext die gleiche Größe wie das Fensterrechteck hat. Jetzt geben Sie das Fensterrechteck an den Druckergerätekontext aus (der im Vergleich zum Fenster klein ist), erzeugen aber eine Schrift mit einer Höhe von 2,2 cm, die (bedingt durch den Abbildungsmodus) immer die gleiche Größe hat.

OnPrint überschreiben

Um das Problem mit den Koordinaten des Client-Bereichs zu lösen, ist das richtige Rechteck für den Drucker statt für das Fenster zu übergeben. Zum Glück ruft das Programmgerüst eine virtuelle Funktion auf, die Sie in Ihrer Ansicht überschreiben können, um alle benötigten Informationen zusammenzutragen. Wie bereits weiter oben erwähnt, handelt es sich hierbei um die Funktion `OnPrint`, die analog zu `OnDraw` ist. Beim Zeichnen in einem Fenster wird `OnDraw` aufgerufen; wenn man auf einen Drucker zeichnet, erfolgt der Aufruf von `OnPrint`. Vielleicht fragen Sie sich, wie der Zeichencode in `OnDraw` ausgeführt wird, um die Anzeige der Beispielgrafik zu drucken. Die Standardimplementierung von `OnPrint` in der Klasse `CView` ruft einfach `OnDraw` auf und übergibt dieser Funktion ihren Druckergerätekontext.

Ihre Funktion `OnPrint` muß `OnDraw` nicht aufrufen. Sie können `OnPrint` überschreiben, um irgend etwas anderes zu zeichnen, aber viele Anwendungen müssen genau das ausdrucken können, was der Benutzer sieht.

Um die virtuelle Funktion `OnPrint` zu überschreiben, führen Sie die folgenden Schritte aus:

1. Gehen Sie im Arbeitsbereich auf die Registerkarte **Klassen**.
2. Klicken Sie auf das oberste Pluszeichen, um die Projektklassen einzublenden.
3. Klicken Sie mit der rechten Maustaste auf die Ansichtsklasse, der Sie die überschriebene `OnPrint`-Funktion hinzufügen möchten (im Beispiel `CPrintItView`), um das Kontextmenü anzuzeigen.
4. Wählen Sie den Befehl **Virtuelle Funktion hinzufügen**, um das Dialogfeld **Überschreiben neuer virtueller Funktionen für Klasse CPrintItView** anzuzeigen.
5. In der Liste **Neue virtuelle Funktionen** sollte die virtuelle Funktion `OnPrint` vorhanden sein.
6. Klicken Sie auf die Schaltfläche **Hinzufügen und Bearbeiten**, um die virtuelle Funktion `OnPrint` umzugestalten.

Die vorgegebene überschriebene Funktion für `OnPrint` sieht folgendermaßen aus:

```
void CPrintItView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
// TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen
CView::OnPrint(pDC, pInfo);
}
```

Als erster Unterschied zur Funktion `OnDraw` fällt der zweite Parameter auf, der Zeiger auf ein `CPrintInfo`-Objekt `pInfo`. Hier finden Sie die Einzelheiten über den aktuellen Druck, insbesondere die benötigten Rechteckkoordinaten für den Druckergerätekontext. Zu `CPrintInfo` gehören zahlreiche nützliche Member-Variablen. Einige davon führt Tabelle C.1 auf.

Tabelle C.1: Member-Variablen von CPrintInfo, die sich auf Druckinformationen beziehen

Variablenname	Beschreibung des Inhalts
m_nCurPage	Die aktuell gedruckte Seite bei mehrseitigen Druckausgaben.
m_nNumPreviewPages	Entweder 1 oder 2, je nach Anzahl der angezeigten Vorschauseiten.
m_rectDraw	Die Rechteckkoordinaten der Druckseite.
m_pPD	Zeiger auf eine CPrintDialog-Klasse, wenn das Dialogfeld Drucken verwendet wird.
m_bDirect	TRUE, wenn das Dialogfeld Drucken umgangen wird.
m_bPreview	TRUE, wenn momentan die Druckvorschau aktiv ist.
m_strPageDesc	Ein Formatstring für die Generierung der Seitennummer.
m_lpUserData	Ein Zeiger, der für die Speicherung von Benutzerdaten verwendet werden kann.

Auf einige andere Variablen in CPrintInfo gehen wir weiter hinten in diesem Anhang ein. Zuerst müssen wir aber die Rechteckkoordinaten des Druckbereiches und nicht des Fensterbereichs ermitteln. Die Elementvariable m_rectDraw speichert die Rechteckkoordinaten der aktuellen Druckseite. Diese Koordinaten kann man beim Druckergerätekontext in der Funktion OnDraw verwenden. Dennoch besteht das Problem, daß diese Struktur nicht an OnDraw übergeben wird. Allerdings lassen sich die Koordinaten in eine Elementvariable der Klasse CPrintItView kopieren.

Fügen Sie nach dem Kommentar `// TODO ...` und noch vor dem Aufruf von `CView::OnPrint` die nachstehenden Zeilen hinzu, um das Rechteck zu speichern:

```
// Rechteckbereich aus pInfo kopieren
if (pInfo) m_rcPrintRect = pInfo->m_rectDraw;
```

Diese Anweisung speichert den zu druckenden Bereich im Element m_rcPrintRect der Klasse CPrintItView. Diese Member-Variable müssen Sie demzufolge noch deklarieren. Klicken Sie dazu im Arbeitsbereich auf der Registerkarte **Klassen** mit der rechten Maustaste auf die Klasse CPrintItView, und wählen Sie aus dem Kontextmenü den Befehl **Member-Variable hinzufügen**. Den Variablentyp geben Sie mit `CRect` und die Deklaration mit `m_rcPrintRect` an. Der Zugriffsstatus sollte **Privat** sein, da keine andere Klasse über dieses interne Rechteck Bescheid wissen muß.

Der Druckergerätekontext

Der an OnPrint übergebene Gerätekontext unterscheidet sich leicht vom Anzeigekontext, da zum einen weniger Farben vorhanden sein können und er wahrscheinlich größer als die Anzeige ist. Davon abgesehen kann man ihn genauso wie den Gerätekontext für den Bildschirm zum Zeichnen einsetzen. Daraus erklärt sich, daß man dieselbe OnDraw-Funktion sowohl zum Drucken als auch zur Anzeige in einem Fenster nutzen kann. Der Aufruf der Basisklasse CView::OnPrint implementiert Code, der genau das bewirkt.

Der Gerätekontext speichert ein Flag, das man über die Funktion IsPrinting abfragen kann und das angibt, ob man in einen bildschirmbasierten Gerätekontext oder einen druckerbasierten zeichnet.

Diesen Unterschied nutzt man, um die gedruckte Ausgabe gegenüber der Bildschirmausgabe zu ändern, oder genauer gesagt, um die Koordinaten für die gedruckte Ausgabe anzupassen.

Für das Beispielprogramm muß man nur noch die Koordinaten `m_rcPrintRect` verwenden, wenn man in der Funktion `OnDraw` druckt. Listing C.2 zeigt den Code, der mit der Funktion `IsPrinting` bestimmt, ob der Client-Bereich des Fensters oder das Rechteck des Druckers zu verwenden ist. Die von der Seitenansicht produzierte Ausgabe zeigt Abbildung C.2.

Listing C.2: Drucken in einen Rechteckbereich in die Standardimplementierung der Funktion `OnDraw` eingefügt

```

1: // Client-Rechteck deklarieren
2: CRect rcClient;
3:
4: // Gerätekontext auf Druckermodus testen
5: if (pDC->IsPrinting())
6: {
7: // Drucken, also Druckerrechteck verwenden
8: rcClient = m_rcPrintRect;
9: }
10: else
11: {
12: // Kein Drucken, also genügt Client-Rechteck
13: GetClientRect(&rcClient);
14: }
15:
16: // In logische Einheiten konvertieren
17: pDC->DPtoLP(&rcClient);

```

Beachten Sie, daß Zeile 5 von Listing C.2 die Funktion `IsPrinting` des Gerätekontextes in einer `if`-Anweisung aufruft. Die Funktion liefert `TRUE`, wenn es sich um einen Druckergerätekontext (oder die Druckvorschau) handelt, und `FALSE` für jeden anderen Gerätekontext. Beim Drucken kann man das gespeicherte Rechteck der Druckseite an `rcClient` zuweisen, wie es Zeile 8 zeigt. Bei einem normalen Bildschirmfenster verwendet man einfach die normale `GetClientRect`-Funktion, um das Client-Rechteck des Fensters zu ermitteln (siehe Zeile 13).

Da das Programm mit einem Abbildungsmodus arbeitet, muß man sowohl die zum Drucken als auch die zur Anzeige verwendeten Rechteckkoordinaten von Geräteeinheiten in logische Einheiten umrechnen. Das erledigt die Funktion `DPtoLP` in Zeile 17. Wenn Sie in die bereits vorhandene Funktion `OnDraw` die Zeilen 4 bis 14 aus Listing C.2 einfügen und die Anwendung erstellen und ausführen, läßt sich die Seitenansicht wie zuvor aufrufen - aber mit besseren Ergebnissen (siehe Abbildung C.2).

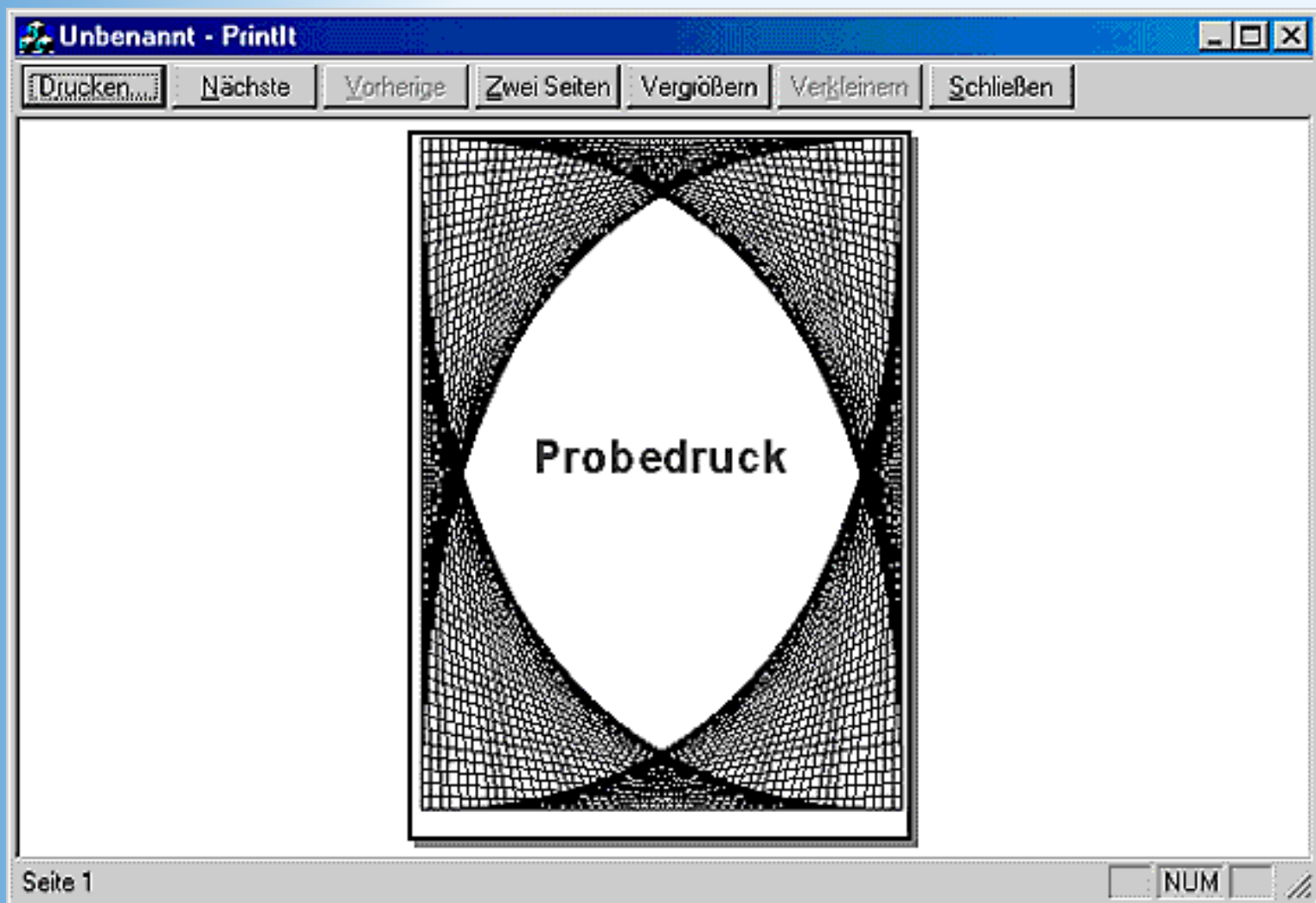


Abbildung C.2:
Seitenansicht mit den Rechteckkoordinaten einer vollständigen Druckseite

Das Höhen-/Seitenverhältnis

Wie aus Abbildung C.2 hervorgeht, ist die gedruckte Ausgabe gestreckt, weil das Papier wesentlich länger und schmaler ist als das Fenster. Die Beziehung zwischen der Breite und der Höhe bezeichnet man als Höhen-/Seitenverhältnis. Um ein Strecken des Bildes in die eine oder andere Richtung zu vermeiden, muß man das gleiche Höhen-/Seitenverhältnis wie beim Bild im Fenster einhalten. Der Code in Listing C.2 versucht das gar nicht erst, was in den meisten Fällen zu einer unbefriedigenden Ausgabe führt. Fügen Sie deshalb den entsprechenden Code hinzu, um das Höhen-/Seitenverhältnis der gedruckten Ausgabe zu gewährleisten.

Am besten ermittelt man zuerst, ob man entweder die Breite oder die Höhe auf den Maximalwert setzt, um die bestmögliche Ausnutzung des Druckbereichs zu erzielen. Abhängig davon verkürzt man die andere Dimension, um das Höhen-/Seitenverhältnis zu wahren.

Dazu braucht man Angaben über die Abmessungen des Papiers und dessen Höhen-/ Seitenverhältnis. Diese Details (und viele weitere) lassen sich über eine Funktion des Gerätekontextes, `GetDeviceCaps`, abrufen. Wenn man die Flags `ASPECTX` oder `ASPECTY` an `GetDeviceCaps` übergibt, kann man die Beziehung zwischen der Breite eines Pixels und dessen Höhe ermitteln. Handelt es sich um eine 1:1-Beziehung, ist das Pixel quadratisch. Andernfalls ist es rechteckig und kann vom eigenen Höhen-/Seitenverhältnis des Bildschirms abweichen. Wenn das Verhältnis von 1:1 abweicht, kann man entscheiden, welche Achse das größte Bild erhält, während man das

Höhen-/Seitenverhältnis des Bildschirms beibehält. Auf diese Weise lassen sich gestreckte Bilder vermeiden.

Listing C.3 zeigt den Code, der genau die beschriebenen Abläufe in der Funktion `OnDraw` realisiert.



Höhen-/Seitenverhältnis von Geräten

Bei den meisten Druckern eignet sich das Höhen-/Seitenverhältnis von 1:1 scheinbar am besten. Wenn man sich aber die Druckergebnisse von Thermodruckern wie etwa in Fax-Geräten genauer ansieht, erkennt man einen deutlichen Unterschied im Höhen-/Seitenverhältnis gegenüber dem Original.

Listing C.3: Modifikationen an der Funktion `OnDraw` zur Wahrung des Höhen-/Seitenverhältnisses bei gleichzeitig größtmöglicher Druckausgabe

```

1: // Client-Rechteck deklarieren und Abmessungen holen
2: CRect rcClient;
3: GetClientRect(&rcClient);
4:
5: // Gerätekontext auf Druckermodus testen
6: if (pDC->IsPrinting())
7: {
8: // Verhältnis Druckerbreite/Fensterbreite ermitteln
9: double dWidthRatio=(double)m_rcPrintRect.Width()/
10: (double)rcClient.Width();
11:
12: // Verhältnis Druckerhöhe/Fensterhöhe ermitteln
13: double dHeightRatio=(double)m_rcPrintRect.Height()/
14: (double)rcClient.Height();
15:
16: // Höhen-/Seitenverhältnis des Geräts berechnen
17: double dAspect=(double)pDC->GetDeviceCaps(ASPECTX)/
18: (double)pDC->GetDeviceCaps(ASPECTY);
19:
20: // Neue relative Höhe ermitteln
21: int nHeight=(int)(rcClient.Height() *
22: dWidthRatio * dAspect );
23:
24: // Neue relative Breite ermitteln
25: int nWidth=(int)(rcClient.Width() *
26: dHeightRatio * (1.0 / dAspect) );
27:
28: // Das gesamte Rechteck festlegen
29: rcClient=m_rcPrintRect;
30:
31: // Beste Anpassung längs oder quer ermitteln
32: if (nHeight > nWidth)
33: {
34: // Längs am besten, also Breite anpassen
35: rcClient.BottomRight().x=

```

```

36: m_rcPrintRect.TopLeft().x + nWidth;
37: }
38: else
39: {
40: // Quer am besten, also Höhe anpassen
41: rcClient.BottomRight().y=
42: m_rcPrintRect.TopLeft().y + nHeight;
43: }
44: }
45:
46: // In logische Einheiten konvertieren
47: pDC->DPtoLP(&rcClient);

```

Beachten Sie, daß sowohl das Bildschirmfenster als auch die bedruckte Fläche auf die Fensterkoordinaten zurückgreifen, die aus der Funktion `GetClientRect` in Zeile 3 stammen. Für die Darstellung auf dem Bildschirm unternimmt der Code nichts weiter und fährt wie gehabt fort.

Bei der Druckausgabe passiert dagegen eine ganze Menge, wenn der Test `IsPrinting` in Zeile 6 das Ergebnis `TRUE` liefert. Zuerst ist das Verhältnis zwischen Papierbreite und Fensterbreite sowie zwischen Papierhöhe und Fensterhöhe zu ermitteln. Das geschieht in den Zeilen 9 und 13, die die Papierabmessungen durch die Fensterabmessungen dividieren.

Als nächstes müssen Sie die Eigenheiten des Höhen-/Seitenverhältnisses des Geräts berechnen. Dazu verwenden Sie die Funktion `GetDeviceCaps` in Zeile 17, um das Verhältnis der Breite zur Höhe im Gerät selbst zu ermitteln und das Ergebnis in `dAspect` zu speichern.

Mit diesen Werten können Sie nun die Breiten- und Höhenkoordinaten des Geräts im Vergleich zu den Fensterabmessungen berechnen, wie es aus den Zeilen 21 und 25 hervorgeht. Diese Berechnung, die das Höhen-/Seitenverhältnis des Geräts für jede Dimension einschließt, liefert die angepaßte Höhe für die volle Seitenbreite oder umgekehrt. Nun müssen Sie entscheiden, ob sich die volle Breite oder Höhe einer Seite am besten eignet und die jeweils andere Dimension anpassen. Die Bedingung in Zeile 32 trifft diese Entscheidung auf der Basis der größeren Breite oder Höhe. Das bedeutet, daß man bei einem hohen, schmalen Fenster besser die volle Höhe des Papiers nutzt und die Breite anpaßt. Wenn Sie umgekehrt ein kurzes, breites Fenster haben, verwendet man besser die volle Breite und paßt die Höhe an. Je nachdem, was besser ist, findet die Anpassung in Zeile 35 oder 41 statt, indem die x- oder y-Koordinate des rechts unten liegenden Punktes auf die angepaßte Breite oder Höhe gesetzt wird.

Alle anderen Abmessungen werden mit der Zuweisung in Zeile 29 auf `rcClient` des Papiers gesetzt, so daß die Anpassung die einzige erforderliche Änderung ist. Nach diesem Codeabschnitt setzt das Programm fort und verwendet das angepaßte Rechteck für die Zeichnung.

Wenn Sie die Anwendung erstellen und ausführen, nachdem Sie die Zeilen in Listing C.3 in die Funktion `OnDraw` eingefügt haben, sollte das Fenster für das Drucken oder die Seitenansicht das gleiche Höhen-/Seitenverhältnis aufweisen wie die Darstellung auf dem Bildschirm. Wenn Sie das Fenster strecken, um es höher als breit zu machen, verwendet die gedruckte Ausgabe die volle Höhe der Seite statt der vollen Breite, wobei aber das korrekte Höhen-/Seitenverhältnis erhalten bleibt.

Seitenumbruch und Seitenausrichtung

Es kommt zwar häufig vor, daß man den Fensterinhalt ausdrucken muß, in der Regel hat man es aber mit der Druckausgabe großer und komplexer, mehrseitiger Dokumente mit den vom Benutzer als so wichtig erachteten Daten zu tun. Auch hier kommt uns das Programmgerüst zu Hilfe und vereinfacht diesen Vorgang mit dem allgemeinen Dialogfeld **Seite einrichten** und einem System zur Seitennummerierung, um den angegebenen Seitenbereich zu drucken und in der Vorschau zu betrachten.

Anfangs- und Endseite festlegen

Bei mehrseitigen Dokumenten gilt unser Augenmerk zuerst den Anfangs- und Endseiten, die auch anzeigen, wie viele Seiten zu drucken sind. Wenn das Drucken beginnt, wird als erstes die virtuelle Funktion `OnPreparePrinting` der Ansichtsklasse des Programmgerüsts aufgerufen. Sie stellt als Parameter das `CPrintInfo`-Objekt `pInfo` bereit. An dieser Stelle begegnen Sie `CPrintInfo` das erste Mal, und hier können Sie das Objekt auch zuerst ändern, um die Druckausgabe an die eigenen Erfordernisse anzupassen. Der Anwendungs-Assistent stellt die Funktion `OnPreparePrinting` automatisch bereit, wenn Sie eine SDI-Anwendung erstellen. In diesem Fall brauchen Sie sich also nicht selbst darum zu kümmern. Die Standardimplementierung können Sie sich ansehen, indem Sie auf das Element `OnPreparePrinting` der Klasse `CPrintItView` auf der Registerkarte **Klassen** doppelklicken.

Die Funktion sieht folgendermaßen aus:

```
BOOL CPrintItView::OnPreparePrinting(CPrintInfo* pInfo)
{
// Standardvorbereitung
return DoPreparePrinting(pInfo);
}
```

Beim Drucken wird die Funktion `DoPreparePrinting` automatisch aufgerufen und ihr der Zeiger `pInfo` auf das `CPrintInfo`-Objekt übergeben. Die Funktion `DoPreparePrinting` richtet den erforderlichen Gerätekontext ein und ruft das Standarddialogfeld **Drucken** auf, wenn Sie drucken (und nicht die Seitenansicht anzeigen). Auf dieses Dialogfeld geht der nächste Abschnitt näher ein. Zunächst einmal können Sie den zu druckenden Seitenbereich einrichten, indem Sie das Objekt `CPrintInfo` vor dem Aufruf von `DoPreparePrinting` modifizieren.

Fügen Sie zu diesem Zweck die folgenden Zeilen vor dem Kommentar `// Standardvorbereitung` ein:

```
pInfo->SetMinPage(2);
pInfo->SetMaxPage(8);
```

Diese beiden Elementfunktionen der Klasse `CPrintInfo` modifizieren das `CPrintInfo`-Objekt, auf das `pInfo` zeigt, um die Startseite auf Seite 2 über `SetMinPage` und die letzte Seite auf Seite 8 via `SetMaxPage` zu setzen.

Wenn man das Dokument jetzt druckt, wird die Funktion `OnPrint` sechsmal aufgerufen. Der einzige Unterschied zwischen jedem dieser Aufrufe ist die Member-Variable `pInfo->m_nCurPage`, die die aktuelle Seite aufnimmt, wenn die Schleife zwischen 2 und 8 läuft.

Je nach der Art der zu schreibenden Anwendung kann sich das Verfahren zur Bestimmung der Seitenanzahl ändern. Wenn Sie Musik-CDs verkaufen und eine Broschüre über Ihren

Produktionsbereich drucken wollen, bringen Sie wahrscheinlich das Cover der CD und einen kurze Inhaltsübersicht auf je einer gedruckten Seite unter. Verkaufen Sie zum Beispiel 120 verschiedene CDs, brauchen Sie 120 Seiten. Wenn Sie dagegen eine umfangreiche öffentliche Ausschreibung mit unterschiedlichen Rechnungsposten und formatierten Einträgen drucken, müssen Sie wahrscheinlich die Höhe aller verschiedenartigen Teile messen und die Seitenzahl nach der Ausführung des Seitenumbruchs berechnen. Wie auch immer, wenn Sie die Seitenzahl ermittelt haben, ist `OnPreparePrinting` die Stelle, wo Sie diese Zahl in das Objekt `CPrintInfo` schreiben.



Das Dialogfeld **Drucken** überspringen

Man muß den Benutzer nicht in jedem Fall mit dem Dialogfeld **Drucken** »ärgern«. Das Dialogfeld läßt sich übergehen, indem man die Variable `pInfo->m_bDirect` in `OnPreparePrinting` auf `TRUE` setzt.

Um den Unterschied zwischen einem ganzseitigen Bericht und einem Fensterausdruck hervorzuheben, können Sie eine ganz andere Zeichnung in der Funktion `OnPrint` als in `OnDraw` implementieren, wie es Listing C.4 zeigt. In dieser `OnPrint`-Funktion wird die Funktion `CView::OnPrint` der Basisklasse überhaupt nicht aufgerufen. Demnach wird auch der Standardaufruf von `OnDraw` nicht ausgeführt. In dieser Implementierung ist damit die Druckausgabe und die Bildschirmausgabe vollkommen unterschiedlich.

Listing C.4: Seitenspezifisches Zeichnen in `OnPrint` implementieren

```
1: void CPrintItView::OnPrint(CDC* pDC, CPrintInfo* pInfo)
2: {
3: // TODO: Speziellen Code hier einfügen und/oder Basisklasse
aufrufen
4:
5: // Schrift erzeugen und auswählen
6: CFont fnTimes;
7: fnTimes.CreatePointFont(720, "Times New Roman", pDC);
8: CFont* pOldFont=(CFont*)pDC->SelectObject(&fnTimes);
9:
10: // Pinsel erzeugen und auswählen
11: CBrush brHatch(HS_CROSS, RGB(64, 64, 64));
12: CBrush* pOldBrush =
13: (CBrush*)pDC->SelectObject(&brHatch);
14:
15: // Seitentext erzeugen
16: CString strDocText;
17: strDocText.Format("Seitennummer %d",
18: pInfo->m_nCurPage);
19:
20: pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
21:
22: // Einige hilfreiche Punktobjekte einrichten
23: CPoint ptCenter=pInfo->m_rectDraw.CenterPoint();
24: CPoint ptTopLeft=pInfo->m_rectDraw.TopLeft();
25: CPoint ptBotRight=pInfo->m_rectDraw.BottomRight();
26:
```

```

27: // Punkte für den Rhombus erzeugen
28: CPoint ptPolyArray[4]=
29: {
30: CPoint(ptTopLeft.x,ptCenter.y),
31: CPoint(ptCenter.x,ptTopLeft.y),
32: CPoint(ptBotRight.x,ptCenter.y),
33: CPoint(ptCenter.x,ptBotRight.y)
34: };
35:
36: // Rhombus zeichnen
37: pDC->Polygon(ptPolyArray,4);
38:
39: // Text zeichnen
40: pDC->TextOut(ptCenter.x,ptCenter.y,strDocText);
41:
42: // Alte Zeichenobjekte wiederherstellen
43: pDC->SelectObject(pOldFont);
44: pDC->SelectObject(pOldBrush);
45: }

```

Die Zeilen 6 bis 12 von Listing C.4 richten die Ressourcen (eine Schrift und einen Pinsel) zum Drucken ein. Eigentlich gibt es dafür eine bessere Stelle. Darauf geht der Abschnitt »GDI-Objekte mit OnBeginPrinting hinzufügen« weiter hinten in diesem Anhang ein.

In der Beispielanwendung gibt Zeile 17 die Seitennummer stellvertretend für den eigentlichen Inhalt einer Seite aus. Bei einer richtigen Anwendung beziehen Sie sich über die Seitennummer wahrscheinlich auf das Dokument und suchen nach einem bestimmten Datenelement. Im weiter oben erwähnten Szenario einer Musik-CD-Produktion kann sich die Seitennummer auf eine bestimmte CD beziehen, und die Zeichenfunktionen übernehmen dann diese Daten. Auf derartig ausgeklügelte Verfahren wollen wir an dieser Stelle verzichten, so daß die aktuelle Seitennummer einfach mit `pInfo->m_nCurPage` diesen Punkt demonstrieren soll.

Die Zeilen 22 bis 37 richten einen Rhombus als Hintergrund ein. Zeile 40 zeichnet den Text mit der aktuellen Seitennummer in die Mitte der Seite. Die ursprünglichen Einstellungen für Schrift und Pinsel stellen die Zeilen 43 und 44 wieder her.

Wenn Sie das Programm nach diesen Änderungen in `OnPrint` erstellen, ausführen und im Menü **Datei** den Befehl **Seitenansicht** wählen, sollten Sie eine Vorschau auf mehrere Seiten über die Schaltflächen **Nächste** und **Vorherige** wie in Abbildung C.3 erhalten. Bei angeschlossenem Drucker können Sie auch das mehrseitige Dokument ausdrucken.

Das Dialogfeld Drucken

Beim Drucken eines mehrseitigen Dokuments erscheint zuerst ein Dialogfeld, in dem Sie die Druckereinstellungen anpassen können (siehe Abbildung C.4). Es handelt sich hierbei um das Standarddialogfeld **Drucken**, das von der Funktion `CView::DoPreparePrinting` aufgerufen wird. Diese Funktion wird wiederum von der überschriebenen Funktion `OnPreparePrinting` aufgerufen. Im Dialogfeld **Drucken** können Sie den zu druckenden Seitenbereich, die Anzahl der Kopien, ein Sortier-Flag, den Zieldrucker und eine ganze Reihe von Eigenschaften für den jeweiligen Drucker einrichten.

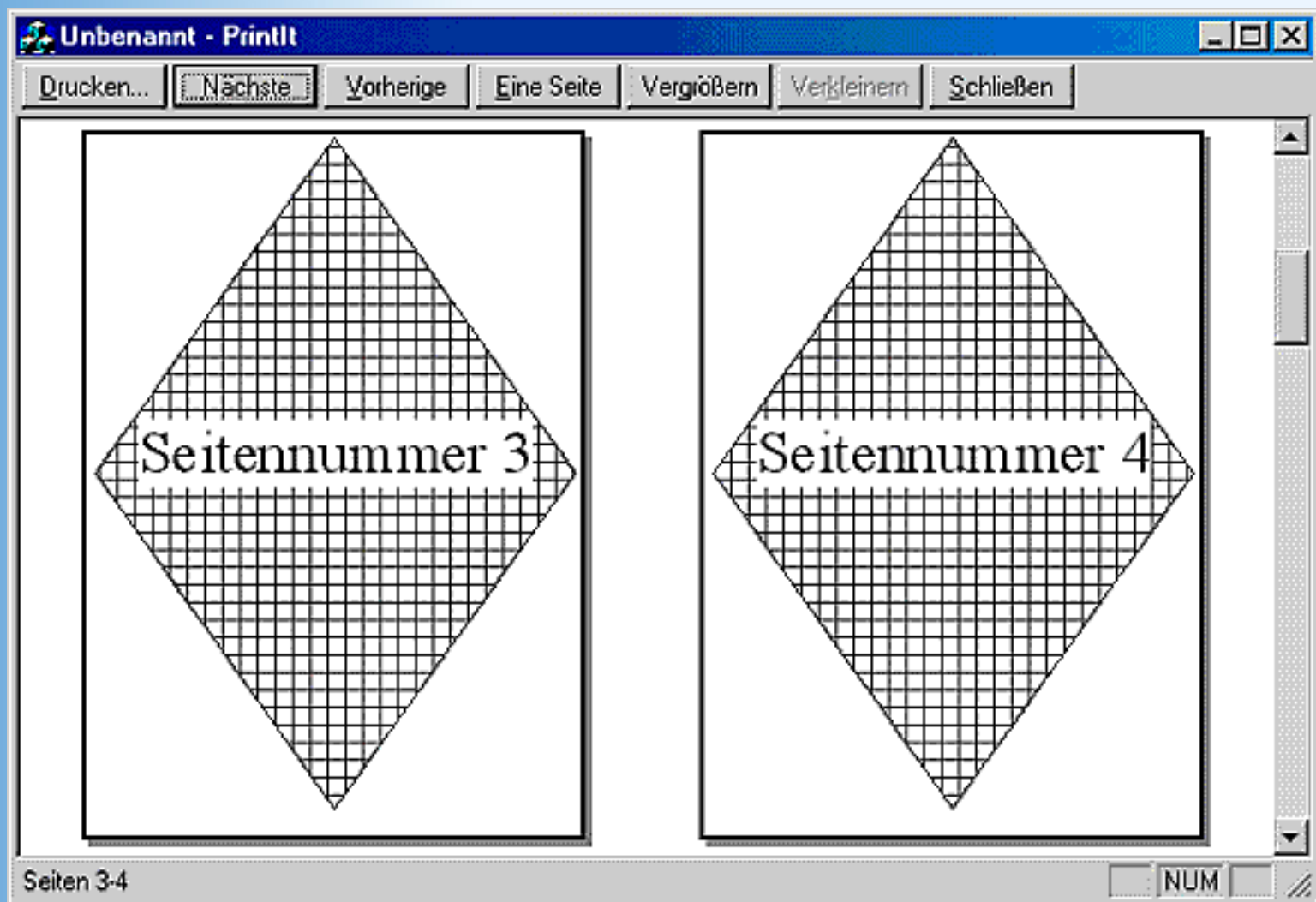


Abbildung C.3:
Die Seitenansicht eines mehrseitigen Dokuments

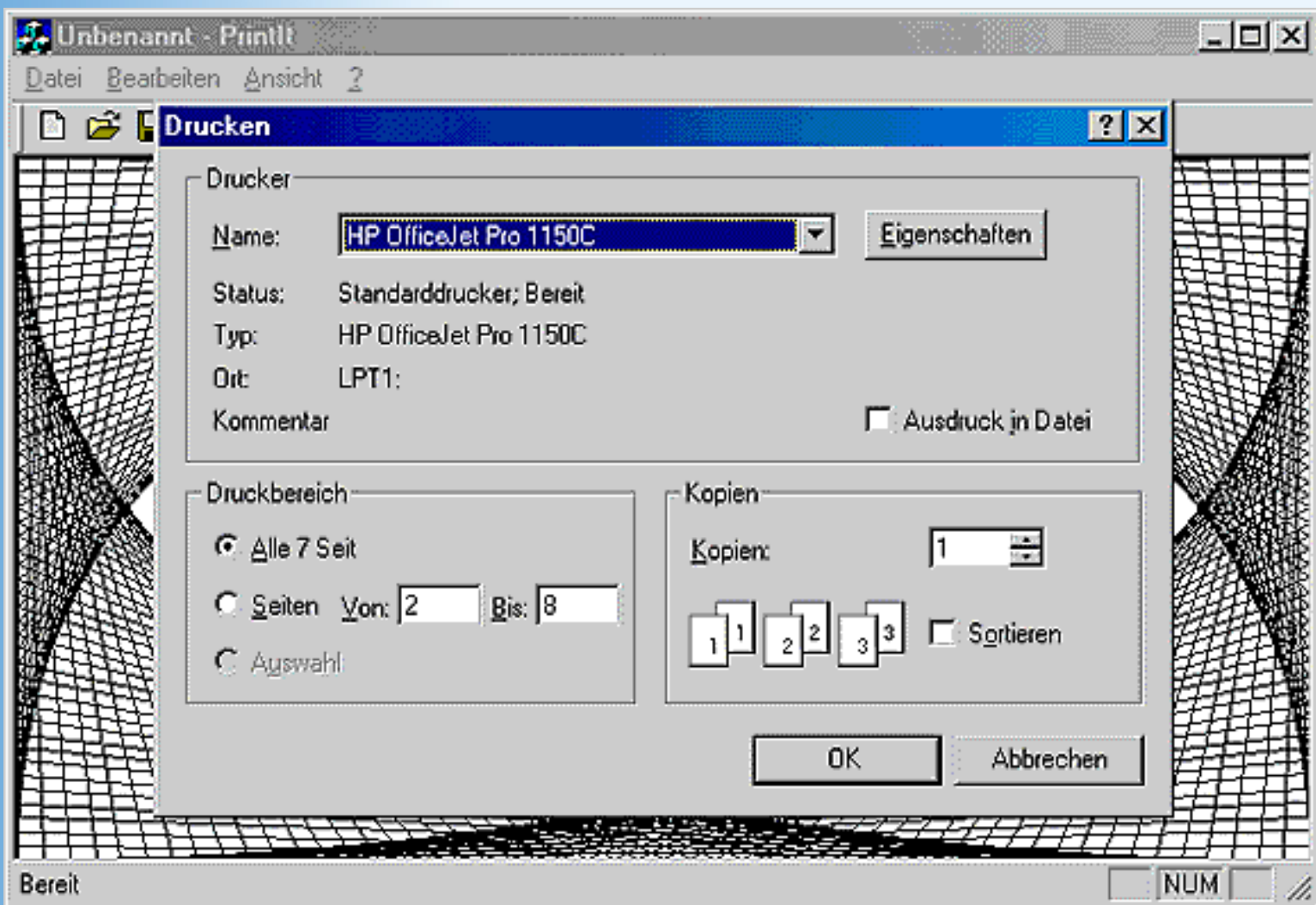


Abbildung C.4:
Das Standarddialogfeld Drucken



Das Kontrollkästchen **Sortieren**

Wenn der Benutzer das Kontrollkästchen **Sortieren** im Dialogfeld **Drucken** ausschaltet, werden die Seiten jeweils von der ersten bis zur letzten ausgegeben, wobei der Druckertreiber diesen Vorgang bei mehreren Kopien automatisch wiederholt. Ist das Kontrollkästchen eingeschaltet, wird jede Seite entsprechend der Anzahl Kopien mehrfach gedruckt, bis es zur nächsten Seite weitergeht. In Ihrem Code brauchen Sie dafür keine besonderen Vorkehrungen treffen - der Druckertreiber muß aber diese Funktion unterstützen. Andernfalls ist das Kontrollkästchen im Dialogfeld **Drucken** deaktiviert und läßt sich nicht nutzen.

Der Benutzer kann über das Dialogfeld die Druckeroptionen ändern. Die Einstellungen aktualisieren dann das Objekt `CPrintInfo`, bevor es an Ihre Anwendung übergeben wird. Sie können das Dialogfeld bei Bedarf mehr oder weniger anpassen. Das hängt von Ihren Anforderungen ab und vom Aufwand, den Sie in diesen Teil der Anwendung stecken wollen.

Wie Sie Tabelle C.1 entnehmen können, gehört zu den Klassenelementen von `CPrintInfo` ein Zeiger `m_pPD`. Dieser zeigt auf eine `CPrintDialog`-Klasse, die eine MFC- Wrapper-Klasse für das Dialogfeld **Drucken** darstellt. Diese Klasse enthält auch ein Element `m_pd`, eine `PRINTDLG`-Struktur mit den Standardeinstellungen, die im Dialogfeld **Drucken** erscheinen. In dieser Struktur sind zahlreiche Elemente enthalten, wie es Listing C.5 zeigt. Damit lassen sich die Vorgaben im Dialogfeld

vollständig anpassen, was sogar bis zu einer vollkommen anderen Vorlage gegenüber der Standardvorlage des Dialogfelds reicht (wenn Sie eine Herausforderung wünschen). Es würde den Umfang dieses Buches sprengen, auf alle Elemente im Detail einzugehen. Eines der herausragenden Elemente ist die Member-Variable `nCopies`. Die vorgegebene Anzahl der Kopien, die das Dialogfeld anzeigt, können Sie über das Element `nCopies` in der Struktur einstellen, unmittelbar bevor die Funktion `CView::DoPreparePrinting` aufgerufen wird. Fügen Sie dazu die folgende Zeile in die Funktion `OnPreparePrinting` ein:

```
pInfo->m_pPD->m_pd.nCopies = 15;
```

Wenn Sie das Dialogfeld **Drucken** öffnen, nachdem Sie diese Zeile hinzugefügt haben, ist die Anzahl der Kopien mit 15 vorgegeben (falls Ihr Drucker oder Druckertreiber mehrere Kopien unterstützt). Die anderen Vorgabewerte können Sie in der Struktur `PRINTDLG` sinngemäß einstellen.

	Die Struktur <code>DevMode</code>
	In der Struktur <code>DevMode</code> sind verschiedene Attribute zusammengefaßt, die die technischen Fähigkeiten und die Konfiguration des Geräts beschreiben. Die Funktion <code>GetDevMode</code> der Klasse <code>CPrintDialog</code> liefert einen Zeiger auf diese Struktur zurück.

Listing C.5: Die Struktur `PRINTDLG`

```
1: typedef struct tagPD {
2:     DWORD lStructSize;
3:     HWND hwndOwner;
4:     HANDLE hDevMode;
5:     HANDLE hDevNames;
6:     HDC hDC;
7:     DWORD Flags;
8:     WORD nFromPage;
9:     WORD nToPage;
10:    WORD nMinPage;
11:    WORD nMaxPage;
12:    WORD nCopies;
13:    HINSTANCE hInstance;
14:    DWORD lCustData;
15:    LPPRINTHOOKPROC lpfnPrintHook;
16:    LPSETUPHOOKPROC lpfnSetupHook;
17:    LPCTSTR lpPrintTemplateName;
18:    LPCTSTR lpSetupTemplateName;
19:    HANDLE hPrintTemplate;
20:    HANDLE hSetupTemplate;
21: } PRINTDLG;
```

Nachdem der Benutzer das Dialogfeld **Drucken** mit **OK** bestätigt hat, können Sie die Änderungen mit Hilfe der in Tabelle C.2 aufgeführten Zugriffsfunktionen der Klasse `CPrintDialog` abrufen. Wenn Sie zum Beispiel die festgelegte Anzahl der Kopien wissen möchten, die der Benutzer vor dem Ausdrucken festgelegt hat, können Sie auf diesen Wert zugreifen, nachdem er von der Funktion `CView::DoPreparePrinting` zurückgegeben wurde (siehe Listing C.6).

Natürlich können Sie auch alle anderen Werte in der `PRINTDLG`-Struktur `pInfo->m_pPD-m_pd` an dieser Stelle testen.

Tabelle C.2: Zugriffsfunktionen von CPrintDialog

Funktionsname	Beschreibung
GetCopies	Liefert die vom Benutzer eingestellte Anzahl der Kopien zurück.
GetFromPage	Liefert die angegebene erste Seite zurück.
GetToPage	Liefert die angegebene letzte Seite zurück.
GetPortName	Liefert den ausgewählten Druckeranschluß, beispielsweise LPT1:.
GetDriverName	Gibt den ausgewählten Druckertreiber (Zieldrucker) zurück.
GetPrinterDC	Gibt einen Gerätekontext für den Drucker zurück.
PrintAll	Liefert TRUE, wenn alle Seiten ausgewählt sind.
PrintCollate	Liefert TRUE, wenn Sortierung gefordert ist.
PrintRange	Liefert TRUE, wenn ein Bereich angegeben ist.
PrintSelection	Liefert TRUE, wenn eine bestimmte Auswahl von Seiten gewählt wird.

Listing C.6: Überprüfung des Standarddialogfelds Drucken nach einer bestimmten Anzahl von Kopien.

```

1: BOOL CPrintItView::OnPreparePrinting(CPrintInfo* pInfo)
2: {
3: pInfo->SetMinPage(1);
4: pInfo->SetMaxPage(10);
5:
6: pInfo->m_pPD->m_pd.nCopies = 3;
7:
8: do
9: {
10: // Testen, ob Benutzer den Druckvorgang abgebrochen hat
11: if (DoPreparePrinting(pInfo) == FALSE)
12: return FALSE;
13:
14: // Benutzer warnen, falls zu viele Kopien angegeben
15: if (pInfo->m_pPD->GetCopies()>5)
16: AfxMessageBox("Bitte nicht mehr als 5 Kopien wählen");
17:
18: // Schleife bis gültige Zahl eingegeben
19: } while(pInfo->m_pPD->GetCopies()>5);
20: return TRUE;
21: }

```

In Listing C.6 liefert die Funktion `CView::DoPreparePrinting` den Wert `FALSE` zurück, wenn der Benutzer auf **Abbrechen** klickt (Zeilen 11 und 12). Andernfalls prüft die Funktion die eingestellten Kopien in Zeile 15 und gibt eine Warnung aus, wenn mehr als fünf Kopien (ein willkürlich gewähltes Kriterium) angegeben sind. Nach Auswertung der Schleifenbedingung in Zeile 19 wird die Schleife solange fortgesetzt, bis der Benutzer eine gültige Anzahl von Kopien eingibt oder **Abbrechen** wählt.

Hochformat und Querformat

Wenn Sie im Menü **Datei** der Anwendung den Befehl **Seite einrichten** wählen, können Sie die vorgegebene Ausrichtung der Seite für den Drucker ändern. Das Dialogfeld bietet die Optionen **Hochformat** und **Querformat**. Um im Querformat zu drucken, sind keinerlei Änderungen am Code erforderlich. Wenn Sie diese Option wählen und dann eine Seitenvorschau starten, sollte der Gerätekontext nun in die gedrehte Seite zeichnen. Solange Ihre Anwendung das Element `rectDraw` des Objekts `CPrintInfo` beachtet, läßt sich das Drucken im Querformat automatisch bewerkstelligen.

GDI-Objekte mit `OnBeginPrinting` hinzufügen

Wie bereits erwähnt, funktioniert zwar der Code von Listing C.4, es gibt aber eine bessere Lösung, um die benötigten Ressourcen zuzuweisen. Momentan ruft das Programm für jede zu druckende Seite die Funktion `OnPrint` auf und erstellt alle Ressourcen von Grund auf neu. Bei dem einfachen Ausdruck des Beispiels wirkt sich das nicht aus, in großen und komplexen Berichten empfiehlt es sich aber, eine Reihe von Ressourcen und andere Berechnungen nur einmal beim Start des Berichts auszuführen. Dann druckt man mehrere Seiten und gibt die Ressourcen nach dem Ende des Berichts wieder frei.

Die virtuelle Funktion `OnBeginPrinting` stellt einen idealen Platz für diese Initialisierung dar. In der Schwesterfunktion `OnEndPrinting` gibt man dann die Ressourcen wieder frei.

Der Aufruf von `OnBeginPrinting` erfolgt nach `OnPreparePrinting`. Hier wird auch zum ersten Mal ein Druckergerätekontext übergeben. Genau dieser Gerätekontext kommt während des Druckvorgangs zum Einsatz, so daß man alle GDI-Objekte und die Koordinaten der Druckseite an diesem Punkt einrichten kann. Der vom Klassen- Assistenten automatisch erzeugte Standardcode liefert eine leere Funktion:

```
void CPrintItView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo*
/*pInfo*/)
{
// ZU ERLEDIGEN: Zusätzliche Initialisierung vor dem Drucken hier
einfügen
}
```

Sehen Sie sich die Funktionsdefinition genauer an. Die Parameter sind eigentlich für den Compiler auskommentiert, so daß Sie eine Warnung bezüglich nicht verwendeter Parameter erhalten, wenn Sie die Anwendung kompilieren. Denken Sie daran, die Kommentare bei diesen Parametern zu entfernen, bevor Sie sie verwenden.

Nunmehr können Sie die Aufrufe für das Erzeugen der GDI-Objekte in die Funktion einfügen, um das Ganze nicht bei jeder Seite erneut ausführen zu müssen:

```
m_fnTimes.CreatePointFont(720,"Times New Roman", pDC);
m_brHatch.CreateHatchBrush(HS_CROSS,RGB(64,64,64));
```

Beachten Sie, daß vor den Objekten `fnTimes` und `brHatch` das Präfix `m_` steht. Diese Namenskonvention weist darauf hin, daß die Objekte auf Klassenebene (in die Klasse eingebettet) und nicht nur lokal (in die Funktion eingebettet) gültig sind. Da man in `OnPrint` auf diese GDI-Objekte zugreifen muß, kann man sie in die Klassendeklaration aufnehmen. Fügen Sie dazu die Schrift- und Pinselobjekte wie folgt in die Klassendeklaration ein:

```
protected:
```

```
CFont m_fnTimes;
CBrush m_brHatch;
```

Die Objekte fügen Sie hinzu, indem Sie entweder im Arbeitsbereich auf der Registerkarte **Klassen** auf die Klasse **CPrintItView** doppelklicken und die Variablen direkt eintragen oder über das Dialogfeld **Member-Variable hinzufügen** in die Klasse aufnehmen.

Beachten Sie, daß der schraffierte Pinsel mit der Funktion **CreateHatchBrush** und nicht mit dem Konstruktor erzeugt wird. Das hat folgenden Grund: Der Pinsel existiert genauso lange wie die Ansicht, aber in der Funktion **OnBeginPrinting** müssen Sie **DeleteObject** aufrufen, so daß die zugrundeliegende GDI-Ressource zwischen zwei Druckausgaben freigegeben wird. Den Code zum Löschen der beiden GDI-Objekte (Schrift und Pinsel) realisieren Sie in der Funktion **OnEndPrinting** mit den folgenden Zeilen:

```
m_fnTimes.DeleteObject();
m_brHatch.DeleteObject();
```

Jetzt sind lediglich noch die lokalen GDI-Objekte von der Funktion **OnPrint** selbst zu entfernen und deren Referenzen durch die Versionen der Member-Variablen zu ersetzen. Dazu ersetzen Sie die lokalen Variablen **fnTimes** von **CFont** und **brHatch** von **CBrush** sowie deren Erstellungsfunktionen und selektieren einfach die vordefinierte Schrift und den Pinsel:

```
CFont* pOldFont = (CFont*)pDC->SelectObject(&m_fnTimes);
CBrush* pOldBrush = (CBrush*)pDC->SelectObject(&m_brHatch);
```

Wenn Sie die Anwendung nach diesen Änderungen erstellen und ausführen, bemerken Sie wahrscheinlich keinen Unterschied. Funktionell ist alles gleich geblieben, aber das Drucken und die Seitenansicht sollten etwas schneller arbeiten. Insbesondere bei einem großen, komplexen 100seitigen Bericht, der auf eine Menge von GDI-Ressourcen zurückgreift, werden Sie bei diesem Verfahren zweifellos eine höhere Druckgeschwindigkeit registrieren.



Die Koordinaten aus OnBeginPrinting

Vielleicht haben Sie versucht, auch die Koordinaten von **OnBeginPrinting** zu speichern. Das funktioniert nicht, weil das Element **m_rectDraw** von **CPrintInfo** in dieser Phase nicht initialisiert wurde und zufällige Koordinaten zurückkommen.

Die Vorbereitung des Gerätekontextes anpassen

Vor dem Aufruf der beiden Funktionen **OnDraw** und **OnPrint** wird die virtuelle Funktion **OnPrepareDC** aufgerufen, die Sie in Ihrer Ansichtsklasse überschreiben können, um Modifikationen am Gerätekontext vorzunehmen, die sowohl für **OnDraw** als auch **OnPrint** zutreffen.

Beispielsweise kann man die Abbildungsmodi oder bestimmte Zeichenmodi einstellen, die sowohl für die Darstellung auf dem Bildschirm als auch den Ausdruck gültig sind. Der Anwendungs-Assistent stellt die überschriebene Funktion nicht bereit, man kann sie aber einfach über das Dialogfeld **Virtuelle Funktion hinzufügen** aufnehmen. Im Beispiel ist den beiden Funktionen **OnDraw** und **OnPrint** die Funktion **SetTextAlign** des Gerätekontextes gemeinsam. In eine **OnPrepareDC**-Funktion können Sie das folgendermaßen hinzufügen:

```
void CPrintItView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    pDC->SetTextAlign(TA_CENTER+TA_BASELINE);
}
```

}

Insbesondere bei der Vorbereitung von WYSIWYG-Ausdrucken kann es vorteilhaft sein, die Abbildungsmodi und Fensterdimensionen in einer gemeinsamen Funktion einzurichten, bevor man die Zeichen- oder Druckfunktion aufruft. Hier bietet sich `OnPrepareDC` an, um den speziellen Initialisierungscode für den Gerätekontext unterzubringen.

Den Druckauftrag abbrechen

Ein weiterer Einsatzfall der Funktion `OnPrepareDC` ist der Aufruf von Escape-Sequenzen für den Drucker oder anderer dokumentspezifischer Funktionen. Wenn Sie einen besonders langen Bericht haben, wollen Sie dem Benutzer sicherlich die Möglichkeit bieten, den Druckvorgang zu beenden und das Drucken abubrechen. Die Funktion `AbortDoc` des Gerätekontextes bricht das Drucken des Dokuments für einen Druckergerätekontext ab. Probieren Sie es aus, indem Sie die folgenden Zeilen in `OnPrepareDC` einfügen und den Druck des Dokuments nach drei Seiten abbrechen:

```
if (pDC->IsPrinting())
if (pInfo->m_nCurPage==3) pDC->AbortDoc();
```

Direktes Drucken ohne Programmgerüst

Dieser Anhang ist bisher nur auf die Unterstützung der Druckfunktionen in SDI- und MDI-Gerüsten eingegangen. Diese Unterstützung fügt sich zwar harmonisch in die Dokument/Ansicht-Architektur ein, es kann aber vorkommen, daß man nur einfach mal so auf den Drucker zugreifen möchte oder kein Programmgerüst zur Hand hat - beispielsweise in einer dialogbasierten Anwendung.

Das Programmgerüst verbirgt die systemnahe Druckunterstützung, die den Grundstein für alle Druckoperationen legt. Dieser Abschnitt erklärt, wie diese Unterstützung arbeitet, und zeigt den Einsatz in einem dialogbasierten Anwendungsbeispiel.

Das Dialogfeld Drucken direkt aufrufen

Im Abschnitt »Das Dialogfeld Drucken« weiter vorn in diesem Anhang haben Sie gesehen, wie die Klasse `CPrintDialog` einen Wrapper für das Standarddialogfeld `PRINTDLG` bereitstellt und wie man diesen von `CView::DoPreparePrinting` aufruft.

Das gleiche Dialogfeld und die gleiche Klasse kann man direkt verwenden, um den Zieldrucker und dessen Standardeinstellungen einzurichten, genauso wie man ein normales modales Dialogfeld einsetzt. Es lassen sich auch die gleichen Zugriffsfunktionen nutzen, um die Standardwerte für Seitennummern und Anzahl der Kopien festzulegen, wie Sie es bereits beim Aufruf der Funktion `DoPreparePrinting` des Programmgerüsts kennengelernt haben.

Listing C.7 zeigt dieses Dialogfeld, über das man bei dialogbasierten Anwendungen direkt den Drucker konfigurieren und dann ein kleines Dokument mit den Standardeinstellungen des Dialogfelds drucken kann.

Der direkte Druckvorgang arbeitet mit den in diesem Listing vorgeführten Funktionen `StartDoc` und `EndDoc`. Der nächste Abschnitt geht näher darauf ein.

Erstellen Sie als Beispiel mit dem Anwendungs-Assistenten eine dialogfeldbasierende Anwendung namens `DlgPrint`, und erzeugen Sie mit dem Klassen-Assistenten eine `OnOK`-Behandlungsroutine,

um den Druckercode zu implementieren, wie es Listing C.7 zeigt.

Listing C.7: DlgPrintDlg.cpp - Das direkte Ausdrucken eines Dokuments in OnOK einer dialogfeldbasierenden Anwendung implementieren.

```

1: void CDlgPrintDlg::OnOK()
2: {
3: // TODO: Zusätzliche Prüfung hier einfügen
4:
5: // Ein CPrintDialog-Objekt konstruieren
6: CPrintDialog dlgPrint(FALSE, PD_ALLPAGES, this);
7:
8: if (dlgPrint.DoModal()==IDOK)
9: {
10: // Den Drucker-DC vom Dialogfeld mit einem
11: // CDC-Objekt verbinden
12: CDC dcPrint;
13: dcPrint.Attach(dlgPrint.GetPrinterDC());
14:
15: // Eine DOCINFO-Struktur erzeugen und füllen
16: DOCINFO myPrintJob;
17: myPrintJob.cbSize = sizeof(myPrintJob);
18: myPrintJob.lpszDocName = "MeinDruckAuftrag";
19: myPrintJob.lpszOutput = NULL;
20: myPrintJob.lpszDatatype = NULL;
21: myPrintJob.fwType = NULL;
22:
23: // Ausdruck des Dokuments starten
24: if (dcPrint.StartDoc(&myPrintJob)>=0)
25: {
26: // Eine Seite beginnen
27: dcPrint.StartPage();
28:
29: // Zeichnen beginnen
30: dcPrint.TextOut(0,0,"Mein kleiner Druckauftrag");
31:
32: // Seite auswerfen
33: dcPrint.EndPage();
34:
35: // Dokument schließen
36: dcPrint.EndDoc();
37: }
38:
39: // Druckergerätekontext löschen
40: dcPrint.DeleteDC();
41: }
42:
43: // Mit Standardverarbeitung von OnOK fortfahren
44: CDialog::OnOK();
45: }

```

Listing C.7 deklariert in Zeile 6 ein `CPrintDialog`-Objekt, das im Konstruktor drei Parameter übernimmt. Der erste Parameter ist ein Flag, das man auf `TRUE` setzt, um das Dialogfeld **Seite einrichten** anzuzeigen. Bei `FALSE` erscheint das Dialogfeld **Drucken**. Der zweite Parameter stellt eine Gruppe kombinierbarer Flags dar, mit denen sich die Einstellungen des Dialogfelds (zu umfangreich, um sie hier zu behandeln) festlegen lassen. Der dritte Parameter ist ein Zeiger auf das übergeordnete Fenster. In diesem Fall kennzeichnet der C++-Zeiger `this`, daß das Dialogfeld das übergeordnete Fenster ist.

Zeile 8 ruft `dlgPrint.DoModal` auf, um dieses Dialogfeld anzuzeigen. Wenn der Benutzer auf **OK** klickt, beginnt das Drucken. Andernfalls wird der Block übersprungen.

Wenn der Benutzer im Dialogfeld **Drucken** auf **OK** geklickt hat, wird ein Gerätekontext für den Drucker erzeugt und in Zeile 13 mit einem `CDC`-Objekt verbunden, um bequemer darauf zugreifen zu können. Denken Sie daran, den Gerätekontext selbst zu löschen, wie es Zeile 40 zeigt.

Fügen Sie den Code des Listings und die Behandlungsroutine in Ihre Beispielanwendung ein, erstellen und starten Sie sie, und klicken Sie im Dialogfeld der Anwendung auf **OK**, um den neuen Code auszuprobieren.

Die Funktionen `StartDoc` und `EndDoc`

Der `CDC`-Gerätekontext umfaßt viele druckerspezifische Funktionen. Bei einem neuen Ausdruck muß Windows ein Spool-Dokument erzeugen, um den Druckauftrag zu speichern und ihn bei Vollständigkeit an den Drucker abzuschicken. Die Funktion `StartDoc` weist Windows an, das Spooling zu starten, während die Funktion `EndDoc` anzeigt, daß das Dokument vollständig ist und an den Drucker geschickt werden kann. Weiter vorn in diesem Anhang haben Sie bereits die Funktion `AbortDoc` kennengelernt, die das Drucken abbricht und den Druckauftrag storniert, statt ihn zum Drucker zu senden.

Listing C.7 ruft in Zeile 24 die Elementfunktion `StartDoc` des Druckergerätekontextobjekts `dcPrint` auf und übergibt dabei einen Zeiger an eine `DOCINFO`-Struktur. Diese Struktur speichert die Details des Druckauftrags. Als einzige Angabe müssen Sie einen Namen für das Spool-Dokument festlegen, was in Zeile 18 geschieht. Beachten Sie das unübliche Element `cbSize`, das die Größe der Struktur speichert. Zeile 17 weist diesem Element den Wert aus `sizeof(myPrintJob)` zu. Eine derartige Konstruktion finden Sie auf der Ebene des Win32-API häufig, weil `DOCINFO` eine alte Struktur im Stil von C ist. Hier wird `cbSize` verwendet, da es unterschiedliche Formen von `DOCINFO` gibt und man diese nur über die Größe auseinanderhalten kann.

Beim Aufruf von `StartDoc` versucht die Funktion, den Druckvorgang zu starten und liefert im Erfolgsfall einen positiven Wert zurück. Die Gründe für ein Scheitern können unterschiedlichster Art sein, beispielsweise zu geringer Platz auf dem Datenträger oder im Hauptspeicher, oder ein beschädigter Druckertreiber, so daß man den Druckvorgang nur fortsetzen sollte, nachdem man den Rückgabecode ausgewertet hat.

Nach dem Ausdruck des Dokuments rufen Sie `EndDoc` auf, wie es Zeile 36 zeigt, um den eigentlichen Ausdruck des Dokuments in die Wege zu leiten.



Den Windows-Spooler überwachen

Den Aufbau des Druckdokuments können Sie verfolgen, indem Sie einen Haltepunkt in die Funktion `OnPrint` oder nach der Funktion `StartDoc` setzen und das Symbol für den Druckerstatus aus der Druckergruppe öffnen, die man über das Windows **Start**-Menü unter **Einstellungen** findet.

Die Funktionen `StartPage` und `EndPage`

Ein weiteres Funktionspaar für den Druckergerätekontext sind `StartPage` und `EndPage`. Mit der Funktion `StartPage` initialisiert man den Gerätekontext, um eine neue Seite zu drucken. Dabei werden einige Einstellungen des Gerätekontextes wie zum Beispiel die aktuelle Position des Grafikcursors zurückgesetzt und die Spooling- Informationen für den Beginn einer neuen Seite gesetzt.

Normalerweise ruft man `StartPage` auf, zeichnet etwas in den Gerätekontext, um die Details der zu druckenden Seite auszugeben, und ruft `EndPage` auf, um die Seite in die Spool-Datei zu schreiben und sie zum Druckdokument hinzuzufügen.

In Listing C.7 steht der Aufruf von `StartPage` in Zeile 27, gefolgt von einer einzelnen Funktion `TextOut`, die etwas auf der zu druckenden Seite ausgibt, und einem Aufruf der Funktion `EndPage` in Zeile 33.

Beim Aufruf von `EndPage` werden die speziellen Druckercode für einen Formularvorschub (Form Feed) an den Spooler gesendet, und das Spool-Dokument registriert eine weitere Druckseite. Die Folge `StartPage` und `EndPage` können Sie für alle Dokumentseiten wiederholen, bevor Sie mit `EndDoc` den Druckvorgang abschließen. Den Druckergerätekontext können Sie für das Zeichnen zwischen den Aufrufen von `StartPage` und `EndPage` in genau der gleichen Weise verwenden, wie Sie `OnPrint` in der SDI-Anwendung eingesetzt haben. Das SDI-Gerüst ruft die gleichen Funktionen auf, verbirgt sie allerdings, und es erscheinen nur die Aufrufe der Funktion `OnPrint` zwischen den Aufrufen für die erste und letzte Seite.

❖ Kapitel Inhalt Index **SAMS** ❖ Top Kapitel ❖

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](http://www.mut.com/lesecke/buecher/vcplus6/kap_c.htm).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Tag D

Ausnahmebehandlung

Ausnahmen

Eine **Ausnahme** (englisch Exception) ist ein Objekt, das Einzelheiten über fehlerhafte Operationen speichert. Das Raffinierte der Ausnahmebehandlung besteht darin, daß man bei einem Fehler in einer Funktion auf niedriger Ebene eine Ausnahme erzeugen und automatisch nach oben bis zu einer aufrufenden Funktion durchreichen lassen kann, wo sich dann ein spezieller Codeabschnitt mit allen derartigen Ausnahmen an einer zentralen Stelle beschäftigt.

Code ausführen und Fehler abfangen

Das System erkennt automatisch bestimmte Fehlerbedingungen und generiert dafür Ausnahmen. Wenn Sie sich in Ihrer Anwendung nicht selbst darum kümmern, »klettern« die Ausnahmen aus Ihrem Code heraus und werden von den Ausnahmebehandlungsmechanismen des Betriebssystems Windows abgefangen. Um sich davon ein Bild zu machen, fügen Sie einfach die folgenden beiden Zeilen in einen beliebigen Codeabschnitt ein und führen ihn aus:

```
CDC* pNullDC = 0;  
pNullDC->SetPixel(0, 0, 0);
```

Die erste Zeile deklariert einen Gerätekontextzeiger `pNullDC` und setzt ihn auf die Speicheradresse null (was für ein Objekt ein denkbar ungeeigneter Platz ist). Auch wenn es offensichtlich an dieser Adresse kein gültiges Objekt gibt, versucht das System beim darauffolgenden Aufruf der Funktion `SetPixel`, das Objekt an der Adresse null zu finden. Die Hardware und Software zur Speicherverwaltung wissen, daß das Programm kein Recht hat, an dieser Adresse etwas zu unternehmen, und lösen eine Zugriffsverletzung für den Speicher aus.

Wenn Sie diese Codezeilen außerhalb des Visual C++-Debuggers ausführen, erscheint ein Dialogfeld, das allen Windows-Benutzern vertraut sein dürfte und in Abbildung D.1 zu sehen ist.

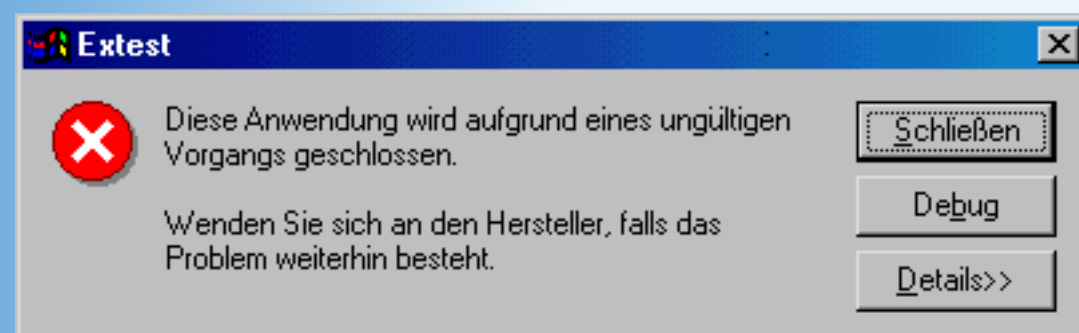
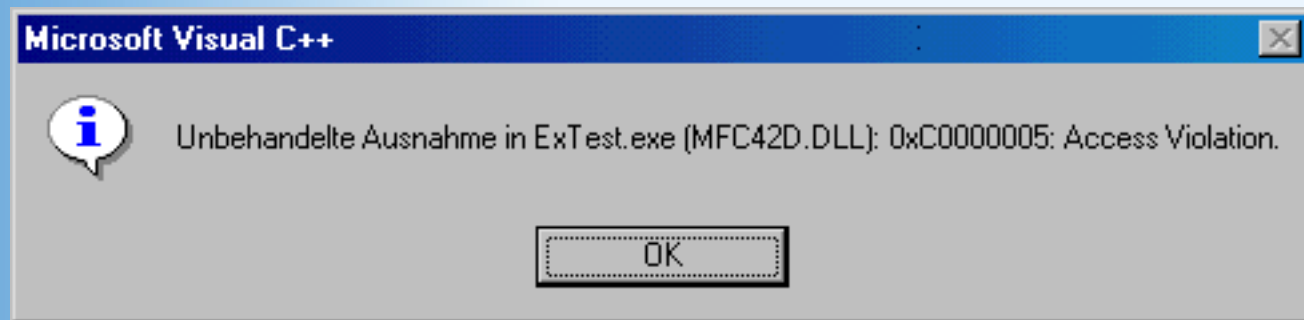


Abbildung D.1:**Das bekannte Dialogfeld bei Zugriffsverletzungen des Speichers**

Wenn Sie dagegen die Anwendung über den Visual C++-Debugger ausführen, fängt der Debugger zuerst die Ausnahme für Sie ab und zeigt statt dessen das Dialogfeld des Visual Studios (siehe Abbildung D.2).

**Abbildung D.2:****Die vom Visual Studio behandelte Ausnahme der Zugriffsverletzung**

Zugriffsverletzungen des Speichers sind sehr schwerwiegende Ausnahmen, die Ihr Programm zum Absturz bringen, ohne daß eine Chance besteht, sie abzufangen. Es gibt viele weniger schwere Ausnahmen, wie etwa die Ausnahme `CFileNotFoundException` für die Arbeit mit Dateien. Diese Ausnahme wird ausgelöst, wenn fehlerhafte Dateioperationen auftreten, beispielsweise das Suchen nach dem Beginn einer nicht geöffneten Datei:

```
CFile fileNotOpen;
fileNotOpen.SeekToBegin();
```

In diesem Fall erscheint ein vom System generiertes Dialogfeld (siehe Abbildung D.3). Wenn Sie auf **OK** klicken, setzt Ihr Programm wie üblich fort.

**Abbildung D.3:****Das Windows-Standarddialogfeld für eine Dateiausnahme**

Statt dem System alles zu überlassen, können Sie die Ausnahme selbst abfangen und in einer weit sanfteren Weise behandeln. Zu diesem Zweck stehen in C++ die Schlüsselwörter `try` und `catch` zur Verfügung. Beide spielen zusammen: zuerst definieren Sie einen Codeblock mit `try` und dann, wenn eine spezifizierte Ausnahme auftritt, wird die im `catch`-Block definierte Aktion ausgeführt (siehe Listing D.1).

Listing D.1: CFileExceptions mit einem try- und einem catch-Block abfangen

```
1: // Einen Codeblock probieren (try)
2: try
3: {
4: CFile fileNotOpen;
```

```

5: fileNotOpen.SeekToBegin();
6: }
7: catch(CFileException* e) // Dateiausnahmen abfangen (catch)
8: {
9: // Ursache der Ausnahme untersuchen
10: if (e->m_cause == CFileException::fileNotFound)
11: AfxMessageBox("Oje, vergessen, die Datei zu öffnen!");
12: e->Delete();
13: }

```

In Listing D.1 sind die Dateioperationen der Zeilen 4 und 5 in einen `try`-Block eingeschlossen. Wenn in diesen Zeilen keine Ausnahme auftritt, setzt der Code ganz normal fort. Wird allerdings eine `CFileException`-Ausnahme ausgelöst, fängt sie das Schlüsselwort `catch` in Zeile 7 ab, und die Variable `e` zeigt dann auf die neue Ausnahme. Zum `CFileException`-Objekt gehört ein `m_cause`-Code (Ursachencode), der genau definiert, warum die Ausnahme aufgetreten ist. Die Prüfung des Codes findet in Zeile 10 statt, und wenn es sich um den Code `CFileException::fileNotFound` handelt, bringt Zeile 11 das Meldungsfeld auf den Bildschirm.

Beachten Sie, daß die Member-Funktion `Delete` der Klasse `CException` (der Basisklasse von `CFileException`) die Ausnahme in Zeile 12 für Sie löscht. Es ist zu gewährleisten, daß Ausnahmen immer gelöscht werden, wenn die Arbeit damit beendet ist.

Der `try`-Block kann Aufrufe an andere Funktionen enthalten und dazu verwendet werden, alle spezifizierten Ausnahmen in einem großen Codeabschnitt der Anwendung abzufangen, wie es Listing D.2 zeigt.

Listing D.2: Ein try-Block kann viele Funktionsaufrufe und Aufrufe aus diesen Funktionen umfassen.

```

1: try
2: {
3: // ... Eine Menge Code
4: DoLotsOfFileHandling();
5: // ... Mehr Code
6: EvenMoreFileHandling();
7: // ... Und noch mehr Code
8: }
9: catch(CFileException* e) // Dateiausnahmen abfangen
10: {
11: // Ursache der Ausnahme untersuchen
12: if (e->m_cause == CFileException::fileNotFound)
13: AfxMessageBox("Oje, vergessen, die Datei zu öffnen!!");
14: e->Delete();
15: }

```

In Listing D.2 könnte die Funktion `DoLotsOfFileHandling` in Zeile 4 bestimmte Dateibehandlungen selbst erledigen und auch andere Funktionen aufrufen, wie es Zeile 6 mit `EvenMoreFileHandling` demonstriert. Sollte eine Dateiausnahme bei einer dieser Dateioperationen auftreten, wandert die Ausnahme nach oben, so daß derselbe `catch`-Block in den Zeilen 9 bis 13 ausgeführt wird, wobei `e` auf das `CFileException`-Objekt zeigt. Schließlich

löscht Zeile 14 die Ausnahme.

Wenn Sie zwei unterschiedliche Ausnahmen vom `try`-Block heraus abfangen möchten, können Sie `catch`-Blöcke hinzufügen, um die einzelnen Ausnahmen getrennt zu behandeln, wie es Listing D.3 verdeutlicht.

Listing D.3: Zwei verschiedene Ausnahmen mit zwei spezialisierten `catch`-Blöcken behandeln.

```

1: try
2: {
3: // Diese Dateioperation ist OK.
4: CMemFile fileMemFile;
5: fileMemFile.SeekToBegin();
6:
7: // Man kann aber keine zwei verschiedenen
8: // Systemressourcen mit demselben Namen haben.
9: CMutex mutex1(0, "Derselbe Name");
10: CSemaphore semaphore1(1,1, "Derselbe Name");
11: }
12: catch(CFileException* e) // Dateiausnahmen abfangen
13: {
14: if (e->m_cause == CFileException::fileNotFound)
15: AfxMessageBox("Oje, vergessen, die Datei zu öffnen!");
16: e->Delete();
17: }
18: catch(CResourceException* e) // Ressourcenausnahmen abfangen
19: {
20: // Ausnahmefehler der Ressourcen melden
21: AfxMessageBox("Oha, doppelter Ressourcenname ");
22: e->Delete();
23: }

```

Listing D.3 erzeugt die Datei in Zeile 4 automatisch, so daß Zeile 5 keine Dateiausnahme auslöst. Da jedoch zwei unterschiedliche Systemressourcen (ein Mutex und ein Semaphor) denselben Namen erhalten, bewirkt das eine `CResourceException`-Ausnahme in Zeile 10, der sich dann der zweite `catch`-Block in Zeile 18 widmet und das Meldungsfeld in Zeile 21 anzeigt. Wenn Sie diesen Code ausprobieren, denken Sie daran, die Anweisung `#include <afxmt.h>` für die Definitionen von `CMutex` und `CSemaphore` vorzusehen.

Wenn Sie Ausnahmen mit einem Rundumschlag abfangen wollen, brauchen Sie nicht für jeden Typ von Ausnahme einen eigenen `catch`-Block vorzusehen, sondern können die Ausnahme der Basisklasse `CException` abfangen. Von dieser Basisklasse sind alle anderen speziellen Ausnahmeklassen abgeleitet (siehe Listing D.4).

Listing D.4: Mit einem `catch`-Block alle Arten von Ausnahmen abfangen

```

1: // Codeblock ausprobieren
2: try
3: {
4: // Eine Menge Code...
5: }
6: catch(CException* e)

```



```

7: {
8: // Allgemeine Fehlermeldung, Einzelheiten in e
9: AfxMessageBox("Irgendwas ist schiefgegangen!");
10: e->Delete();
11: }

```

In Zeile 6 kommt die Basisklasse `CException` statt einer speziellen Ausnahme wie `CFileException` oder `CResourceException` zum Einsatz. Den Typ der Anwendung können Sie mit der Funktion `IsKindOf` innerhalb des `catch`-Blocks testen. Mit den folgenden Zeilen läßt sich zum Beispiel herausfinden, ob eine Dateiausnahme ausgelöst wurde:

```

if (e->IsKindOf(RUNTIME_CLASS(CFileException)))
AfxMessageBox("File Exception");

```

Da Ausnahmen von `CObject` abgeleitet sind, unterstützen sie die Informationen der MFC-Laufzeitklassen. Mit Hilfe von `DECLARE_DYNAMIC` und `IMPLEMENT_DYNAMIC` werden die Klasseninformationen im abgeleiteten Ausnahmeobjekt zusammengefaßt, so daß man mit der Funktion `IsKindOf` auf einen speziellen Klassentyp prüfen kann. Das Makro `RUNTIME_CLASS` verwandelt Klassennamen in einen Zeiger auf ein `CRuntimeClass`-Objekt für das angegebene Objekt. Die Member-Funktion `IsKindOf` gibt dann `TRUE` zurück, wenn das aufgerufene Objekt zu dieser Laufzeitklasse gehört.

Der Abschnitt »MFC-Ausnahmetypen« weiter hinten in diesem Anhang geht darauf ein, wie man ausnahmespezifische Informationen von allen MFC-Ausnahmetypen in einem `catch`-Block abfangen kann.

Systemressourcen freigeben

Dieses Prinzip zum Abfangen von Ausnahmen erweist sich als nützlich, wenn man Fehler aus umfangreichen Codeabschnitten erkennen und behandeln will. Man kann sich dadurch eine Menge Zeilen für individuelle Fehlerbehandlungsroutinen sparen, aber man muß dennoch alle Systemressourcen freigeben, die man in den Zeilen vor dem Auftreten der Ausnahme zugewiesen hat.

Ausnahmen auslösen

Wenn man einen bestimmten Codeabschnitt in einen `try`-Block einschließt, kann man Ausnahmen abfangen, die in diesem Codeabschnitt entstehen. Der zugehörige `catch`-Block behandelt dann die Ausnahme. Man kann auch die Ausnahme innerhalb eines `catch`-Abschnitts erneut zu einem höhergelegenen `catch`-Abschnitt, der den ersten einschließt, auslösen.

Verschiedene `AfxThrow...`-Funktionen generieren automatisch verschiedene Arten von MFC-Ausnahmen und lösen sie bis zur nächsten `catch`-Ebene aus, beispielsweise `AfxThrowFileException` oder `AfxThrowMemoryException`. Darauf geht der Abschnitt »MFC-Ausnahmetypen« näher ein. Allerdings erzeugen diese Funktionen - mit dem Schlüsselwort `new` von C++ - eine neue Instanz eines speziellen, von `CException` abgeleiteten Objekts. Die Ausnahme wird dann mit dem Schlüsselwort `throw` ausgelöst, wie es das Codefragment in Listing D.5 zeigt.

Listing D.5: Eine Ausnahme mit dem Schlüsselwort `throw` auslösen

```

1: try
2: {
3: DoSomeFileHandling();

```



```

4: }
5: catch(CFileException* e)
6: {
7: e->ReportError();
8: e->Delete();
9: }
10:
11: return TRUE;
12: }
13:
14: BOOL bSomethingWentWrong = TRUE;
15:
16: void CExceptionalDlg::DoSomeFileHandling()
17: {
18: // Funktionen zur Dateibehandlung
19: if (bSomethingWentWrong == TRUE)
20: {
21: CFileException* pException =
22: new CFileException(CFileException::generic);
23: throw(pException);
24: }
25:
26: // Weitere Funktionen zur Dateibehandlung
27: }

```

In Listing D.5 umschließt der `try`-Block einen Aufruf der Funktion `DoSomeFileHandling` in Zeile 16. Diese Funktion kann bestimmte Prozeduren zur Dateibehandlung implementieren und eine Ausnahme auslösen, wenn die Fehlerbedingung in Zeile 19 den Wert `TRUE` ergibt. Zeile 22 erzeugt ein neues `CFileException`-Objekt mit Übergabe des Flags `CFileException::generic` an den Konstruktor und löst dann in Zeile 23 das neue Objekt aus, das der `catch`-Abschnitt in Zeile 5 abfangen soll.

Das Erzeugen eines neuen, von `CException` abgeleiteten Objekts und die darauffolgende Verwendung des Schlüsselwortes `throw` bildet die Grundlage des Auslösemechanismus für Ausnahmen. Die speziellen Details, die die Ursache des Fehlers kennzeichnen, können dem `CException`-Objekt zugeordnet werden, oder man kann zusätzliche Informationen hinzufügen, indem man eine Klasse von der Basisklasse `CException` ableitet und zusätzliche Variablen zum Speichern weiterer Informationen aufnimmt.

Der `catch`-Block kann dann entscheiden, ob der Fehler zu schwerwiegend ist, um auf dieser Ebene behandelt zu werden. In diesem Fall lässt sich die Ausnahme zu einer höheren Ebene eines umschließenden `catch`-Blocks erneut auslösen. Man kann das Schlüsselwort `throw` (ohne Parameter) aus einem `catch`-Block heraus verwenden, um die Ausnahme erneut auszulösen, bevor man die Ausnahme löscht. Beispielsweise lässt sich in Listing D.5 das Schlüsselwort `throw` einbauen, um die Ausnahme zu einem `catch`-Block auf höherer Ebene neu auszulösen statt die Ausnahme zu löschen:

```

e->ReportError();
throw;

```

Nach der Meldung des Fehlers wird dann die Ausnahme erneut für einen umschließenden `try`-Block

zum Abfangen ausgelöst. Wenn Sie diese Verschachtelung nicht implementiert haben, fängt die globale MFC außerhalb des `catch`-Blocks die Ausnahme ab. Mit diesem Verschachtelungsmechanismus kann man die Schwere des Fehlers bestimmen und geeignete Aktionen zur Wiederherstellung auf verschiedenen Hierarchieebenen des Programms implementieren.

Ausnahmen löschen

Wie Sie gesehen haben, sind Sie grundsätzlich dafür verantwortlich, neue Ausnahmen zu erzeugen, und Sie müssen auch diese Objekte löschen, wenn Sie die Ausnahmen behandelt haben. Wenn Sie eine der MFC-Ausnahmen löschen, sollten Sie nicht das normale C++-Schlüsselwort `delete` verwenden (wie bereits gezeigt), da die Ausnahme ein globales Objekt oder ein Objekt auf dem Heap sein kann. Zu diesem Zweck verfügt die Basisklasse `CException` über eine `Delete`-Funktion, die zuerst prüft, ob die Ausnahme zu löschen ist. Der Erzeuger der Ausnahme kann festlegen, ob die Ausnahme zu löschen ist, indem er `TRUE` als (einzigen) Parameter `b_AutoDelete` an den Konstruktor der Klasse `CException` übergibt.

MFC-Ausnahmetypen

Die Microsoft Foundation Classes verfügen über verschiedene vordefinierte, von `CException` abgeleitete Klassen, die während der verschiedenen Arten von MFC-Operationen zum Einsatz kommen. Davon haben Sie bereits `CFileException` und `CResourceException` kennengelernt. Der folgende Abschnitt geht näher auf diese verschiedenen Klassen und die Auslösung der Ausnahmen ein. Jede Klasse basiert auf der Klasse `CException` und erweitert die Funktionalität von `CException` für unterschiedliche Arten der Ausnahmebehandlung. Sie können auch eigene Ausnahmeklassen von `CException` ableiten. Eine allgemeine `CUserException` ist für benutzerorientierte Anwendungsausnahmen vorgesehen.

Die Basisklasse CException

`CException` selbst hat einen Konstruktor, der das bereits erwähnte Flag `AutoDelete` übernimmt und wie folgt definiert ist:

```
CException(BOOL b_AutoDelete);
```

Wenn man eine Instanz von `CException` oder einer abgeleiteten Klasse mit `new` erzeugt, sollte man sicherstellen, daß das Flag auf `TRUE` gesetzt ist, damit die Ausnahme mit dem C++-Schlüsselwort `delete` gelöscht wird. Andernfalls sollte eine globale oder stackbasierte Ausnahme `TRUE` übergeben, so daß sie nur dann gelöscht wird, wenn sie den Gültigkeitsbereich verliert (am Ende einer Funktion oder eines Programms, wo die Klasse deklariert ist).

Die Basisklasse enthält die Funktion `Delete` und zwei Funktionen zur Fehlermeldung. Mit der Funktion `GetErrorMessage` kann man die Fehlermeldung in einem vordefinierten Puffer speichern und die ID einer Hilfsmeldung festlegen, die dem Benutzer die zum Fehler gehörende kontextabhängige Hilfe anzeigt. Der erste Parameter der Funktion ist die Adresse eines Zielpuffers, der die zugeordnete Fehlermeldung aufnimmt. Der zweite Parameter legt die Maximalgröße des Puffers fest, damit die im Puffer gespeicherten Meldungen nicht den Pufferbereich überschreiten. Mit dem dritten, optionalen Parameter können Sie die ID der kontextabhängigen Hilfe als `UINT`-Wert spezifizieren.

Mit Hilfe dieser Funktion läßt sich eine Fehlermeldung formatieren, die sich mehr auf die Anwendung

bezieht:

```
char msg[512];
e->GetErrorMessage(msg, sizeof(msg));
CString strMsg;
strMsg.Format("In MyApp ist folgender Fehler aufgetreten: %s", msg);
AfxMessageBox(strMsg);
```

Der C++-Operator `sizeof` in der Funktion `GetErrorMessage` liefert die Größe eines Arrays oder einer Variablen zurück. Wenn Sie also das `msg`-Array ändern, brauchen Sie keine weiteren Änderungen im Code vornehmen. Die Meldung wird dann in das `CString`-Objekt `strMsg` formatiert und in einem Meldungsfeld angezeigt.

Die Funktion `ReportError` zeigt den Meldungstext direkt im bekannten Meldungsfeld für Ausnahmen an und wird vom `catch`-Block verwendet:

```
e->ReportError();
```

Speicherausnahmen

Die Ausnahme `CMemoryException` wird automatisch ausgelöst, wenn die Ausführung einer Operation mit dem C++-Schlüsselwort `new` scheitert. Diese Ausnahme können Sie auch selbst mit Hilfe der Funktion `AfxThrowMemoryException` auslösen. Die Bedeutung dieser Ausnahme bezieht sich ausschließlich darauf, daß Windows keinen weiteren Hauptspeicher über `GlobalAlloc` oder andere Funktionen zur Speicherreservierung bereitstellen kann. Das ist für jedes Programm eine höchst unerfreuliche Situation. Gewöhnlich behandelt man diese Ausnahme so, daß das Programm in gepflegter Manier beendet wird, wobei man alle Speicher- und Systemressourcen freigibt. In seltenen Fällen kann man eine Wiederherstellung versuchen, wenn man einen großen reservierten Speicherblock zur Verfügung hat und ohne Schaden für die Aktivitäten des Benutzers freigeben kann.

Aufgrund dieser einzigartigen Ausnahmeform erweitern keinerlei Ursachenattribute oder spezielle Funktionen die Funktionalität der Klasse `CException`.

Die automatische Auslösung der Ausnahme `CMemoryException` bei einer Zuweisung mit `new` können Sie mit den folgenden Zeilen nachvollziehen:

```
MEMORYSTATUS mem;
GlobalMemoryStatus(&mem);
BYTE* pBig = new BYTE[mem.dwAvailVirtual+1];
```

Das Strukturelement `mem.dwAvailVirtual` von `MEMORYSTATUS` nimmt den gesamten verfügbaren Speicher auf, nachdem die Funktion `GlobalMemoryStatus` die Details abgerufen hat. Das `new` auf der nächsten Zeile fordert ein Byte mehr als möglich an und löst damit die Ausnahme aus.

Ressourcenausnahmen

Die Ausnahme `CResourceException` wird an vielen Stellen ausgelöst, wo Systemressourcen knapp werden, wie Sie im Beispiel mit Mutex und Semaphoren in Listing D.3 gesehen haben. Wenn Sie diese Ausnahmen selbst auslösen möchten, verwenden Sie die korrespondierende Funktion `AfxThrowResourceException`.

Windows kann die angeforderte Ressource nicht finden oder bereitstellen und gibt keine weitere

Hilfestellung. Daher hat die Klasse keine weiteren Funktionen oder Attribute.

Datei- und Archivausnahmen

In Listing D.5 haben Sie bereits `CFileException` kennengelernt. Das ist wahrscheinlich eine der komplizierteren MFC-Ausnahmen, da beim Dateizugriff eine ganze Menge schief gehen kann. Die Ausnahme können Sie selbst mit der Funktion `AfxThrowFileException` auslösen. Die Funktion übernimmt drei Parameter, einen erforderlichen und zwei optionale. Der erste, erforderliche Parameter, `cause`, gibt den Ursachencode für die Ausnahme an. Dieser Code wird in die Member-Variable `m_cause` der Dateiausnahme gestellt, damit man darauf in einem `catch`-Block zurückgreifen kann.

Tabelle D.1 zeigt eine Liste der verschiedenen Ursachencodes. Mit dem zweiten Parameter, `lOSError`, kann man einen Betriebssystemfehlercode spezifizieren, um ihn in der Elementvariablen `m_lOSError` der Ausnahme ablegen zu lassen. Dieser `long`-Wert kann bei der detaillierten Aufklärung eines Fehlers hilfreich sein, indem man die Liste der Dateizugriffsfehler des Betriebssystems selbst heranzieht. Der dritte Parameter, `strFileName`, wird in die Member-Variable vom Typ `String` der Ausnahme übertragen, um den Namen der Datei zu kennzeichnen, auf die der Zugriff beim Auftreten des Fehlers erfolgte.

Tabelle D.1: Die `m_cause`-Codes von `CFileException`

Ursachencode (Cause Code)	Bedeutung
<code>CFileException::none</code>	Kein Fehler.
<code>CFileException::generic</code>	Kein Fehlercode angegeben.
<code>CFileException::tooManyOpenFiles</code>	Zu viele Dateien gleichzeitig geöffnet.
<code>CFileException::fileNotFound</code>	Kann die angegebene Datei nicht finden.
<code>CFileException::badPath</code>	Der angegebene Pfadname ist ungültig.
<code>CFileException::invalidFile</code>	Versuch, einen ungültigen Datei-Handle zu verwenden.
<code>CFileException::badSeek</code>	Suchen-Operation gescheitert.
<code>CFileException::endOfFile</code>	Ende der Datei erreicht.
<code>CFileException::diskFull</code>	Kein Platz mehr auf dem Datenträger.
<code>CFileException::hardIO</code>	Hardwarefehler aufgetreten.
<code>CFileException::accessDenied</code>	Erlaubnis für den Zugriff auf die Datei verweigert.
<code>CFileException::directoryFull</code>	Das Verzeichnis enthält zu viele Dateien und kann keine weitere aufnehmen.
<code>CFileException::removeCurrentDir</code>	Kann das aktuelle Arbeitsverzeichnis nicht löschen.
<code>CFileException::lockViolation</code>	Kann keinen bereits gesperrten Bereich der Datei noch einmal sperren.
<code>CFileException::sharingViolation</code>	Ein gemeinsam genutzter Bereich ist gesperrt oder lässt sich nicht gemeinsam nutzen.

Weiterhin ist die statische Elementfunktion `ThrowOsError` verfügbar, die eine Dateiausnahme auf der Basis eines Fehlercodes des Betriebssystems auslöst und konfiguriert. Der Funktion `ThrowOsError` übergeben Sie den Fehlercode des Betriebssystems als ersten Parameter und einen optionalen Dateinamen als zweiten Parameter. Eine weitere Elementfunktion, `ThrowErrno`, bewirkt das gleiche, verwendet aber die `errno`- Fehlercodes im Stil von UNIX als einzigen Parameter (aus der Header-Datei `Errno.h`). Da es sich um statische Funktionen handelt, setzen Sie sie mit statischem Gültigkeitsbereich ein, um Ausnahmen mit Zeilen wie den folgenden auszulösen:

```
CFileException::ThrowOsError(ERROR_BAD_PATHNAME); // Ungültiger Pfad
CFileException::ThrowErrno (ENOSPC); // Datenträger voll
```

Eine weitere statische Elementfunktion, `OsErrorToException`, konvertiert automatisch zwischen den Fehlercodes des Betriebssystems und den Ursachencodes von `CFileException`. Man übergibt den Fehlercode des Betriebssystems, und die Funktion liefert den entsprechenden Ursachencode zurück. Die korrespondierende Funktion `ErrnoToException` führt das gleiche aus, wenn man einen `errno`-Fehlercode übergibt.

Wenn man Archive mit der Klasse `CArchive` nutzt, behandelt man normalerweise die Fälle `CFileExceptions` und `CArchiveException` in Verbindung: Viele der Operationen von `CArchive` sind mit den zugrundeliegenden Datei- und Dateizugriffsfunktionen verbunden. `CArchiveException` hat eine eigene Elementvariable `m_cause`, um die für das Archiv spezifischen Ursachencodes aufzunehmen, wie sie Tabelle D.2 zeigt. Man kann Archivausnahmen in eigener Regie über die Funktion `AfxThrowArchiveException` auslösen. Die Funktion erfordert als Parameter einen Ursachencode und einen `lpszArchiveName`-Stringzeiger für das Archivobjekt, das die Ausnahme auslöst.

Tabelle D.2: Die `m_cause`-Codewerte von `CArchiveException`

Ursachencode	Bedeutung
<code>CArchiveException::none</code>	Kein Fehler.
<code>CArchiveException::generic</code>	Die spezielle Ursache wurde nicht angegeben.
<code>CArchiveException::badSchema</code>	Die falsche Version eines Objekts wurde gelesen.
<code>CArchiveException::badClass</code>	Die Klasse des zu lesenden Objekts wurde nicht erwartet.
<code>CArchiveException::badIndex</code>	Ungültiges Dateiformat.
<code>CArchiveException::readOnly</code>	Versuch, in ein zum Lesen geöffnetes Archiv zu schreiben.
<code>CArchiveException::writeOnly</code>	Versuch, in einem zum Speichern geöffneten Archiv zu lesen.
<code>CArchiveException::endOfFile</code>	Das Ende der Datei wurde beim Lesen unerwartet erreicht.

Datenbankausnahmen

Für Datenbanken gibt es zwei Ausnahmeklassen: `CDBException` dient dem ODBC-basierten Datenbankzugriff, und `CDAOException` kommt beim DAO-basierten Datenbankzugriff zum Einsatz. Mit der Funktion `AfxThrowDBException` kann man diese Ausnahmen selbst auslösen. Die Funktion erfordert drei Parameter. Der erste, `nRetCode`, spezifiziert einen der zahlreichen Rückgabecodes aus der Datenbank, um den Fehlertyp zu definieren (diese Codes finden Sie in der

ODBC-Dokumentation). Der zweite Parameter, `pDB`, ist ein Zeiger auf die Datenbank, die mit der Ausnahme verbunden ist, und der dritte Parameter, `hstmt`, ist ein ODBC-Handle für das SQL-Anweisungsobjekt, dessen Ausführung die Ausnahme ausgelöst hat.

Der Typ `RETCODE` ist vom `CDBException`-Objekt über dessen Elementvariable `m_nRetCode` verfügbar. Man kann auch auf einen besser verständlichen Fehlertext im Elementstring `m_strError` und den vom ODBC-Treiber selbst im Element `m_strStateNativeOrigin` zurückgegebenen Fehlertext zugreifen.

Die Klasse `CDAOException` hat eine korrespondierende Funktion `AfxThrowDaoException`, die die DAO-Ausnahmeobjekte auslösen kann. Die Funktion übernimmt lediglich zwei optionale Parameter. Der erste, `nAfxDaoError`, ist ein DAO-spezifischer Fehlercode, der auf Probleme mit DAO selbst hinweist (siehe Tabelle D.3). Der zweite Parameter ist ein OLE-SCODE-Wert, der den Rückgabecode aus einem DAO-basierten OLE-Aufruf darstellt (siehe dazu den Abschnitt »OLE-Ausnahmen«).

Tabelle D.3: DAO-spezifische Fehlercodes von `nAfxDaoError`

Fehlercode	Bedeutung
<code>NO_AFX_DAO_ERROR</code>	Die Ausnahme geht auf ein DAO-spezifisches Problem zurück: Sie sollten das bereitgestellte <code>CDaoErrorInfo</code> -Objekt und den SCODE-Wert überprüfen.
<code>AFX_DAO_ERROR_ENGINE_INITIALIZATION</code>	Das Microsoft Jet-Datenbankmodul ist bei der Initialisierung gescheitert.
<code>AFX_DAO_ERROR_DFX_BIND</code>	Eine Adresse des DAO-Datensatzfeldaustauschs (DFX) ist ungültig.
<code>AFX_DAO_ERROR_OBJECT_NOT_OPEN</code>	Die abgefragte Tabelle ist nicht geöffnet.

Die `CDAOException`-Klasse hat drei Elementattribute: `m_score`, das einen zugeordneten OLE-SCODE-Wert mit der versuchten Operation aufnimmt, oder `S_OK`, wenn die OLE-Operation erfolgreich war. Das Element `m_nAfxDaoError` nimmt einen der DAO-spezifischen Werte aus Tabelle D.3 auf. Der Wert `m_pErrorInfo` ist ein Zeiger auf eine `CDaoErrorInfo`-Struktur, die einen Fehlercode, beschreibende Fehlerstrings und eine Hilfekontext-ID speichert und folgendermaßen definiert ist:

```
struct CDaoErrorInfo
{
    long m_lErrorCode;
    CString m_strSource;
    CString m_strDescription;
    CString m_strHelpFile;
    long m_lHelpContext;
};
```

Über diese Struktur lassen sich die meisten Details der speziellen Datenbankfehler finden, die sich auf DAO-Ausnahmen beziehen.

DAO-Ausnahmen können mehrere Fehler gleichzeitig beschreiben, so daß man mit der Funktion `GetErrorCount` ermitteln kann, wie viele Fehler referenziert werden. Die anderen Fehler lassen sich dann erhalten, indem man der Funktion `GetErrorInfo` einen Null-basierten Index auf den speziellen Fehler übergibt. Nach dem Aufruf von `GetErrorInfo` mit einem bestimmten Index in dem Bereich, den die Funktion `GetErrorCount` zurückgegeben hat, wird `m_pErrorInfo` aktualisiert, um auf das betreffende Objekt zu zeigen, so daß man diese Werte abrufen kann.

OLE-Ausnahmen

Die zwei Typen von OLE-Ausnahmen werden durch zwei Klassen repräsentiert: die Klasse `COleException` ist normalerweise für Server-seitige OLE-spezifische Operationen vorgesehen, die Klasse `COleDispatchException` wird bei Client-seitigen `IDispatch`-basierten Operationen wie etwa dem Aufruf von ActiveX-Objektfunktionen verwendet.

Die einfachere der beiden ist die Klasse `COleException`, die sich durch Aufruf der Funktion `AfxThrowOleException` generieren läßt, wobei man der Funktion einen OLE-`SCODE`-Wert (Statuscode) übergibt. Ein OLE-`SCODE`-Wert ist ein 32-Bit-Fehlercode, der alle Arten von Fehlern repräsentiert, die aus einer OLE-Funktion herrühren.

Dieser Wert stammt wahrscheinlich aus dem Rückgabecode einer Funktion, die auf einer der Schnittstellen eines OLE-Objekts aufgerufen wird. Der `SCODE`-Wert wird dann im Element `m_sc` gespeichert, um ihn innerhalb eines `catch`-Blocks untersuchen zu können.

Die statische Member-Funktion `Process` übernimmt ein Ausnahmeobjekt und überführt diese Ausnahme in einen `SCODE`-Wert, um die Ausnahme zu repräsentieren.

Die `COleDispatchException`-Klasse verwendet man in Verbindung mit der OLE- Schnittstelle `IDispatch`. Die Klasse wird durch die Funktion `AfxThrowOleDispatchException` ausgelöst. Die Funktion hat zwei Formen, die beide mit zwei erforderlichen und einem optionalen Parameter arbeiten. Der erste Parameter für beide Formen ist ein `wCode`-Wert vom Typ `WORD`, der einen anwendungsspezifischen Fehlercode darstellt. Der zweite Parameter ist in der ersten Form ein `lpszDescription`-Stringzeiger auf eine verbale Beschreibung des Fehlers und in der zweiten Form ein `nDescriptionID`-Wert vom Typ `UINT` für eine Ressourcen-ID der verbalen Fehlerbeschreibung. Der letzte, optionale Parameter ist eine Hilfekontext-ID.

Diese Werte sind dann im Objekt `COleDispatchException` als Member-Variablen `m_wCode`, `m_strDescription` und `m_dwHelpContext` verfügbar. Wenn man einen Hilfekontext spezifiziert und eine Hilfedatei verfügbar ist, füllt das Programmgerüst einen `m_strHelpFile`-String, der die Hilfedatei kennzeichnet. Der Name der Anwendung, die den Fehler produziert hat, läßt sich ebenfalls über das Attribut `m_strSource` ermitteln.

Wenn man eine Ausnahme aus einem OLE-Objekt auslöst, beispielsweise einem ActiveX-Steuerelement, zeigt Visual Basic oder eine andere Anwendung, die das Steuerelement oder das Objekt verwendet, die Ausnahmedetails an.

Nicht unterstützte Ausnahmen

Die von der Klasse `CNotSupportedException` repräsentierten Ausnahmeobjekte werden generiert, wenn man nicht unterstützte Funktionsmerkmale der MFC, des Betriebssystems oder anwendungsspezifischer Funktionen anfordert. Wenn Sie diese Ausnahme selbst auslösen wollen, verwenden Sie die Funktion `AfxThrowNotSupportedException`, die keinerlei Parameter

übernimmt. Darüber hinaus gibt es bei dieser Ausnahme keine erweiterten Elemente oder Funktionen - genau deshalb heißt es **nicht unterstützt**.

Benutzerausnahmen

Mit der Klasse `CUserException` können Sie anwendungsspezifische Ausnahmeobjekte generieren. Beispielsweise läßt sich damit ein Vorgang anhalten, wenn der Benutzer in einer Anwendung eine bestimmte Option wählt. Wenn Sie zum Beispiel mit dem Anwendungs-Assistenten arbeiten, können Sie jederzeit <Esc> drücken, um den ganzen Prozeß abubrechen. Microsoft hat wahrscheinlich mit `CUserException` gearbeitet, um die <Esc>-Taste zu erkennen und dann ein Objekt für Benutzerausnahmen auszulösen.

Diese Ausnahme läßt sich durch einen Aufruf der Funktion `AfxThrowUserException` auslösen und dann in den üblichen `try`- und `catch`-Blöcken auffangen. In der MFC gibt es verschiedene Stellen, wo diese Ausnahme ausgelöst wird, beispielsweise während der Gültigkeitsprüfung eines Dialogfelds oder wenn eine Datei zu groß ist, um sie im Editor anzuzeigen.

Eigene Ausnahmeklassen generieren

Ihre eigenen Ausnahmeklassen können Sie von `CException` ableiten und spezielle Erweiterungen der Funktionalität hinzufügen. Listing D.6 zeigt die Klassendefinition für eine derartige Ausnahmeklasse, die die normale Funktionalität um eine `CString`-Variable `m_strMessage` erweitert und damit ermöglicht, daß Sie Ihre eigene Meldung spezifizieren können, wenn Sie die Ausnahme konstruieren.

Listing D.6: In `CustomException.h` implementierte Klassendefinition für `CCustomException`

```
1: // Datei: CustomException.h
2: // Header-Datei für CCustomException
3:
4: class CCustomException : public CException
5: {
6: DECLARE_DYNAMIC(CCustomException);
7:
8: public:
9: CCustomException(CString strMessage);
10:
11: CString m_strMessage;
12: };
```

In Listing D.6 ist die Klasse in ihrer eigenen Header-Datei `CustomException.h` implementiert und in Zeile 4 von `CException` abgeleitet. Das Makro `DECLARE_DYNAMIC` in Zeile 6 stellt die Laufzeitinformationen der von `CObject` abgeleiteten MFC-Klassen bereit, die erforderlich sind, um den Ausnahmetyp in einem `catch`-Block für alle Ausnahmen bestimmen zu können. Die Konstruktordefinition in Zeile 9 übernimmt den Parameter `strMessage` vom Typ `CString`, damit Sie die benutzerdefinierte Ausnahme mit der Nachricht, die in der auf Zeile 11 deklarierten `CString`-Variablen gespeichert wird, erzeugen können.

Die entsprechende Implementierung der Klasse `CCustomException` ist in Listing D.7 zu sehen.

Listing D.7: Implementierung der Klasse `CCustomException`

```

1: // CustomException.cpp
2: // Implementierung für CCustomException-Ausnahme
3:
4: #include "stdafx.h"
5: #include "CustomException.h"
6:
7: IMPLEMENT_DYNAMIC(CCustomException, CException);
8:
9: CCustomException::CCustomException(CString strMessage)
10: : m_strMessage(strMessage)
11: {
12: }

```

In Listing D.7 sind die üblichen Header-Dateien eingebunden, und das Makro `IMPLEMENT_DYNAMIC` in Zeile 7 implementiert die Funktionen für die Laufzeitinformationen der MFC-Klassen. Der Konstruktor in Zeile 9 übernimmt den `strMessage`-Parameter und initialisiert in Zeile 10 die Elementvariable `m_strMessage` mit diesem Stringwert.

Die benutzerdefinierte Ausnahmeklasse können Sie dann in Ihrer Anwendung einsetzen, wie es Listing D.8 zeigt.

Listing D.8: Einsatz der neuen Klasse `CCustomException`

```

1: try
2: {
3: // Irgendwas geht schief
4: CCustomException* pCustomEx =
5: new CCustomException("Benutzerdefinierter Fehler aufgetreten ");
6: throw(pCustomEx);
7: }
8: catch(CCustomException* e)
9: {
10: // Auf erweiterten m_strMessage-String zugreifen
11: AfxMessageBox(e->m_strMessage);
12: e->Delete();
13: }

```

In Listing D.8 wird ein neues `CCustomException`-Objekt mit dem anwendungsspezifischen Fehlertext in den Zeilen 4 und 5 erzeugt und in Zeile 6 ausgelöst. Zeile 8 fängt die Ausnahme mit dem Schlüsselwort `catch` ab, und Zeile 11 zeigt die verwendeten benutzerdefinierten Informationen in einem Meldungsfeld an. Die Ausnahme wird dann in Zeile 12 gelöscht.

Wenn Sie diesen Code ausprobieren, denken Sie daran, daß der Implementierungscode eine `#include`-Anweisung für die Header-Datei `CustomException.h` enthalten muß, um die Klassendefinition abzurufen:

```
#include "CustomException.h"
```

Ein Imprint des [Markt&Technik Buch- und Software-Verlag GmbH](#).
Elektronische Fassung des Titels: Visual C++ 6 in 21 Tagen, ISBN: 3-8272-2035-1

Tag E

Debugger und Profiler

Informationen zum Debuggen und Durchsuchen erzeugen

Ein großer Teil der Anwendungsentwicklung besteht eigentlich aus der Fehlersuche im Programm. Die gesamte Softwareentwicklung ist ein straffer Zyklus von Anwendungsentwurf, Implementierung und Fehlersuche.

Visual C++ bietet eine umfangreiche Umgebung für die Fehlersuche und zahlreiche Debugging-Werkzeuge, die bei der Programmentwicklung unentbehrlich sind. Man kann damit Probleme schnell erkennen, den Inhalt von Variablen überwachen und den Programmfluß durch den eigenen und den MFC-Code verfolgen.

Werkzeuge wie das Programm Spy++ zeigen, welche Nachrichten Windows an die Anwendung übergibt, und man kann auch untersuchen, mit welchen Steuerelementen der Benutzeroberfläche und welchen Windows-Stilen die Anwendungen arbeiten.

Die Modi Debug und Release

Der Compiler unterscheidet zwei Hauptmodi, die man für das Erstellen der Anwendung konfigurieren kann: Debug und Release. Diese Modi lassen sich ändern, indem man im Menü **Projekt** den Befehl **Einstellungen** wählt oder (Alt)+(F7) drückt. Es erscheint das Dialogfeld **Projekteinstellungen** (siehe Abbildung E.1). Die Haupteinstellungen des Projekts sind auf oberster Ebene zu sehen und lassen sich ändern, indem man eine der im Kombinationsfeld aufgeführten Optionen markiert. Ist eine Einstellung markiert, beziehen sich alle Änderungen, die Sie an den Optionen auf den Registerkarten im rechten Teil des Dialogfelds vornehmen, auf diese Konfiguration. Wenn Sie die Anwendung erstellen, gelten dabei die aktuellen Konfigurationseinstellungen. Haben Sie *Alle Konfigurationen* gewählt, wirken die Änderungen für alle Konfigurationen gleichzeitig.

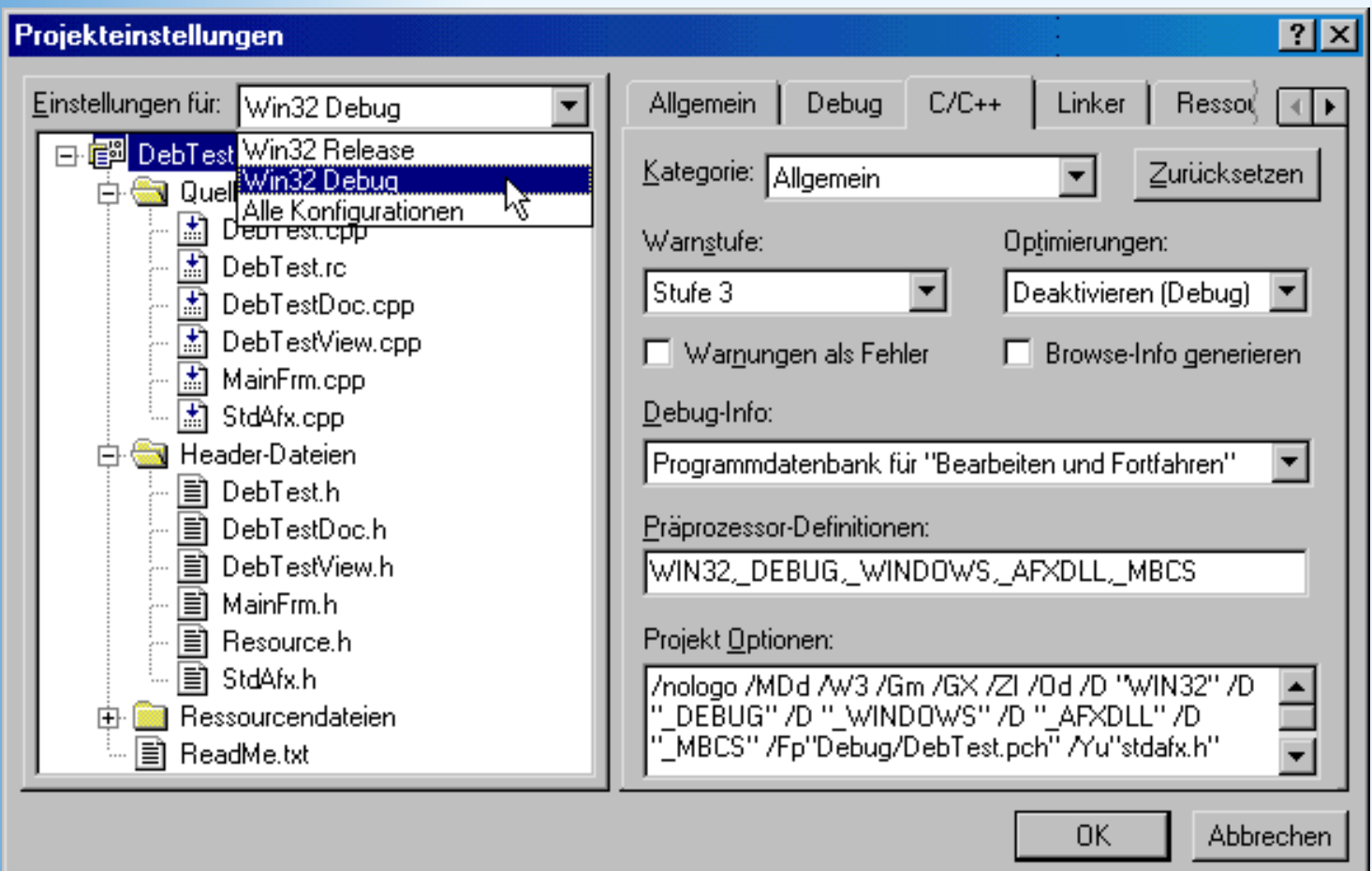


Abbildung E.1:
Die Registerkarte C/C++ des Dialogfelds Projekteinstellungen

Sobald Sie ein neues Projekt anlegen, stehen sowohl die Release- als auch die Debug- Konfiguration zur Verfügung. Die beiden Modi erzeugen unterschiedlichen Objektcode. In der Konfiguration für den Debug-Modus entsteht ein großes und relativ langsames ausführbares Programm. Das hängt damit zusammen, daß eine Unmenge von Debugging-Informationen in das Programm eingebunden werden und alle Optimierungen des Compilers deaktiviert sind.

Wenn Sie dasselbe Programm im Release-Modus erneut kompilieren, erhalten Sie ein kleines und schnell laufendes Programm, können aber nicht den Quellcode im Schrittbetrieb durchgehen oder irgendwelche Debugging-Meldungen erhalten.

Während der Entwicklungsphase einer Anwendung läßt man den Compiler normalerweise im Debug-Modus, damit man Probleme im Code schnell und einfach einkreisen und beseitigen kann. Haben Sie die Anwendung fertiggestellt und bereiten sie für den Vertrieb vor, können Sie die Konfiguration auf Release-Modus stellen und ein kleines, schnelles Programm für die Benutzer produzieren.



Testen im Release-Modus

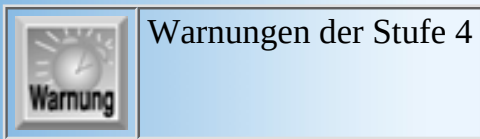
Nachdem Sie die Anwendung im Release-Modus erstellt haben, sollten Sie sie umfassend testen, bevor sie in den Vertrieb geht. Fehler können beispielsweise dadurch entstehen, daß man maßgeblichen Programmcode in ASSERT-Makros (weiter hinten in diesem Anhang behandelt) beibehält. Im Release-Modus werden aber diese Makros entfernt. Darüber hinaus können auch Probleme aufgrund von Geschwindigkeits- oder Speicheroptimierungen auftauchen.

Optionen und Stufen für das Debuggen

Auf der Registerkarte **C/C++** des Dialogfelds **Projekteinstellungen** lassen sich verschiedene Optionen und Stufen für das Debuggen einstellen. Diese Dialogseite erreichen Sie über den Befehl **Einstellungen** des Menüs **Projekt** (oder durch Drücken von (Alt)+(F7)). Im Dialogfeld **Projekteinstellungen** aktivieren Sie dann die Registerkarte **C/C++**.

In der Kategorie *Allgemein* sind folgende Elemente verfügbar:

- Warnstufe: Legt den Umfang der Warnhinweise des Compilers während der Kompilierung fest. Tabelle E.1 führt die verfügbaren Werte auf. Voreingestellt ist Stufe 3, eine recht empfindliche Stufe. Eingefleischte C++-Programmierer bestehen allerdings auf der Stufe 4, um die meisten Warnungen über mögliche Probleme vom Compiler zu erhalten. Die Optionen Stufe 1 und Keine sollte man nur in Ausnahmefällen verwenden, da hier der Compiler nur schwerwiegende Warnungen (oder überhaupt keine) anzeigt.



Auf dieser Stufe führt sogar der vom Microsoft-eigenen Anwendungs-Assistenten generierte Code zu Warnmeldungen (auch wenn es sich nur um nicht genutzte Funktionsparameter handelt, die man getrost ignorieren kann).

- Warnungen als Fehler: Bei eingeschaltetem Kontrollkästchen erscheinen Warnmeldungen als Fehler, die dann den Compiler stoppen.
- Browse-Info generieren: Wenn Sie dieses Kontrollkästchen einschalten, generiert der Compiler Informationen, mit denen man Funktionen, Symbole und Klassenbeziehungen lokalisieren kann, die in einem Durchsuchen-Fenster (siehe dazu den nächsten Abschnitt) angezeigt werden. Leider erhöht sich durch das Generieren dieser hilfreichen Informationen die Kompilierungszeit insbesondere bei großen Projekten (wo man sie gerade am nötigsten braucht).
- Debug-Info: Hier legen Sie die Stufe der vom Compiler generierten Debug-Informationen fest, wie es Tabelle E.2 zeigt.
- Optimierungen: Im Debug-Modus stellt man normalerweise Deaktivieren (Debug) ein, weil sich Optimierungen mit den Vorgängen bei der Fehlersuche schlecht vertragen und die Kompilierung verlängern. Im Release-Modus dagegen können Sie sich entscheiden, ob Sie die Geschwindigkeit der Anwendung maximieren oder minimieren wollen (oder den Standardwert Anpassen wählen, um das beste aus beidem zu erhalten).
- Präprozessor-Definitionen: Dieses Feld zeigt feststehende Definitionen an, die beim Kompilieren des Programms definiert werden. Diese Definitionen können Sie in Verbindung mit den Präprozessor-Befehlen `#ifdef`, `#else` und `#endif` einsetzen, um Codeabschnitte in bestimmten Konfigurationen zu kompilieren. Die Definition `_DEBUG` ist im Debug-Modus per Voreinstellung gesetzt. Damit kann man Code, der nur im Debug-Modus auszuführen ist, kompilieren, wie etwa im folgenden Beispiel:

```
int a = b * c / d + e;
#ifdef _DEBUG
CString strMessage;
strMessage.Format("Zwischenergebnis (Summe): %d", a);
AfxMessageBox(strMessage);
#endif
```

- Der Code für das Meldungsfeld wird nur dann kompiliert und ausgeführt, wenn Sie die Anwendung im Debug-Modus erstellen. Wenn Sie in den Release-Modus wechseln, wird der Code nicht in das

ausführbare Programm aufgenommen.

- **Projekt Optionen:** Der Compiler selbst läuft als Konsolenanwendung und konvertiert die Optionen des Visual Studios in mehrere Flags, die an die Befehlszeile übergeben werden. Man kann in diesem Eingabefeld Flag-Werte für obskure Compiler-Einstellungen hinzufügen, die keinen Schalter in der Benutzeroberfläche haben.

Tabelle E.1: Warnstufen des Compilers

Stufe	Ausgegebene Warnung
Keine	Keine
Stufe 1	Nur die schwerwiegendsten.
Stufe 2	Einige weniger schwerwiegende.
Stufe 3	Standardstufe (alle sinnvollen Warnungen).
Stufe 4	Sehr empfindlich (gut für Perfektionisten).

Tabelle E.2: Einstellungen der Debug-Info

Einstellung	Generierte Debug-Infos
Keine	Produziert keine Debug-Informationen - normalerweise für die Release-Modi vorgesehen.
Nur Zeilennummern	Generiert nur Zeilennummern, die sich auf den Quellcode beziehen, für Funktionen und globale Variablen. Allerdings verringern sich die Erstellungszeit und die Größe der ausführbaren Datei.
C7-kompatibel	Generiert Debug-Informationen, die mit Microsoft C 7.0 kompatibel sind. Schreibt alle Debug-Informationen in die ausführbaren Dateien, vergrößert sie damit, erlaubt aber ein komplett symbolisches Debuggen.
Programmdatenbank	Diese Einstellung erzeugt eine Datei mit der Erweiterung <code>.pdb</code> , die alle verfügbaren Debug-Informationen aufnimmt, aber keine Informationen zum Bearbeiten und Fortsetzen erzeugt.
Programmdatenbank für »Bearbeiten und Fortsetzen«	Die standardmäßige und übliche Debug-Einstellung. Sie produziert eine <code>.pdb</code> -Datei mit der höchsten Debug-Stufe und erzeugt die notwendigen Informationen für das Merkmal Bearbeiten und Fortsetzen.

Browse-Infos

Mit dem Hilfsprogramm Quellcode-Browser können Sie Ihren Quellcode im Detail inspizieren. Dieses Werkzeug ist unschätzbar, wenn Sie den Code eines anderen Programmierers untersuchen oder zu Ihrem eigenen Werk zurückkehren, mit dem Sie eine ganze Zeitlang nichts zu tun hatten.

Um das Programm zu starten, drücken Sie (Alt)+(F12) oder wählen im Menü **Extras** den Befehl

Quellcode-Browser. (Wenn Sie das erste Mal dieses Programm aufrufen, erscheint die Frage, ob Sie das Projekt neu erstellen möchten, um die Browse-Informationen zu generieren.)

Das erste Dialogfeld des Quellcode-Browsers fragt nach einem Bezeichner, nach dem zu suchen ist (wie es Abbildung E.2 zeigt). Dabei kann es sich um den Namen einer Klasse, einer Struktur, einer Funktion oder globalen oder lokalen Variablen in Ihrer Anwendung handeln. Nachdem Sie einen Bezeichner eingegeben haben, ist die Schaltfläche **OK** aktiviert, und Sie können nach Einzelheiten zu diesem Bezeichner fahnden.

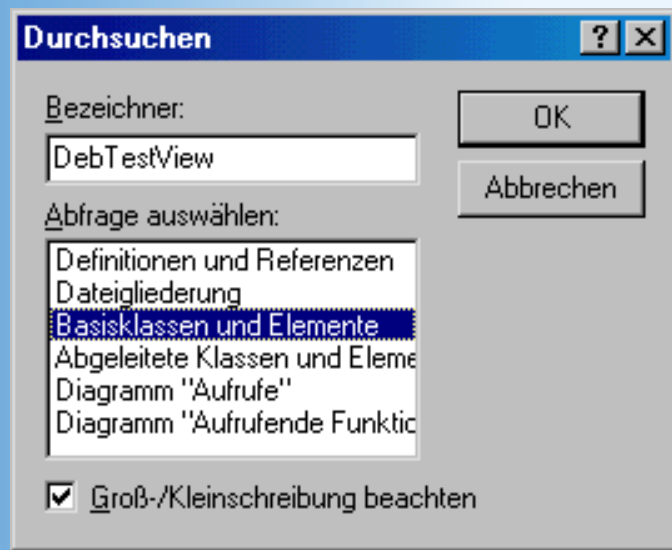


Abbildung E.2:
Das Dialogfeld Durchsuchen fordert ein zu suchendes Symbol an.

Im Listenfeld **Abfrage auswählen** finden Sie verschiedene Optionen bezüglich der Details, die zum gewählten Symbol gehören. Es stehen folgende Optionen zur Wahl:

- **Definitionen und Referenzen:** Diese Option zeigt alle Dateien, die Referenzen auf den angegebenen Bezeichner enthalten, und ob es sich um Referenzen auf den Bezeichner (Stellen, wo er im Code erscheint) oder die Definitionen (Stellen, wo der Bezeichner definiert ist) handelt, wie es Abbildung E.3 zeigt. Indem Sie auf eine der Referenzen oder Definitionen doppelklicken, wird der Code, auf den sich die Referenz oder die Definition bezieht, geladen und im Editor des Visual Studios an der angegebenen Position angezeigt. Das ist sehr nützlich, um alle Stellen zu verfolgen, an denen eine bestimmte Variable oder Funktion vorkommt.

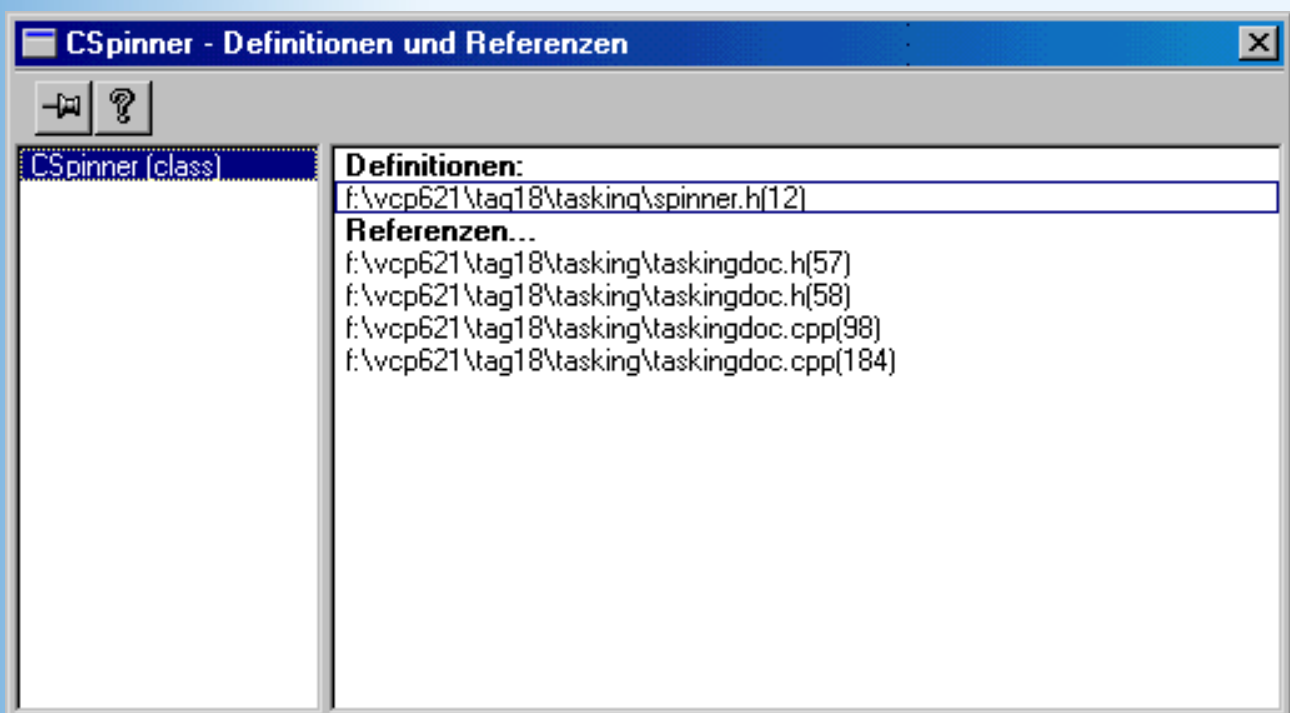
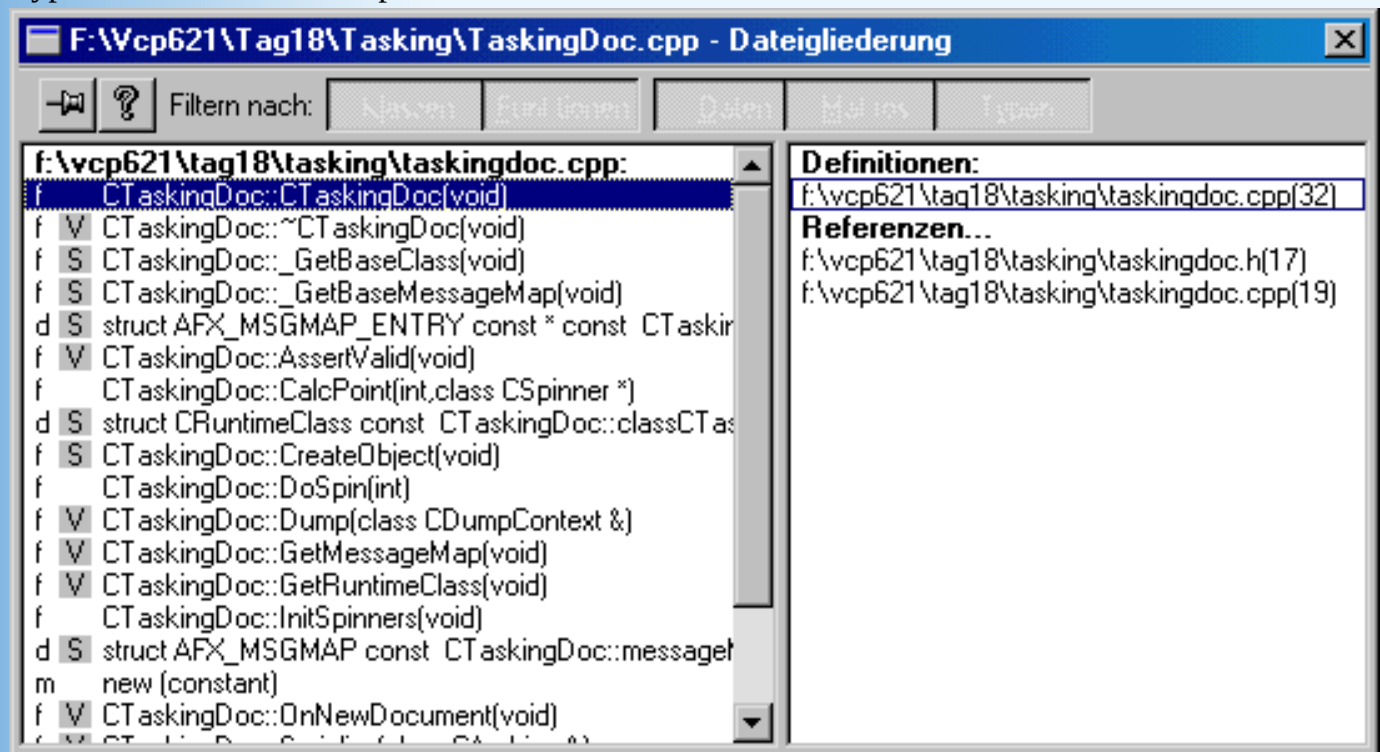


Abbildung E.3:

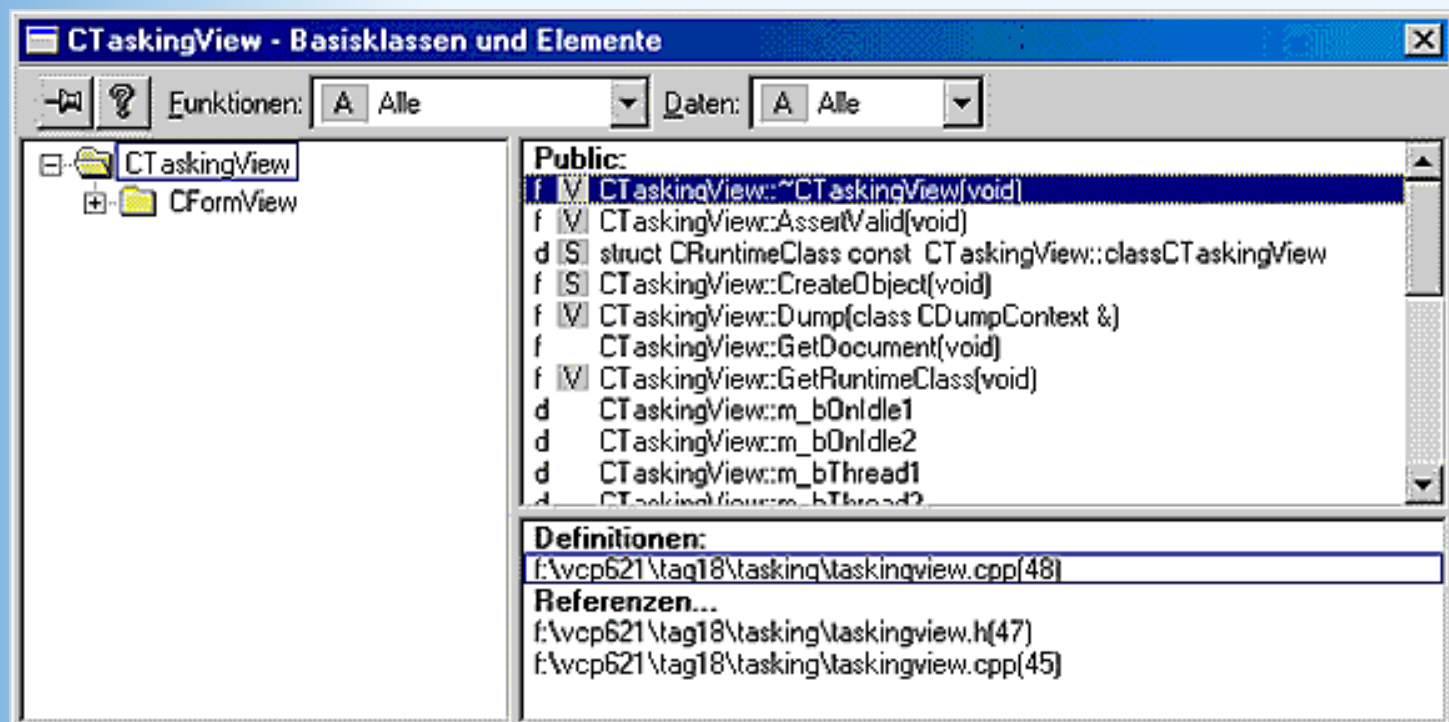
Der Quellcode-Browser zeigt hier Definitionen und Referenzen an.

- **Dateigliederung:** Diese Option zeigt alle Klassen, Daten, Funktionen, Makros und Typen, die in der angegebenen Datei (Bezeichner) definiert sind, wie es aus Abbildung E.4 ersichtlich ist. Die einzelnen Typen lassen sich mit entsprechenden Schaltflächen am oberen Rand des Browser-Fensters filtern.

**Abbildung E.4:**

Die Ansicht der Dateigliederung des Quellcode-Browsers

- **Basisklassen und Elemente:** Das ist vermutlich die nützlichste Option des Quellcode-Browsers. Indem man eine Klasse als Bezeichner angibt, werden die gesamte Hierarchie und die Elementfunktionen der Klassen sowie die Variablen auf jeder Hierarchiestufe angezeigt (siehe Abbildung E.5). Man kann die Filteroptionen auch so einstellen, daß nur bestimmte Typen von Elementfunktionen und Variablen erscheinen.

**Abbildung E.5:**

Die Ansicht der Basisklassen und Elemente des Quellcode-Browsers

- Abgeleitete Klassen und Elemente: Diese Ansicht ist ebenfalls sehr nützlich und zeigt alle Klassen, die von der angegebenen Klasse abgeleitet sind, sowie deren eigene Elementfunktionen und Variablen. Den Quellcode-Browser kann man auch auf MFC-Klassen anwenden, um einen tieferen Einblick in die MFC-Implementierung zu gewinnen, wie es Abbildung E.6 mit der MFC-Klasse CWnd zeigt.

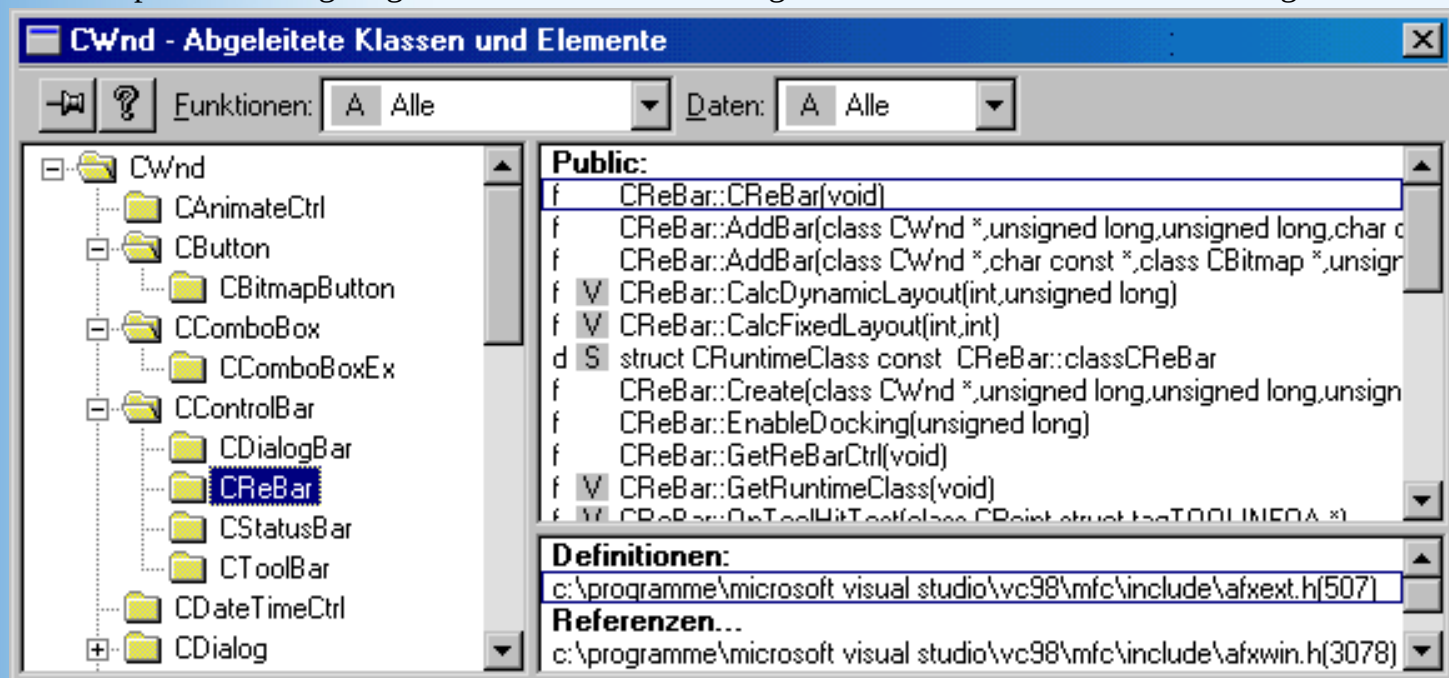


Abbildung E.6:

Die Ansicht der abgeleiteten Klassen und Elemente des Quellcode-Browsers zeigt Klassen, die von CWnd abgeleitet sind.

- Diagramm »Aufrufe«: Diese Option zeigt alle Funktionen, die von einem angegebenen Bezeichner aufgerufen werden, und die Dateien, in denen sie definiert und implementiert sind. Damit kann man leicht den möglichen Programmfluß verfolgen.
- Diagramm »Aufrufende Funktion«: Die korrespondierende Option zeigt alle Funktionen, die den angegebenen Bezeichner aufrufen. Damit kann man die möglichen Aufrufer der angegebenen Funktion verfolgen.

Remote- und Just-In-Time-Debugging

Zum Debugger gehören Werkzeuge, mit denen man ein laufendes Programm auf einem Remote-Computer (sogar über das Internet via TCP/IP) debuggen kann. Das ist hilfreich, wenn man die Anwendung in einer anderen Umgebung als der Entwicklungsmaschine testen möchte. Dazu müssen Sie über genau die gleichen Versionen der .dll- und .exe-Dateien auf beiden Maschinen verfügen. Nach dem Laden des Projekts, können Sie es über ein gemeinsam genutztes Verzeichnis von der Remote- Maschine aus debuggen, indem Sie im Dialogfeld **Projekteinstellungen** auf der Registerkarte **Debug** in das Eingabefeld **Ausführbares Programm für Debug-Sitzung** den Pfad und Dateinamen der lokalen .exe-Datei eintragen.

Der Pfad zur .exe-Datei ist auch in das Eingabefeld **Pfad und Name des ausführbaren Remote-Programms** im unteren Teil der Registerkarte **Debug** einzutragen. Das Eingabefeld **Arbeitsverzeichnis** lassen Sie dabei leer. Dann können Sie den Monitor des Remote-Debuggers auf dem Remote-Computer starten, indem Sie das Programm MSVCMON . EXE aufrufen. Eine Verbindung zum Remote-Computer stellen Sie über den Befehl **Remote-Verbindung des Debuggers** im Menü **Erstellen** her.

Im Dialogfeld **Remote-Verbindung** können Sie als Verbindung *Lokal* für eine Sitzung mit gemeinsam genutztem Verzeichnis oder *Netzwerk (TCP/IP)* beim Debuggen über eine TCP/IP-Verbindung wählen. (Um die Adresse festzulegen, klicken Sie auf die Schaltfläche **Einstellungen**.) Damit wird eine Verbindung zum

Remote-Monitor hergestellt, der die Remote-Debugging-Sitzung startet.



Die Dateien für den Remote-Debugger installieren

Um den Remote-Debugger-Monitor auf der Remote-Maschine auszuführen, sind die folgenden Dateien erforderlich: `MSVCMON.EXE`, `MSVCRT.DLL`, `TLN0T.DLL`, `DM.DLL`, `MSVCP50.DLL` und `MSDIS100.DLL`. Die Dateien finden Sie in ihrem installierten Unterverzeichnis `...\Microsoft Visual Studio\Common\MSDev98\bin` bzw. in `\Windows\System` (die Datei `MSVCP50.DLL`).

Beim Just-In-Time-Debugging können Sie ein Programm debuggen, das normal (nicht über den Debugger) läuft und dann ein Problem verursacht. Wenn Sie Visual C++ auf einem Computer installiert und diese Option aktiviert haben, wird jedes Programm, das einen Fehler verursacht, in eine neue Sitzung des Visual Studios zum Debuggen geladen und zeigt den Code, der den Absturz verursacht hat.

Lustig wird es, wenn Visual Studio selbst abstürzt, daraufhin eine weitere Sitzung von sich selbst lädt und eine Assemblercode-Ansicht zeigt, wo der Absturz im Original stattgefunden hat, damit Sie sich auf die Fehlersuche machen können. Es kann sehr hilfreich sein, die eigenen Anwendungen auf Fehler zu untersuchen, wenn sie unerwartet abstürzen (normalerweise bei einer Vorführung für Ihren Chef). Um diese Option einzuschalten, wählen Sie aus dem Menü **Extras** den Befehl **Optionen**. Im Dialogfeld **Optionen** gehen Sie auf die Registerkarte **Debug** und schalten das Kontrollkästchen **Just-In-Time-Debugging** ein.

Das Kontrollkästchen **OLE RPC-Debugging** auf dieser Registerkarte ist ebenfalls sehr hilfreich, wenn Sie COM- und DCOM-Anwendungen entwickeln, weil der Debugger damit in einen prozeßexternen Funktionsaufruf von Programmen oder DLLs springen und eine zweite Instanz des Debuggers starten (oder aktivieren) kann, um den anderen Prozeß zu verarbeiten. Die Steuerung kehrt dann zurück, wenn die Rückkehr aus der Remote-Funktion stattfindet. Das Ganze funktioniert in Netzwerken und mit verschiedenen Computern.

Ablaufverfolgung und Einzelschrittausführung

Eines der nützlichsten Features der Debugging-Umgebung von Visual C++ ist der interaktive Einzelschrittbetrieb. Damit läßt sich der Code zeilenweise ausführen und der Inhalt der Variablen untersuchen. Außerdem kann man Haltepunkte setzen, so daß das Programm bis zu diesem Haltepunkt läuft und dort stoppt. Von hier aus kann man im Einzelschrittbetrieb arbeiten, bis man das Programm normal fortsetzen möchte.

Trace- und Assertion-Anweisungen unterstützen ebenfalls die Suche nach Programmfehlern. Mit Trace-Anweisungen kann man Meldungen und Variablen aus dem Programm heraus im Ausgabefenster anzeigen, während das Programm durch diese Trace-Anweisungen läuft. Assertions erlauben es, das Programm anzuhalten, wenn eine Bedingung nicht erfüllt ist, obwohl sie eigentlich erfüllt sein müßte.

Das Makro TRACE

TRACE-Makros können Sie an den verschiedensten Stellen in Ihr Programm einbauen, um etwa zu kennzeichnen, daß bestimmte Teile des Codes ausgeführt wurden, oder um den Inhalt von Variablen an diesen Punkten anzuzeigen. Die TRACE-Makros werden in der Debug-Konfiguration in den Code kompiliert und im Ausgabefenster auf der Registerkarte **Debug** angezeigt, wenn Sie das Programm über den Debugger ausführen.

Die TRACE-Makros brauchen Sie für die Release-Version des Programms nicht zu entfernen, da sie der Compiler beim Erstellen automatisch aus dem Zielobjekt ausschließt.

Mit einem Formatstring als erstem Parameter an das TRACE-Makro können Sie einfache Meldungen anzeigen oder den Inhalt von Variablen ausgeben. Der Formatstring entspricht im Aufbau völlig den Strings, die man an die Funktionen `printf` oder `CString::Format` übergibt. Es lassen sich verschiedene spezielle Formatcodes festlegen wie `%d` zur dezimalen Anzeige einer Zahl, `%x` zur hexadezimalen Anzeige einer Zahl oder `%s` zur Anzeige von Strings. Die darauffolgenden Parameter müssen dann in der entsprechenden Reihenfolge der Formatcodes stehen. Beispielsweise bewirkt der Code

```
int nMyNum = 60;
char* szMyString = "Mein String";
TRACE("Zahl = %d, oder %x (hex). Der String lautet: %s\n",
nMyNum, szMyString);
```

die Ausgabe der folgenden Trace-Zeile:

Zahl = 60, oder 3c (hex). Der String lautet: Mein String

Listing E.1 demonstriert, wie man mit dem Makro TRACE den Inhalt eines Arrays vor und nach dem Sortieren durch einen zwar nicht sehr effizienten aber einfachen Sortieralgorithmus anzeigt.

Wenn Sie den in Listing E.1 dargestellten Code ausprobieren möchten, erstellen Sie mit dem Anwendungs-Assistenten das Gerüst einer einfachen SDI-Anwendung. Fügen Sie den Code über der Elementfunktion `OnNewDocument` der Dokumentklasse hinzu, und rufen Sie ihn über `DoSort()` in Ihrer Funktion `OnNewDocument` auf.

Die Anwendung können Sie über den Debugger ausführen, um die Ausgaben zu verfolgen. (Wählen Sie **Erstellen** / **Debug starten** / **Ausführen**.)

Stellen Sie sicher, daß das Ausgabefenster sichtbar ist (beispielsweise durch Wahl von **Ansicht** / **Ausgabe**), und aktivieren Sie hier die Registerkarte **Debug**.

Listing E.1: DebTestDoc.cpp - Eine einfache Sortierroutine, die die Verfahren zum Debuggen demonstriert

```
1: void Swap(CUIntArray* pdwNumbers,int i)
2: {
3:     UINT uVal = pdwNumbers->GetAt(i);
4:     pdwNumbers->SetAt(i, pdwNumbers->GetAt(i+1));
5:     pdwNumbers->SetAt(i+1,uVal);
6: }
7:
8: void DoSort()
9: {
10:    CUIntArray arNumbers;
11:    for(int i=0;i<10;i++) arNumbers.Add(1+rand()%100);
12:
13:    TRACE("Vor dem Sortieren\n");
14:    for(i=0;i<arNumbers.GetSize();i++)
15:        TRACE("[%d] = %d\n",i+1,arNumbers[i]);
16:
17:    BOOL bSorted;
18:    do
19:    {
20:        bSorted = TRUE;
21:        for(i=0;i<arNumbers.GetSize()-1;i++)
22:        {
23:            if (arNumbers[i] > arNumbers[i+1])
24:            {
```

```

25: Swap(&arNumbers,i);
26: bSorted = FALSE;
27: }
28: }
29: } while(!bSorted);
30:
31: TRACE("Nach dem Sortieren\n");
32: for(i=0;i<arNumbers.GetSize();i++)
33: TRACE("[%d] = %d\n",i+1,arNumbers[i]);
34: }

```

Listing E.1 sortiert ein Array von Zufallszahlen (zwischen 1 und 100), die in Zeile 11 generiert werden. Die Zeilen 13 bis 15 geben den Inhalt des Arrays vor dem Sortieren mit Hilfe von TRACE-Anweisungen aus. Die Zeilen 17 bis 29 sortieren das Array, indem paarweise aufeinanderfolgende Zahlen ausgetauscht werden, wenn sie nicht der geforderten Sortierrichtung entsprechen (und zwar durch Aufruf der Funktion Swap in Zeile 25). Die Funktion Swap in den Zeilen 1 bis 6 übernimmt einen Zeiger auf das Array sowie die aktuelle Position und vertauscht dann die beiden Zahlen an dieser Position.

Nach dem Sortieren wird der Inhalt des Arrays erneut im Ausgabefenster durch die TRACE-Anweisungen in den Zeilen 31 bis 33 angezeigt.

Die Trace-Ausgabe des Programms erscheint im Ausgabefenster des Visual Studios, wie es Tabelle E.3 angibt.

Tabelle E.3: Ausgabe des Sortierprogramms

Vor dem Sortieren	Nach dem Sortieren
[1] = 42	[1] = 1
[2] = 68	[2] = 25
[3] = 35	[3] = 35
[4] = 1	[4] = 42
[5] = 70	[5] = 59
[6] = 25	[6] = 63
[7] = 79	[7] = 65
[8] = 59	[8] = 68
[9] = 63	[9] = 70
[10] = 65	[10] = 79

Die Makros ASSERT und VERIFY

Mit dem Makro ASSERT kann man sich vergewissern, ob Bedingungen TRUE sind. Als Parameter übergibt man an ASSERT einen Ausdruck, der entweder TRUE oder FALSE liefert. Wenn der Ausdruck gleich TRUE ist, läuft alles bestens. Liefert der Ausdruck FALSE, stoppt das Programm, und es erscheint das Dialogfeld **Debug Assertion Failed!** (siehe Abbildung E.7). Das Dialogfeld bietet die Optionen **Beenden** des Programms, **Wiederholen** des Codes oder **Ignorieren** der Annahme. Außerdem zeigt es das Programm, die Quellcodedatei und Zeilennummer an, wo die Annahme (Assertion) fehlerhaft war. Wenn Sie **Beenden** wählen, wird die Sitzung beendet. **Wiederholen** ist die wahrscheinlich nützlichste Option, weil der Compiler

dann den Code zeigt, wo das Makro `ASSERT` gescheitert ist und Sie die Ursache herausfinden können. Wenn Sie bereits die Annahme kennen oder sich nicht darum kümmern möchten, wählen Sie Ignorieren und setzen das Programm fort, was dann allerdings zu einem schlimmeren Fehler führen kann.

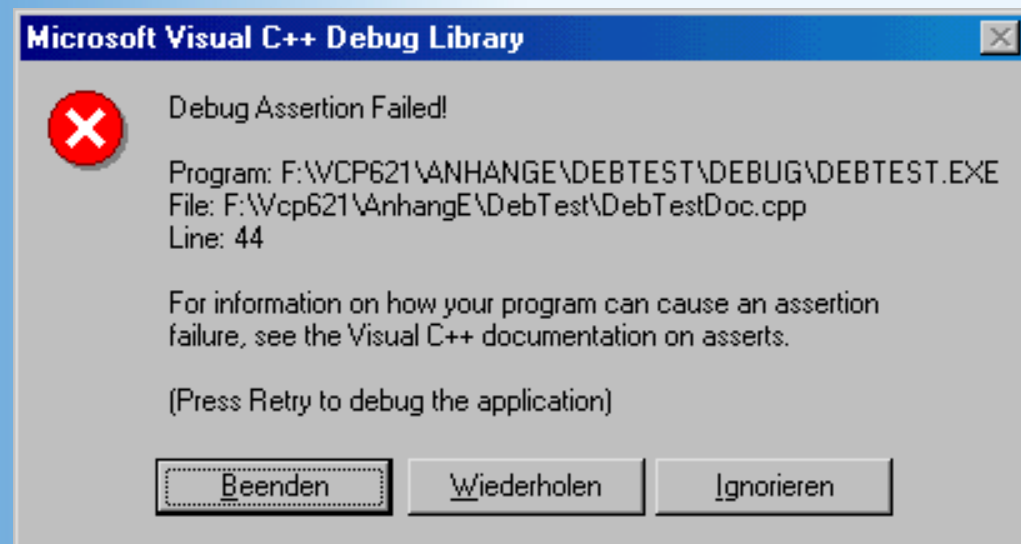


Abbildung E.7:
Das Dialogfeld Debug Assertion Failed! hilft beim Aufspüren von Bugs

Mit `ASSERT` prüft man oftmals, ob Eingabeparameter an Funktionen korrekt sind. Beispielsweise können Sie die (in Listing E.1 dargestellte) Funktion `Swap` robuster gestalten, indem Sie deren Eingabeparameter überprüfen. Fügen Sie dazu `ASSERT`-Makros am Beginn der Funktion `Swap` wie folgt hinzu:

```
ASSERT(pdwnumbers);
ASSERT(i >= 0 && i < 10);
```

Damit ist sichergestellt, daß der Zeiger auf das Zahlenarray ungleich null ist und daß die Position zum Vertauschen zwischen 0 und 9 liegt. Wenn eine dieser Bedingungen nicht korrekt ist, wird das Dialogfeld **Debug Assertion Failed!** angezeigt. Mit dieser Technik lassen sich Fehler aufspüren, die durch die Übergabe falscher Parameter an Funktionen entstehen. Es empfiehlt sich, mit dem Makro `ASSERT` zu prüfen, ob die übergebenen Werte an die einzelnen Funktionen mit den eigenen Erwartungen übereinstimmen.

Ein anderes Makro, `ASSERT_VALID`, kann man bei von `CObject` abgeleiteten Klassen - wie bei den meisten MFC-Klassen - verwenden. Dieses Makro führt einen umfangreichen Test des Objekts und seines Inhalts aus, um sicherzustellen, daß das gesamte Objekt in einem korrekten und gültigen Zustand ist. Auf das zu überprüfende Objekt übergibt man einen Zeiger wie im folgenden Beispiel:

```
ASSERT_VALID(pdwnumbers);
```

Das Makro `ASSERT_KINDOF` prüft bei von `CObject` abgeleiteten Klassen, ob der Klassentyp korrekt ist. Beispielsweise kann man testen, ob ein Zeiger auf das Ansichtsobjekt von der richtigen Ansichtsklasse ist:

```
ASSERT_KINDOF(CYourSpecialView, pYView);
```

Das Dialogfeld **Debug Assertion Failed!** erscheint, wenn das Objekt nicht vom korrekten Klassentyp oder aller davon abgeleiteten Typen ist.

Achten Sie darauf, keinerlei Code, der für die normale Programmausführung erforderlich ist, in `ASSERT`-Makros unterzubringen, da sie in der Release-Version ausgeschlossen werden. Eine allgemeine Quelle für Fehler bei Anwendungen im Release-Modus, die man schwer auffinden kann, ist etwa folgender Code:

```
int a = 0;
ASSERT(++a > 0);
if (a > 0) MyFunc();
```

Im Debug-Modus inkrementiert dieser Code die Integer-Zahl `a` in der `ASSERT`-Zeile und ruft dann in der folgenden Zeile die Funktion `MyFunc` auf, weil `a` größer als `0` ist. Angenommen, Ihr Verkaufsteam ist darauf versessen, Ihre neue Anwendung vorzuführen. Da es im Debug-Modus keinerlei Probleme gegeben hat, gehen Sie davon aus, daß alles in Ordnung ist. Also kompilieren Sie die Anwendung ruhigen Gewissens im Release-Modus und geben sie an die Verkaufsabteilung weiter. Diese zeigt die Anwendung dem Kunden, und die Anwendung stürzt ab! Alles nur deshalb, weil das `++a` nicht ausgeführt wird - der Release-Modus schließt die `ASSERT`-Zeilen aus.

Das Makro `VERIFY` unterstützt Sie bei diesem Problem. `VERIFY` funktioniert wie `ASSERT` und bringt im Debug-Modus das gleiche Dialogfeld **Debug Assertion Failed!** auf den Bildschirm, wenn der Ausdruck `0` ist. Im Release-Modus wird der Ausdruck allerdings trotzdem ausgeführt, wobei aber das Ergebnis `0` nicht zur Anzeige des Dialogfelds **Debug Assertion Failed!** führt. Vielleicht neigen Sie nun dazu, `VERIFY` immer zu verwenden, wenn Sie einen Ausdruck auswerten, und `ASSERT`, wenn Sie nur beim Debuggen eine Überprüfung vornehmen wollen. Wenn Sie im vorherigen Beispiel `ASSERT` durch `VERIFY` ersetzen, läuft der Code auch im Release-Modus ordnungsgemäß:

```
VERIFY(++a > 0);
```

Höchstwahrscheinlich setzen Sie aber `VERIFY` ein, um Rückgabecodes aus Funktionen zu prüfen:

```
VERIFY(MyFunc() != FALSE);
```

Haltepunkte und Einzelschrittbetrieb

Ein Großteil der Probleme läßt sich wahrscheinlich am effektivsten mit den Debugging-Werkzeugen für den Einzelschrittbetrieb und das Setzen von Haltepunkten lösen. Die Unterstützung für verschiedene Typen von Haltepunkten und die verfügbaren Informationen im Einzelschrittbetrieb sind in Visual C++ sehr ausgeklügelt. Hier können wir nur eine Vorstellung davon vermitteln, wie leistungsfähig dieses Werkzeug zur Fehlersuche ist.

Den Schlüssel für den Einzelschrittbetrieb bilden die Haltepunkte. Man kann einen Haltepunkt an jeder beliebigen Stelle im Code setzen und dann das Programm über den Debugger starten. Ist der Haltepunkt erreicht, erscheint der Code im Editorfenster an der Position des Haltepunkts. Ab hier können Sie die Programmausführung im Einzelschrittbetrieb oder im normalen Ausführungsmodus fortsetzen.

Um einen Haltepunkt zu setzen, markieren Sie die betreffende Codezeile (indem Sie mit dem Bearbeitungscursor im Editorfenster in dieser Zeile klicken) und klicken dann auf das Symbol **Haltepunkt** in der **Minileiste erstellen** (siehe Abbildung E.8) oder drücken (F9). Noch komfortabler lassen sich Haltepunkte setzen oder entfernen, indem Sie im Menü **Bearbeiten** den Befehl **Haltepunkte** wählen, um das Dialogfeld **Haltepunkte** zu öffnen (siehe Abbildung E.9). Ein gesetzter Haltepunkt ist durch einen roten Kreis neben der spezifizierten Zeile gekennzeichnet. Haltepunkte kann man nur für gültige Codezeilen festlegen, so daß Visual Studio manchmal einen von Ihnen gesetzten Haltepunkt automatisch auf die am nächsten liegende gültige Codezeile verschiebt.



Abbildung E.8:

Einen Haltepunkt fügen Sie in den Code über die Minileiste erstellen oder durch Drücken von (F9) ein.

Das Programm läuft dann normal, bis es den Haltepunkt erreicht. Hier stoppt die Programmausführung, und ein Pfeil zeigt auf die Zeile mit dem Haltepunkt. Jetzt können Sie über die Symbolleiste **Debug** im Einzelschrittbetrieb weiterarbeiten, wie es Abbildung E.10 zeigt.

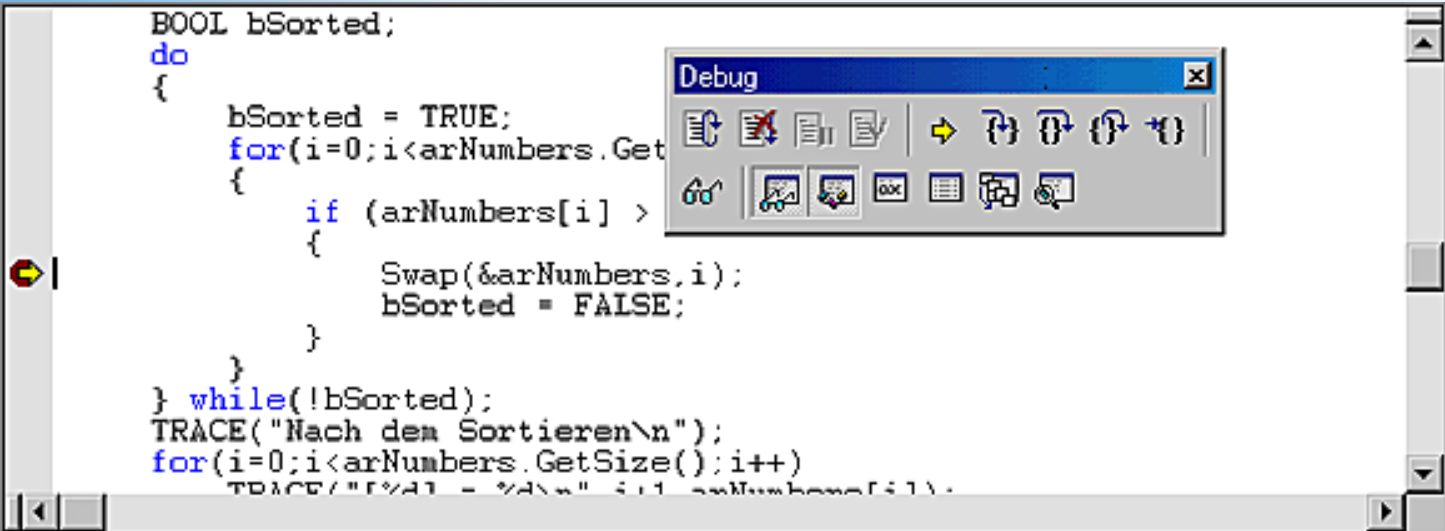


Abbildung E.10:**Der Debugger hat am Haltepunkt gestoppt und erlaubt die Ausführung des Programms im Einzelschrittbetrieb über die Symbolleiste Debug.**

Wenn der Debugger gestoppt hat, lässt sich der Inhalt der meisten Variablen anzeigen, indem man einfach den Cursor über die Variable im Editorfenster führt. Der Inhalt der Variablen erscheint dann in einer Art QuickInfo an der Cursorposition. Nähere Angaben zum Inhalt kann man erhalten, wenn man die Variable in das Überwachungsfenster zieht, wie es der nächste Abschnitt erläutert.

Um den Code in Einzelschritten zu durchlaufen, verwenden Sie die vier Schaltflächen mit den geschweiften Klammern auf der Symbolleiste **Debug** oder wählen im Menü **Debug** einen der Schrittbefehle aus. Die verfügbaren Schrittoptionen zeigt Tabelle E.4. Die Befehle sind sowohl im Menü **Debug** als auch auf der Symbolleiste **Debug** enthalten.

Tabelle E.4: Im Einzelschrittbetrieb verfügbare Schrittoptionen

Symbol	Schrittoption	Zugriffstaste	Wirkung
	In Aufruf springen	F11	Der Debugger führt die aktuelle Zeile aus. Befindet sich der Cursor über einem Funktionsaufruf, tritt der Debugger in die Funktion ein.
	Aufruf als ein Schritt	F10	Wie bei »In Aufruf springen«. Befindet sich jedoch der Cursor über einem Funktionsaufruf, führt der Debugger die Funktion mit normaler Geschwindigkeit aus und stoppt bei Rückkehr aus der Funktion. Man führt die Funktion praktisch in einem Schritt aus.
	Ausführen bis Rücksprung	Umschalt+F11	Der Debugger führt die restlichen Codezeilen der Funktion mit normaler Geschwindigkeit aus und stoppt bei der Rückkehr in die aufrufende Funktion.
	Ausführen bis Cursor	Strg+F10	Der Debugger führt das Programm normal aus, bis die festgelegte Cursorposition erreicht ist. Um die Position festzulegen, klicken Sie auf die Zeile, bis zu der der Debugger das Programm ausführen soll.
	Ausführen	F5	Setzt die Programmausführung mit normaler Geschwindigkeit fort, bis der Debugger den nächsten Haltepunkt erreicht.
	Debug beenden	Umschalt+F5	Stoppt den Debugger und kehrt in den Bearbeitungsmodus zurück.
	Erneut starten	Strg+Umschalt+F5	Startet das Programm erneut von Anfang an und stoppt auf der allerersten Codezeile.
	Ausführung anhalten		Hält ein bei normaler Geschwindigkeit ablaufendes Programm an.

	Code-Änderungen zuweisen	Alt+F10	Diese Option erlaubt es, den Code zu kompilieren, nachdem man Änderungen während einer Debug-Sitzung durchgeführt hat. Anschließend kann man das Debuggen von der letzten Position aus wiederaufnehmen.

Mit diesen Befehlen können Sie den Programmablauf überwachen und den Inhalt der Variablen verfolgen, wie sie der Code manipuliert. Der gelbe Pfeil im Editorfenster zeigt immer auf die als nächstes auszuführende Anweisung.

Der nächste Abschnitt beschäftigt sich näher mit den Debug-Fenstern, die Sie bei angehaltenem Debugger verwenden können.

Bearbeiten und Fortsetzen

Ein wesentliches neues Merkmal von Visual C++ 6 ist die Fähigkeit, den Code zu bearbeiten und dann fortzusetzen. Das bedeutet, daß man den Code ändern oder bearbeiten kann, während der Debugger angehalten hat. Nach der Bearbeitung wird der Befehl **Code-Änderungen zuweisen** im Menü **Debug** (sowie das entsprechende Symbol auf der Symbolleiste **Debug**) aktiviert. Sie können dann den Befehl (oder die entsprechende Schaltfläche) **Code-Änderungen zuweisen** wählen, um die vorgenommenen Änderungen am Code zu kompilieren und dann das Debuggen mit dem geänderten Code fortzusetzen. Auf diese Weise lassen sich während des Debuggens Programmfehler beseitigen, wobei man mit der Fehlersuche an der gleichen Stelle des Codes und vor allem mit den gleichen Variableneinstellungen fortsetzen kann. Insbesondere beim Debuggen größerer und komplexer Programme kommen die Vorteile dieses neuen Funktionsmerkmals voll zur Geltung.

Programmvariablen überwachen

Abbildung E.11 zeigt die Debug-Fenster **Überwachung** und **Variablen**. Die Fenster zeigen den Inhalt von Variablen an, wenn der Debugger gestoppt hat. Um die Fenster einzusehen, wählen Sie aus dem Menü **Ansicht** den Befehl Debug-Fenster und anschließend den jeweiligen Eintrag des aufklappenden Untermenüs oder klicken auf die Symbole der Symbolleiste.

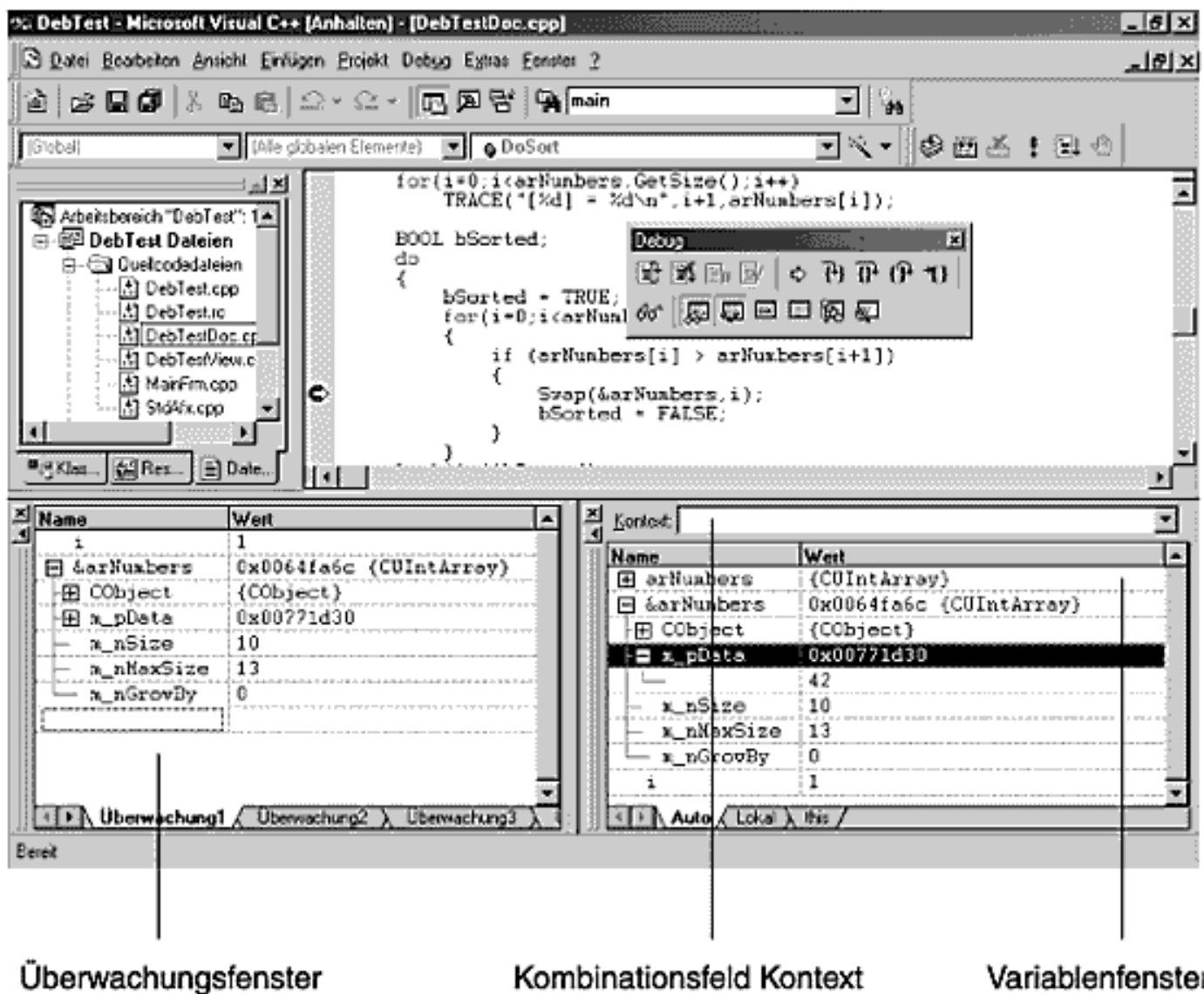


Abbildung E.11:
Das Überwachungsfenster zeigt den Inhalt von Variablen während des Debuggens an.

Das Variablenfenster zeigt immer die lokalen Variablen der im Kombinationsfeld **Kontext** am oberen Rand des Fensters aufgeführten Funktion an. Um zur aktuellen Funktion zu gelangen, können Sie die Liste des Kombinationsfelds öffnen. Es erscheinen dann alle Funktionen, die nacheinander aufgerufen wurden. Es handelt sich hierbei um den sogenannten Aufruf-Stack. Er zeigt den aktuellen Kontext innerhalb des Programms mit einer Liste von Funktionen, die aufgerufen wurden, um zur momentan ausgeführten Funktion des Programms zu gelangen, wo der Debugger gestoppt hat. Wenn Sie eine andere Funktion wählen, werden die relevanten lokalen Variablen für diese Funktionsebene angezeigt.

Die Objektzeiger kann man erweitern, indem man auf das Plusymbol neben dem Zeigernamen klickt. Der spezielle C++-Zeiger **this** wird immer für Elementfunktionen von Klassen angezeigt und lässt sich öffnen, um alle Elementvariablen für das aktuelle Objekt anzuzeigen.

In das Überwachungsfenster kann man Variablennamen über die Tastatur eintragen oder Variablen aus dem Editorfenster ziehen (nachdem man sie mit dem Mauszeiger markiert und invertiert hat). Die in den angezeigten Variablen gespeicherten Werte sind solange zu sehen, bis die Variablen den Gültigkeitsbereich verlassen (das heißt, nicht mehr für die momentan untersuchte Funktion relevant sind).

Es lassen sich auch einfache Typumwandlungen und Array-Indizes im Überwachungsfenster eingeben, um die zugehörigen Werte anzuzeigen. Klicken mit der rechten Maustaste schaltet die angezeigten numerischen Werte zwischen hexadezimaler und dezimaler Darstellung um. Bei der schrittweisen Untersuchung des

Programms werden die im Überwachungsfenster angezeigten Variablen jeweils aktualisiert, so daß man verfolgen kann, wie das Programm die Variablen ändert.

Andere Fenster des Debuggers

Weitere Fenster des Debuggers sind über **Ansicht / Debug-Fenster**, das Menü **Debug** bzw. über die Symbole auf der Symbolleiste **Debug** erreichbar. Zu diesen Fenstern gehören:

- **Schnellüberwachung:** Klickt man auf eine Variable im Listing und wählt Schnellüberwachung oder drückt (^)+(F9), erscheint der Inhalt der markierten Variablen. Man kann Variablen auch direkt eingeben und dann auf die Schaltfläche Zu Überwachung hinzufügen klicken, um sie in das Fenster Überwachung 1 zu übertragen.
- **Register:** Das Fenster Register zeigt die aktuellen Werte im Registersatz der CPU. Diese Informationen sind dann interessant, wenn man Probleme auf der Ebene von Maschinen- oder Assemblercode aufspürt.
- **Speicher:** Dieses Fenster zeigt den Speicher im Adreßbereich der Anwendung an. In den jeweiligen Spalten stehen die Adresse, die Hexadezimalwerte und die Zeichenwerte für jeweils 8 Byte. Die Darstellung läßt sich zwischen Byte, Short und Long umschalten, indem man mit der rechten Maustaste klickt, um die entsprechenden Befehle des Kontextmenüs einzublenden.
- **Aufrufliste:** Dieses Fenster zeigt die Liste der Funktionen, die auf dem Weg bis zur aktuellen Funktion aufgerufen wurden, und die Parameterwerte, die an jede Funktion übergeben wurden. Damit kann man untersuchen, wie der Programmablauf eine bestimmte Funktion erreicht hat. Wenn man auf eine der Funktionen doppelklickt, erscheint im Editorfenster die Position, wo der Funktionsaufruf im Code erfolgt.
- Wo der Quellcode nicht verfügbar ist, sind die Funktionseinträge wie folgt dargestellt:

KERNEL32! bff88f75()

- Klickt man auf diese Einträge, erscheint der Assemblercode statt dem C++-Code.
- **Disassemblierung:** Mit diesem Befehl kann man die Darstellung umschalten - entweder C++-Code gemischt mit Assemblercode oder nur C++-Code. Wenn der Quellcode nicht verfügbar ist, erscheint nur der Assemblercode.

Zusätzliche Debugging-Tools

Neben den integrierten Debugger-Werkzeugen gibt es verschiedene eigenständige und nützliche Hilfsprogramme. Diese lassen sich über das Menü **Extras** und den jeweiligen Menübefehl starten.

Die zusätzlichen Programme dienen vor allem dazu, spezielle Elemente des Betriebssystems wie Windows-Nachrichten, laufende Prozesse und registrierte OLE-Objekte zu verfolgen, um die verfügbaren Informationen beim Debuggen einer Anwendung zu erweitern.

Spy++

Spy++ gehört zweifellos zu den nützlichsten dieser Werkzeuge. Mit Spy++ kann man die hierarchischen Beziehungen von übergeordneten zu untergeordneten Fenstern, die Position und Flageinstellungen für Fenster und die Basisklassen der Fenster sehen. Außerdem lassen sich die Nachrichten überwachen, die Windows an ein Fenster schickt.

Beim Start von Spy++ erscheinen alle Fenster auf dem aktuellen Desktop, deren gleichgeordnete Fenster und die Basisklassen der Fenster für jedes Objekt (siehe Abbildung E.12). In der in Abbildung E.12 dargestellten Ansicht wurde ein Bildlauf nach unten bis zur CD-Wiedergabe von Microsoft Windows durchgeführt. Spy++ zeigt alle Schaltflächen und Kombinationsfelder, die wiederum selbständige Fenster als untergeordnete

Fenster zum Hauptfenster der CD-Wiedergabe sind.

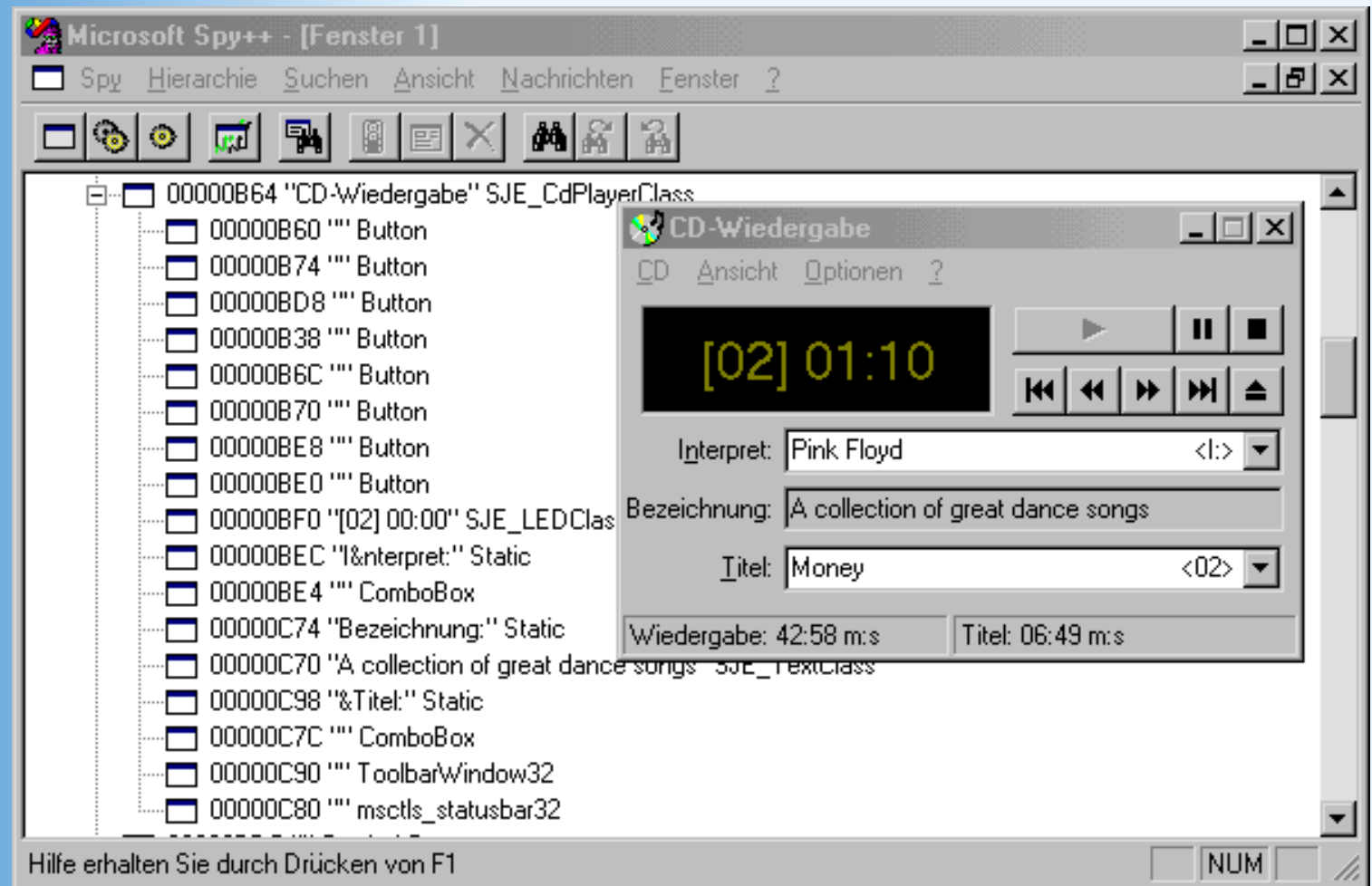


Abbildung E.12:
Das Fenster von Spy++ mit dem Abschnitt CD-Wiedergabe des Windows-Desktop.

Das Menü **Spy** enthält folgende Befehle:

- **Nachrichten:** Die Nachrichtenansicht ist wahrscheinlich die nützlichste Option, da man damit die Nachrichten beobachten kann, die an ein Fenster (einschließlich der eigenen Anwendung) geschickt werden. Außerdem kann man die Nachrichten filtern, damit man nicht von einer Lawine der Nachrichten von Mausbewegungen überrollt wird.
- Um die Nachrichtenansicht zu verwenden, wählen Sie diesen Befehl, der das Dialogfeld **Nachrichtenoptionen** öffnet (siehe Abbildung E.13). Das Suchwerkzeug (das Symbol im Abschnitt **Fenstersuchwerkzeug**) können Sie über jedes Fenster im System ziehen. Bei der Verschiebung zeigt es die Einzelheiten des Fensters an. Spy++ hebt auch das ausgewählte Fenster hervor, so daß man die Rahmen- und Client-Fenster sehen kann. Wenn man das zu betrachtende Fenster lokalisiert hat, ist das Werkzeug einsatzbereit. Jetzt lassen sich auch die anderen Registerkarten verwenden, um Filteroptionen und Formate der Ausgabe einzustellen. Wenn Sie die Arbeit beendet haben, klicken Sie auf **OK**, um das Dialogfeld **Nachrichtenoptionen** zu schließen.

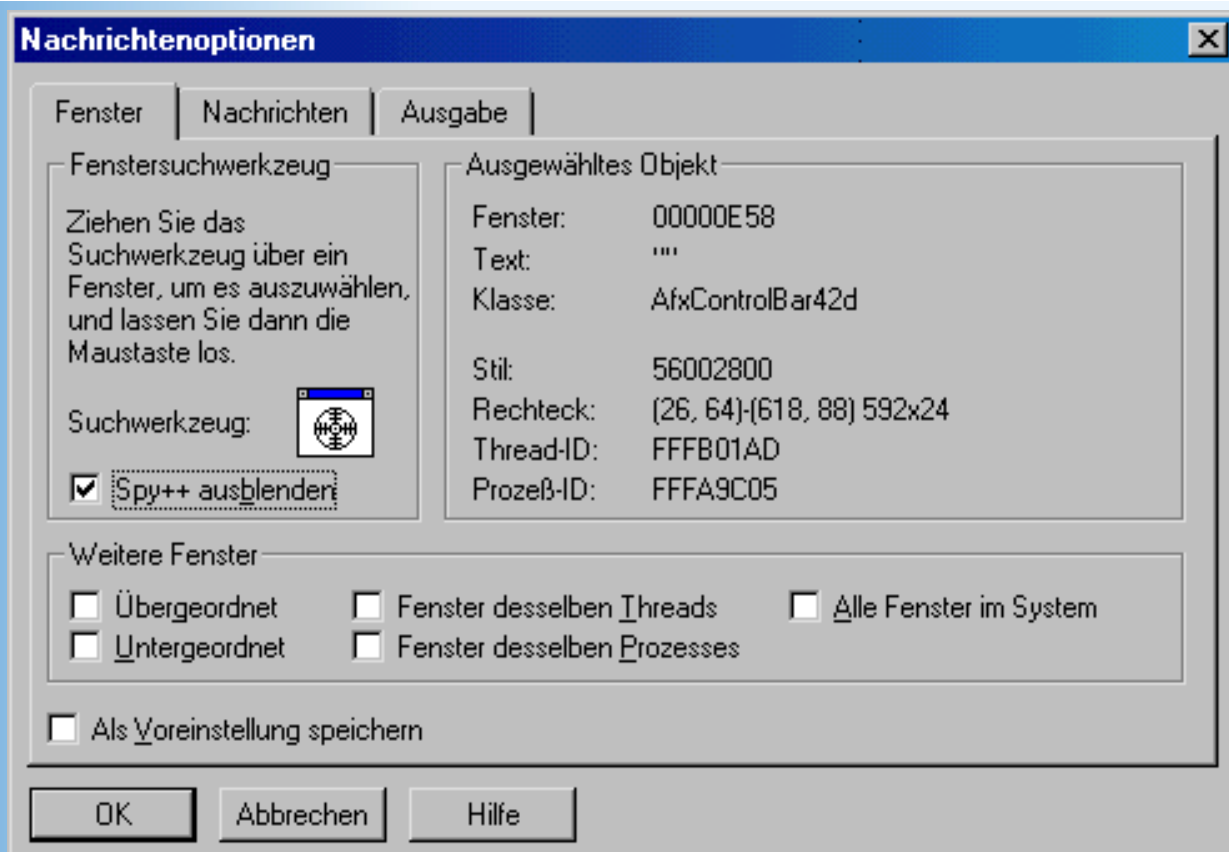


Abbildung E.13:

Mit den Nachrichtenoptionen von Spy++ lassen sich Fenster lokalisieren.

- Die Ausgabe in Abbildung E.14 zeigt die Nachrichten, die von der Symbolleiste einer normalen SDI-Anwendung stammen. Wie Sie sehen, erhalten Sie viele Nachrichten zu Mausbewegungen und Cursortests, es ist aber auch die bekannte Nachricht `WM_LBUTTONDOWN` mit den zugehörigen Positionsparametern zu erkennen.

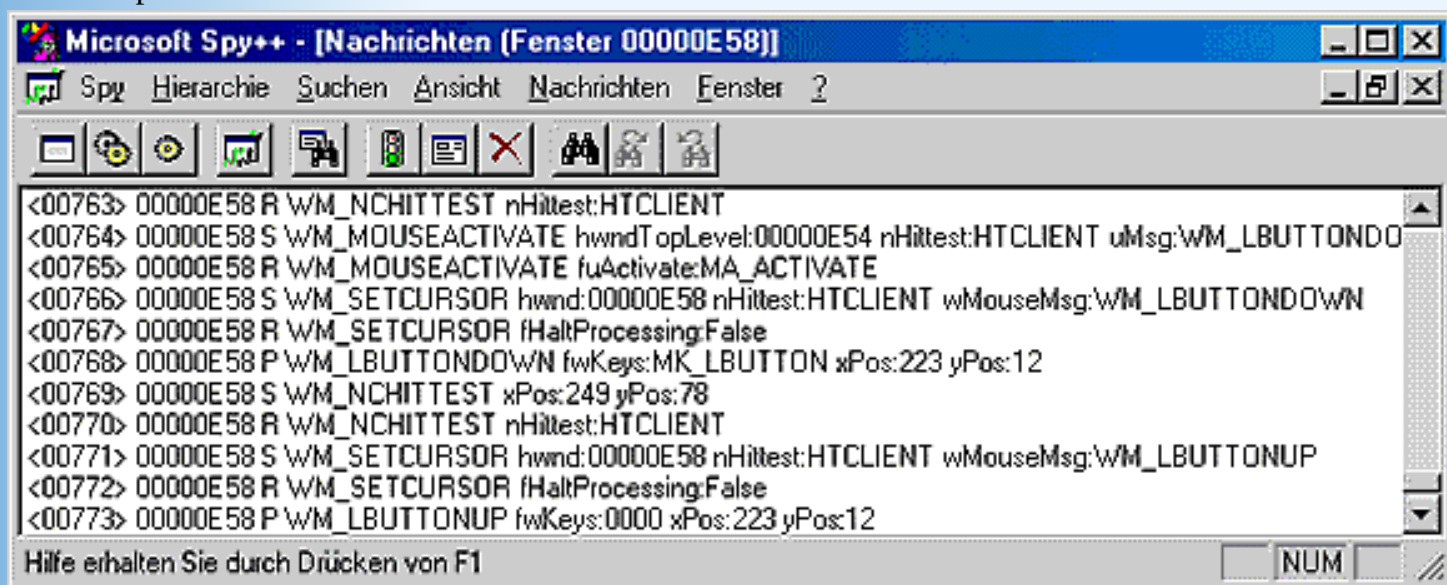


Abbildung E.14:

Windows-Nachrichten für eine Symbolleiste, die durch Spy++ angemeldet ist.

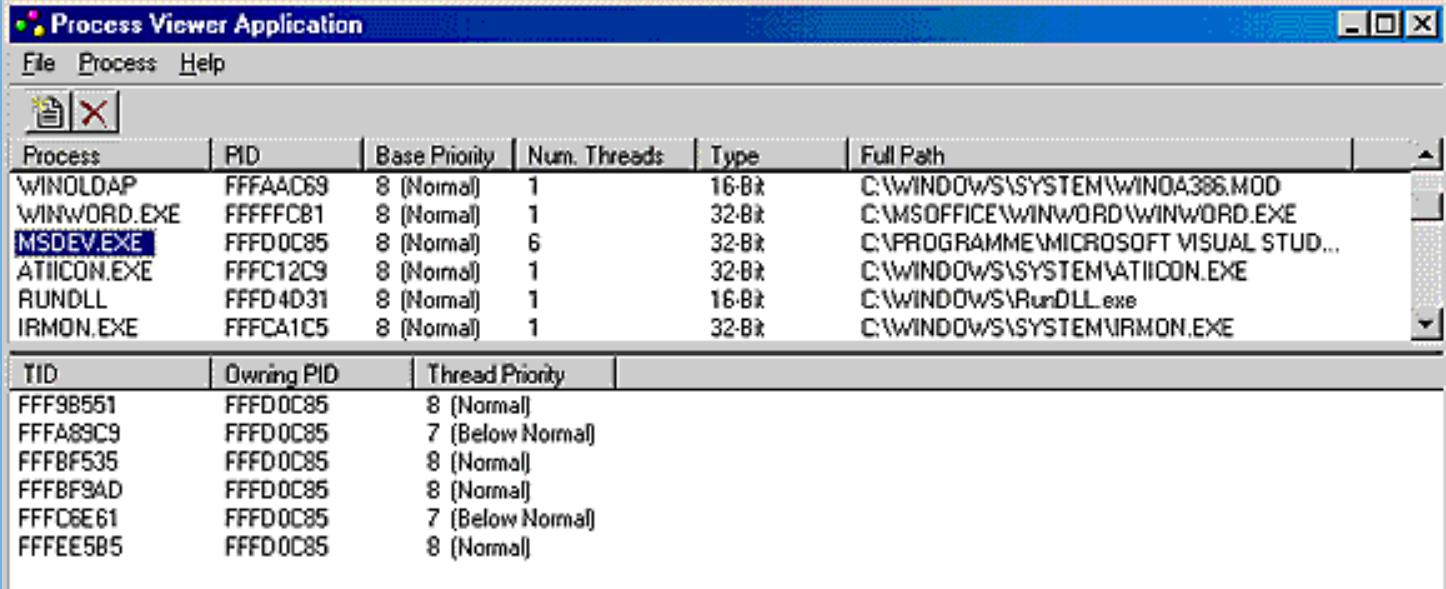
- Fenster:** Die Fensteransicht ist die bereits in Abbildung E.12 dargestellte Ansicht mit dem Layout und der Struktur des Windows-Desktop. Wenn man auf eines dieser Fenster doppelklickt, erscheint ein Eigenschaftsblatt mit den Positionsinformationen und Attributen des ausgewählten Fensters. Um die Informationen zu aktualisieren, müssen Sie im Menü Fenster den Befehl Aktualisieren wählen.
- Prozesse:** In der Prozeßansicht lassen sich alle laufenden Programme überwachen. Diese Ansicht kann geöffnet bleiben, um jeden Thread und alle mit diesen Threads verbundenen Fenster anzuzeigen.

- **Threads:** Die Option Threads zeigt die gleichen Einzelheiten ohne die Hierarchieebenen der Prozesse, so daß man jeden auf dem System laufenden Thread und die von jedem Thread besessenen Fenster sehen kann.

Spy++ läßt sich aufgrund der umfangreichen Funktionen an dieser Stelle nicht erschöpfend behandeln. Das Programm ist aber als Werkzeug unübertroffen, wenn Sie sich näher mit der Struktur der Hierarchie und der Nachrichtenverarbeitung unter Windows beschäftigen möchten. Mit Spy++ können Sie eine Menge Kenntnisse sammeln, indem Sie sich einfach einmal kommerzielle Anwendungen ansehen. Spy++ ist auch ein hervorragendes Hilfsmittel, um Probleme in der eigenen Anwendung mit der Nachrichtenverarbeitung aufzuspüren, um sich zu vergewissern, daß die Fenster die korrekten Nachrichten erhalten, und um zu sehen, wie diese Nachrichten aufeinanderfolgen.

Prozess-Viewer

Mit der Anwendung Prozess-Viewer (PView95.exe) lassen sich alle Prozesse detaillierter als mit Spy++ anzeigen. Um das Programm zu starten, wählen Sie **Start / Programme / Microsoft Visual Studio 6.0 / Microsoft Visual Studio 6.0-Dienstprogramme / Prozess-Viewer**. Die Anwendung listet die Prozesse auf, die auf Ihrem Computer laufen. Um die Liste zu sortieren, klicken Sie auf den jeweiligen Spaltenkopf. Markieren Sie dann einen Prozeß, um alle damit verbundenen Threads anzuzeigen. Abbildung E.15 zeigt die Prozeßanzeige, die mit der ausgewählten Anwendung Developer Studio (MSDEV.EXE) läuft und die Vielzahl der zugehörigen Threads anzeigt.



The screenshot shows the 'Process Viewer Application' window. The top menu bar includes 'File', 'Process', and 'Help'. Below the menu is a toolbar with icons for file operations. The main window is divided into two panes. The top pane displays a list of running processes with columns: Process, PID, Base Priority, Num. Threads, Type, and Full Path. The process 'MSDEV.EXE' is selected and highlighted. The bottom pane shows the threads for the selected process, with columns: TID, Owning PID, and Thread Priority.

Process	PID	Base Priority	Num. Threads	Type	Full Path
WINOLDAP	FFFAAC69	8 (Normal)	1	16-Bit	C:\WINDOWS\SYSTEM\WINOAS386.MOD
WINWORD.EXE	FFFFFC81	8 (Normal)	1	32-Bit	C:\MSOFFICE\WINWORD\WINWORD.EXE
MSDEV.EXE	FFFD0C85	8 (Normal)	6	32-Bit	C:\PROGRAMME\MICROSOFT VISUAL STUD...
ATIICON.EXE	FFFC12C9	8 (Normal)	1	32-Bit	C:\WINDOWS\SYSTEM\ATIICON.EXE
RUNDLL	FFFD4D31	8 (Normal)	1	16-Bit	C:\WINDOWS\RunDLL.exe
IRMON.EXE	FFCA1C5	8 (Normal)	1	32-Bit	C:\WINDOWS\SYSTEM\IRMON.EXE

TID	Owning PID	Thread Priority
FFF9B551	FFFD0C85	8 (Normal)
FFFA89C9	FFFD0C85	7 (Below Normal)
FFF8F535	FFFD0C85	8 (Normal)
FFF8F9AD	FFFD0C85	8 (Normal)
FFFC6E61	FFFD0C85	7 (Below Normal)
FFFE5B5	FFFD0C85	8 (Normal)

Abbildung E.15:
Prozeßanzeige zeigt MSDEV.EXE und dessen Threads.

Der OLE/COM-Objektkatalog

Das Dienstprogramm OLE/COM-Objektkatalog zeigt alle in Ihrem System registrierten OLE/COM-Objekte einschließlich der ActiveX-Steuerelemente, Typenbibliotheken, eingebetteten Objekte, Automatisierungsobjekte und vieler anderer Kategorien.

Man kann sogar Instanzen der verschiedenen Objekte erzeugen und deren Schnittstellen im Detail ansehen. Der OLE/COM-Objektkatalog ist hilfreich, wenn man OLE/ COM-Anwendungen entwickelt oder nach einem undefinierten ActiveX-Steuerelement sucht.

Die MFC-Ablaufverfolgung (MFC Tracer)

Mit dem Dienstprogramm MFC-Ablaufverfolgung (siehe Abbildung E.16) kann man die normale Verfolgung stoppen oder spezielle Ablaufverfolgungen in die Protokollierung aufnehmen. Wenn Sie dieses Werkzeug wählen, erscheint eine Gruppe von Kontrollkästchen, über die Sie die Optionen zur Ablaufverfolgung einstellen können.

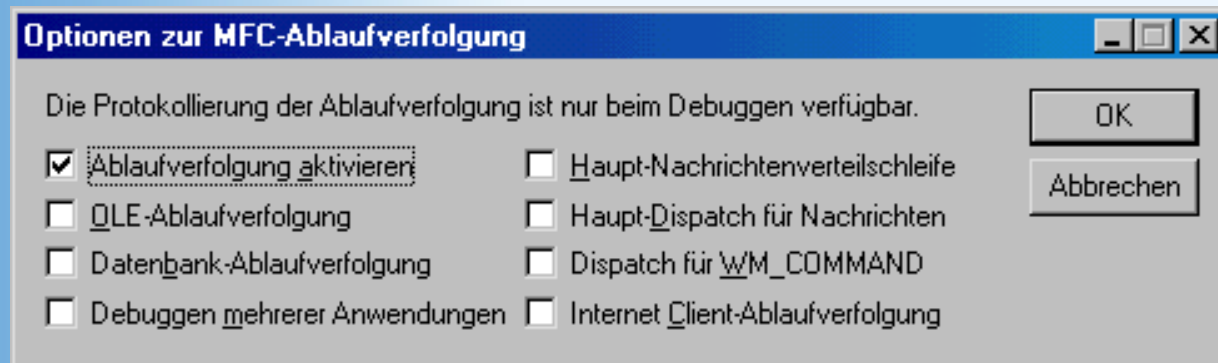


Abbildung E.16:
Die Optionen der MFC-Ablaufverfolgung

Es lassen sich Windows-Nachrichten, Datenbanknachrichten, OLE-Nachrichten und viele andere Stufen der Ablaufverfolgung hinzufügen, um ausgewählten Problemen auf die Spur zu kommen. Diese Nachrichten werden dann durch den MFC-Code für die verschiedenen ausgewählten Flags generiert.

Sie können sogar die normale Ablaufverfolgung, die von Ihrer Anwendung generiert wird, unterdrücken, wenn Sie das Kontrollkästchen **Ablaufverfolgung aktivieren** ausschalten.