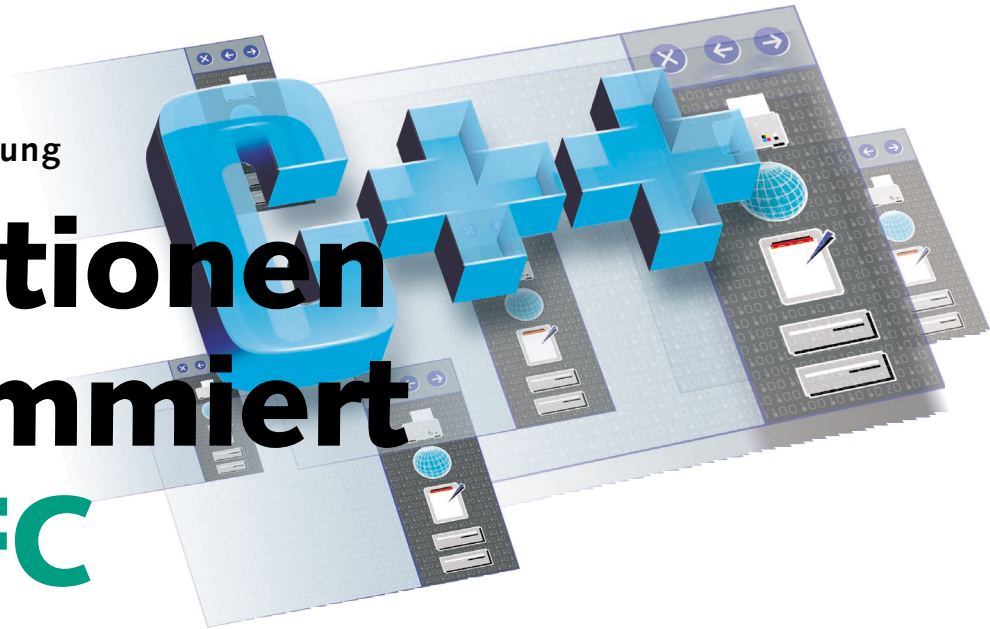




Anwendungsprogrammierung

Interaktionen programmiert mit MFC



Nun geht's an die Programmierung eines echten Windows-Programms, mit allem Drum und Dran: Sie programmieren ein kleines interaktives Malprogramm, das die gemalten Elemente in mehreren Fenstern gleichzeitig anzeigen kann.

THOMAS WÖLFER

Kommandozeile ade, richtige Windows-Programme sind ereignisgesteuert. Für solche Anwendungsprogrammierung stellt VC++ in erster Linie die MFC-Bibliothek zur Verfügung. Die Entwicklung von Programmen mit MFC unterscheidet sich dabei dramatisch von der "klassischen" Art, Programme für Windows zu schreiben.

■ Grundlegendes zu den MFC

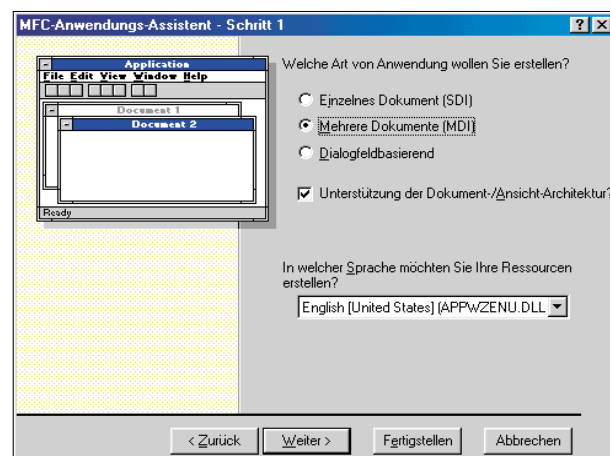
Früher, also vor der Verfügbarkeit von MFC, wurden die meisten Windows-Programme unter Benutzung der ganz normalen Windows API geschrieben. Bei der Windows API handelt es sich um ein C-Interface, mit dem die Funktionen des Betriebssystems zur Verfügung gestellt werden. Dabei gibt es zum Beispiel eine Funktion *CreateWindow()*, mit der ein Fenster erzeugt werden kann, es gibt eine Funktion *ShowWindow()*, die ein Fenster anzeigt, und so weiter.

Ein ebenfalls wichtiger Bestandteil eines klassischen Windows-Programms ist die Message-Loop, eine Schleife, in der alle von Windows eingehenden Nachrichten behandelt werden, typischerweise mit einem großen *switch/case*-Statement. Die Arbeit, all diese Elemente von Hand ins Anwen-

derprogramm einzubauen und entsprechende Funktionen zu programmieren, nimmt die MFC dem Programmierer ab. Dazu steht nicht nur die Bibliothek al-

MDI- oder SDI-Anwendung bevorzugen, wie das Layout der Fenster sein soll und auch, welche Elemente diese Fenster haben sollen. Dazu zählt zum Beispiel ein Parameter, mit dem Sie festlegen können, ob Ihr Hauptfenster eine Statuszeile haben soll oder nicht. Diese Parameter werden im Assistenten in mehreren Schritten nach und nach festgelegt. Sind alle bestimmt, erzeugt der Assistent mehrere Quellcode- (cpp) und Header-Dateien (.h) und bei Bedarf auch einen rudimentären Satz an Windows-Ressourcen.

Bei diesen Windows-Ressourcen handelt es sich um eine ganz spezielle Art von Daten. Bisher haben Sie in Ihrem Beispielprogramm zwischen Daten, die zum Beispiel in Form von konstanten Strings auftauchen und ausführbarem Code unterschieden. Beiden Windows-Ressourcen handelt es sich im Wesentlichen auch um konstante Daten, die im Programm eingebettet werden. Allerdings werden diese auf eine spezielle Art eingebettet: Nämlich so, dass Sie die Ressourcen spä-

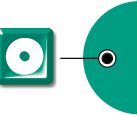


DAS BEISPIELPROGRAMM wird als Windows-MDI-Anwendung angelegt. Damit können multiple Dokumente und Ansichten gleichzeitig angezeigt werden.

leine zur Verfügung, sondern VC++ stellt dem Programmierer noch zwei spezielle Assistenten zur Seite: den Anwendungs-Assistenten und den Klassen-Assistenten.

Der Anwendungs-Assistent kommt dabei nur einmal zum Zuge. Mit diesem Assistenten wird ein Grundgerüst einer Windows-Anwendung anhand von bestimmten Parametern erzeugt. Sie können zum Beispiel festlegen, ob Sie eine

ne ganz spezielle Art von Daten. Bisher haben Sie in Ihrem Beispielprogramm zwischen Daten, die zum Beispiel in Form von konstanten Strings auftauchen und ausführbarem Code unterschieden. Beiden Windows-Ressourcen handelt es sich im Wesentlichen auch um konstante Daten, die im Programm eingebettet werden. Allerdings werden diese auf eine spezielle Art eingebettet: Nämlich so, dass Sie die Ressourcen spä-



ter unter Verwendung eines eindeutigen Identifizierers vom Typ Integer ansprechen können. Mit anderen Worten: Jede Ressource hat eine eigene Nummer und mit dieser Nummer können Sie die Ressource dann laden.

Es gibt verschiedene Arten von Windows-Ressourcen. Dazu zählen zum Beispiel Menüs und Bitmaps, aber auch Dialogboxen. Die Windows-Ressourcen werden beim Bauen des Programms von einem speziellen Werkzeug, dem Ressource Compiler, in ein binäres Format übersetzt und dann später vom Linker in dieser binären Form ins Programm eingebunden. Im Zuge des Beispielprogramms lernen Sie auch, wie Sie diese in der ausführbaren Datei vorliegenden Ressourcen zur Laufzeit des Programms benutzen können.

Die vom Anwendungs-Assistenten erzeugten Quellcode-Dateien implementieren bereits eine vollständig lauffähige Windows-Anwendung mit allem Drum und Dran: Man kann die Dateien direkt übersetzen und das Programm testen.

Es hat dann bereits – zumindest sofern diese Option bei der Erzeugung ausgewählt wurde – die Möglichkeit, mehrere Fenster gleichzeitig anzuzeigen, verfügt über eine Druckvorschau und hat auch ein Menü, in dem die wichtigsten Befehle enthalten sind. Echte Anwendungslogik fehlt allerdings – schließlich kann der Assistent nicht erraten, was Ihr Programm eigentlich tun soll.

Im Gegensatz zum Anwendungs-Assistenten, der ein Einwege-Tool ist, kommt der Klassen-Assistent bei der Arbeit am Programm immer wieder zum Einsatz.

Mit diesem Werkzeug legen Sie zum Beispiel fest, welche Funktion aufgerufen werden soll, wenn ein bestimmter Menübefehl ausgewählt wurde. Sie können mit diesem Werkzeug auch neue Klassen definieren oder vorhandene erweitern bzw. bearbeiten. Auch dies werden Sie im Zuge der Programmierung des Beispiels erlernen.

Das Beispielprogramm wird eine einfache MDI-Anwendung. Sie können dabei mehrere Dokumente gleichzeitig geöffnet haben und für jedes Dokument mehrere Fenster gleichzeitig geöffnet halten. In den Fenstern werden Sie interaktiv Kreise und Striche zeichnen. Die Daten, die zu diesen graphischen Elementen gehören, werden in einer Dokumentenklasse abgelegt.

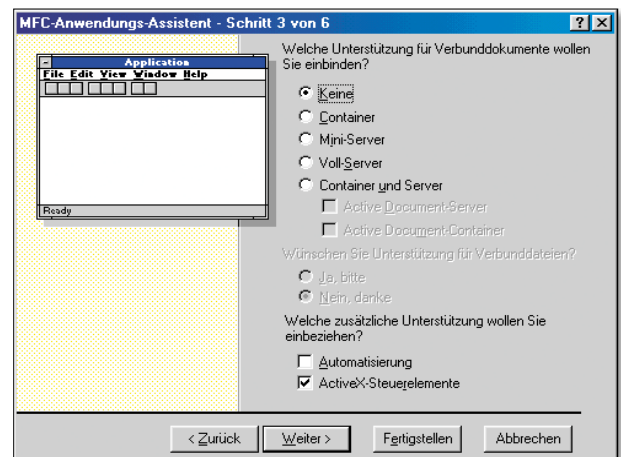
■ Anwendung erstellen

Zunächst brauchen Sie für das nun kommende Programm ein neues VC++-Projekt. Anders als bisher wählen Sie als Projekttyp diesmal nicht *Win32 Konsolen-Anwendung* aus, sondern den Projekt-Typ *MFC-Anwendungsassistent (exe)*. Daraufhin erscheint der eingangs bereits genannte MFC-Anwendungs-Assistent. Hier sind verschiedene Optionen in mehreren Schritten auszuwählen. Im Wesentlichen können Sie dabei die vorgeschlagenen Default-Werte einfach übernehmen, einzig beim ersten Schritt, in

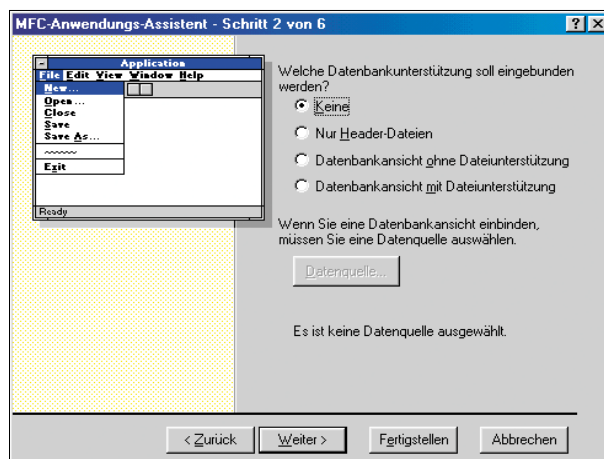
gabewerte und am Ende erzeugt der Assistent den benötigten Quellcode.

Wenn Sie damit fertig sind, können Sie Ihr erstes eigenes Windows-Programm auch schon gleich übersetzen und ausprobieren. Das geht genau wie bei den Kommandozeilenprogrammen. Hier gibt es also nichts umzuwöhnen.

Sie starten den Übersetzungsvorgang einfach mit *[F5]* und VC++ wird das fertig übersetzte Programm gleich im Debugger starten. Wie Sie dann schnell feststellen können, hat das Programm schon



BEI DIESEM DIALOG zeigen sich die negativen Seiten der deutschen Version von VC++: Es wurden teilweise auch Begriffe übersetzt, für die es einfach keine deutschen Worte gibt. Mit "Verbunddokumenten" sind zum Beispiel OLE Compound Documents gemeint.



MFC UNTERSTÜTZT AUCH verschiedene Arten des Datenbankzugriffs, allerdings werden diese fürs Beispielprogramm nicht benötigt.

dem der Typ der Anwendung ausgewählt wird, müssen Sie eine Veränderung vornehmen: Hier wählen Sie die MDI-Anwendung aus. Ansonsten verwenden Sie, wie gesagt, einfach die Vor-

einen erstaunlichen Funktionsumfang: Sie können bereits mehrere Fenster öffnen, haben eine Druckvorschau und auch der Dialog zum Öffnen von Dateien sowie der *About*-Dialog können bereits angezeigt werden. Nun ist es also an der Zeit, eigenen Programmcode hinzuzufügen. Dazu müssen Sie aber ein paar Dinge über die vorliegenden Klassen wissen. Der große Trick bei MFC-Programmen ist immer der, den eigenen Code an der richtigen Stelle einzufügen.

■ MFC-MDI – Grundlagen

Eine MFC-MDI-Anwendung setzt sich aus einer kleinen Handvoll an Klassen zusammen. Welche das sind, können Sie mit der *ClassView* im Arbeitsbereich schnell überblicken. Zum einen ist da eine Klasse, die von *CWinApp* abgeleitet wurde. Diese Klasse kapselt die Anwendung und enthält nur Code, der auf der Anwendungsebene wichtig ist. Dazu zählt zum Beispiel das Öff-

nen des Hauptfensters und das Initialisieren von Anwendungsdaten. Ferner stellt die Anwendungsklasse Methoden zum Lesen und Schreiben von Daten im anwendungsbezogenen Teil der Registry zur Verfügung.

Damit Sie einen Überblick über die zur Verfügung stehende Funktionalität bekommen, sollten Sie die Online-Hilfe nicht nur zur Anwendungsklasse, sondern auch zu den anderen verwendeten Klassen, zumindest in der Übersicht, einmal überfliegen. Jetzt wäre dazu der geeignete Zeitpunkt.

Zusätzlich zur Anwendungsklasse gibt es noch vier weitere umfangreiche Klassen:

- die MainFrame-Klasse,
- die ChildFrame-Klasse,
- die View-Klasse und
- die Document-Klasse.

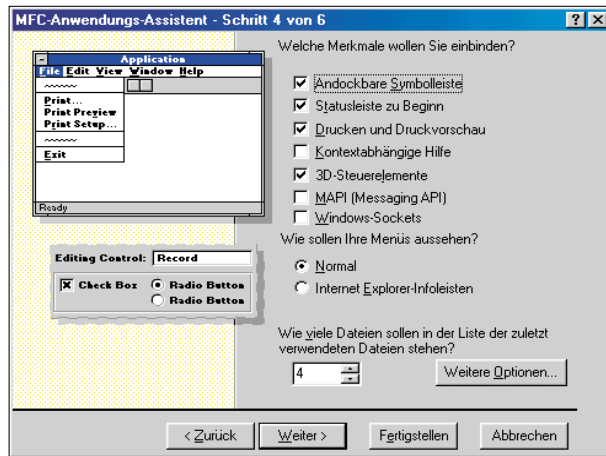
Die MainFrame-Klasse kapselt das Hauptfenster Ihrer Anwendung und kümmert sich dabei um Dinge, wie die Anzeige des Menüs und der Toolbars. Im Beispielprogramm werden Sie diese Klasse nicht verändern.

Auch die ChildFrame-Klasse bleibt im Beispielprogramm unangetastet. Diese Klasse kapselt den Bereich, in dem die einzelnen Views angezeigt werden. Sie ist also unmittelbar in die MainFrame-Klasse eingebettet und nur für die Verwaltung der MDI-Kindfenster zuständig. (Bei einer SDI-Anwendung gibt es diese Klasse nicht, da kümmert sich das MainFrame selbständig um die View-Verwaltung.)

Die View- und die Document-Klasse werden von Ihnen im Zuge des Beispiels verändert. Die Document-Klasse ist für die Verwaltung der in der Anwendung anfallenden Daten zuständig. Nachdem Sie beim Beispielprogramm Linien und Kreise erzeugen werden, wird das Document die Start- und Endpunkte sowie die Kreismittelpunkte speichern – natürlich nicht als Zahlenwerte, sondern in Form von Objekten. Doch dazu später mehr.

Die View-Klasse ist zuständig für die Visualisierung der im Document ent-

haltenen Daten. Mit anderen Worten: Hier werden die Linien und Kreise tatsächlich gemalt, zumindest in der Theorie,

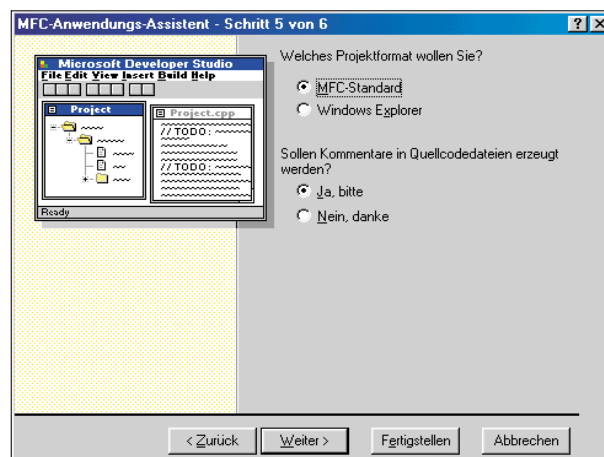


IN DIESEM SCHRITT DES ASSISTENTEN können sie verschiedene Parameter fürs Programm einstellen. Dazu zählt zum Beispiel auch, ob das Programm eine Statusleiste haben soll oder nicht.

denn in der Praxis werden sich Ihre graphischen Objekte selbst malen können.

■ Nächster Schritt: Eigener Code

Das Beispiel soll wie bereits erwähnt ein kleines Graphikprogramm werden, mit dem Linien und Kreise gezeichnet werden können. Dazu braucht man verschiedene Dinge, und das sind die folgenden:



FÜRS BEISPIELPROGRAMM wird das MFC Standard Layout ausgewählt.

- ein Objekt, das eine Linie kapselt,
- ein Objekt, das einen Kreis kapselt,
- eine Funktion, mit der das Erzeugen neuer Linien angestoßen werden kann,

- eine Funktion, mit der das Erzeugen neuer Kreise angestoßen werden kann,
- eine Möglichkeit zur Speicherung der Linien- und Kreisobjekte im Dokument,
- verschiedene Ressourcen, wie zum Beispiel Menü-Einträge und Cursor

Zunächst zu den wichtigsten Elementen: dem Kreis- und dem Linien-Objekt. Diese Objekte sind auf eine recht ähnliche Art abbildbar, denn beides sind geometrische Formen, die sich durch zwei Punkte beschreiben lassen. Im Fall der Linie sind das der Start und der Endpunkt, und im Fall des Kreises sind das der Mittelpunkt sowie ein Punkt auf dem Kreis. (Im Beispiel wird zwar in Wirklichkeit eine Ellipse gemalt, doch tut das der Sache keinen Abbruch.)

Nun sollen beide Elemente am Bildschirm durch den späteren Anwender interaktiv gezeichnet werden. Das bedeutet für den Programmierer, dass diese Objekte in nur teilweise vollständigem Zustand lebensfähig sein müssen.

Bei einer Linie soll sich das Beispiel wie folgt verhalten: Der Anwender des Programms wählt den Befehl zum Erzeugen einer neuen Linie. Dadurch soll der Mauscursor eine neue Form annehmen, um dem Anwender mitzuteilen, dass er nun mit der Definition der Linie beginnen kann. Außerdem muss zu diesem Zeitpunkt bereits ein Linienobjekt erzeugt werden – es hat allerdings noch keinen Start- und keinen Endpunkt und kann dementsprechend auch noch nicht gemalt werden.

Nun bewegt der Anwender die Maus über der Arbeitsfläche und klickt mit dem linken Mausknopf an irgendeine Stelle. Diese Aktion muss mehrere Dinge auslösen: Das Linien-Objekt geht nun in einen neuen Zustand

über, denn es besitzt jetzt einen Startpunkt, allerdings noch keinen Endpunkt. Bewegt der Anwender die Maus weiter über die Arbeitsfläche soll ständig eine Linie ausgegeben werden, die

über, denn es besitzt jetzt einen Startpunkt, allerdings noch keinen Endpunkt. Bewegt der Anwender die Maus weiter über die Arbeitsfläche soll ständig eine Linie ausgegeben werden, die



den bereits festgelegten Startpunkt als den einen Endpunkt und die aktuelle Position der Maus als den anderen Endpunkt verwendet. Klickt der Anwender nochmals mit der linken Maustaste, soll das Linienobjekt wieder in einen neuen Zustand übergehen. Es ist nun vollständig definiert und kann in der Dokumenten-Klasse gespeichert werden. Ferner können nun alle Views mit den neuen Daten upgedated und der aktuelle laufende Befehl beendet werden.

Soviel zur Theorie beim Programmablauf – nun ist das Ganze in Code zu gießen. Nachdem das Linien- und das Kreisobjekt im Wesentlichen die gleichen Eigenschaften haben, bietet es sich für diese Objekte an, eine Basisklasse zu implementieren. Diese trägt im Beispielcode den Namen "Graphic".

Ein Graphik-Objekt hat im Wesentlichen drei Eigenschaften: Zwei davon sind die Punkte, die später entweder eine Linie oder einen Kreis definieren, die dritte Eigenschaft beschreibt den aktuellen Zustand des Objekts: Es hat entweder noch gar keine Daten, oder aber, es befindet sich im Zustand der Initialisierung. (Der Anwender hat zwar be-

reits einmal aber noch nicht das zweite Mal geklickt.) Die Punkte müssen ferner von außen gesetzt werden können,

der Zustand muss von außen sowohl erfragbar als auch setzbar sein. Ferner sollten die Punkte den abgeleiteten Klassen

LISTING 1

```
class Graphic : public
↳CObject
{
public:
    ..Graphic()
    ..{
        ....m_state = State::S_NONE;
    ..}
    ..virtual void Draw( CDC*
↳pDC) = 0;
    ..virtual int OnSetCursor()
    ..{
        ....return 0;
    ..}
    ..enum State
    ..{
        ....S_NONE,
        ....S_INIT,
        ....S_DRAWING,
        ....S_FINISH
    ..};
    ..void SetState( State s)
    ..{
        ....m_state = s;
    ..}
    ..State GetState() const
    ..{
        ....return m_state;
    ..}
    ..void SetPoint1( CPoint p)
    ..{
        ....m_ptBegin = p;
    ..}
    ..void SetPoint2( CPoint p)
    ..{
        ....m_ptEnd = p;
    ..}
private:
    ..State m_state;
protected:
    ..CPoint m_ptBegin;
    ..CPoint m_ptEnd;
};
```

DIE VOM ASSISTENTEN ERZEUGTEN DATEIEN

Wenn Sie eine MDI-Anwendung mit dem Anwendungs-Assistenten erzeugen, produziert dieser Vorgang eine ganze Reihe von Dateien. Nachdem es immer hilfreich ist zu wissen, was sich in welcher Datei befindet, hier die kurze Übersicht.

ReadMe.Txt:

In dieser Datei sind einfach nur ein paar grundlegende Informationen über die Art der erzeugten Anwendung abgelegt.

Stdafx.h:

Diese Header-Datei wird für die einfachere Verwendung von vorkompilierten Header-Dateien erzeugt. Diese Art der Compilierung führt zu einer besseren Übersetzungsgeschwindigkeit beim Bauen der Anwendung.

Stdafx.cpp:

Auch diese Datei wird nur für die vorkompilierten Header benötigt.

resource.h:

Diese Datei enthält die Präprozessoren-Definition der vom Assistenten automatisch erzeugten Ressourcen.

MainFrm.h:

In diesem Header ist das Hauptfenster der Anwendung deklariert. Innerhalb des

Hauptfensters befinden sich das Menü und alle Fenster, die den Dokumenteninhalte anzeigen.

MainFrm.cpp:

Diese Datei enthält die Definition der Methoden und Klassen aus MainFrm.h

ChildFrm.h und ChildFrm.cpp:

Diese beiden Dateien enthalten die Deklaration und Definition des "ChildFrame" Windows. Dieses Window ist unmittelbar im MainFrame Fenster eingebettet und verwaltet die einzelnen Dokumenten-Fenster.

???View.h und ???View.cpp:

Diese beiden Dateien enthalten die Deklaration und die Definition der Views Ihrer Anwendung. Die Views sind die Fenster, in denen die Daten visualisiert werden.

???Doc.h/???Doc.cpp:

Diese beiden Dateien enthalten die Deklaration und Definition des "Dokuments" Ihrer Anwendung. Das Dokument ist das Objekt, das für die Verwaltung der Daten Ihrer Anwendung zuständig ist.

???h / ???cpp:

Diese beiden Dateien enthalten die Deklaration und Definition Ihrer Anwendung. Alle Anwendungen bei MFC müssen eine Ins-

tanz einer von der MFC-Anwendungsklasse abgeleiteten Klasse haben. Dabei handelt es sich auch um die Instanz, die das tatsächliche Programm ausmacht. Gewinnt diese Instanz Scope, so wird das Programm gestartet, verliert sie den Scope, so wird das Programm beendet.

???rc:

In dieser Datei befinden sich die vordefinierten Ressourcen in einem Format, das sowohl von Mengen als auch von den Resource-Editoren des VC++ verstanden werden kann. Trotzdem sollten Sie diese Datei immer nur mit dem Resource-Editor und nicht mit dem Texteditor bearbeiten.

???dsw und .dsp:

Das sind die Workspace und die Projekt-Dateien. Nachdem Sie nur ein Projekt in Ihrem Workspace haben, liegt auch nur eine Projekt-Datei vor. Die Projekt-Datei enthält Informationen über die am Projekt beteiligten Dateien und die zugehörigen Compiler-Einstellungen.

Ferner erzeugt der Assistent noch ein "Res"-Verzeichnis. Darin befinden sich die Ressourcen, die nicht in textueller Form abbildbar sind. Dazu gehört zum Beispiel auch das Icon des Programms.

direkt zur Verfügung stehen, während der Zustand nur in der Basisklasse direkt verfügbar sein soll. Das führt zum

Entwurfsvorliegt. Enthält eine Klasse eine Funktion, die pure virtual ist, nennt man die Klasse eine "abstrakte" Basisklasse, denn es ist nicht möglich, eine solche Klasse zu instanzieren: Es können nur Instanzen abgeleiteter Klassen, in der die reine virtuelle Funktion auch implementiert ist, hergestellt werden. In diesem Fall ist die pure-virtual-Methode eine Methode mit dem Namen *Draw()*. Diese Methode muss also auf jeden Fall von den abgeleiteten Klassen implementiert werden – der Grund dafür ist einleuchtend, denn in dieser Funktion soll die Geometrische Form ausgegeben werden. Wie aber eine konkrete geometrische Form anhand der beiden Punkte entsteht, das wissen nur die konkreten ab-

geleiteten Klassen. Hier gibt es aber eine Default-Implementierung, die den Wert "0" liefert. Wird die Funktion später in einer der abgeleiteten Klassen implementiert, wird sich zur Laufzeit des Programms der Cursor nach dem ersten Klick verändern (beim Kreis ist das der Fall), während er sich bei der Klasse, die diese Funktion nicht implementiert nicht verändert.

Schließlich sehen Sie noch eine *Get*- und eine *Set*-Funktion für den State, also den Zustand des Graphic-Objekts sowie zwei Funktionen zum Setzen der beiden Punkte. Fertig ist die Basisklasse.

Nun kommen die abgeleiteten Klassen mit den Namen "Line" und "Circle". Beide implementieren die virtuelle Funktion *Draw()*. Bei der Linie sieht diese Funktion so aus:

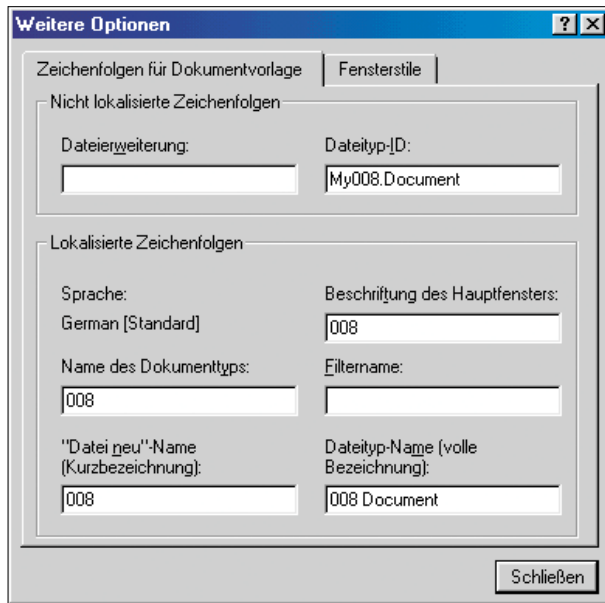
```
void Draw( CDC* pDC)
{
    pDC->MoveTo( m_ptBegin);
    pDC->LineTo( m_ptEnd);
}
```

Beide erhalten einen Parameter vom Typ Zeiger auf CDC. Dabei steht "CDC" für "C Device Context". (Das vorgestellte C ist ohne Bedeutung und bei allen MFC-Klassen Konvention.) Dieser "Gerätekontext" enthält Methoden, mit denen gezeichnet werden

kann. Auch von dieser Klasse sollten Sie einmal die Online-Hilfe bemühen, denn die gebotenen Möglichkeiten gehen doch weit über die Ausgabe eines einfachen Striches mit einer Startposition (*MoveTo*) und einer Endposition (*LineTo*) hinaus.

Bei der Implementierung von *Draw* in der Kreisklasse kommen noch die Hilfsklassen *CSize* und *CRect* zum Zuge. Diese kapseln eine Größe und ein Rechteck und werden an dieser Stelle zur Berechnung des Rechtecks verwendet, das den Kreis einschließt.

Das ist darum notwendig, weil die entsprechende Funktion zur Anzeige eines Kreises des CDC eben vier Eckpunkte eines umhüllenden Rechtecks (hier Quadrat) als Parameter erwartet.



BEI DEN WEITEREN OPTIONEN des Projekts können Sie im Assistenten auch noch angeben, wie die Datei-Erweiterung der Dateien Ihres Programms lauten soll.

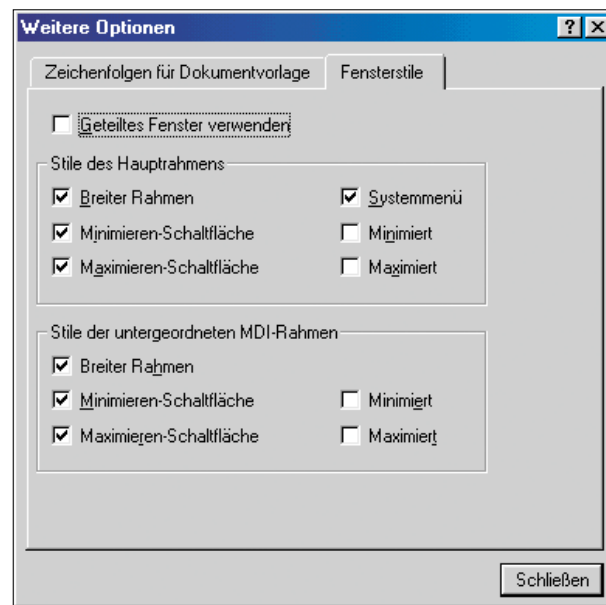
Aufbau der Klasse, wie Sie es in Listing 1 sehen.

Hier können Sie verschiedene bisher noch nicht angesprochene Sprachelemente von C++ sehen. Zunächst einmal wird die "Graphics"-Klasse von einer Klasse namens "CObject" abgeleitet. Das ist bei Objekten, die zu speichernden Daten enthalten innerhalb einer MFC-Anwendung immer sinnvoll – auch wenn das in diesem Beispiel nicht notwendig gewesen wäre. Das Ableiten von "CObject" hat mehrere Implikationen und eine davon ist die, dass ein solches Objekt sehr einfach gespeichert und geladen werden kann. Das wird aber im Beispielprogramm gar nicht getan.

Im Folgenden sehen Sie den Konstruktor des Graphic-Objekts. Dieser tut nicht sehr viel, sondern initialisiert nur die Member-Variable *m_state*, die den aktuellen Zustand eines Graphic-Objekts aufnimmt mit *S_NONE*. Dieser State stammt aus einer Enumeration, die ebenfalls Teil des Graphic-Objekts ist.

Dann sehen Sie zwei Funktionen, die beide mit dem Schlüsselwort "virtual" eingeleitet werden. Die eine enthält außerdem hinten den Zusatz "=0". Eine solche Funktion nennt man "pure virtual", weil keinerlei Default-Implementierung vorliegt.

Enthält eine Klasse eine Funktion, die pure virtual ist, nennt man die Klasse eine "abstrakte" Basisklasse, denn es ist nicht möglich, eine solche Klasse zu instanzieren: Es können nur Instanzen abgeleiteter Klassen, in der die reine virtuelle Funktion auch implementiert ist, hergestellt werden. In diesem Fall ist die pure-virtual-Methode eine Methode mit dem Namen *Draw()*. Diese Methode muss also auf jeden Fall von den abgeleiteten Klassen implementiert werden – der Grund dafür ist einleuchtend, denn in dieser Funktion soll die Geometrische Form ausgegeben werden. Wie aber eine konkrete geometrische Form anhand der beiden Punkte entsteht, das wissen nur die konkreten ab-



MIT DIESEM DIALOG legen Sie das Startverhalten Ihres Programms fest und bestimmen außerdem welche Elemente die einzelnen Fenster haben sollen.

geleiteten Klassen: Bei der Kreisklasse ist das logischerweise anders, als bei der Linienklasse.

Die andere virtuelle Funktion wird später zum Setzen des Cursors verwen-



Nun sollen später erzeugte Elemente im Dokument gespeichert werden, und das bietet sich in Form einer Liste an. Auch typensichere Listen können sie mit MFC anlegen, dazu gibt es eine passende Template-Klasse. Um diese verwenden zu können, müssen Sie in der Datei `stdafx.cpp` allerdings eine weitere Header-Datei inkludieren, und zwar den Header `afxtempl.h` (für AFX Templates).

Nachdem die templatisierten Listenklassen aber immer ein wenig aufwändig hingeschrieben werden müssen, sollten Sie für Ihre Liste aus `Graphic`-Objekten besser noch einen Typen definieren. Das geht mit dem folgenden Statement, das Sie auch in der Datei `graphic.h` vorfinden, in dem beim Beispielcode die drei Klassen definiert sind:

```
typedef CTypedPtrList
<CPtrList, Graphic*> GPtrList;
```

So viel also zu den Objekten, die im Programm erzeugt werden. Nun brauchen Sie noch eine Methode, um diese Objekte zu speichern, und dafür ist die Dokumenten-Klasse da. Allerdings wollen Sie die Objekte später nicht nur im Dokument speichern, sondern auch darüber iterieren können. Das ist zum Beispiel dann notwendig, wenn Sie alle im Dokument abgelegten Objekte anzeigen wollen.

Zunächst müssen Sie also im Header-File der Dokumenten-Klasse ein neues Member einfügen. Das tun Sie im `private`-Bereich und nennen dieses Member `m_list`. Als Datentyp verwenden Sie den per `typedef` zuvor definierten Typ

LISTING 2

```
void AddObject( Graphic* g)
{
    m_list.AddTail( g);
}
POSITION GetHeadPosition()
{
    return m_list.GetHeadPosition();
}
Graphic* GetNext( POSITION& pos)
{
    return m_list.GetNext( pos);
}
```

`GPtrList`. Dann erweitern Sie das Dokument um eine Methode zum hinzufügen eines neuen Objekts sowie um Methoden zum Iterieren über alle Objekte in der Liste – natürlich als öffentliche Methoden, also „public“ (siehe Listing 2).

Damit wäre auch die Speicherung der Objekte geklärt. Nun müssen Sie noch erzeugt und angezeigt werden. Das passiert in der View-Klasse.

Objekte erzeugen

Zum Erzeugen neuer Objekte benötigen Sie auch einen – bzw. zwei – Befehle in Ihrem Menü. Die fügen Sie mit dem Menü-Editor einfach ein, zum Beispiel unter der Bezeichnung *Kreis erzeugen* und *Linie erzeugen*. Diese Befehle müssen noch mit Quellcode in Ihrer View verbunden werden: Dabei ist es erwünscht, wenn die Auswahl des entsprechenden Befehls dazu führt, dass eine Methode aus Ihrer View-Klasse aufgerufen wird.

Die darunterliegenden Logik, die sich darum kümmert, dass

genau das passiert, besorgt MFC. Alles was Sie also tun müssen, ist die Zuordnung herzustellen. Damit beginnen Sie, indem Sie im Menü-Editor mit der rechten Maustaste auf einen der neu eingefügten Menü-Befehle klicken und den Befehl *Klassen Assistent* auswählen. Dieser Assistent wird dann gestartet und präsentiert sich in Form einer Dialogbox.

Hier ist der symbolische Name des von Ihnen ausgewählten Befehls schon ausgewählt. Sie müssen nun noch die Klasse auswählen, die die Befehlsbearbeitung erledigen soll. Das ist in diesem Fall Ihre View-Klasse. Danach klicken Sie auf den Button zum Erzeugen und Bearbeiten der Methode – und laden dann ganz von selbst im Quellcode-Editor innerhalb der Funktion, deren Namen Sie noch im Klassen-Assistenten angegeben haben. Das Gleiche machen Sie auch mit dem zweiten Menü-Befehl: Hier vergeben Sie aber einen anderen Funktionsnamen.

In der Funktion zum Erzeugen einer neuen Linie legen Sie nun als erstes eine neue Linie an. Die müssen Sie sich aber irgendwo merken, und dazu verwenden Sie einen Zeiger auf `Graphic`, den Sie als privates Member Ihrer View-Klasse definieren. Nachdem die neue Linie zunächst in einem undefinierten Zustand ist, teilen Sie dies auf dem Objekt mit:

```
m_pGraphic = new Line();
m_pGraphic->SetState(
    Graphic::S_NONE);
```

Außerdem soll der Anwender ein visu-

MDI ODER SDI – DIE UNTERSCHIEDE

Im Wesentlichen gibt es zwei unterschiedliche Arten von Windows-Anwendungen: Solche, die nur aus einer Dialogbox bestehen und solche, die echte Fenster haben, die man in der Größe verändern kann. Bei letzteren unterscheidet man zusätzlich zwischen einem „MDI“- und einem „SDI“-Interface. Das MDI steht dabei für „Multiple Documents Interface“, während SDI „Single Document Interface“ bedeutet.

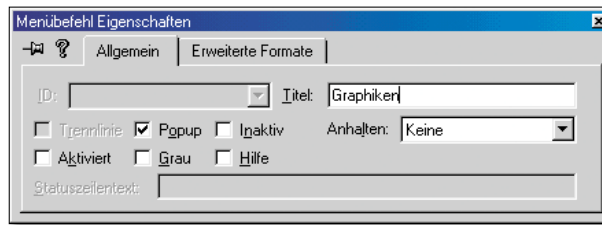
Eine MDI-Anwendung kommt also mit mehreren geöffneten Dokumenten gleichzeitig klar, während eine SDI-Anwendung immer nur ein Dokument auf einmal geöffnet haben kann. Will man mehrere Dokumente mit einer SDI-Anwendung gleichzeitig bearbeiten, müssen eben mehrere Kopien der Anwendung geladen werden.

Ein Beispiel für eine typische MDI-Anwendung sind alte Versionen von Word für Windows, während aktuelle Versionen von Word typische Vertreter des SDI-Ansatzes sind. Prinzipiell ist es so, dass MDI-Anwendungen ein wenig komplizierter zu

programmieren sind, als SDI-Anwendungen – das ist aber meist nicht der Grund dafür, weshalb sich ein Programmierer für das eine oder das andere Modell entscheiden sollte. Der wahre Grund ist eher eine Frage des User-Interface-Designs: Kommt der Zielanwender eher damit klar, immer nur ein Dokument bearbeiten zu können, oder kann er auch ein Programm bedienen, das mehrere Dokumente gleichzeitig geöffnet haben kann. Mit diesem Wissen kann man übrigens auch leicht die Kompetenz der „Word“-Kunden von Microsoft ermitteln.

Bei MFC ist die Unterscheidung zwischen SDI und MDI noch aus einem anderen Grunde wichtig. Bei einer SDI-Anwendung ist es nämlich deutlich schwieriger mehrere Fenster zu verwalten – auch wenn man nur ein Dokument verwenden will: Im Wesentlichen läuft es dabei darauf hinaus, dass man mit MFC immer eine MDI-Anwendung bauen sollte, wenn man sein Dokument in mehreren Fenstern gleichzeitig anzeigen lassen will.

elles Feedback erhalten. Das ist am einfachsten, indem Sie den Cursor entsprechend verändern. Dazu müssen Sie zunächst einmal einen Cursor haben. Den erzeugen Sie mit dem entsprechenden Editor, und der startet, wenn Sie im Arbeitsbereich unter den Ressourcen einen neuen Cursor per rechter Maustaste einfügen. Abgesehen davon, dass Sie den Cursor zeichnen müssen, müssen Sie



IM MENU-EDITOR können Sie nicht nur den Text für den Menübefehl, sondern auch den Text für die Statuszeile festlegen.

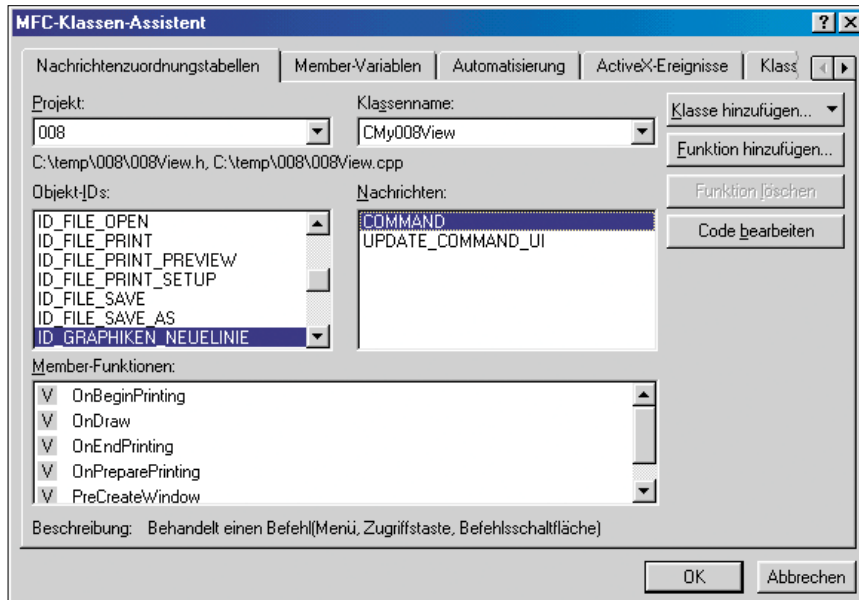
Anlegen einer neuen Linie reagiert – nur passiert sonst noch nicht viel. Den Rest der Arbeit erledigen Sie in zwei Methoden der View-Klasse, in denen Sie die Windows Events für Mausektionen behandeln.

Alle Aktionen, die

im System ausgeführt werden, lösen Ereignisse (Events) aus, und alle Ereignisse die Ihre Anwendung betreffen können Sie behandeln. Dazu gehört es zum Beispiel auch, wenn der Mauscursor über Ihrem Anwendungsfenster bewegt wird. Wenn Sie an einem Ereignis interessiert sind – und in diesem Fall sind Sie an der Mausektion und an einer gedrückten linken Maustaste interessiert – teilen Sie das dadurch mit, dass Sie einen Event-Handler für das entsprechende Ereignis programmieren. Am einfachsten ist das wiederum mit dem Klassen-Assistenten, denn dort können Sie sowohl die Klasse auswählen, die das Ereignis behandeln soll, als auch das zu behandelnde Ereignis selbst auswählen.

In diesem Fall wählen Sie als Klasse wieder die View-Klasse aus, und dann erzeugen Sie für die Events MOUSEMOVE und LBUTTONDOWN eine Methode. Diese erhalten die Namen *OnLButtonDown()* und *OnMouseMove()* und sind nun

nur noch mit Code zu füllen. Die einfachere der beiden Methoden ist *OnMouseMove()*. Hier müssen Sie nur überprüfen, ob der Zeiger auf das Graphic-Objekt gesetzt oder 0 ist. Ist der Zeiger gesetzt, und ist der Zustand des Graphik-Objekts nicht *S_NONE*, muss es aufgrund der Mausektion neu gezeichnet werden. Das geht, indem Sie dem Objekt einen neuen zweiten Punkt geben, und dann die Zeichenfläche invalidieren. Dadurch wird einfach alles neu gezeichnet (siehe dazu Listing 3 auf der nächsten Seite).



MIT DEM KLASSEN-ASSISTENT binden Sie Programmcode an Ressourcen. So legen Sie hier zum Beispiel fest, welche Funktion bei einem Klick auf einen Menübefehl aufgerufen werden soll.

ihm auch einen symbolischen Namen geben, und den brauchen Sie gleich danach. Unterhalb der Zeile in der View-Klasse, in der Sie das neue Linien-Objekt in den Zustand *S_NONE* versetzt haben, laden Sie nämlich nun den Cursor und teilen Windows mit, dass Sie ihn verwenden möchten. Damit das gelingt, müssen Sie zunächst eine spezielle Hilfsfunktion der Window-Klasse namens *SetCapture()* aufrufen:

```
SetCapture();
```

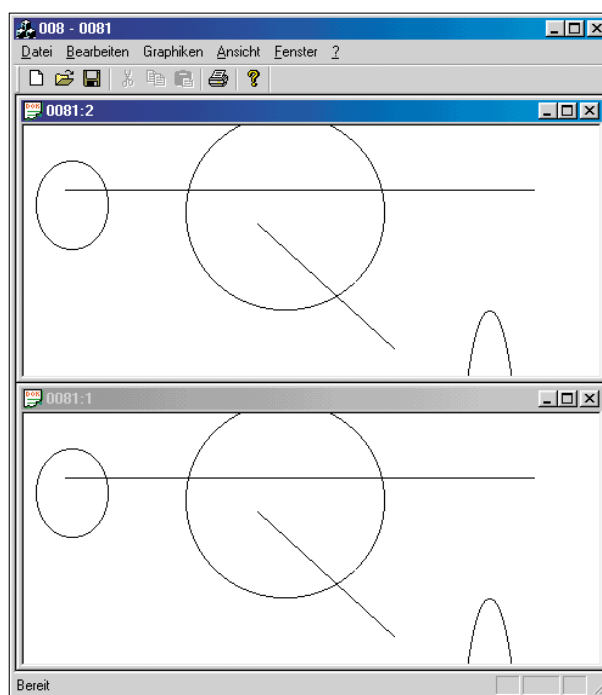
Dann laden Sie den Cursor unter Verwendung des symbolischen Namens mit einer Hilfsfunktion aus der Anwendungsklasse. Einen Zeiger auf Ihre Instanz dieser Klasse erhalten Sie wiederum mit der Hilfsfunktion *AfxGetApp()*:

```
HCURSOR hc = AfxGetApp() ->
    LoadCursor(
        IDC_CURSOR_LINE);
```

Schließlich müssen Sie den Cursor noch setzen, und das geht mit Funktion *SetCursor()*:

```
::SetCursor(hc);
```

Nun wird also bereits auf den Befehl zum



DAS FERTIGE MALPROGRAMM kann Kreise und Linien in mehreren Fenstern gleichzeitig anzeigen. Gemalt wird natürlich interaktiv.



Die Behandlung des Button-Down-Events ist ein wenig aufwändiger, denn hier müssen Sie zwei Fälle behandeln: Entweder, es handelt sich um den ersten Mausklick für das ak-

tuelle Objekt, oder es handelt sich bereits um den zweiten, abschließenden Klick.

Im ersten Fall müssen Sie nur den ersten Punkt im Objekt setzen und bei Be-

darf einen neuen Cursor installieren (siehe Listing 4).

Im zweiten Fall müssen Sie natürlich den zweiten Punkt im Objekt setzen, und dann das Objekt im Dokument eintragen. Ferner müssen Sie den Zeiger auf das Graphic-Objekt aus der View auf 0 setzen (sonst wird es immer weiter gemalt) und den Cursor auf seinen Ausgangszustand zurücksetzen (siehe dazu Listing 5).

Schließlich müssen Sie sich noch darum kümmern, dass alle aktuell geöffneten View-Fenster neu ausgegeben werden, damit das neu hinzugekommene Objekt auch sichtbar ist:

```
pDoc->UpdateAllViews( this );
```

Wenn Sie dies nun ausprobieren, werden Sie allerdings einen Umstand recht schnell bemerken: Man sieht nichts, von Linien oder Kreisen weit und breit keine Spur. Das hat auch einen guten Grund, denn Sie haben zwar die komplette Infrastruktur für das Erzeugen eines neuen Objekts und auch für die Speicherung im Dokument programmiert – nur Code, der nun wirklich das Zeichnen besorgt, den gibt es nicht.

Das mag nun ein bisschen verwirrend sein, denn schließlich haben Ihre beiden von Graphic abgeleiteten Klassen eine *Draw()*-Methode. Doch aufrufen müssen Sie sie selbst und zwar in der Methode *OnDraw()* Ihrer View-Klasse, die vom Assistenten bereits erzeugt worden ist.

Der benötigte Code ist hier aber trivial. Alles, was Sie brauchen, ist ein Zeiger auf Ihr Dokument – mit dem können Sie dann über alle im Dokument befindlichen Objekte iterieren (siehe dazu Listing 6).

Nun wird die ganze Sache schon etwas sichtbarer: Sowohl das interaktive Eingeben, als auch das Anzeigen in mehreren Ansichten gleichzeitig funktioniert: Fertig ist Ihr erstes eigenes Windows-Programm. ✓ UR

LISTING 3

```
void CMy008View::OnMouseMove(UINT nFlags, CPoint point)
{
    ..if( m_pGraphic)
    ..{
        ....if( m_pGraphic->GetState() != Graphic::S_NONE)
        ....{
            .....m_pGraphic->SetPoint2( point);
            .....Invalidate();
            ....}
        ..}
    ..CView::OnMouseMove(nFlags, point);
}
```

LISTING 4

```
if( m_pGraphic->GetState() == Graphic::S_NONE)
{
    m_pGraphic->SetPoint1( point);
    m_pGraphic->SetState( Graphic::S_INIT);
    int idCursor = m_pGraphic->OnSetCursor();
    if( idCursor != 0)
    {
        ::SetCursor( AfxGetApp()->LoadCursor( idCursor));
    }
}
```

LISTING 5

```
m_pGraphic->SetPoint2( point);
CMy008Doc* pDoc = GetDocument();
pDoc->AddObject( m_pGraphic);
m_pGraphic = 0;
SetCursor( NULL);
ReleaseCapture();
```

LISTING 6

```
CMy008Doc* pDoc = GetDocument();
for( POSITION pos = pDoc->GetHeadPosition(); pos!=NULL;)
{
    Graphic* g = pDoc->GetNext( pos);
    g->Draw( pDC);
}
if( m_pGraphic) m_pGraphic->Draw( pDC);
```

